Tree-Based Symmetric Key Broadcast Encryption

A thesis submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the degree of
Doctor of Philosophy in Computer Science

Author:
Sanjay Bhattacherjee

Supervisor:
Prof. Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B. T. Road, Kolkata,
West Bengal, India - 700 108.

To my family.

Acknowledgements

I have always believed that life has been very kind to me. I have probably almost always got more than I deserved. This thesis is a culmination of all that I have earned through my associations with the people around me. I take up this opportunity to account for all that I have received from the innumerable contributors in my life till date. This is more of a remembral and hence is in no way exhaustive accounting.

After my graduation, I worked in the industry for two years. I joined the M. Tech. (CS) course in 2007 at the Indian Statistical Institute, Kolkata. Prof. Palash Sarkar was one of the faculty members teaching in the first semester. Within the first few days of attending Palash-da's classes, I knew that he was one of those people I would aspire to be like. It has been a privilege being his student and then an even more enriching experience to have him as my thesis supervisor first during M. Tech. and then during PhD. His inspiration has played a consistent role in shaping up my academic contours. While working together, he knew almost every time how to get the best out of me. His meticulous discipline, academic depth and breadth, clarity of thought and consistent effort are few of his many qualities I would like to succeed in emulating some day. Most cherished of all that I have received from him is his unending support, guidance and encouragement - academic and otherwise.

My PhD days would not have been as enjoyable without the presence of Dr. Kishan Chand Gupta. Academic discussions with Kishan-da are always very revealing because of his practice of digging things up to insane depths. He has been a patient teacher, an ever supportive friend, and a guide in many ways of life.

I extend my sincerest gratitude to Prof. Bimal Roy who has been providing relentless support in driving forward the research on Cryptology happening in India and ISI Kolkata becoming its largest hub. I shall be grateful to Prof. Rana Barua and Dr. Mridul Nandi for their courses and talks that shaped my understanding of Cryptology in many ways. Rana-da's humble ways and incredible capability of abstraction and Mridul-da's emphasis on details of concepts are things I would love to inculcate. I have also learnt many aspects of the subject from Prof. Subhamoy Maitra. I adore his capability to portray difficult concepts in a very lucid manner. I would like to thank Subhamoy-da for his support during my years at ISI.

Dr. Sanjit Chatterjee provided me the opportunity for a short academic visit to IISc. During the three weeks I spent at IISc, I underwent intense reading and had very fruitful discussions with Sanjit-da. I would love to pursue those directions of research in near future. I would also like to thank Dr. Soumitra Sanadhyay, Dr. Goutam Paul, Dr. Sushmita Ruj, Dr. Sumanta Sarkar for the useful discussions I have had with them during the course of my PhD.

Some people are so much a part and parcel of one's life that it becomes hard to separate out their contributions. Somindu C. R. and Subhabrata Samajder are two such people. We have been friends since our M. Tech. days and their continuous support has enabled me keep my chin up even during the most difficult times.

My return to academics is largely contributed to two individuals - Srimanta Bhattacharya and Dinesh Layek. They were my colleagues at my workplace in Kolkata who inspired me to pursue higher studies. I will always be thankful to them that they found me worthy of their advice.

Having seniors like Dr. Rishiraj Bhattacharya and Dr. Sumit Pandey has been a privilege all through. I got one of my first insights of Cryptology from Sumit-da. His unending support and helping nature has earned him reverence from all his friends and juniors including me. Rishi-da has come up with crucial advices at important junctures of my ISI life. Starting from the first day we sat on the football ground together to this day, he has always pushed me ahead and helped me take longer strides.

Having friends like Avik Chakraborty, Binanda Sengupta, Indranil Ghosh Ray, Nilanjan Dutta, Raju Maiti and Tapas Pandit at one's workplace makes life very enjoyable. A few words with friends like Dr. Sourav Sengupta, Dr. Santanu Sarkar, Kaushik Chakraborty, Mrinal Nandi, Samiran Bag, Satrajit Ghosh, Shashank Singh and Subhadeep Banik feels refreshing. Dr. Ashish Coudhury and Dr. Arpita Patra have visited ISI on many occasions. Knowing them has been a pleasure all along. During my travel to the Seventh International Workshop on Coding and Cryptography, Paris in April 2011, Ashish-da ensured that I had very little to worry about.

I would like to thank the Cryptology Research Society of India for all its support. A special thanks to Amitabha Sinha for his tireless cooperation in making life at ISI very smooth. I am thankful to the Director's Office, the Dean's Office, the ASU Office and other administrative offices at ISI for their cooperation all through.

Dr. Mandar Mitra is one of those teachers in ISI who have influenced me most. His technical soundness was just the first of the numerous impressions he has left on me. His

eloquence and depth of knowledge made me enjoy his classes to the brim. But above all, I would be ever indebted to him for being a living personification of humility and modesty in my life. Prof. Sandip Das is another such teacher. I have seen no one try harder to get the best out of even the most inattentive student as Sandip-da used to. His untiring efforts and dedication as a teacher would be a model to follow. I have been extremely lucky to get teachers like Prof. Aditya Bagchi, Prof. Amitabha Bandopadhyay, Prof. Bhabani P. Sinha, Prof. Bhargab B. Bhattacharya, Prof. Krishnendu Mukhopadhyay, Prof. Subhash Nandy, Dr. Arijit Bishnu, Dr. Pinakpani Pal and Dr. Utpal Garain.

Although I did not get an opportunity to attend the classes of Prof. Bhabotosh Chanda, my first academic tryst with ISI was due to him. During my graduation, I thoroughly enjoyed doing a summer training at his lab. It was a pleasure having academic discussions with him. I would like to extend a special note of thanks to Prof. Sushmita Sur Kolay and Nandakumar Narayanmangalam for helping me get an internship opportunity at ARM during my M. Tech. It was there I had my first practical experience of doing research work.

I have had pleasing associations with quite a few other faculty members of ISI. Prof. Anup Diwanji, Prof. Indraneel Dasgupta, Prof. Mousumi Bose, Prof. Pradipta Bandopadhyay, Prof. Smarajit Bose, Prof. Sumitra Purkayastha, Prof. Tapas Samanta, Dr. Ansuman Banerjee, Dr. Atanu Ghosh, Dr. Diganta Mukherjee, Dr. Indranil Mukhopadhyay, Dr. Samarjit Das, Dr. Sourabh Ghosh and Dr. Swagatam Das are amongst them. I would like to thank ISI, its faculty members and other staff, students, research scholars and others for providing and maintaining an amicable environment for studies and research.

Talking of ISI and the time I spent here, an absolutely inseparable thing would be the friends I have made here. It all started on the days during admission to M. Tech. and then during the first few classes. Amit Tripathi, Ayyappadas A. M., Dr. Aritra Banik, Dr. Chiranjit Chakraborty, Dr. Sandeep Kumar Dey, Dr. Kalikinkar Mandal, Dr. Nargis Pervin, Dr. Pulak Purkait, Mrinmoy Ghorai, Rajnish Kumar, Santanu Bhowmick, Swarup Chattopadhyay and all other M. Tech. batch-mates, seniors and juniors were an integral part in shaping up my life then on. From group-study sessions to the Bonhooghly excursions to the most adventurous campus endeavors - we have been accomplices in everything. Being part of the numerous extra-curriculars especially our annual cultural event "All-Go-Rhythm" (a part of which was the play "Bheem Badh") was one of the most memorable times at ISI.

I have spent hours at a stretch at the research scholars' hostel sharing very memorable times with Dr. Debasmita Basu, Dr. Navonil De Sarkar, Dr. Sayantan Dutta, Dr. Srikanta

Kundu and Debojyoti Mazumdar. I have had an absolutely enjoyable hostel life of more than five years, that has endowed me with very dear friends. Some of them are Aleya Bose, Amiya Bhowmick, Anindita Chatterjee, Apurba Das, Arnab Hazra, Arvind Nambiar, Bidesh Das, Biman Chakraborty, Bipul Islam, Chintan Parmar, Debanjan Majumdar, Dr. Abhijit Mandal, Dr. Aparajita Bakshi, Dr. Buddhananda Banerjee, Dr. Kaushik Kundu, Dr. Kushal Banik Choudhury, Dr. Minati De, Dr. Prosenjit Das, Dr. Rajat Subhra Hazra, Dr. Rituparna Basak, Dr. Sabyasachi Mukhopadhyay, Dr. Shamsher Singh, Dr. Sourav Rana, Dr. Sriparna Ganguly, Dr. Subhendu Chakraborty, Dr. Suchismita Roy, Dr. Sudip Samanta, Dr. Sutripta Sarkar, Dr. Trishita Ray Barman, Gopakumar Achuthankutty, Hirak Sarkar, Kavita Suresh Parab, Keya Dutta, Koustav Sarkar, Mahamitra Das, Mannu Dwivedi, Manusree Mahato, Minerva Mukhopadhyay, Mithun Bhowmik, Narayan Rakshit, Parikshit De, Paromita Dubey, Pramita Bagchi, Prasenjit Ghosh, Priyam Biswas, Rajashree Bhattacharyya, Reshmi Mitra, Ritwik Bhattacharya, Roshni Roy, Ruchira Biswas, Sandip Sarkar, Sayan Roy, Sebanti Sengupta, Sedigheh Mirzaei Salehabadi, Shivani Santosh, Somnath Ghatak, Sourav Kumar Sasmal, Suman Sarkar, Tanmay Das, Tomojoy Ghosh and Tridip Sardar. Anindita Ray, Atanu Ghosh, Arindam Pal, Raghu Teja, Dr. Shalini Datta, Esita Chattopadhyay, Richa Singh, Partha Pratim Kundu, Sarmistha Das, Shrinka Sen, Soham Chakraborty, Soumen Nandi and Tanmay Basu are some of the dear ones I made friends with beyond the hostel walls.

I can not forget the contribution of all my teachers from Don Bosco School, Liluah, in shaping my basic mettle. I would also like to thank the faculty members of the Dept. of Information Technology, Jadavpur University for introducing me to the basics of computer science. I have been very lucky to get life-long friends at DBL and JU. Tanmoy Roychoudhury is one of the most treasured of those friends. One out of the innumerable times Tanmoy has helped me was during my travel to New York to attend the Real World Cryptography Workshop in January 2014. Tanmoy and his wife Ritwika happily endured a lot of pain in helping me arrange my stay there that in turn took away all my worries.

A special word of thanks goes out for Mehlam Shakir, Atanu Ghosh and Pradip Biswas. Mel is a prolific entrepreneur. I thoroughly enjoyed working in his project at Dynamic Digital Technology, Kolkata. Other than the fact that he was a very supportive client, he kept emphasizing on the importance of solving problems in baby steps. I shall always remember this lesson in life. I did learn a lot from Atanu-da and Pradip-da who were my project managers at the company. Vinay Gupta, the General Manager of the company was always very supportive.

I was lucky to be trained in the sport Taekwondo by Grandmaster Pradipta Kumar Roy and Grandmaster Ruma Roy Choudhury. It was my only significant formal training in any sport and I believe to have acquired quite a few life lessons from it. I attempted learning very little football from Trijit Das who is our football coach at ISI. Trijit-da has been ever encouraging and helpful on and off the field.

My friend Amartya Chatterjee with whom I was introduced during my years in ISI has been a big support at very difficult times.

A major source of entertainment during all these years of my PhD has been the situational comedy series on television called "The Big Bang Theory". I would like to thank its creators Chuck Lorry, Bill Prady and the entire team of "The Big Bang Theory" for providing a very unique exposition of geeks on-screen.

My extended family has stood beside me and supported me to the best of their abilities. My uncles, aunts and in-laws have always been very concerned about my career especially during PhD. I would like to take up this opportunity to remember my paternal grandfather Late Sachindranath Bhattacharjee. He has been a huge inspiration in many ways right from my childhood. I feel extremely lucky to have my maternal grandfather Kalidas Samajder standing strong beside me till date with all his inspiration, love and affection. I simply can't resist being pampered by my grandmothers Mayarani Samajder and Monimala Bhattacharjee. I would also like to thank all my brothers and sisters for their love and support all along. Dr. Aniruddha Roy, Mousumi Choudhury and Dwaipayan Roy are amongst those people I always call when in distress - old or new. My brother-in-law Soumyajit Mukherjee and kins Avisek Sarkar and Joydeep Sarkar are new additions to that list.

I really have no words to thank two people - my mother and my wife. My mother's contributions in my life are so vast that I would only belittle them by mentioning a few. However, I would like to explicitly mention the fact that if I have ever demonstrated empathy, patience and perseverance it was just a part of my Maa Sabita Bhattacherjee living within me. She carries the heart that aches the most seeing me face difficulties and leaves no stone unturned to find me true solace. My wife Sanchari Mukherjee has put up with my weird self and ideas all along and ushered unending effort to keep me resolute and always aim higher in life. I believe this is something she carries over from my father-in-law Pradip Mukherjee and mother-in-law Subhra Mukherjee. Thank you Mummum and Babai.

Last but in no way the least, I would like to thank my father Atirindra Nath Bhattacherjee who has had the biggest influence on my academics. Baba is the one person who has taught me lessons in “learning” right from my childhood that I otherwise probably would never have had. His emphasis on dedication and reflecting on subjects by asking questions has probably been the most important lesson that I have received in my career. Thank you Baba! I would surely not have been a fraction of what I am today without you.

Date: 31st July, 2014.

Update: I would like to thank the two anonymous reviewers for their appreciations, suggestions and detailed comments on my thesis that have greatly helped in fine-tuning my thesis.

In February 2015, I joined the AriC team in the LIP laboratory at ENS-Lyon for working on the Programme Avenir Lyon Saint-Etienne (PALSE) project headed by Dr. Benoît Libert. During my stay in Lyon, I have worked on revising the two papers [BS15, BS14b] that are part of this thesis and were under submission. One of these papers [BS15] have been accepted. The other work [BS14b] is still under submission. The results of this second work in particular have been improved significantly from what has been described in this thesis. I have also worked on the revision of my thesis based on the comments from the reviewers during this period. I would like to thank Prof. Damien Stehlé and Dr. Benoît Libert and all other team-mates for their cooperation.

Date: 31st July, 2015.

Acronyms

AACS Advanced Access Content System. 6

ABTSD Augmented Binary Tree Subset Difference. 16

AES Advanced Encryption System. 25

BE Broadcast Encryption. 2

CS Complete Subtree. 44

CSS Content Scrambling System. 227

CTSD Complete Tree Subset Difference. 10

DRM Digital Rights Management. 6

DVD Digital Versatile Disc / Digital Video Disc. 6

HD High Definition. 6

HS Halevy-Shamir. 6

KPS Key Predistribution Scheme. 39

LA Licensing Authority. 226

LSD Layered Subset Difference. 6

NNL Naor-Naor-Lotspiech. 5

SD Subset Difference. 5

List of Tables

4.1 CTSD scheme: Comparison of the expected header lengths with the NNL-SD scheme (that assumes dummy users). 70

4.2 CTSD scheme: Boundary conditions for the recurrences $T(n, r, h)$ and $N(n, r, h)$. 74

4.3 Listing a few values of r and their corresponding n_r. 83

4.4 CTSD scheme: Comparison (using the expected header length finding algorithm) between the expected header lengths of the NNL-SD scheme (assuming dummy users) and the CTSD scheme. 96

4.5 NNL-SD scheme: Data for the limiting value of the expected header length. . 102

4.6 CTSD scheme: Comparison of communication overhead in bytes for the NNL-SD and the CTSD scheme ($n \leq 256$). 103

4.7 CTSD scheme: Comparison of communication overhead in bytes for the NNL-SD and the CTSD scheme ($n \leq 2048$). 103

4.8 CTSD scheme: Comparison of communication overhead in bytes for the NNL-SD and the CTSD scheme ($n \leq 16384$). 104

4.9 CTSD scheme: The expected header lengths for $r = 2$. 104

5.1 CTLSD scheme: Number of storage minimal layering strategies. 117

5.2 CTLSD scheme: Storage minimal layering strategies for $\ell_0 = 12$. 117

5.3 CTLSD scheme: Comparison of user storage and expected header lengths between e-HS LSD and SML. 119

5.4 SML layerings: The table Tab containing all storage minimal layerings up to $\ell_0 = 32$. 122

5.5 CTLSD scheme: Comparison of user storage and average header length for SD, e-HS LSD and the constrained minimization layering. 126

5.6 CTLSD scheme: Comparison of average header length for $r < r_{\min}$ between e-HS layering strategy and the constrained minimization layering strategy. . 127

5.7 CTLSD scheme: Comparison of the storage and the expected header lengths for the CTSD and the CTLSD (with constrained minimization layering) schemes. 139

6.1 kSD scheme: Comparison of the expected header lengths for number of users that are not powers of k. 168

6.2 kSD scheme: Cyclotomic cosets for $k = 3$, $k = 4$ and $k = 5$ used in storage reduction. 179

6.3 kSD scheme: Reductions of user storage achieved for $k = 3$, $k = 4$ and $k = 5$. 184

6.4 kSD scheme: Ranges of the number of users for which the reduced storage due to $k = 2$ is greater than that of $k = 3$. 187

6.5 kSD scheme: Comparison of the user storage and mean header length for $2 \leq k \leq 8$. 189

6.6 kSD scheme: The fixed ratios of MHL_k/r (for any n) corresponding to the varying ratio r/n for each k. 189

6.7 kSD scheme: Values of the threshold δ_k after which the expected header length is better for arity k compared to arity 2. 190

7.1 a-ABTSD scheme: Effect of reduction of user storage for $2 \leq a \leq 4$. 218

7.2 a-ABTSD scheme: User storage and mean header lengths in the complete a-ABTSD scheme for $1 \leq a \leq 4$. 221

7.3 a-ABTSD scheme: Values of the ratio MHL_a/r (for any n) corresponding to the varying ratio r/n for each a. 221

List of Figures

1.1 Symmetric key framework. 1

1.2 Symmetric key broadcast encryption framework. 2

1.3 Blocks of the message each sent in a new session. 3

1.4 A session made of the encrypted message (body) and multiple encryptions of the session key (header). 4

1.5 Each circle or ellipse represents the collection of subsets of a BE scheme. The Singleton Set scheme has the smallest collection that is contained in every other scheme. The Power Set scheme has the collection of all possible subsets of users. 9

2.1 NNL-SD scheme: Example of a full binary tree $\mathcal{T}^0$ with 16 users. 27

2.2 NNL-SD scheme: An example of a subset difference (SD) subset $S_{i,j}$ that has leaves of the subgraph $\mathcal{T}^i \setminus \mathcal{T}^j$. 28

2.3 NNL-SD scheme: Key assignment for a subset $S_{i,j}$. 29

2.4 NNL-SD scheme: Technique to assign user secret I_u for the user u. 31

2.5 NNL-SD scheme: Cover generation algorithm. 33

2.6 HS-LSD scheme: Splitting of an NNL-SD subset $S_{i,j}$ into two. 37

2.7 HS-LSD scheme: Key assignment for the split subsets. 37

4.1 CTSD scheme: Example of a complete binary tree $\mathcal{T}^0$ with $n = 13$ users. . . . 64

4.2 CTSD scheme: Example of an SD subset $S_{1,7} = \mathcal{T}^1 \setminus \mathcal{T}^7$. 65

4.3 CTSD scheme: Secrets stored by a user. 67

4.4 CTSD scheme: Plot showing the impact of dummy users on the mean header length. 69

4.5 CTSD scheme: Non-full subtrees of a complete tree. 71

4.6 CTSD scheme: Plot showing the variation of the maximum header length with varying values of r. 81

4.7 CTSD scheme: In computing the expected header length, the event of a node generating an SD subset. 92

5.1 CTLSD scheme: Example of a layered complete tree with 13 users, showing the dividing path. 129

5.2 CTLSD scheme: In computing the expected header length, the event of a node generating an SD subset. 133

6.1 kSD scheme: Example of an SD subset $S_{0,\{5,6\}}$ in a full tree of arity 3 for 27 users. 143

6.2 kSD scheme: Derived seeds of descendant nodes from the seed of an ancestor node. 145

6.3 kSD scheme: Key assignment to a subset $S_{i,J}$. 146

6.4 kSD scheme: Example of a subset cover $\mathcal{S}_c$ for a given set $\mathcal{R}$ of revoked users. 152

6.5 kSD scheme: Example showing that the upper bound of $2r - 1$ on the header length is tight for $k = 3$ and $r = 4$. 160

6.6 kSD scheme: Example where the header length of 4-ary is smaller than 2-ary. 160

6.7 kSD scheme: Example where the header length of 4-ary is greater than 2-ary. 161

6.8 kSD scheme: In computing the expected header length, the event of a node generating an SD subset. 163

6.9 kSD scheme: The tree structure $T^{(3)}$ for $k = 3$, used in storage reduction. . . 180

6.10 kSD scheme: The tree structure $T^{(4)}$ for $k = 4$, used in storage reduction. . . 182

6.11 kSD scheme: The tree structure $T^{(5)}$ for $k = 5$, used in storage reduction. . . 182

6.12 kSD scheme: Plot showing how MHL_k/r varies with r/n. 190

7.1 a-ABTSD scheme: Example where all nodes of a subtree being revoked implies that it is sufficient to revoke the root of the subtree. 197

7.2 a-ABTSD scheme: Example where only certain nodes of a subtree being revoked implies that it will not be appropriate to revoke the root of the subtree. 197

7.3 a-ABTSD scheme: Example subset from the collection $\mathcal{S}$ and the "moving up" operation. 199

7.4 a-ABTSD scheme: Example of a subset cover $\mathcal{S}_c$ for a given $\mathcal{R}$. 209

7.5 a-ABTSD scheme: Example to show that for $a = 2$, the upper bound $2r - 1$ of the header length is tight. 209

7.6 a-ABTSD scheme: The augmented structure T_4 for $a = 2$. 215

7.7 a-ABTSD scheme: Plot showing how MHL_a/r varies with r/n. 222

Contents

1 Introduction **1**

1.1 Thesis Plan and our Contributions 7

1.1.1 The Complete Tree Subset Difference Scheme and its Analysis 10

1.1.2 The (Layered) Complete Tree Subset Difference Scheme and its Analysis 12

1.1.3 Generalizations of the Subset Difference Scheme Using Trees of Higher Arity 14

1.1.4 The Augmented Binary Tree Subset Difference Scheme 16

2 Background and Preliminaries **19**

2.1 The Subset Cover Revocation Framework 20

2.1.1 The Subset Difference Scheme 27

2.1.2 The Layered Subset Difference Scheme 35

3 Previous and Related Works **39**

3.1 Seminal Works 41

3.2 Tree-Based Schemes 42

3.3 Traitor Tracing Techniques 47

3.4 Code-Based Traitor Tracing 49

3.5 Key Predistribution Based Schemes 50

3.6 Combinatorial Works 52

3.7 Public Key BE 54

3.8 Miscellaneous 61

4 The Complete Tree Subset Difference Scheme and its Analysis **63**

4.1 Introduction 63

4.2 The Complete Tree Subset Difference Method 64

4.2.1 Key Assignment to each Subset $S_{i,j}$ in $\mathcal{S}$ 66
4.2.2 Dummy Users and the Associated Penalty 68
4.3 Combinatorial Analysis of the SD and CTSD Methods 69
4.3.1 Some Notation 71
4.3.2 Recurrences $N(n, r, h)$ and $T(n, r, h)$ 72
4.3.3 Algorithms to Compute $N(n, r, h)$ and $T(n, r, h)$ 74
4.3.4 Upper Bounds on the Header Length 76
4.3.5 Generating Function 84
4.4 Expected Header Length in the CTSD and SD Methods 89
4.4.1 Asymptotic Analysis of the Expected Header Length for the SD Method 97
4.4.2 Other Experimental Results 103
4.5 Conclusion 105

5 The (Layered) Complete Tree Subset Difference Scheme and its Analysis 109
5.1 Introduction 109
5.2 General Layering Strategy 110
5.2.1 The HS Layering with Residual Bottom Layer 112
5.2.2 The e-HS Layering Strategy 113
5.2.3 Root at a Non-Special Level 113
5.2.4 Storage Minimal Layering 115
5.2.5 Constrained Minimization of User Storage 120
5.3 Header Length 128
5.3.1 Tackling Arbitrary Number of Users 128
5.3.2 Maximum Header Length 130
5.3.3 Expected Header Length 131
5.4 Conclusion 138

6 Generalization of the Subset Difference Scheme Using Trees of Higher Arity 141

6.1 Introduction . . . 141

6.2 The k-ary Tree Subset Difference Scheme . . . 142

6.2.1 Initiation . . . 142

6.2.2 Cover Finding Algorithm . . . 149

6.2.3 Traitor Tracing . . . 155

6.3 Header Length Analysis . . . 156

6.3.1 Expected Header Length . . . 161

6.4 Tackling Arbitrary Number of Users . . . 168

6.5 Reducing User Storage . . . 171

6.5.1 The Case $k = 3$ . . . 171

6.5.2 The Case $k = 4$ . . . 175

6.5.3 The Technique for General k . . . 176

6.6 The Layered k-ary Tree Subset Difference Scheme . . . 184

6.6.1 Storage Minimal Layering . . . 185

6.7 A Comparative Study . . . 188

6.8 Conclusion . . . 190

7 The Augmented Binary Tree Subset Difference Scheme 193

7.1 Introduction . . . 193

7.1.1 Some Notation . . . 194

7.2 The a-Augmented Binary Tree Subset Difference Scheme . . . 196

7.3 Cover Finding Algorithm . . . 203

7.4 Other Issues . . . 210

7.4.1 Accommodating an Arbitrary Number of Users . . . 210

7.4.2 Traitor Tracing . . . 211

7.5 Reducing User Storage . . . 212

7.5.1 The Basic Idea . . . 213

7.5.2 The Case $a = 2$ . . . 214

7.5.3 General Case . . . 216

7.5.4 Full Resilience . . . 218

7.6 Experimental Studies . . . 219

7.7 Conclusion . . . 223

8 Applications, Implementation Aspects and Future Directions 225

8.1 Real-Life Applications of BE . . . 225

8.1.1 Content Protection in Optical Discs . . . 226

8.1.2 Pay-TV . . . 231

8.1.3 File Sharing in Encrypted File Systems . . . 232

8.1.4 Sending Encrypted Email to Mailing Lists . . . 233

8.1.5 Online Content Sharing and Distribution . . . 233

8.1.6 Online Gaming . . . 234

8.1.7 Web-Based Electronic Commerce . . . 235

8.1.8 Peer-to-Peer DRM . . . 235

8.1.9 Military Broadcasts . . . 236

8.1.10 Home Networks . . . 236

8.1.11 Mobile Broadcast . . . 238

8.2 Practical Impact of our Contributions . . . 239

8.3 Implementations . . . 241

8.4 Possible Future Directions . . . 242

Chapter 1

Introduction

In symmetric key cryptography, it is assumed that there are two parties Alice and Bob and there is an insecure communication channel between them as shown in Figure 1.1. A cryptographic system can be used to achieve secure communication between these two parties. This cryptographic system assumes that there is a secret K called the *key* that is known only to Alice and Bob and no one else. The message to be communicated is called the *plaintext* and is denoted by M. The cryptographic system has an encryption algorithm $Enc(M, K)$ used by the sender that takes as input a plaintext message M and the secret key K and gives as output a *ciphertext* C. The receiver uses the decryption algorithm $Dec(C, K)$ that takes as input the ciphertext C and the secret key K to recover the plaintext message M. Since the secret key K is not known to anybody other than Alice and Bob, no one else can succeed in decrypting the ciphertext C with non-negligible probability.

Now consider a scenario where there are $n+1$ parties such that one of them is the sender and the remaining n are receivers as shown in Figure 1.2. The sender here is called the *center* who broadcasts encrypted messages to the n receivers called the *users* of the system. Let $\mathcal{N}$ be the set of users. In a particular session, some of the users are *privileged* and hence they can correctly decrypt the message. The decryption privilege of the remaining users are *revoked*. Let $\mathcal{R}$ be the set of revoked users. Assuming there are $r = |\mathcal{R}|$ revoked users, the

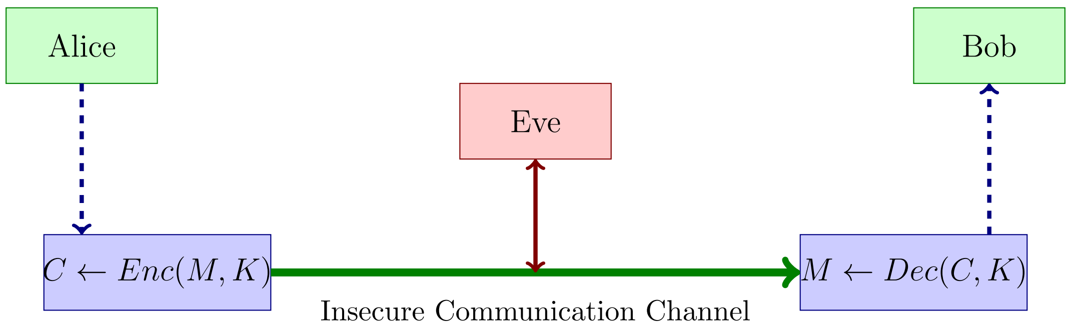


Figure 1.1: Symmetric key framework.

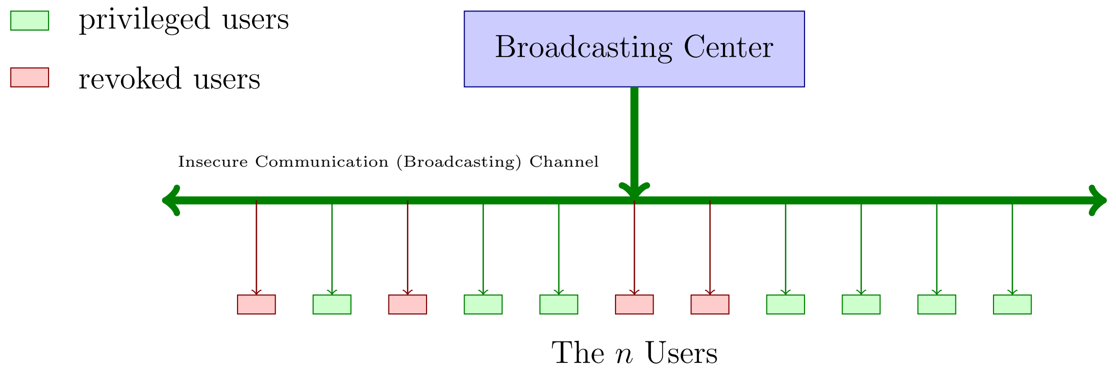


Figure 1.2: Symmetric key broadcast encryption framework.

remaining $n-r$ users are privileged. The cryptographic framework that ensures the working of the above system is called **Broadcast Encryption (BE)**.

A Broadcast Encryption (BE) scheme allows the center to efficiently broadcast encrypted information so that only the privileged users in $\mathcal{N} \setminus \mathcal{R}$ can decrypt the message correctly. The privileged set can be any subset of $\mathcal{N}$. In the two-party system, we have seen that a single secret key is shared between Alice and Bob while the algorithms *Enc* and *Dec* and all other parameters in the system are public. The use of this secret key ensures that no third party will be able to correctly decrypt the ciphertext. We first look at two basic techniques for designing a BE scheme using such a two-party symmetric key encryption scheme.

Singleton Set Scheme. In the first technique, a unique secret key is assigned to every user in $\mathcal{N}$. Each of these secret keys are known to the broadcasting center. The two-party symmetric key scheme can hence be used to communicate between the center and each user. Hence, the center encrypts the plaintext message M using the secret key of each privileged user in $\mathcal{N} \setminus \mathcal{R}$. All these $n-r$ encryptions of M are broadcast through the common public channel. Only a user in $\mathcal{N} \setminus \mathcal{R}$ should be able to decrypt the plaintext message M from the portion of this broadcast intended for itself. In this scheme, each user needs to store only a single secret key. However, the communication overhead is $O(n-r)$.

Power Set Scheme. In the second technique, a unique secret key is assigned to every subset in $\mathcal{N}$. As before, each of these secret keys are known to the broadcasting center. A user is given all those secret keys that correspond to subsets in which the user belongs.

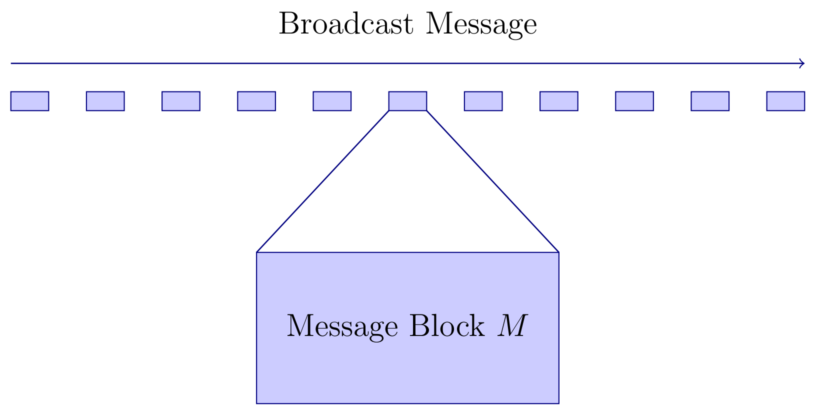


Figure 1.3: Broadcast message divided into blocks, each sent in a new session.

This time, the two parties involved in any communication would be the center and the set of privileged users. The center needs to encrypt the plaintext message M only once using the secret key of the privileged subset $\mathcal{N} \setminus \mathcal{R}$. Hence, a user should be able to decrypt the broadcast if and only if it belongs to $\mathcal{N} \setminus \mathcal{R}$. In this scheme, the number of secret keys to be stored by a user is exponential in n. However, the plaintext has to be encrypted only once.

From these two schemes it is clear that in a BE system the users are given some secret information before the start of the broadcast. This could be the actual decryption keys (as in the schemes described above) or some information from which it can derive the decryption keys. A user uses this information for decrypting relevant encrypted digital content. This secret information occupies some storage space in the user equipment. The storage requirement per user is one of the important parameters of a BE scheme.

In a typical BE scheme, the entire digital data to be broadcast is divided into blocks as shown in Figure 1.3. Each such block is called a message and each message is broadcast in a new *session*. For each session, a new random key called the *session key* K_s is used to encrypt the corresponding message M to be broadcast. The session key in turn, is encrypted a number of times using user keys and these multiple encryptions of the session key are sent as the *header* of the encrypted message. An encrypted session is shown in Figure 1.4. The transmission overhead of the scheme is determined by the number of encryptions of the session key in the header. This is called the *header length* and we denote this quantity by h. This header length h is another important parameter of a BE scheme.

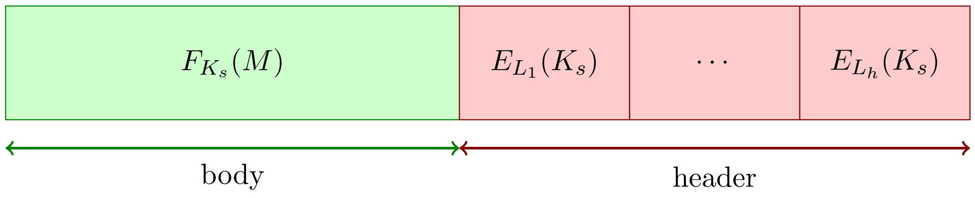


Figure 1.4: An encrypted session where K_s is the session key and L_i's denote keys with which the session key is encrypted.

A few more points to be noted:

- In both these schemes, keys are assigned to subsets of users. In the first scheme, each subset is a singleton set containing a single user $u \in \mathcal{N}$. The user storage requirement is minimum while the header length is maximized. In the second scheme, every non-empty subset of $\mathcal{N}$ is assigned a unique key. Here, the user storage requirement is maximized while the header length is minimum. In effect, these are two ends of a hierarchy of optimization between the user storage and the header length of the BE system. Other schemes may be obtained by assigning keys to only certain subsets of $\mathcal{N}$.

- *Resilience* is an important feature to be considered in the design of BE schemes. In a BE scheme, an individual revoked user should not be able to decrypt the broadcast individually. However for certain schemes, an adversary may be able to use the secret information (decryption keys) of a set of revoked users to derive some additional information. With this additional information, it may be able to decrypt the encrypted broadcast. In a *t-resilient scheme*, an adversary may have the secrets of at most t revoked users and yet will not be able to decrypt the content correctly. In a *fully resilient scheme*, even if an adversary has the decryption keys of all the remaining non-privileged users in the system, it will not be able to correctly decrypt the content.

- A crucial requirement for a BE scheme is that it should facilitate *dynamic revocation* of any subset of users. In other words, after the BE system has been initialized and has started working, the center should be able to revoke a privileged user's decryption capabilities from a certain point of time. The decision could be based on their subscription or privilege status. At the start of the next session, the center makes sure that the new session key is not encrypted using the keys of any of the revoked users including the newly revoked ones.

- A BE scheme is said to be *stateless* if the secret keys distributed initially need not be updated as new users are revoked or provided decryption privileges. On the other hand, a stateful BE scheme would allow the user secret to be updated from time to time.

- When a BE scheme is put to practical use, the user devices may get compromised. The leaked secret keys of these compromised devices can be used to build *pirate* devices with decryption privileges or re-broadcast the secret to many unauthorized users. The pirate devices can decrypt the broadcasts correctly even though they are not supposed to. Hence, for the system to keep working, there should be a mechanism to identify these leaked keys and revoke them, so that future broadcasts cannot be decrypted by the pirate devices carrying those keys. The technique of identifying the compromised keys from a pirate device by treating it as a black-box[1] is called *traitor tracing*.

  It is to be noted here that a traceability scheme that will be able to trace traitors, may not have revocation capabilities for taking away decryption privileges and vice-versa. However for most practical scenarios, either of the two properties would be rendered useless without the other.

The NNL-SD and HS-LSD Schemes. Broadcast Encryption was first introduced in [Ber91] followed by [FN93]. There have been several works in this area (discussed in details in Chapter 2) since then, but the most popular scheme out of these is the tree-based Subset Difference (SD) method of Naor-Naor-Lotspiech (NNL) [NNL01, NNL02]. The NNL-SD scheme is fully resilient against any number of revoked users colluding together. The scheme also allows the users to be stateless and hence, they do not have to update their individual secret information with every session. The decryption privileges of a user in the system may be dynamically revoked or reinstated. Since it is a symmetric key based scheme, it is very efficient in terms of encryption and decryption time. User storage requirement is $O(\log^2 n)$ the transmission overhead is linear in the number r of revoked users. The NNL-SD scheme offered the simplest algorithm and the best trade-offs for use in both real-time

[1] A *black-box* is a device which can be viewed in terms of its input, output and transfer characteristics without any knowledge about its internal implementation, engineering or data contained.

applications like Pay-TV and non-real time applications like content protection in optical discs. Further, the scheme itself is quite elegant and reasonably easy to implement. This scheme was adopted as part of the Advanced Access Content System (AACS) standard for content protection in High Definition (HD) Digital Versatile Disc / Digital Video Disc (DVD) and Blu-ray discs [AAC].

The NNL-SD scheme is defined for n users where n is a power of two, i.e., $n = 2^{\ell_0}$ for some $\ell_0 \geq 0$. The users are considered to be the leaves of a *full binary*[2] *tree* having ℓ_0 levels. Let i be a node in this tree and j be a node in the subtree of i. Now consider the set of users that are leaf nodes in the subtree rooted at i but are not in the subtree rooted at j. The subsets are called Subset Difference (SD) subsets. All SD subsets that can be formed in the binary tree are assigned keys. We will see in Chapter 2 that each user in the SD scheme needs to store $\ell_0(\ell_0 + 1)/2$ k-bit strings where k is the key length of the underlying symmetric key cryptosystem. If r users are revoked, then the worst case header length (i.e., the number of encryptions of the session key) is $2r - 1$ [NNL01, NNL02], while the average case header length was experimentally found to be at most $1.25r$ [NNL01, NNL02].

A later work by Halevy-Shamir (HS) [HS02] introduced a variant of the SD method called the Layered Subset Difference (LSD) scheme. The basic idea is to partition the tree into several layers which gives the name of the scheme. A different trade-off is obtained. User storage is reduced in the HS-LSD method to $\ell_0^{3/2}$ but, the worst case header length grows to $4r - 3$. In [HS02], based on simulation results, it is remarked that the average header length is around $2r$. Compared to the SD method, the LSD method reduces the user storage at the cost of increasing the header length.

Applications of BE. Applications of BE systems have been discussed in details in Chapter 8. Here we provide a very brief overview of these applications so that the parametric requirements of a BE system can be understood.

The application of BE systems is pretty wide in the implementation of Digital Rights Management (DRM) [DRMa] for content protection in digital data distribution technologies such as Pay-TV, Internet or mobile video broadcast, optical discs, etc. DRM systems can in general be modelled as follows. There is a set of users and a center which broadcasts the copyrighted digital content. As shown in Figure 1.1, for each block of data the center decides

[2]The arity of a tree corresponds to the maximum number of children that a node in the tree may have. In a binary tree, each node has at most two children.

on a set of privileged users which should be able to decrypt it while the revoked users should not be able to do so. Other than DRM systems, BE may also be used for broadcasting secret instructions to military outposts from a base station or to ensure that only privileged users in a file-sharing system gets access to files.

In real-time scenarios like Pay-TV, Internet or mobile video broadcast, the number of users can vary from a few thousands to millions. For other real-time applications of BE like broadcasts from a military base station, the number of users will be a few tens or hundreds. The BE scheme that is used in real time scenarios as above, has to be efficient in terms of the transmission overhead associated with each message as also the encryption and decryption times and storage of user keys.

For non-real-time applications like content protection in Blu-Ray discs and HD-DVDs [AAC], the requirements from a BE scheme are somewhat different. Here, the transmission overhead is the additional information stored in the physical media that is used for decrypting the content. Storage space in discs is no more a constraint nowadays. Further, since encryption does not happen in real-time, improving the encryption time is also not very important. On the other hand, reducing the user storage and decryption time is still important.

1.1 Thesis Plan and our Contributions

The Context. Our central focus has been the tree-based symmetric key BE schemes that are based on the subset difference based technique. These schemes have the following distinguishing features.

- They are *fully resilient* to collusion of users. This offers a stronger security guarantee against revoked users compared to t-resilient schemes.

- They allow broadcasts *to any set* $\mathcal{N} \setminus \mathcal{R}$ *of privileged users* which implies that any set $\mathcal{R}$ of users can be revoked. The center determines the set $\mathcal{R}$ of revoked users at the beginning of each session. Hence, these schemes allow *dynamic revocation of users.*

- They are *stateless* and hence do not require the keys stored in user devices to be updated. This reduces the cost of tamper-resistant hardware used in user devices.

- All these schemes have corresponding *traitor tracing techniques*. By revoking the users whose keys have been leaked, the system's security can be retained.

These features together make them arguably the most useful schemes for long-term implementations at various scales.

One disadvantage of these schemes is that they do not allow *dynamic joining or leaving of the users* from the system. As a result all these schemes have to fix the total number of users during the initiation of the scheme. We understand that it is difficult to come up with a stateless scheme that allows users to join or leave the system. A user leaving a system may be realized by the permanent revocation of that user. However, there is no known method of adding new users to the system without updating the keys of the existing users (as the keys of the new subsets have to be provided to the existing users). As a work-around, stateless schemes assume the total number n of users at the outset during the initiation of the scheme. The actual number of users of the system may be much smaller than n. The remaining users of the system are assumed to be *dummy*. As users are added to the system, these dummy users are associated with real users.

Scope for Optimizations. With the above-mentioned desirable features in mind, our objective has been to work on various optimizations of the subset-difference based BE schemes. We performed detailed (combinatorial as well as probabilistic) analysis of these schemes which has played a crucial role in understanding the scope for optimizations that were available. The two most important as well as competing parameters of any BE scheme are *the header length* (communication overhead) and the amount of *user storage* required.

The fundamental behaviour of all BE schemes is primarily determined by *the choice of subsets of users to which keys are assigned.* This choice determines the optimization between the header length and the user storage to a large extent. Keeping the other parameters like decryption time under control is also important. In addition to the choice of subsets, the user storage also depends on the *technique by which keys are assigned* to the subsets.

The Singleton Set scheme and the Power Set scheme are at the two ends of the spectrum of possible schemes. As the choice of the collection of subsets is varied, we get different BE schemes. Typically as the number of subsets to which keys are assigned is increased, the header length decreases while the user storage increases and vice-versa. However, this may not always be true as we will see later in the thesis.

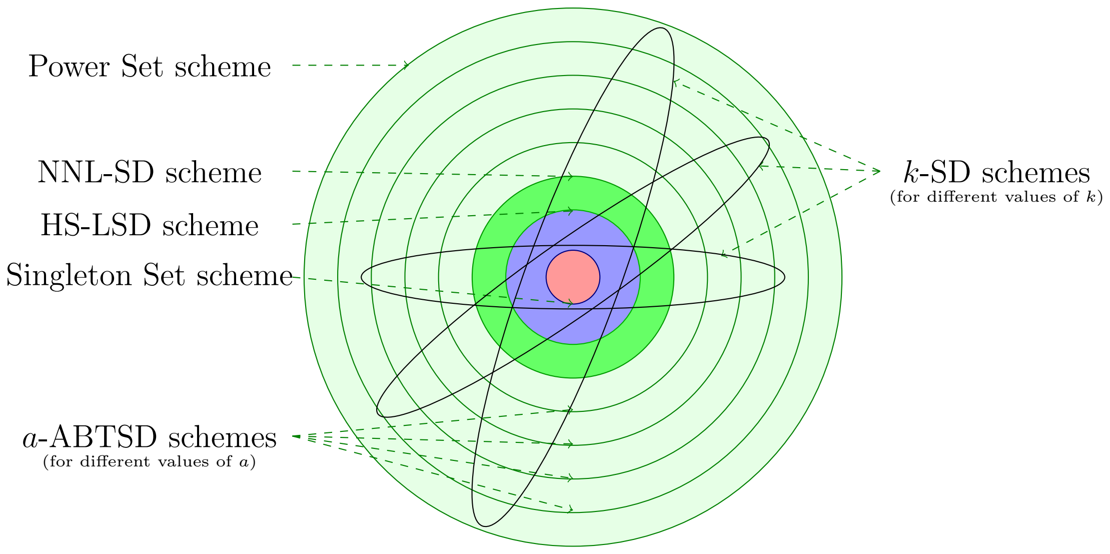


Figure 1.5: Each circle or ellipse represents the collection of subsets of a BE scheme. The Singleton Set scheme has the smallest collection that is contained in every other scheme. The Power Set scheme has the collection of all possible subsets of users.

In Figure 1.5 we represent the relationship between the collection of subsets that are assigned keys in various schemes. The singleton subsets are present in every scheme while the collection for the Power Set scheme contains all possible subsets that may be assigned keys. All other schemes fall between these two ends. The collections of the NNL-SD scheme and the HS-LSD scheme have also been indicated. In due course we will come to know of the other schemes that have been indicated alongwith.

Our Goal. Let us now understand the overall goal of this thesis. We have succeeded in improving both the header length and the user storage individually by different kinds of generalizations of the NNL-SD and HS-LSD schemes. Although these improvements are not asymptotic, they are significant as far as practical numbers are concerned and our results are the state-of-the-art in both these directions. However, improving both the parameters together could not be done. The best we could achieve was to improve one parameter while restricting the increase of the other.

We believe that the contributions of this thesis may be used to achieve significant gains in various practical applications of BE. Some of these applications are listed in *Chapter 8*. This chapter also provides a summary of our results and their practical impact.

In the following, we provide a brief summary of the other chapters which appear in the thesis. In *Chapter 2*, we provide the necessary preliminary material required in the later chapters. In *Chapter 3*, we list the previous and related works in BE. The next four chapters provide details of the four papers [BS13, BS14a, BS15, BS14b] this thesis is based on. The following is an overview of these four chapters.

1.1.1 The Complete Tree Subset Difference Scheme and its Analysis

Chapter 4 consists of the work done in [BS13]. We develop tools for detailed analysis of the subset-difference based technique for choosing subsets that are assigned keys. These tools are used for the detailed understanding of this technique and how it may be extended. There are three major contributions in this work.

Arbitrary Number of Users. We broaden the scope of use of the NNL-SD scheme. The NNL-SD scheme and all follow-up works [HS02, GST04, PB06, AK08, MMW09] assume the total number of users n to be a power of two. When implementing the NNL-SD scheme for applications such as Pay-TV, it is possible that the number of users in the system will be arbitrary. As mentioned before, the center assumes the existence of dummy users to make the number of users a power of two. We relax this restriction to allow any arbitrary number of users in the system by introducing the Complete Tree Subset Difference (CTSD) scheme. The CTSD scheme is based on the NNL-SD scheme and subsumes it while eliminating the requirement of dummy users in the system. When the number of users in the CTSD method is a power of two, it becomes exactly the same as the NNL-SD scheme. Inclusion of dummy users results in the expected header length of the NNL-SD scheme to be more than the CTSD scheme for practical values of n and r.

It is to be noted that an implementation that uses the NNL-SD scheme can easily shift to using the CTSD scheme with minimal change in the software implementation. This is because the internal tree structure used for assigning keys to subsets of users in the NNL-SD scheme remains almost the same in the CTSD scheme.

Combinatorial Analysis of The CTSD Scheme. The importance of the NNL-SD scheme motivates the study of its combinatorial properties. We carry out such a study for the CTSD scheme and the results so obtained also apply to the NNL-SD scheme. A new approach is used for the detailed combinatorial analysis. A method is proposed to count the number, $N(n, r, h)$, of ways that r out of n users can be revoked to get a header length of h in the CTSD scheme. This counting is formulated using two recurrences. Using these recurrences, a dynamic programming based algorithm is developed to compute $N(n, r, h)$ in polynomial time. Previous to our work, to compute $N(n, r, h)$ for the NNL-SD method, one would have to run the SD algorithm on the possibly exponentially many $\binom{n}{r}$ revocation patterns. Further combinatorial results that we obtain are as follows.

1. The worst case header length for a given r in the NNL-SD scheme was shown to be $2r - 1$ in [NNL01, NNL02]. We show that the worst case header length for the CTSD scheme and hence for the NNL-SD scheme is $\min(2r - 1, \lfloor n/2 \rfloor, n - r)$.

2. Given r, we characterize the minimum number of users, n_r, that need to be in a system using the CTSD method, that can give rise to the maximum header length of $2r - 1$. For the special case of the NNL-SD method the expression for n_r was obtained in [MMW09].

3. For the special case when n is a power of two i.e., for the NNL-SD scheme, we use the recurrences to obtain a generating function for the sequence. Earlier, a generating function of a slightly different form was obtained in [PB06] using direct arguments.

Probabilistic Analysis of The CTSD Scheme. We propose a simple and efficient algorithm for computing the expected header length for a given n and r in the CTSD and hence the NNL-SD method. The algorithm requires $O(r \log n)$ multiplications and $O(1)$ space. Due to its efficiency, this algorithm allows the computation of the expected header length for values of n ranging from a few hundreds to millions. This provides a useful tool to practitioners implementing either the NNL-SD or the CTSD method.

For the NNL-SD scheme, as n goes to infinity through powers of two, we provide an expression H_r for the limiting upper bound on the expected header length $H_{n,r}$. The value of H_r can be computed using $O(r)$ multiplications. Computing this value for different r shows that H_r is always less than $1.25r$. The only previously known upper bound on the expected header length in the NNL-SD scheme for r revoked users was proved to be $1.38r$ in [NNL01, NNL02]. They also commented that experimental results indicated that the bound is probably $1.25r$. Our analysis of the expected header length shows that proving the precise limiting upper bound is more complicated than anticipated in [NNL01, NNL02].

1.1.2 The (Layered) Complete Tree Subset Difference Scheme and its Analysis

In *Chapter 5*, we work with the idea of layering the levels of the underlying binary tree $\mathcal{T}^0$ of the NNL-SD scheme [NNL01, NNL02]. This idea of layering was introduced in [HS02]. A layering strategy is a choice of levels of the underlying binary tree which are said to be *special*. Layering in general reduces the user storage while increasing the (worst case and average) header length. The Halevy-Shamir (HS) layering works for $n = 2^{\ell_0}$ users where ℓ_0 is a perfect square. This limits its usage to very specific number of users $(2^4, 2^9, 2^{16}, 2^{25})$. Two natural extensions of the HS layering strategy that work for values of ℓ_0 that may not be a perfect square (and hence subsume the HS layering strategy) are considered. While both have the same storage requirement, one of them is experimentally seen to have lower average header length. We call this the extended HS or e-HS layering. We also propose a general layering strategy where any set of levels of the tree may be considered to be special and hence would denote a layering strategy.

Storage Minimal Layering. The first major problem that we tackle is whether the user storage can be lowered further than the e-HS layering strategy. To this end, we introduce the notion of storage minimal layering. For such a strategy, the user storage requirement is the minimum possible that can be obtained from 2-way splitting of NNL-SD subsets using layerings. An $O(\ell_0^3)$ time and $O(\ell_0^2)$ space dynamic programming algorithm is presented to compute storage minimal layerings. In the HS layering strategy, the root node of the user tree is treated as a special level. We show that removing this condition yields a scheme where the user storage is significantly reduced while the effect on the average header length is negligible. The resulting storage minimal schemes result in user storages which are between 18% to 24% lower than that required by the (extended) Halevy-Shamir layering scheme. We note that our work does not provide any asymptotic improvement in user storage compared to the Halevy-Shamir scheme. Rather, our work provides concrete improvement in user storage for all practical values of n and also an algorithm to compute the corresponding layering strategies.

Constrained Minimization Layering. Simply minimizing user storage is only one aspect of the problem. We consider the constrained minimization problem whereby one tries to minimize the user storage but, without increasing the actual values of the average header length significantly beyond that achieved by the NNL-SD scheme. This is a difficult problem to solve analytically. Instead, we show how to tackle the problem empirically. Given some idea about the number of users that would be revoked, we show how one may use this information to design a layering strategy for which the average header length is almost as small as the NNL-SD scheme. The user storage for such a layering scheme is significantly less than that of the NNL-SD scheme. Concrete practical examples are provided and it is shown how to tackle this problem for any practical value of the number of users.

Probabilistic Analysis of General Layering Strategy. We describe an algorithm to compute the expected header length of the layering based NNL-SD schemes assuming any general layering strategy. This algorithm works for all possible values of the number of users (and not only those values which are powers of two). Assuming that r out of n users are revoked uniformly at random, our algorithm computes the expected header length in $O(r \log^2 n)$ time and $O(\log n)$ space. A simulation based approach can also be used to estimate the average header length. In this approach, for a fixed n and r, a set of r users are randomly revoked and the cover generation algorithm is applied to compute the corresponding header length. This process is repeated many times and the average of the different header lengths is taken to be an estimate of the actual value of the expected header length. Each run will require $O(n)$ space (and hence also $O(n)$ time) to compute the cover and hence the header length. In contrast, our algorithm does away with the need of performing such a simulation study. Given n and r, it directly computes the expected header length when r out of n users are uniformly revoked. Since r will be much smaller than n for practical scenarios, our algorithm will be faster and require much less space. The algorithm is of interest in its own right as it will be a useful tool to practitioners who may wish to quickly calculate the average header length for different broadcast scenarios.

1.1.3 Generalizations of the Subset Difference Scheme Using Trees of Higher Arity

In *Chapter 6*, we extend the ideas of NNL to k-ary[3] trees for any $k \geq 2$. Our treatment is general and unified, i.e., the same approach works for all values of k. Suppose n is a power of k, i.e., $n = k^{\ell_0}$ for some $\ell_0 \geq 1$ and consider the users to be the leaf nodes of a full k-ary tree of height ℓ_0. Let $j_1, \ldots, j_c$, $1 \leq c \leq k$, be a set of sibling nodes in this tree and i is an ancestor of these nodes. Consider the set S of leaf nodes in the subtree formed by taking away the subtrees rooted at $j_1, \ldots, j_c$ from the subtree rooted at i. So, the set S is formed as a subset difference of two sets of users. In the summary of the NNL-SD scheme above, we have seen that subsets of users arising in this manner are called Subset Difference (SD) sets. The identification of the SD sets is a key aspect of obtaining the k-ary tree scheme. This idea extends the idea of SD sets introduced for binary trees in [NNL01, NNL02].

Why k-ary Trees? We mentioned earlier that as more subsets are assigned keys, the header length of a scheme reduces while the storage requirement increases. An intuition behind considering k-ary trees with $k > 2$ is that the number of SD sets grows with increasing k (Figure 1.5) and so the header length may come down at the cost of increasing the user storage. This, however, does not turn out to be entirely true. Working out the details of the scheme and the resulting analysis shows up a rich complexity of behavior which is not apparent at the outset. We provide an extensive analysis of the scheme covering the following points.

Cover Generation Algorithm. Given a set of revoked users, the center has to find the subsets of users whose union would be the set of privileged users. The session key K_s will be encrypted using keys of only these subsets. This set of subsets is called the *subset cover* $\mathcal{S}_c$ and the algorithm to find the subset cover is called the *cover generation algorithm* or the *cover finding algorithm*. We develop a single cover generation algorithm which works for all k. This is an intuitively simple algorithm which uses just an array as the underlying data structure. Specializing this algorithm for $k = 2$ yields the cover finding algorithm given in [NNL01, NNL02]. The description of the algorithm turns out to be considerably simpler than that of [NNL01, NNL02].

[3]A node in a k-ary tree may have at most k child nodes.

Traitor Tracing. The NNL paper [NNL01, NNL02] provides a mechanism for tracing traitors. With some modification, this idea also fits the k-ary BE scheme. It turns out that compared to binary trees, for $k \geq 3$, tracing traitors can be done more efficiently (i.e.; with fewer number of queries).

Header Length. For k-ary trees with n users, the maximum header length of a transmission with r revoked users is shown to be $\min(2r-1, n-r, \lceil n/k \rceil)$. Somewhat surprisingly, the first component, i.e., $2r-1$ is not affected by k. We show that the bound of $2r-1$ is indeed achieved for values of k greater than 2. Average case analysis of the header length is done under the assumption that the revoked set of users is distributed uniformly among the set of all users. With this assumption, we derive an expression for the expected header length. The method is to compute the probability that any internal node generates a subset in the header. Summing over all these probabilities provide the expected header length. The expression for the expected header length can be computed in $O(r \log n)$ time and $O(1)$ space. We have implemented the algorithm to compute the expected header length and provide representative values to show the average header lengths for different values of k.

User Storage. During the initiation of the scheme, the center provides each user with sufficient information so that it is able to generate any key corresponding to an SD set of which it is a member. This information is measured in terms of the number of m-bit seeds that are required to be stored by any user. Here m is the size of the key of the underlying symmetric cipher. The work of NNL provides a clever way to use a pseudo-random generator so that user storage consists of $1 + \lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)/2$ seeds. The direct combination of this idea with the SD sets of a k-ary tree makes the user storage to be $1+(2^{k-1}-1)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds. We show that a modification based on the use of cyclotomic cosets modulo $2^k - 1$ reduces the user storage to $1+(\chi_k - 2)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds, where χ_k is the number of cyclotomic cosets modulo $2^k - 1$.

Tackling Arbitrary Number of Users. When n is not a power of k, we show that a complete k-ary tree structure can be used to construct the BE scheme. This is an analogue of complete binary trees used in data structures. Average header length analysis of such schemes is performed using simulation studies.

Simulation Study of the Header Length. We perform a simulation study of the average header length for $n = 10^x$ ($x = 3, \ldots, 8$) users and for $k = 2, \ldots, 8$. Experimental results indicate that there is a cut-off value δ_k such that for $r/n > \delta_k$, the average header length of the k-ary scheme is less compared to that of the binary tree based scheme. Further, the value of δ_k decreases as k increases. This suggests that by increasing k, it is possible to reduce the header length for lower values of r. This can be important for applications such as Pay-TV systems. The trade-off is a one-time moderate increase in user storage.

1.1.4 The Augmented Binary Tree Subset Difference Scheme

The key idea behind the work in *Chapter 7*, is to assign keys to more subsets in addition to the collection of the NNL-SD scheme. More specifically, union of subsets which are already in the NNL-SD collection are assigned keys. As a result, if the subset cover due to the NNL-SD scheme has subsets whose combination has been newly assigned a key, then those subsets are replaced in the cover by their union. Consequently, the header length decreases.

In order to include these additional subsets in the collection, an additional tree structure is assumed at each node in $\mathcal{T}^0$. This structure directly relates a node with its descendants at a height a below it in $\mathcal{T}^0$. Our scheme is parameterized by a and is hence called the *a-Augmented Binary Tree Subset Difference (ABTSD)* scheme. For $a = 1$, this scheme is exactly the same as the NNL-SD scheme. For a given value of a, the user storage for the scheme is $O(\log^2 n)$. As the value of a is increased, the user storage increases in concrete terms. It has been proved that for any given set of revoked users, the header length for $a > 1$ is at most as large as the NNL-SD scheme. Hence, it follows from the result in [BS13] that the worst case header length for the scheme is $\min(2r-1, \lfloor n/2 \rfloor, n-r)$.

The a-ABTSD scheme is extended to accommodate an arbitrary number of users using a complete binary tree instead of a full tree. The cover generation algorithm is simulated for this more general complete tree version, to find the performance of the a-ABTSD scheme in terms of communication overhead. It is observed that the expected header length for any given number of revoked users r, decreases as a increases. For example, for $n = 10^6, r = 4 \times 10^5$, the expected header length for $a = 1$ is 2.29 times that of $a = 3$. The storage requirement increases from around 3.28KB for $a = 1$ to around 94.13KB for $a = 3$. From the simulation studies, we observe that, for a given ratio of r/n, the expected header length of the a-ABTSD scheme with $a > 1$ is a fixed fraction of that of $a = 1$. A technique is proposed to mitigate the increase in user storage with increasing a. It is also argued that the efficiency of the traitor tracing mechanism for this scheme does not deteriorate with increasing a.

Chapter 2

Background and Preliminaries

As mentioned in Chapter 1, the two most important and influential works in the area of symmetric key broadcast encryption are [NNL01, NNL02] and [HS02]. Almost all known BE schemes fall under the Subset Cover Revocation Framework that was introduced in [NNL01, NNL02]. The Subset Difference (SD) scheme that has been suggested by the AACS [AAC] standard for digital rights management in optical discs was also introduced in [NNL01, NNL02]. The Layered Subset Difference (LSD) scheme of [HS02] resulted in asymptotic improvement of the user storage requirement of the NNL-SD scheme at the cost of increased worst case and average header lengths.

In this thesis, we work within the ambit of the Subset Cover Framework. We have done detailed combinatorial and probabilistic analysis of the SD and LSD schemes. Additionally, we have proposed various generalizations of these schemes that can be instantiated for improved user storage and header length. Hence, in this chapter we describe in details the Subset Cover Revocation Framework, the Subset Difference scheme and the Layered Subset Difference scheme.

Basic Notations. Before we start describing the various schemes and their analysis, we give a brief summary of the most commonly used notations in this thesis. This listing is not intended to be exhaustive but it should give a fair idea about their usage. Any new notation introduced in the thesis, has been defined explicitly at appropriate places.

We use the usual set notations and logical operators. All indexing variables are indicated appropriately in the different contexts. All logarithms considered have base 2.

For a BE scheme, the set of all users in the system is denoted by $\mathcal{N}$ and the set $\mathcal{R} \subseteq \mathcal{N}$ denotes the set of revoked users. The cardinalities of these sets are $n = |\mathcal{N}|$ and $r = |\mathcal{R}|$. The underlying tree structure is denoted by $\mathcal{T}^0$. The 0 in the superscript indicates the label of the root node of $\mathcal{T}^0$. A node of the tree is in general denoted by lowercase letters i, j, etc. and sometimes by u, v, etc. Hence, for a node i, the subtree of $\mathcal{T}^0$ that is rooted at node i is denoted by $\mathcal{T}^i$ and the number of users in the subtree is denoted by λ_i. The nodes at the same distance from the root node are said to be at the same level and the level numbers are denoted by ℓ. A path in the tree is denoted by $\mathcal{P}$. The maximum number of child nodes of any node in a tree is called the arity of the tree. The arity of the tree is denoted by k.

A subset of the set $\mathcal{N}$ of users is denoted by S while the empty set is denoted by ϕ. The collection of all such subsets of $\mathcal{N}$ that are assigned keys is denoted by $\mathcal{S}$. The set of subsets to which a user u belongs is denoted by $\mathcal{S}_u$. A subset difference subset is in general denoted by $S_{i,J}$ where J is a set of nodes in the subtree $\mathcal{T}^i$. It is to be noted that in the NNL-SD scheme, $|J| = 1$.

The secret information that is stored by a user is denoted by I_u while the header length is denoted by h. Pseudo-random generators defined as hash functions are denoted by G and H. The seeds used as inputs to these functions are denoted by the letter L or are written as *seed*. We use uppercase letters W, X, Y, Z to denote random variables unless otherwise stated explicitly.

2.1 The Subset Cover Revocation Framework

The Subset Cover Revocation Framework assumes a *center* that encrypts a message M and broadcasts it to a set $\mathcal{N}$ of *users* where $|\mathcal{N}| = n$. This set of users contains all the possible recipients of the broadcast. A subset $\mathcal{R}$ of these users are revoked. A broadcast encryption algorithm under this framework consists of three parts:

- *scheme initiation* - each user $u \in \mathcal{N}$ is assigned the secret information I_u that will allow them to decrypt messages intended for them;
- *broadcasting algorithm* - that takes as input the message M, the set $\mathcal{R}$ of revoked users and $\bigcup_{u \in \mathcal{N} \setminus \mathcal{R}} I_u$ and outputs the ciphertext C. C is broadcast to all the users in $\mathcal{N}$;
- *decryption algorithm* - that runs at the user end. It takes as input the ciphertext C and the secret information I_u that the user u had received during initiation and attempts to decrypt C. A privileged user in $\mathcal{N} \setminus \mathcal{R}$ should be able to get back the original message M, while any coalition of revoked users in $\mathcal{R}$ should not be able to get back the correct message from C.

Scheme Initiation. During initiation, a collection $\mathcal{S} = \{S_1, \ldots, S_w\}$ of subsets are defined, where each $S_j \subseteq \mathcal{N}$. A set $S_j \in \mathcal{S}$ has an associated key and any subset of $\mathcal{N}$ which is not in $\mathcal{S}$ does not have any key associated with it. Each subset S_j is assigned a long-lived key L_j. For a user u, let $\mathcal{S}_u = \{S_j \in \mathcal{S} : u \in S_j\}$. User u is given secret information I_u such that it can construct the key L_j associated with any set $S_j \in \mathcal{S}_u$. However, I_u may not explicitly contain the long-lived key L_j, as we will see in the Subset Difference scheme and all the related schemes in this thesis.

Broadcasting Algorithm. Once the scheme has been initiated, and the user secrets have been distributed, the center can now start broadcasting. During broadcast, the set of privileged users $\mathcal{N} \setminus \mathcal{R}$ is partitioned into pairwise disjoint subsets $S_{i_1}, \ldots, S_{i_h}$ each taken from the collection $\mathcal{S}$. This partition is called the *subset cover* $\mathcal{S}_c$. In other words,

$$\mathcal{N} \setminus \mathcal{R} = \bigcup_{j=1}^{h} S_{i_j}$$

where each $S_{i_j} \in \mathcal{S}$ and $\mathcal{S}_c = \{S_{i_1}, \ldots, S_{i_h}\}$. The message to be broadcast is divided into blocks each sent in a new session. For a message block M, the broadcasting algorithm uses two encryption functions:

- A function $F : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ to encrypt the message M with a *session key* $K \in \mathcal{K}$; $F_K(\cdot) \triangleq F(K, \cdot)$ is length preserving. The function F is length preserving so that there is no loss of information or redundant communication overhead. The session key is a random string chosen afresh for each new message M.
- A function $E : \mathcal{K}_1 \times \{0,1\}^m \to \{0,1\}^m$ to encrypt the session key K with a long-lived key $L \in \mathcal{K}_1$ corresponding to the subset S_j ($\in \mathcal{S}_c$) of users; $E_L(\cdot) \triangleq E(L, \cdot)$ is length preserving.

In order to broadcast the message M, the center chooses a random session key K_s and encrypts M as $F_{K_s}(M)$. This session key has to be communicated to the privileged users in $\mathcal{N} \setminus \mathcal{R}$ so that they can correctly decrypt K_s and in turn decrypt M from this encrypted form. To that end, the center finds the subset cover $\mathcal{S}_c = \{S_{i_1}, \ldots, S_{i_h}\}$. Let $L_{i_1}, \ldots, L_{i_h}$ be the long-lived keys that were assigned to each of these subsets in $\mathcal{S}_c$. The center then encrypts the session key K_s with each of these keys L_{i_j}. The session key has to be encrypted

h times i.e., once for each set in $\mathcal{S}_c$. The h encryptions of the session key are sent along with $F_K(M)$ as a *header* for the encrypted message. The header also has information to identify the subsets S_{i_j} that form the cover $\mathcal{S}_c$. The size h of the header is determined by the number of sets in $\mathcal{S}_c$. We are going to refer to this size as the *header length.* The encrypted message $F_K(M)$ along with the header forms the ciphertext C. The header length is a key efficiency parameter that resembles the transmission overhead of the scheme. The resultant ciphertext C is a tuple $\langle \textit{header}, \textit{body} \rangle$. The *body* is the encryption $F_{K_s}(M)$ of the message block M for that session. The *header* part contains the encryptions $E_{L_{i_j}}(K_s)$ of the session key K_s for each subset $S_{i_j} \in \mathcal{S}_c$ and the identifier i_j for that subset.

$$C = \langle [i_1, i_2, \ldots, i_h, E_{L_{i_1}}(K_s), E_{L_{i_2}}(K_s), \ldots, E_{L_{i_h}}(K_s)], F_{K_s}(M) \rangle.$$

Decryption Algorithm. During decryption, a user u has to first find from the header, the identifier i_j such that $S_{i_j} \in \mathcal{S}_u$. Next, from the encryption of the session key $E_{L_{i_j}}(K_s)$ in the header, it will extract the session key K_s. It then derives the long-lived key L_{i_j} from the secret information I_u it had acquired during initiation. Once it has L_{i_j}, it decrypts the session key K_s. The user can hence decrypt the message M from $F_{K_s}(M)$. In case a user is revoked and hence does not belong to any of the sets in $\mathcal{S}_c$, it will not be able to decrypt K_s from the header or M from the body for that matter.

Two parameters are of crucial interest. The size of the secret information I_u that is to be stored by a user u and the average or expected length of a broadcast header which amounts to the communication overhead. Basic intuition tells us that as the number of elements in $\mathcal{S}$ grows, it should be possible to cover the privileged set $\mathcal{N} \setminus \mathcal{R}$ with fewer elements from $\mathcal{S}$ and so the average header length will decrease. On the other hand, as $\mathcal{S}$ grows, the size of $\mathcal{S}_u$ also grows and this should lead to an increase in the size of I_u. Thus, the average header length and the user storage are two competing parameters.

Security. Proving the security of the subset cover framework starts with assumptions on the underlying primitives. With these assumptions, it has to be shown that if the key assignment technique of the subset cover algorithm satisfies the *key indistinguishability* property, then we get a secure encryption of the message.

Underlying Primitives. The overall security of the BE scheme is expressed as a function of the security provided by the underlying symmetric key encryption functions F_{K_s} and E_{L_i}. The security requirements of these two methods are different, since F_{K_s} uses short-lived keys (only for a session) whereas E_{L_i} uses long-lived ones (for the lifetime of the scheme). The assumptions on the security of these primitives are as follows.

A *feasible adversary* [1] $\mathcal{B}$ chooses a message M and receives for a randomly chosen K_s, one of the following: (a) $F_{K_s}(M)$ or (b) $F_{K_s}(R_M)$ where R_M is a random message of the same length as M. It is assumed that $\mathcal{B}$ is able to distinguish between these two encryptions with negligible probability bounded above by ϵ_1. In other words,

$$|\Pr[\mathcal{B} \text{ outputs `a'} | F_{K_s}(M)] - \Pr[\mathcal{B} \text{ outputs `a'} | F_{K_s}(R_M)]| \leq \epsilon_1.$$

The long-term encryption method should withstand a more severe attack in the following sense. A feasible adversary $\mathcal{B}$ for a random key L_i gets to choose adaptively polynomial many inputs and examine ciphertexts encrypted with E_{L_i} and similarly provide ciphertexts and examine the decryptions too. Then, it chooses a random plaintext K and receives one of (a) $E_{L_i}(K)$ or (b) $E_{L_i}(R_K)$ where R_K is a random string of length $|K|$. It is assumed that $\mathcal{B}$ is able to distinguish between these two encryptions with negligible probability bounded above by ϵ_2. In other words,

$$|\Pr[\mathcal{B} \text{ outputs `a'} | E_{L_i}(K)] - \Pr[\mathcal{B} \text{ outputs `a'} | E_{L_i}(R_K)]| \leq \epsilon_2.$$

Key Assignment. A secure subset cover algorithm requires the key assignment technique to have the key indistinguishability property. This property requires that the key L_i assigned to a subset $S_i \in \mathcal{S}$ is indistinguishable from a random key given all the secret information of all users in $\mathcal{N} \setminus S_i$.

Definition 1. *Let $\mathcal{A}$ be a subset cover revocation algorithm that defines the collection $\mathcal{S}$ of subsets of $\mathcal{N}$. Let $\mathcal{B}$ be a feasible adversary that selects an $S_i \in \mathcal{S}$ and then receives the I_u for all $u \in \mathcal{N} \setminus S_i$. Then $\mathcal{A}$ is said to satisfy the key indistinguishability property if the probability that $\mathcal{B}$ distinguishes (a) L_i (the long-lived key of the set S_i that was chosen by $\mathcal{B}$) from (b) a random key R_{L_i} of the same length $|L_i|$ is negligible and bounded above by ϵ_3. In*

[1] An adversary that is computationally bounded.

other words,

$$|\Pr[\mathcal{B} \text{ outputs `}a\text{'}|L_i] - \Pr[\mathcal{B} \text{ outputs `}a\text{'}|R_{L_i}]| \leq \epsilon_3.$$

All *information theoretic* [2] key assignment schemes in which the key for each subset in $\mathcal{S}$ is chosen independently, satisfies this property with $\epsilon_3 = 0$.

It is to be noted here that the key indistinguishability property as per Definition 1 implies full resilience of the subset cover revocation algorithm.

The key indistinguishability property of the key predistribution technique implies the following lemma.

Lemma 1. *For any $S_i \in \mathcal{S}$, let $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ be all the subsets of S_i that are in $\mathcal{S}$; let $L_{i_1}, L_{i_2}, \ldots, L_{i_t}$ be their corresponding keys. For any adversary $\mathcal{B}$ that chooses an $S_i \in \mathcal{S}$ and receives I_u for all $u \in \mathcal{N} \backslash S_i$, if $\mathcal{B}$ attempts to distinguish between (a) the keys $L_{i_1}, L_{i_2}, \ldots, L_{i_t}$ from (b) random strings $R_{L_{i_1}}, R_{L_{i_2}}, \ldots, R_{L_{i_t}}, |L_{i_j}| = |R_{L_{i_j}}|$, then*

$$|\Pr[\mathcal{B} \text{ outputs `}a\text{'}|L_{i_1}, L_{i_2}, \ldots, L_{i_t}] - \Pr[\mathcal{B} \text{ outputs `}a\text{'}|R_{L_{i_1}}, R_{L_{i_2}}, \ldots, R_{L_{i_t}}]| \leq t \cdot \epsilon_3.$$

The definition of security of a revocation scheme is as follows.

Definition 2. *Consider an adversary $\mathcal{B}$ that gets to:*

1. *Select adaptively a set $\mathcal{R}$ of revoked users and obtain I_u for all $u \in \mathcal{R}$ as follows.*
   - *$\mathcal{B}$ may adaptively select messages $M_1, M_2, \ldots$ and corresponding revocation sets $\mathcal{R}_1, \mathcal{R}_2, \ldots$ and observe the encryption of M_i when the revoked set is $\mathcal{R}_i$. The users in $\mathcal{R}_i$ may or may not be corrupted.*
   - *$\mathcal{B}$ can create ciphertexts to see how the (non-corrupted) users decrypt it.*
   - *$\mathcal{B}$ then asks to corrupt a receiver u and obtains I_u.*

   The adaptive corruption of users is repeated $|\mathcal{R}|$ times for any $u \in \mathcal{N}$ with the other steps getting repeated accordingly a bounded number of times.

2. *Choose a message M as the challenge plaintext and a set $\mathcal{R}$ of revoked users that must include all the users it has corrupted at least (if not more).*

[2] Where keys are chosen uniformly and independently at random as opposed to them being generated by a computationally secure primitive from a short random seed.

$\mathcal{B}$ then receives the encryption for a message M' and the revoked set $\mathcal{R}$. It has to guess if M' is (a) the message M it chose or (b) a random string R_M of length $|M|$.

We say that a revocation scheme is secure if for any (probabilistic polynomial time) adversary $\mathcal{B}$ as above, the probability that $\mathcal{B}$ distinguishes between the two cases (a) and (b) is negligible.

The Security Theorem. The main security theorem below shows that the key indistinguishability property is sufficient for a scheme in the subset cover framework to be secure in the sense of Definition 2.

Theorem 2. *[NNL01, NNL02] Let $\mathcal{A}$ be a subset cover revocation algorithm where the key assignment satisfies the key indistinguishability property (Definition 1) and where E and F satisfy the aforementioned security requirements. Then $\mathcal{A}$ is secure in the sense of Definition 2 with security parameter $\delta \leq \epsilon_a + 2h_{max}w(\epsilon_2 + 4w\epsilon_3)$, where w is the total number of subsets in the collection $\mathcal{S}$ of subsets for the scheme and h_{max} is the maximum size of a cover.*

The proofs for Lemma 1 and Theorem 2 use hybrid arguments and algebraic manipulations that are mostly routine. We skip the proofs here and refer to [NNL01, NNL02] for details.

Choices for Functions E_{L_i} and F_{K_s}. Two symmetric key encryption functions are used in the subset cover revocation framework. The function F_{K_s} is used to encrypt a message block with the session key K_s and E_{L_i} is used to encrypt K_s with the long-lived key L_i of the set $S_i \in \mathcal{S}$. Here we mention some of the schemes that may be used as E_{L_i} or F_{K_s}.

Block Ciphers. The Advanced Encryption System (AES) program was announced by NIST[3] in 1997 to replace the ageing Data Encryption Standard (DES). There were five AES finalists namely: MARS [BCD$^+$99], RC6 [RRYS98], Rijndael [DR02], Serpent [ABK98] and Twofish [SKW$^+$98, SKW$^+$99]. Rijndael developed by Joan Daemen and Vincent Rijmen was the final winner. Rijndael is a family of ciphers with different key and block sizes. It has an iterated structure based on the *SPN*[4] framework. The round function of the SPN structure

[3] National Institute of Standards and Technology, USA.
[4] Substitution-Permutation-Network

is composed of (1) a subkey addition layer, (2) an Sbox layer and (3) a bit permutation layer. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different *input key*[5] lengths: $128, 192$ and 256 bits. The case of AES-128 encrypts 128-bit blocks under a key of length 128 bits. AES-128 is composed of 10 rounds that repeat four elementary mappings (SubBytes for the Sbox layer; ShiftRows and MixColumns for the permutation layer; AddRoundKey for subkey addition) on blocks seen as 4×4-byte matrices.

Light-Weight Block Ciphers. AES is not suitable for extremely constrained environments such as RFID tags and sensor networks [CMM13]. Light-weight block ciphers are to be used in such scenarios. PRESENT [BKL$^+$07] is one of the most popular light-weight block ciphers. It is a 31-round block cipher that works on 64-bit blocks and input keys may be 80 or 128 bits long. The design is based on a simple SPN structure. Instead of having 16 unique Sboxes, PRESENT uses a single 4×4-bit Sbox. This helps in speed-up of the cipher by reducing circuit complexity. Other light-weight block ciphers include CLEFIA-128 [SSA$^+$07], DES(X)L [LPPS07], HIGHT [HSH$^+$06], IDEA [LM90], KATAN & KTANTAN [CDK09], KLEIN [GNL11], LBLOCK [WZ11], LED [GPPR12], mCrypton [LK05], MIBS [ISSK09], Piccolo [SIH$^+$11], SEA [SPGQ06], SKIPJACK [ski98], TEA & XTEA [WN94] and TWINE [SMMK12].

Stream Ciphers. The eSTREAM project was co-ordinated by ECRYPT[6] for the design of new stream ciphers. The project finished in April 2008 with the publication of a portfolio of new stream ciphers. There were four proposals that were suited to fast encryption in software (so-called Profile 1) while four others offered particularly efficient hardware implementation (so-called Profile 2). The portfolio has been revisited and revised periodically and consequently the algorithms have matured. The current 2012 eSTREAM portfolio contains seven algorithms: HC-128 [Wu08], Rabbit [BVZ08], Salsa20/12 [Ber08], Sosemaunk [BBC$^+$08], Grain [HJMM08], MICKEY 2.0 [BD08] and Trivium [CP08].

[5] The input key is further used to get an *expanded key* using the key-scheduling algorithm of AES.

[6] European Network of Excellence in Cryptology was a 4-year European research initiative launched on 1 February 2004 with the stated objective of promoting the collaboration of European researchers in information security and especially in cryptology and digital watermarking [Wik].

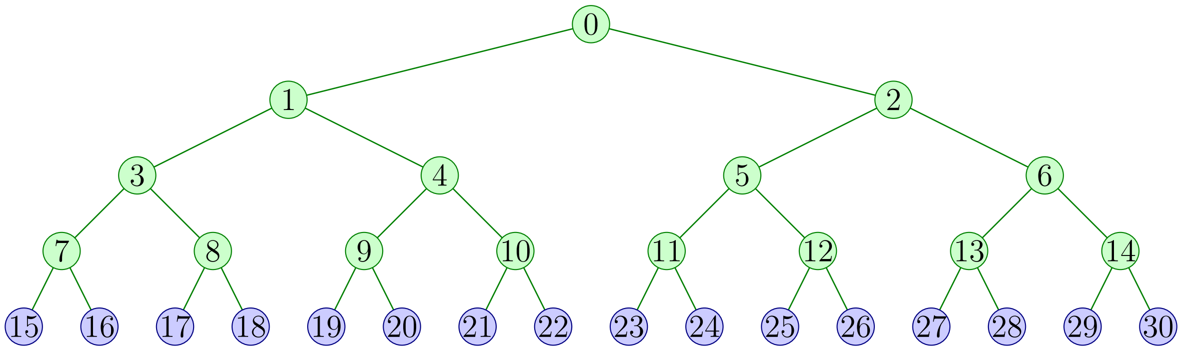


Figure 2.1: An example of a full binary tree $\mathcal{T}^0$ with 16 users.

2.1.1 The Subset Difference Scheme

In [NNL01, NNL02], Naor-Naor-Lotspiech introduced the Subset Difference (SD) scheme as an instance falling under the Subset Cover framework.

Scheme Initiation. The SD scheme assumes the number n of users to be a power of 2, say $n = 2^{\ell_0}$. Each user is associated with a leaf of a full binary tree and all the n leaf nodes are at the bottom-most level[7]. The full binary tree $\mathcal{T}^0$ has a root node at the top-most level $\mathcal{T}^0$ as shown in Figure 2.1. All non-leaf nodes have exactly two children. The nodes in $\mathcal{T}^0$ are identified by labels as follows. The root node is labeled as 0. For a non-leaf (also called internal) node i, its two children are labeled as $2i + 1$ and $2i + 2$. The subtree rooted at a node i of $\mathcal{T}^0$ is denoted by $\mathcal{T}^i$. A node i in $\mathcal{T}^0$ represents the users at the leaf level of the tree $\mathcal{T}^i$. We shall sometimes denote this set of users by the notation $\mathcal{T}^i$ for the subtree.

The Collection $\mathcal{S}$. The SD scheme introduces a major novelty in defining $\mathcal{S}$ and assigning keys to the subsets in $\mathcal{S}$ such that there is a compact way of representing I_u. Let i be a non-leaf node in $\mathcal{T}^0$ and j be a non-root node in $\mathcal{T}^i$. By $\mathcal{T}^i \setminus \mathcal{T}^j$ we denote the subgraph obtained by taking away $\mathcal{T}^j$ from $\mathcal{T}^i$. Let $S_{i,j}$ be the set of leaf nodes of $\mathcal{T}^i \setminus \mathcal{T}^j$. Figure 2.2 shows an example of such a set $S_{i,j}$. Then for the SD scheme, the collection $\mathcal{S}$ consists of the subsets $S_{i,j}$ for all possible choices of node i and all possible nodes $j \neq i$ in the subtree

[7] The set of all nodes in the tree that are at a fixed distance from the root is defined as a *level*. The level number of a node is defined as the difference between the height of the tree and the length of the path from the root to that node.

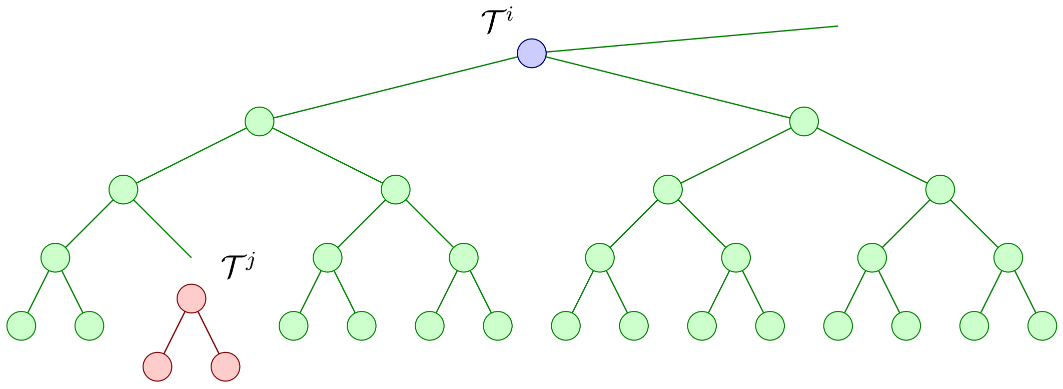


Figure 2.2: An example of a subset of the form $S_{i,j}$ (leaf nodes of the subgraph $\mathcal{T}^i \setminus \mathcal{T}^j$ shown in green).

$\mathcal{T}^i$. These subsets are called SD subsets.

Key Assignment. A clever algorithm is used to define the key associated with an SD subset $S_{i,j}$. The set of all users $\mathcal{N}$ is assigned a random key. This key is used if there are no revoked users. Next, each internal node i in $\mathcal{T}^0$ is assigned an independent and uniform random string $seed_i$. A cryptographically strong pseudo-random generator (PRG) $G : \{0,1\}^m \to \{0,1\}^{3m}$ is used to assign seeds derived from $seed_i$. Let $G(seed)$ be written as the concatenation of 3 m-bit strings $G_L(seed)$, $G_M(seed)$ and $G_R(seed)$. Let a node i have some (random or derived) $seed_i$. The left child $2i+1$ gets $seed_{i,2i+1} = G_L(seed_i)$ and the right child $2i+2$ gets $seed_{i,2i+2} = G_R(seed_i)$. This $seed_{i,2i+1}$ (respectively $seed_{i,2i+2}$) is further used to find the seeds derived from $seed_i$ for all other nodes in $\mathcal{T}^{2i+1}$ (respectively $\mathcal{T}^{2i+2}$). The derived seed for a node j from $seed_i$ of an ancestor node i is denoted as $seed_{i,j}$. The key for the subset $S_{i,j}$ is defined as $L_{i,j} = G_M(seed_{i,j})$. For example, let the node j in $\mathcal{T}^i$ be reached from node i by the moves 'left', 'left' and 'right' as shown in Figure 2.3. Then the seed of j derived from $seed_i$ is $seed_{i,j} = G_R(G_L(G_L(seed_i)))$ and the key associated with the set $S_{i,j}$ is $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$ as shown in Figure 2.3. This easily extends to any appropriate pair of nodes i and j. The string $L_{i,j}$ is an m-bit string and the value of m is determined by the key size of the underlying encryption algorithm.

This key assignment may alternatively be done using a hash function (in place of the PRG) and the technique may be viewed as follows. A key K_0 is assigned to the subset $\mathcal{N}$.

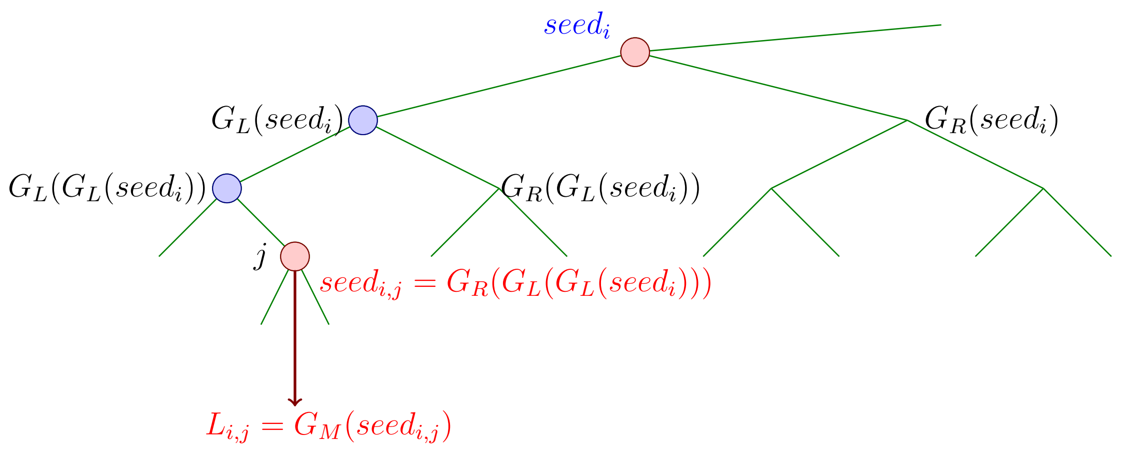


Figure 2.3: Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

For key assignment to the other subsets $S_{i,j} \in \mathcal{S}$, a cryptographic hash function

$$G : \{0, 1, 2\} \times \{0, 1\}^m \to \{0, 1\}^m \tag{2.1}$$

is chosen by the center and is made available to all users in the system. Here m is the key-size of the underlying symmetric cipher. For $t = 0, 1, 2$, let $G_t(\cdot) \triangleq G(t, \cdot)$. Each subset $S_{i,j} \in \mathcal{S}$ is assigned a key as follows.

- Every internal node i in $\mathcal{T}^0$ is assigned a uniform random m-bit seed L_i.
- All non-root nodes j in the subtree $\mathcal{T}^i$ derive seeds from L_i in the following manner. Let $j = t_0, \ldots, t_p = i$ be the sequence of nodes in the path from j to i. Then for $\imath = p - 1, \ldots, 0$, $t_\imath = 2t_{\imath+1} + s_\imath$ where $s_\imath \in \{1, 2\}$. Define the derived seed $L_{i,j}$ associated to $S_{i,j}$ to be $L_{i,j} \triangleq G_{s_0}(\cdots G_{s_{p-2}}(G_{s_{p-1}}(L_i)) \cdots)$.
- The key $K_{i,j}$ associated to the subset $S_{i,j}$ is defined to be $K_{i,j} \triangleq G_0(L_{i,j})$.

User Information I_u. Recall that users are at the leaf level of the tree. The leaf level is numbered 0 and level numbers increase up to ℓ_0 which is the level number of the root. For any user u, the user storage I_u is defined in the following manner. Consider the path from the node u to the root and let i be a node on this path at level $\ell > 0$ of the tree. Let $i_1, \ldots, i_\ell$

be the siblings[8] of the nodes on the path from u to i (including u but not including i). Then for each such i, user u gets $seed_{i,i_1}, seed_{i,i_2}, \ldots, seed_{i,i_\ell}$. Figure 2.4 shows an example where the ancestor i of u is at level $\ell = 4$ and hence receives 4 seeds derived from $seed_i$. The value of ℓ varies from 0 to ℓ_0 and so each user gets $\ell_0(\ell_0+1)/2$ seeds. The total size of I_u is $m\ell_0(\ell_0+1)/2$ bits where m is the size of the seed of the PRG. Since m is fixed, it is enough to consider only the number of derived seeds stored by a user as determining the size of user storage.

Correctness. The derived seeds provided to a user are sufficient for the user to construct the key corresponding to any $S_{i,j}$ to which it belongs. To see this suppose that i is a node on the path from u to the root and j is a node in the subtree rooted at i such that $u \in S_{i,j} = \mathcal{T}^i \setminus \mathcal{T}^j$. Since u is not in $\mathcal{T}^j$ and both u and j are in $\mathcal{T}^i$, the paths to the root from these two nodes intersect for the first time at some node v which is also in $\mathcal{T}^i$. Let v_1 be the first node in the path from v to j. Then v_1 is the sibling of some node v_2 in the path from u to i and so u has $seed_{i,v_1}$. From $seed_{i,v_1}$, u can generate $seed_{i,j}$ by applying G_L and G_R appropriately and so can generate $L_{i,j} = G_M(seed_{i,j})$. This $L_{i,j}$ is the key corresponding to the set $S_{i,j} = \mathcal{T}^i \setminus \mathcal{T}^j$. So, u can generate keys for any subset $S_{i,j}$ to which it belongs.

Security. In order to prove security for the scheme, it has to be shown that the key indistinguishability condition (as in Definition 1) holds for this method. A user u will not be able to find the key of any subset $S_{i,j}$ if $u \notin S_{i,j}$. In other words, each key $L_{i,j}$ of a subset $S_{i,j}$ in the scheme is indistinguishable from a random key for all $u \notin S_{i,j}$. If $u \notin S_{i,j}$, then either (1) $u \notin \mathcal{T}^i$ or (2) $u \in \mathcal{T}^j$.

If $u \notin \mathcal{T}^i$, then it is not assigned any $seed_{i,j}$ derived from $seed_i$. Consequently, for any set $S_{i,j}$ the key $L_{i,j}$ is (information theoretically) independent of all I_u for $u \notin \mathcal{T}^i$. For that matter, any part of I_u of any user that has not been derived from $seed_i$ is (information theoretically) independent of $L_{i,j}$.

If $u \in \mathcal{T}^j$, it is to be argued that $L_{i,j}$ remains (information theoretically) independent of the combined secret information of all $u \in \mathcal{T}^j$. The combined information of all $u \in \mathcal{T}^j$ would contain $seed_{i,j'}$ for (1) all j' that are directly attached to (also called "hanging off") the path joining i and j and (2) $j' \in \{2j+1, 2j+2\}$, the children of j. These seeds are sufficient to derive all other seeds in the combined information of $u \in \mathcal{T}^j$. Moreover, there

[8] A *sibling* of a node in a rooted tree is defined as any other node with the same parent node in the tree.

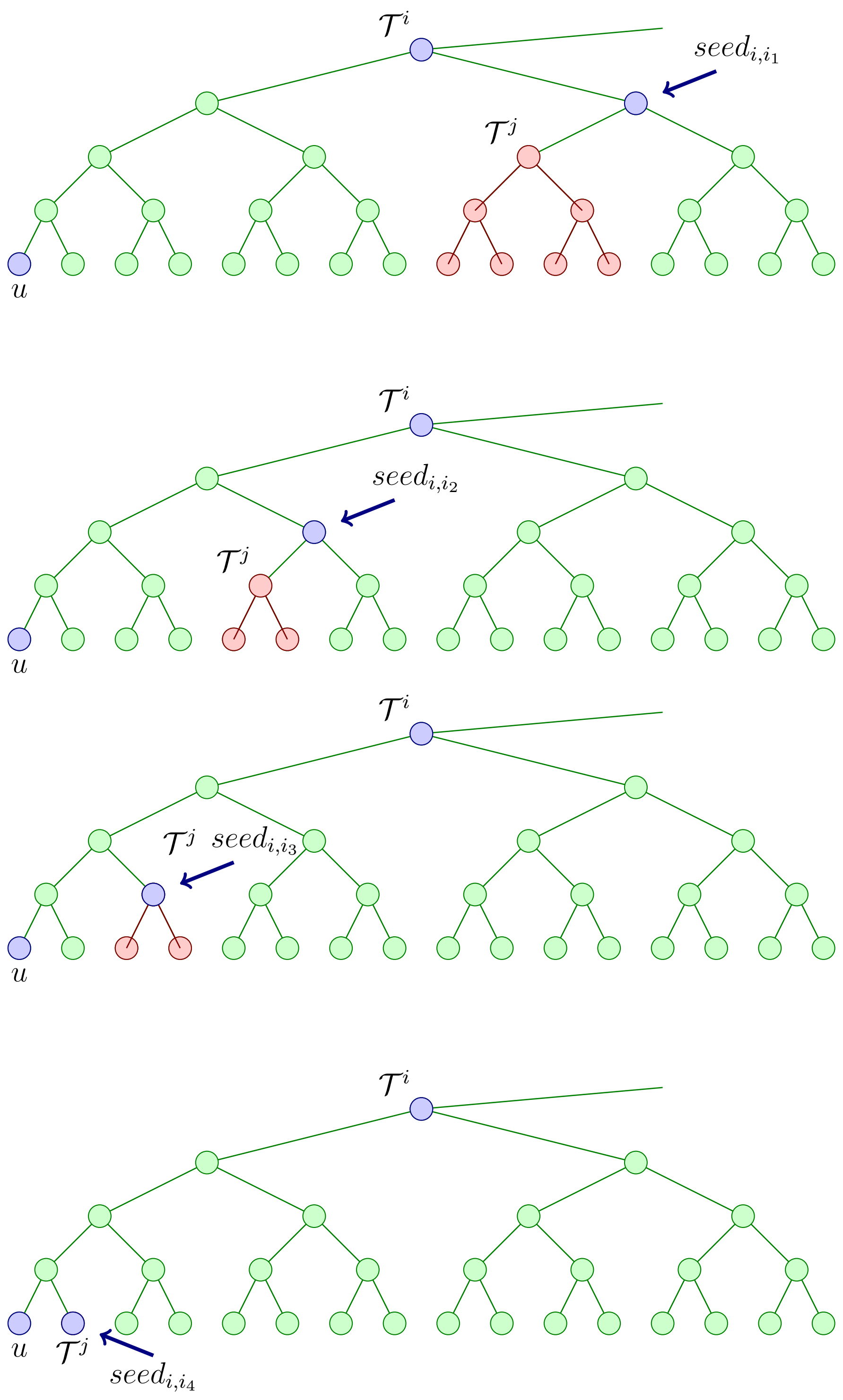


Figure 2.4: User u gets 4 seeds derived from $seed_i$ for nodes that are directly attached to the path between u and i. From each such derived seed, keys of many subsets ($S_{i,j}$ indicated respectively for each of the four cases) can be generated.

can be at most $\log n$ such seeds. It is important to note here that none of these seeds are derived from one another since they were generated independently using the PRG G. Let ϵ_4 be the bound on distinguishing outputs of G from random strings. (In case a hash function is used in place of a PRG, ϵ_4 would be the bound on the probability of finding a collision for the hash function. It is to be noted that the security argument for the scheme remains the same if G is assumed to be a random oracle and may be modified appropriately for the use of specific hash functions.) Using a hybrid argument similar to the one used in the proof of Lemma 1, it can be shown that the probability of distinguishing $L_{i,j}$ from a random string can be at most $\epsilon_4/\log n$. Assuming ϵ_4 is negligible, we get that the key $L_{i,j}$ is indistinguishable from a random string for any $u \notin S_{i,j}$.

Broadcasting Algorithm. For a given set $\mathcal{R}$ of revoked users, the center finds the subset cover $\mathcal{S}_c$. If $\mathcal{R} = \phi$, the set $\mathcal{N}$ of all users forms the cover. Otherwise, the cover finding algorithm runs iteratively as follows. The revoked users are leaves of $\mathcal{T}^0$. It finds two revoked leaves j_1 and j_2 such that their first (lowest in terms of level number) common ancestor i has no other revoked leaf in its subtree. Let i_1 (respectively i_2) be the child of i on the path joining j_1 (respectively j_2) with i. Subsets S_{i_1,j_1} (provided $i_1 \neq j_1$) and S_{i_2,j_2} (provided $i_2 \neq j_2$) are added to the cover and hence subtrees rooted at nodes i_1 and i_2 are deleted. The algorithm keeps running as above assuming the common ancestor i to be a newly revoked leaf along with the previous ones until all the privileged users are covered. If only one non-root revoked node j remains in the tree, the subset $S_{0,j}$ is added to the cover. The session key is thus encrypted for all these subsets in the cover $\mathcal{S}_c$. Figure 2.5 demostrates this algorithm.

Decryption Algorithm. On receiving a ciphertext C, a privileged user u needs to identify from the header the subset $S_{i,j} \in \mathcal{S}_c$ to which it belongs. It then derives the key $L_{i,j}$ from I_u and decrypts K_s from the header. Using K_s, it decrypts the plaintext message block M for the session.

Traitor Tracing. Attackers of a BE scheme may either create a pirate decryption box or re-broadcast the copyrighted material. In [NNL01, NNL02], the authors only talk about tracing a traitor that has participated in creating a pirate decryption box. For the other

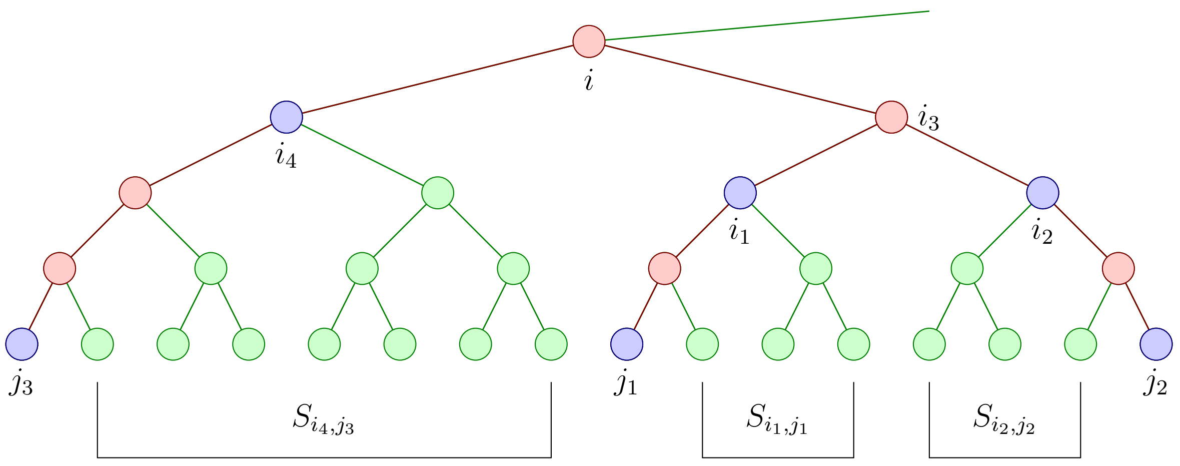


Figure 2.5: The Cover Finding Algorithm of the NNL-SD scheme is demonstrated. First the two nodes j_1 and j_2 (with lowest common ancestor i_3) gives rise to the subsets S_{i_1,j_1} and S_{i_2,j_2}. As a result, the nodes in the subtree $\mathcal{T}^{i_3}$ is covered and hence deleted with only the root i_3 remaining in $\mathcal{T}^0$ and is a newly revoked leaf. Next the two nodes i_3 and j_3 (with lowest common ancestor i_3) gives rise to the subset S_{i_4,j_3} only as the subtree rooted at i_3 is empty. Hence, the remaining nodes in the subtree $\mathcal{T}^i$ is covered and deleted with only the root node i remaining in $\mathcal{T}^0$ and is a newly revoked leaf.

re-broadcasting attack, no immediate solutions were provided by Naor et. al.[9]

Hence, in the context of the SD scheme and all related or derived schemes, the traitor tracing mechanism will be expected to identify leaked user keys from a pirate decoder by treating it as a *black-box*. In [NNL01, NNL02], it was shown that traitor tracing can be done on any scheme that assigns keys to subsets which satisfy the *bifurcation property*. The bifurcation property states that *given any subset that is in the collection $\mathcal{S}$ and hence has been assigned a key, it is possible to partition the set into two (or a constant number of) almost equal subsets from $\mathcal{S}$*. (It is to be noted that the partitioning refers to that of an SD set and not of the collection $\mathcal{S}$, and the equality is with respect to the size of each of these SD subsets.) The *bifurcation value* was defined to be the ratio of the size of the largest subset to that of the set itself. For the BE schemes of [HS02, BS13, BS14a], the subsets used in these schemes all belong to the collection $\mathcal{S}$ for the NNL-SD scheme with the same number of users. Hence, their respective traitor tracing mechanisms are almost the same as

[9] A separate traitor tracing scheme to defend against re-broadcasting attacks may be used in parallel with the tracing mechanism for pirate decoders that was proposed in [NNL01, NNL02]. As mentioned in [JL09], the trace and revoke scheme of [JL07] may be useful that way.

the NNL-SD scheme.

The *tracing algorithm* for the NNL-SD scheme works as follows. It assumes that there is a good *subset tracing procedure* that will test the capability of a decryption box for its ability to correctly decrypt a transmission intended for a particular subset cover $\mathcal{S}_c = \{S_{i_1,j_1}, \ldots, S_{i_h,j_h}\}$. The process of testing the decryption capability of the box for a subset cover $\mathcal{S}_c$ is called a *query*. A query succeeds if the box is able to decrypt the transmission. The subset tracing procedure also identifies the set $S_{i',j'} \in \mathcal{S}_c$ containing a traitor.

The traitor is thus in the subtree $\mathcal{T}^{i'}$ but not in $\mathcal{T}^{j'}$. The node j' is either in the left subtree $\mathcal{T}^{2i'+1}$ or the right subtree $\mathcal{T}^{2i'+2}$ of i'. First, we assume j' to be in the left subtree $\mathcal{T}^{2i'+1}$. It can be easily seen that $S_{i',j'} = S_{2i'+1,j'} \cup S_{i',2i'+1}$. The tracing algorithm fires two more queries. The set $S_{i',j'}$ in the cover is first replaced with $S_{2i'+1,j'}$ and then with $S_{i',2i'+1}$ and respective queries are fired on the pirate decryption box. The case in which the decryption box successfully decrypts the content, tells which of these two subsets (that $S_{i',j'}$ has been divided into) contains the traitor. Similarly, if j' is in the right subtree $\mathcal{T}^{2i'+2}$, two separate decryption capability test queries have to be fired for $\mathcal{S}_c$ containing $S_{i',2i'+2}$ and $S_{2i'+2,j'}$ instead of $S_{i',j'}$. The result of these tests will tell which of the two sets has the traitors. The algorithm would thus work recursively on the subset cover where $S_{i',j'}$ is replaced by its subset that contains the traitor.

The number of queries required by the traitor tracing algorithm depends on the bifurcation value. At every step of the traitor tracing algorithm, a subset $S_{i',j'}$ of users that contains a traitor is divided into two subsets as mentioned above. One of the two subsets of $S_{i',j'}$ would be $S_{i',2i'+1}$ or $S_{i',2i'+2}$. Hence, the bifurcation value ratio will be largest when the total number of users in $S_{i',j'}$ is the smallest. For the partition $S_{i',j'} = S_{2i'+1,j'} \cup S_{i',2i'+1}$, the total number of users will be minimal for $j' \in \{2i'+3, 2i'+4\}$ (i.e.; when j' is a child of $2i'+1$). Similarly, for the partition $S_{i',j'} = S_{2i'+2,j'} \cup S_{i',2i'+2}$, when j' is a child of $2i'+2$, that is when the total number of users will be minimal. Hence, it can be seen that the bifurcation value of the NNL-SD scheme is $2/3$.

The size of the remaining subset from which the traitors have to be traced decreases with the bifurcation value of the sets in the collection $\mathcal{S}$ for a BE scheme. Hence, the traitor tracing algorithm will be more efficient.

2.1.2 The Layered Subset Difference Scheme

The point of the LSD scheme is to reduce the user storage in the SD scheme at the cost of increasing the header length. Reduction in the user storage is achieved by reducing the size of $\mathcal{S}$. As in the SD scheme, the LSD scheme also considers the number of users to be of the form 2^{ℓ_0} where the users form the leaves of a full binary tree. The major difference between the SD and the LSD schemes is that in the LSD scheme the levels of the tree are partitioned into *layers*. Some of the levels are marked as *"special"*. The collection of levels between (and including) two consecutive special levels is called a layer. The levels are numbered with the bottom-most level having the number 0, increasing to the top as in the SD scheme description of Section 2.1.1. The *length* of a layer is the difference between the numbers of the special levels enclosing the layer.

The Halevy-Shamir Layering Strategy

The layering strategy described in [HS02] is as follows:

> "The root is considered to be at a special level, and in addition we consider every level of depth $k \cdot \sqrt{\log(n)}$ for $k = 1 \ldots \log(n)$ as special (wlog, we assume that these numbers are integers)."

We call this the Halevy-Shamir (HS) layering strategy. It assumes $\sqrt{\ell_0}(= \sqrt{\log n})$ to be an integer and hence ℓ_0 to be a perfect square. The "wlog" in the above statement is valid when one is interested in asymptotic analysis. For concrete values of n, the paper does not describe how to choose a layering scheme. This restricts the use of the scheme to very limited values of n (of the form 2^{ℓ_0} where $\ell_0 = 4, 9, 16, 25$). On the other hand, the authors of [HS02] consider the case of $n = 2^{28}$ users and suggest a layering strategy with layers of size $6, 6, 6, 5$ and 5. However, they do not give any general description of how to choose the layer lengths when ℓ_0 is not a perfect square. We take up this issue later in Chapter 5.

As a consequence of layering, an SD subset $S_{i,j}$ is defined to be in $\mathcal{S}$ if either of the following two conditions hold:

- node i is at a special level;
- or, node i is not at a special level but, node j is in the same layer as level i.

This reduces the size of $\mathcal{S}$ and consequently the size of I_u also reduces as we explain below. The distribution of seeds is done as follows. Suppose that u is a user (i.e., a leaf node) and i is a node at level ℓ in the path from u to the root and $i_0, \ldots, i_{\ell-1}$ are the siblings of the nodes in the path from u to i. If ℓ is a special level, then u is given $seed_{i,i_0}, \ldots, seed_{i,i_{\ell-1}}$ as in the SD scheme. Suppose ℓ is not a special level. Let ℓ' be the first special level below i and consider the segment of the path from u to i which lies between ℓ' and ℓ. Suppose $i_{\ell'-1}, \ldots, i_{\ell-1}$ are the siblings of the nodes on this segment. Then u gets $seed_{i,i_{\ell'-1}}, \ldots, seed_{i,i_{\ell-1}}$ derived from $seed_i$. The net effect is that if i is not at a special level, it generates seeds only up to the next special level (and not up to the bottom-most level). This leads to the reduction in the user storage.

The reduction in user storage is achieved at the cost of an increase in the header length. Suppose i is not at a special level and j is in the sub-tree rooted at i but not in the same layer as i. The SD scheme would associate the set $S_{i,j}$ to such an (i, j) pair. In the LSD scheme, this set is not present. Instead, the header computation algorithm will cover this set in the following manner. Let k be the node in the first special level as one moves down the path from i to j. The sets $S_{i,k}$ and $S_{k,j}$ are both present in the LSD scheme and it is easy to see that

$$S_{i,j} = S_{i,k} \cup S_{k,j}.$$

This can be viewed as a two-way split of the set $S_{i,j}$. Figure 2.6 shows the splitting of the subset $S_{i,j}$ of Figure 2.2. The key assignment to the subsets $S_{i,k}$ and $S_{k,j}$ in Figure 2.6 is shown in Figure 2.7. The work [HS02] also considers the possibility of multi-way split. But, the authors conclude that this leads to further reduction in user storage only for impractical values of the number of users. In this thesis, we will not consider multi-way split.

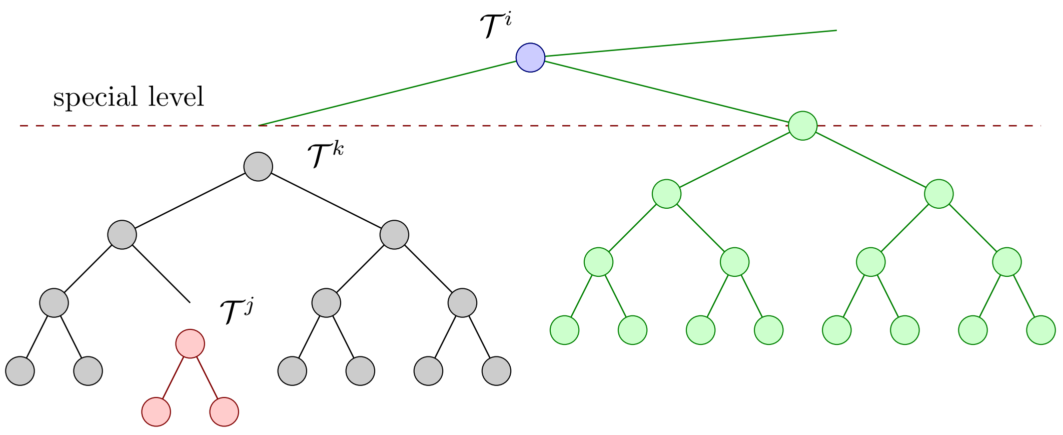


Figure 2.6: The subset $S_{i,j}$ split into $S_{i,k}$ (green leaves) and $S_{k,j}$ (grey leaves).

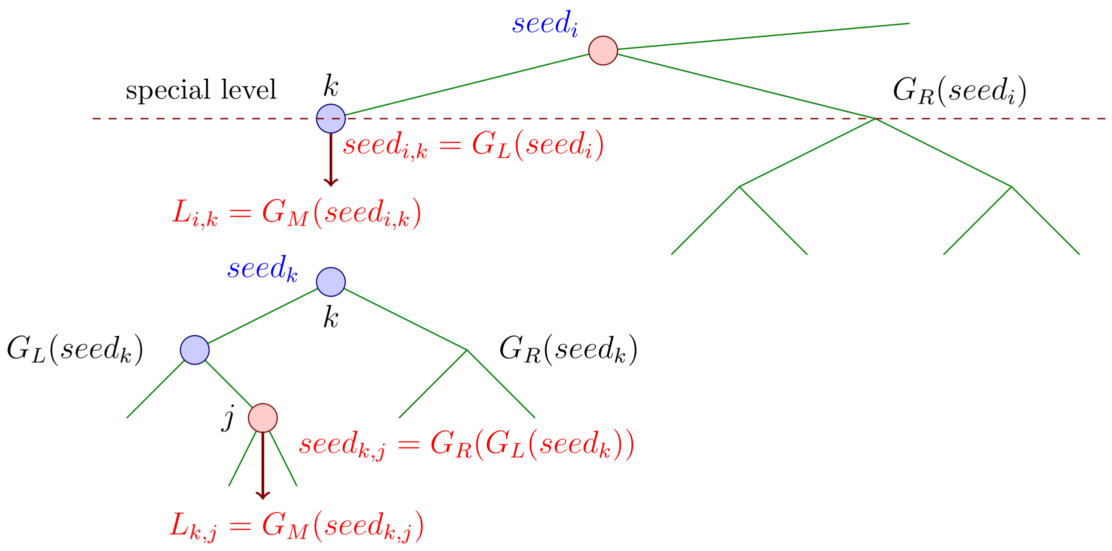

$seed_i$
special level
k
$G_R(seed_i)$
$seed_{i,k} = G_L(seed_i)$
$L_{i,k} = G_M(seed_{i,k})$
$seed_k$
k
$G_L(seed_k)$
$G_R(seed_k)$
j
$seed_{k,j} = G_R(G_L(seed_k))$
$L_{k,j} = G_M(seed_{k,j})$

Figure 2.7: Key for $S_{i,k}$ is $L_{i,k} = G_M(G_L(seed_i))$ and for $S_{k,j}$ is $L_{k,j} = G_M(G_R(G_L(seed_k)))$.

Chapter 3

Previous and Related Works

Although the NNL-SD scheme is the most popular of all BE schemes, there have been several other significant works in this area. In this chapter, we shall look at some important works on and related to BE. These works have been classified primarily based on the underlying techniques and functionalities. Limited by our knowledge and interest, this listing is nowhere close to being exhaustive. However, it should give the reader a fair idea of the directions of research in this area.

The scope for obtaining hierarchies of optimization determined by the choice of the collection $\mathcal{S}$ of subsets to which keys are assigned and the subsequent main optimization goal of this thesis has been mentioned in Chapter 1. The intent of this chapter is to play a supporting role towards this goal in the following way. The listing of the previous and related works and categorising them under different functional groups along with their descriptions, will set the perspective and point out where the results of the thesis stand with respect to the known results in the area leading to the current state-of-the-art. Amongst other possible directions of research in each of these functional categories, it will also be interesting to obtain hierarchies of optimizations in each of the functional groups wherever appropriate. Such a study is beyond the direct scope of this thesis.

Before we look at the characterization of the related works on BE, let us briefly state the correlation between BE and two other related functional requirements of similar practical scenarios: *key predistribution schemes* and *traitor tracing schemes.*

Key Predistribution Schemes. In BE, the message body is encrypted with a session key for the users in the set $\mathcal{N} \setminus \mathcal{R}$. This session key is shared with the users in $\mathcal{N} \setminus \mathcal{R}$ usually by appending a header to the encrypted body that contains several encryptions of the session key. There are schemes that assume the session key will be shared before the start of the broadcast. A scheme where the session key is generated ahead of the start of a session (say by exchanging messages amongst the privileged users) is called a Key Predistribution Scheme (KPS). Hence, when the broadcast starts, each user in $\mathcal{N} \setminus \mathcal{R}$ would already have a common key established using the KPS algorithm. As part of the broadcast, no additional header will be required. Several KPS schemes shall be mentioned and discussed in this chapter.

However, KPS schemes require user equipment to identify the set $\mathcal{N} \setminus \mathcal{R}$ at run time and if necessary communicate with each other to establish the common key. Hence, the overall communication overhead goes up. They also typically require updating the keys of the users. Hence, these schemes require specialized tamper-resistant memory that can be updated. The user secrets are updated using rekeying messages. This rekeying event requires all users to be connected at a time which may not be a practicable assumption in all scenarios [NNL01, NNL02]. Hence, some mechanism is always needed to ensure individual updates. To quote from [NNL01, NNL02],

> *"Taking the stateless approach gets rid of the need for such a mechanism (of updating states individually): simply add a header to each message denoting who are the legitimate recipients by revoking those who should not receive it. In case the number of revocations is not too large this may yield a more manageable solution. This is especially relevant when there is a single source for the sending messages or when public-keys are used."*

Traitor Tracing Schemes. According to [CFN94, CFNP00],

the traitor *or* traitors *is the (set of) authorized user(s) who allow other unauthorized parties to obtain the data;*

the unauthorized parties are called pirate users*.*

The traitor may have leaked its secret keys to build pirate decryption boxes. It may also choose to distribute the data by re-transmitting them to the pirates. Identifying the traitor is termed as *traitor tracing* as was explained in Section 2.1.1 of the previous chapter.

Data that is to be delivered to some and protected from others, has to be encrypted. That is where a BE scheme is functional. The data distribution center thus gives the authorized parties cryptographic keys to decrypt the encrypted data. However, this does not stop the authorized users from transferring the secret decryption keys or the decrypted data to an unauthorized party.

Pirate decryption boxes may be be created using these leaked secret keys. In order to identify which of the authorized users' keys have been leaked, one has to get hold of a pirate decryption box. Traitor tracing schemes would run tests on the pirate decoder treating it as a black-box.

If the secrets of each user is unique, the traitor that has leaked its key can be made evident. However, if more than one user shares the same set of secret keys, it becomes mathematically impossible to uniquely identify the traitor.

BE schemes *may or may not* have associated techniques to trace traitors and combat piracy. In case they do and a traitor is identified, its secret keys are revoked dynamically. Any future broadcast will not authorize decryption by these traitor devices. Hence, the pirate decryption methods will be rendered useless.

This revocation of decryption privileges of a traitor may be done by updating the secret keys of the remaining privileged users in a stateful system. In a stateless system, the center only needs to ensure that future broadcasts cannot be decrypted using the keys of the revoked traitor.

3.1 Seminal Works

We have already described two of the most popular works on Broadcast Encryption - the one by Naor-Naor-Lotspiech [NNL01, NNL02] and the work by Halevy-Shamir [HS02]. However, this area of research was set rolling by papers almost a decade before these two works. These seminal works have been listed here.

How to Broadcast a Secret; Berkovits (Eurocrypt, 1991) [Ber91]. The idea of a broadcasting center wanting to transmit a secret to some subset of its listeners was introduced in [Ber91]. As a first basic solution, the center can re-encrypt the message or a random key (analogous to the session key) that is used to encrypt the message, individually (and "in parallel") for each of the users using their separate secret keys. This scheme is same as the Singleton Subset scheme discussed in Chapter 1. However, *a true broadcast scheme* was defined in [Ber91] to be *"one in which the same broadcast message contains the same information for each and every listener"*. The intended recipients should be able to decrypt the secret while the others cannot.

A general technique to design a (true) broadcasting scheme based on Shamir's "k out of n" secret sharing scheme [Sha79] was also proposed in [Ber91]. For broadcasters with more computational resources, the above scheme reduces to a vector-based formulation related to Brickell's secret sharing scheme [Bri89]. This vector based scheme allows several variations that provide optimizations between the computation time at the center and the transmission overhead.

Broadcast Encryption; Fiat, Naor (Crypto, 1993) [FN93]. The term Broadcast Encryption (BE) was coined by Amos Fiat and Moni Naor in [FN93]. The idea of *resilience* in a BE scheme was also introduced in this work. A BE scheme is said to be k-resilient if a coalition of users of size up to k cannot obtain any secret of the remaining users in the system. They acknowledged the fact that a BE scheme should allow transmission to a *dynamic* set of privileged users. It was also pointed out that the relevant parameters that one would want to optimize in a BE scheme are: (1) communication overhead, (2) user storage, and (3) decryption time at the user end.

Zero-message schemes were defined in [FN93] where after the scheme has been initiated, if a user $u \in \mathcal{N} \setminus \mathcal{R}$ knows the identities of the privileged users in $\mathcal{N} \setminus \mathcal{R}$ it would be able to compute a common key with the center without any additional transmission from the center. As a first, they constructed zero-message schemes with low resilience. The first basic scheme worked like one-time pad for each subset of $\mathcal{N}$ of size at most k. Two other zero-message schemes were proposed using cryptographic assumptions like the existence of one-way functions and security of RSA. Using these schemes, more general schemes with higher resilience were constructed (using a family of hash functions) which were not zero-message schemes and would hence require additional transmission from the center to the users.

3.2 Tree-Based Schemes

We have already described the two most important tree-based schemes: NNL-SD and HS-LSD. Here are a few more important key predistribution and broadcast encryption schemes that assume an underlying tree structure with which the users are associated.

Logical Key Hierarchy; Wallner, Harder, Agee (RFC 2627, NSA, 1999) [WHA99]. The Logical Key Hierarchy (LKH) key predistribution technique [WHA99] is a novel solution to the key predistribution and re-keying problem. The users are assumed to be associated with leaves of a rooted binary tree structure. Each node in this tree represents a subset of users under its subtree. A hierarchy of keys is created using this underlying tree. Each user is secretly given one of the keys at the bottom of the tree-based hierarchy. The key of an internal node in the tree is encrypted with all of its children keys, and all of these ciphertexts are broadcast to the group. Each member can decrypt the key of a parent node along the path from its leaf to the root, since it has the key of the child node on that path. For balanced trees, the length of the path from a user to the root is logarithmic in the group size. Hence, each user stores $\log_2 n$ keys. When a user joins or leaves the system, $\log_2 n$ keys have to be broadcast by the center corresponding to all the ancestors of the corresponding user, through $2\log_2 n + 1$ re-keying messages. The LKH method achieves logarithmic broadcast size, user storage, and computational cost. Due to the re-keying technique, the system is *forward secure* (new members cannot decrypt transmissions previous to their inception) as well as *backward secure* (evicted colluding members cannot decrypt new transmissions post eviction). A generalization and improvement of the LKH scheme was given in [CMN99] by Canetti et al.

The LKH scheme and all previous related works assumed the underlying structure to be static. The Time-Varying Heterogeneous LKH scheme of [Mih03] employs a reconfigurable underlying structure and a related divide-and-conquer technique to achieve trade-offs like a large reduction of the storage and processing overload in lieu of a small increase of the communication overhead.

In [Pin04], the state update transmission requirement per user eviction of the LKH scheme was improved from $\log_2 n$ keys to $O(\log_2 t)$ where t is the size of each key.

This approach of a hierarchy of keys organized as a rooted tree was discovered independently by [WGL00] at about the same time as [WHA99]. Additionally, protocol design, implementation and performance analysis were considered in [WGL00]. With a hierarchy of keys, there may be many different ways to construct rekeying messages and securely distribute them to users. The authors designed protocols for users to join and leave the system using these rekeying strategies. Empirical results from the implementations of these rekeying strategies and protocols showed that these protocols were scalable to larger number of users.

Key Establishment in Large Dynamic Groups Using One-Way Function Trees; McGrew, Sherman (IEEE-TSE, 2003) [SM03]. The one-way function tree (OFT) scheme takes a bottom-up approach where the new keys are derived from the leaves up to the root. In [CGI+99], a variation of OFT was proposed that was called One-way Function Chain (OFC) in [SM03]. In OFC, there is always a functional relationship among the node secrets along the path in the key tree from some leaf to the root. In both OFT and OFC, the node secrets and node keys are different unlike the LKH scheme. A length-doubling PRG is used to compute the node secret and node keys. The left half $f(\mathit{seed})$ of the output is used to construct the node secret and the right half $g(\mathit{seed})$ is used to construct the node key as follows. Let x_i and x_j be the node keys of two sibling nodes in the tree. The node key x of their parent is computed as

$$x = f(x_i) \oplus f(x_j)$$

and the node secret is computed as $g(x)$. This functional chain changes over time and will hold for the last leaf whose user was removed. This effectively halves the broadcast overhead in the OFT scheme as compared to the LKH scheme for a single user eviction. This scheme is forward as well as backward secure even for arbitrarily large number of evicted users.

The Complete Subtree Scheme; Naor, Naor, Lotspiech (Crypto, 2001) [NNL01, NNL02]. Before the Subset Difference scheme of [NNL01, NNL02] that has been described in Section 2.1.1, the NNL paper had proposed a simpler BE scheme. It was called the Complete Subtree (CS) scheme. It falls under the subset cover framework described in 2.1. Like the NNL-SD scheme, it assumes an underlying full binary tree $\mathcal{T}^0$ with the users at its leaf nodes. Each node in this tree is assigned a uniform random key for the subset of users under it. A user gets the keys of all nodes from its leaf to the root node of $\mathcal{T}^0$. Hence, the user storage requirement is $O(\log n)$. The subset cover is found by finding maximal subtrees of $\mathcal{T}^0$ that do not contain any revoked user. In [NNL01, NNL02], this is described using a reduced subgraph of $\mathcal{T}^0$ called the *Steiner Tree* $ST(\mathcal{R})$ containing only the nodes and edges on the paths between revoked users and the root of $\mathcal{T}^0$. The users in a subtree of $\mathcal{T}^0$ that "hang off" from this subgraph $ST(\mathcal{R})$ form a subset of the cover $\mathcal{S}_c$. It turns out that the maximum header length of the CS scheme is $O(r \log n/r)$.

One may note the similarity of the CS scheme with that of the LKH key predistribution scheme described above. The nodes of the underlying binary tree in both the schemes represent subsets of users and are assigned some secret information that will be held only by the users in that subtree. However, the CS scheme is stateless while the LKH scheme is stateful. We have already discussed the advantages of the stateless BE schemes over stateful key predistribution schemes in Chapter 1.

BE Schemes with Underlying Trees of Arity > 2 **[Asa02, FKTS08].** Two public key BE schemes were proposed in [Asa02] that assigned keys to subsets following the Complete Subtree (CS) method of [NNL01]. While the CS method assumed an underlying binary tree, the schemes in [Asa02] were allowed to have arity greater than or equal to two. It utilized the master-key technique of Chick-Tavares in [CT89]. This scheme obtained the header size $O(r(\log_a(n/r) + 1))$ and required $O(1)$ user storage. However, the key computing technique required multiplication of large primes and hence was quite inefficient. The paper does not discuss how to extend the Subset Difference technique of [NNL01] for trees of arity greater than two.

The Subset Difference technique of [NNL01] was extended for ternary trees in [FKTS08]. The key assignment technique of [FKTS08] however could not be extended to higher arities. To quote from their paper (page 236 of the WISA 2008 proceedings):

> *"However, in a general a-array tree with $a \geq 4$, there exists sets of nodes that are inconsecutive ... Our hash chain approach fails with regard to these inconsecutive points. Thus, the construction of a coalition resistant a-array SD method with reasonable communication, computation, and storage overhead is an open issue."*

Analysis of Complete Subtree and Subset Difference Based Schemes [PB06, EOPR08, AK08, MMW09]. An analysis of the expected header length of the SD and LSD schemes was done in [PB06]. They proposed generating functions for counting the number of ways p users out of total n users can be given access privilege so that the header length will be h. Using this generating function, they found equations to compute the expected header length for a given n and r. However, they admitted that their equations were "complex to compute and difficult to gain insight from". Consequently, they went forward to find *approximations* for the same.

The analysis of the expected header length in [PB06] was continued in [EOPR08] to show that the standard deviations are small compared to the means as the number of users gets large.

Other combinatorial studies of the SD method have been performed in [MMW09, AK08]. In particular, the accurate values of the maximum possible header length for a given n and varying ranges of r for the NNL-CS and NNL-SD schemes [NNL01, NNL02] were found in [MMW09]. They also did comparative analysis of the NNL-CS and the NNL-SD schemes, establishing the worst-case broadcast size for both these schemes.

Stateful Subset Cover [CGZ+04, JKL06]. According to [CGZ+04], statelessness comes at a cost in terms of storage and message overhead when the number of privileged users is much smaller than the total number of users. Rather than maintaining a large static key tree $\mathcal{T}^0$ that accommodates all potential users, they used a smaller dynamic key tree for only currently privileged users. Current privileged users were assigned dynamically to the positions in $\mathcal{T}^0$ rather than using a fixed pre-assignment. The smaller key tree requires less storage and dynamic assignment achieves a smaller rekeying cost. They empirically compared performances and showed that the dynamic scheme significantly improved the performance as compared to the NNL-SD scheme, reducing by half the rekey communication cost when the number of privileged users were much smaller than the total number of users. Compared to the NNL-SD scheme, the dynamic SD scheme did not need to know the maximum number of potential group members in advance.

In [JKL06], it was shown how a key server using a BE scheme falling under the subset cover framework, can establish a common session key K_s for a dynamically changing group (i.e., multiple members can join and leave together). We know already from Section 2.1 that subset cover schemes define a family $\mathcal{S}$ of subsets of $\mathcal{N}$, where each subset is associated with a key. To distribute a new session key K_s, the key server generates the subset cover $\mathcal{S}_c$ and encrypts K_s multiple times using the key L_i of each subset $S_i \in \mathcal{S}_c$. In [JKL06], they presented a technique where an additional *state key* is encrypted along with the new session key. These new keys are held only by the current privileged users of the new session. Thus, the scheme is stateful where at the time of distribution of a new session key, the state key is used to transform all subset keys for the privileged users of that session. Since only current privileged users have access to the state key, the key server does not need to avoid covering all of $\mathcal{R}$, but only those who were recently removed in the previous session (and thus have a current state key). This improves the transmission efficiency of the SD scheme whose header length is linear in the number r of revoked users. This technique could be applied to any scheme that comes under the subset cover framework. It was applied on the SD scheme [NNL01, NNL02] and the punctured interval scheme [JHC+05, CJKY08].

Efficient Tree-Based Revocation in Groups of Low-State Devices [GST04]. Several new techniques for BE were provided in [GST04] under the log-key restriction. Both the static (zero-state) and dynamic (low-state) versions were proposed. Their static scheme achieved communication overhead exactly the same as the NNL-SD scheme while the user storage requirement was reduced to $O(\log n)$ and the decryption time increased to $O(n)$. The reduction in storage as compared to the NNL-SD scheme is due to the technique used for assignment of keys to the subsets. For the key assignment, instead of a top-own traversal from the root directly to a node, *left and right preorder traversals* were used so that each user had to store only 2 seeds instead of $O(\log n)$ seeds for each ancestor as in the NNL-SD scheme. On the other hand, the decryption required tree traversal that would take $O(n)$ time. The seed assignment could also be restricted within a set of $(\log n)/k$ consecutive levels (for a fixed constant k) while there would be k such sets of levels. This resulted in the *stratified subset difference* scheme for which the header length and the user storage would grow k times while the decryption time required is at the best $O(n^{1/k})$.

3.3 Traitor Tracing Techniques

The traitor tracing technique (based on the bifurcation property of the subsets that have been assigned keys) for the NNL-SD scheme has been described before in Section 2.1.1. Here, we identify some of the other techniques that have been used in the literature.

(Threshold) Traitor Tracing; Chor, Fiat, Naor (Crypto, 1994) [CFN94], (IEEE-IT, 2000) [CFNP00], Naor, Pinkas (Crypto, 1998) [NP98]. Traitor tracing was first defined in [CFN94, CFNP00] as the technique to identify the leaked secret keys (of traitor devices) that are present in a pirate decoder by running experiments on it as a black-box. They introduced *k-resilient traceability schemes* that would identify from a pirate decoder, at least one traitor device and not accuse innocent parties even if up to k traitors colluded and combined their keys. In a *fully-resilient scheme*, at least one traitor can be traced from any pirate decoder that decrypts with non-negligible probability. In a *threshold tracing scheme*, if the pirate decoder decrypts with probability less than 1 but above some threshold, the scheme will be able to trace at least one traitor. They provided several k-resilient traceability schemes (some were fully-resilient and others were threshold tracing schemes) that used hash functions and any arbitrary symmetric key cryptosystem.

The underlying security assumptions were either information theoretic or were derived from the security of the respective symmetric key cryptosystems. They observed that threshold tracing schemes were more efficient than fully resilient schemes.

Efficient Trace and Revoke Schemes; Naor, Pinkas (Fin. Crypto., 2000; IJIS, 2010) [NP00, NP10]. An efficient revocation scheme based on secret sharing is designed that can revoke up to r users and is secure against their coalition. The scheme is efficient in terms of user storage, communication overhead and computation of the new common group key by virtue of the fact that none of these parameters depended on n. Traitor tracing techniques are also developed for this scheme. Additionally, they introduce the idea of *self-enforcement* for deterring users from revealing their keys to others. The self-enforcement property is obtained by giving each user a personal key, which contains some sensitive private information (say the user's credit card number). This personal key is required for the decryption of the content. It is reasonable to assume that users would be reluctant to disclose such personal and sensitive keys to pirates. Such deterrence of the users from leaking their secret keys may not succeed in preventing unintentional compromise of the secrets happening without the user's knowledge (like hack of the user device).

Dynamic Traitor Tracing; Fiat, Tassa (JoC, 2001) [FT01]. In scenarios where compromised keys are identified periodically, traitors have to be traced *dynamically*. In [FT01], such scenarios are considered where instead of a pirate decoder being constructed, a pirate re-broadcasts the original content to pirate users. This is accomplished by the use of *watermarking techniques* [1]. In their scheme the content is broken into segments and marked so that a segment re-broadcasted by the pirates, can be linked to a particular subgroup of users. Mark allocation for a segment is determined when the re-broadcast from the previous segment is observed. They showed that by careful design of the mark allocation scheme it is possible to detect all traitors. Quoting from [FT01], *"the watermarking problem is to generate multiple versions of watermarked content so that, given a black market copy of that content, the watermarks embedded in that copy would lead to the identification of its source"*. A broadcaster can watermark the original content to create different versions. These watermarks are used to trace the traitor devices from which keys were leaked. These dynamic schemes are based on some "feedback" from the pirate network and decides the number and

[1] Usually called *fingerprinting*.

identity of active traitors on the fly.

3.4 Code-Based Traitor Tracing

Coding Constructions for Blacklisting Problems without Computational Assumptions; Kumar, Rajagopalan, Sahai (Crypto, 1999) [KRS99]. One-time revocation of up to r users, secure against a coalition of all of them was proposed in [KRS99]. A constructive scheme using algebraic-geometric codes was proposed that required communication overhead of $O(r^2)$ and user storage of $O(rn)$. Another scheme based on polynomials was proposed with communication overhead $O(rn)$ and user storage $O(rn)$.

Sequential Traitor Tracing; Safavi-Naini, Wang (Crypto, 2000) [SNW00]. This work considers the same scenario as Fiat and Tassa [FT01] and proposes a new type of traceability scheme, called *sequential traitor tracing*. Here the marking allocation is predetermined and is independent of the re-broadcasted segment. It does not use the feedback signal used for mark allocation in [FT01] and hence, (i) it will not be vulnerable to *delayed rebroadcast attack* (where the attackers do not rebroadcast immediately, but decide to record the content and rebroadcast it at a later time), and (ii) it does not require real-time computation for mark allocation and so allows very short time slots. This is very attractive as it allows segments to be shortened and hence the overall convergence time reduces. The scheme is analyzed and two general constructions are given: one based on a special type of function family and the other on error correcting codes. The convergence time of these schemes is obtained and show that the scheme based on error correcting codes has a convergence time which is the same as the best known result for dynamic schemes.

Coding Theory Based Traceability Techniques; Staddon, Stinson, Wei (IEEE-IT, 2001) [SSW01b], Silverberg, Staddon, Walker (Asiacrypt, 2001; IEEE-IT 2003) [SSW01a, SSW03]. In [SSW01b], the authors suggested that codes may be introduced into copyrighted material transmitted using BE in order to implement traceability of schemes. Codes with identifiable parent property (IPP), traceability (TA) codes, frameproof (FP) codes, and secure-frameproof (SFP) codes were studied and equivalent formulations using structures such as perfect hash families were proposed.

In [SSW01a, SSW03], traceability schemes based on error-correcting codes were constructed. The tracing technique was based on list decoding algorithms and hence was much faster compared to the previously known traceability techniques. The traitors could be identified in time polynomial in the length of the underlying code rather than the number of codewords.

3.5 Key Predistribution Based Schemes

Unconditionally Secure Key Distribution and Broadcast Encryption; Blundo, Mattos, Stinson (Crypto, 1996; TCS, 1998) [BMS96, BMS98], Stinson (DCC, 1997) [Sti97], Stinson, Wei (SACrypt, 1998; SIAMDM, 1998) [SW98a, SW98b], Stinson, Trung (DCC, 1998) [SvT98]. All these works view most networks as broadcast networks where all users have access to the data flowing through it. To ensure confidentiality in such a network, only the intended users should be able to decrypt them correctly. A common key is needed to encrypt the plaintext message (1) when a center wants to broadcast secretly to a subset of users, or (2) when a subset of users want to communicate through a private conference. To do this, BE can be used to distribute the common key to all privileged users from a center (trusted authority or TA). This common key can also be dynamically computed through interactions among the privileged set of users through schemes are called Interactive Key Distribution (IKD) schemes. There have been several works [BMS96, BMS98, Sti97, SvT98, SW98a, SW98b] that have proposed IKD schemes and used them as part of BE schemes.

In [BMS96, BMS98] families of unconditionally secure BE and IKD schemes were proposed that could be used for a single broadcast or a single key distribution. The user storage and communication overhead of these schemes were recognized as the two most important parameters of these schemes. These families provide trade-offs between these two parameters.

These one-time schemes could be modified to general t-time schemes. In [Sti97], construction of key predistribution schemes by combining Mitchell-Piper IKD patterns [MP88] with resilient functions was described. Resilient functions were used to make IKD schemes more efficient. A general method to combine IKD schemes along with secret sharing schemes to get BE schemes was presented. Construction of the Fiat-Naor BE scheme [FN93] using this method was also proposed.

The work in [Sti97] was further extended in [SvT98] using combinatorial structures like orthogonal arrays, perpendicular arrays, Steiner systems and universal hash families. In [SW98a, SW98b] traceability of the above schemes were investigated and then key distribution schemes with more efficient traceability were proposed.

BE Schemes from Linear Algebraic Techniques for Key Predistribution; Padro, Gracia, Mollevi, Morillo (DCC, 2002; DAM, 2003) [PGMM02, PGMM03]. A new model for key predistribution based on linear algebraic techniques was proposed in [PGMM02] that provides a common mathematical formulation of the framework for key predistribution. The security of these schemes do not depend upon any computational assumption. The assignment of keys to subsets of users in these schemes depends upon a choice of vectors in some vector space. From such a scheme, a key predistribution scheme for the corresponding dual structure (obtained by exchanging privileged and revoked subsets) can also be found.

A method to construct a family of broadcast encryption schemes from linear key predistribution schemes was provided in [PGMM03]. These schemes were hence called linear broadcast encryption schemes. All previously known BE schemes could be obtained in this manner.

BE and KPS Schemes from PRGs [NNL01, NNL02, HS02, AKI03]. The NNL-SD [NNL01, NNL02] and the HS-LSD [HS02] have been discussed in details in Section 2.1.1 and Section 2.1.2 respectively. In [AKI03] the authors found a generic method to construct BE schemes and KPSs from pseudo-random sequence generators (PRGs) by observing a general *"sequential key-derivation patterns"* for doing so. Using this method, they found a technique to construct BE schemes that would support an arbitrary number of users while at the same time be secure against any set of colluding users. The NNL-SD and the HS-LSD schemes are special cases of this method. Using their techniques they improved the user storage of the NNL-SD and the HS-LSD schemes while maintaining the same communication overhead.

A dynamic subset difference scheme was devised in [CGZ$^+$04] where the underlying tree has only the currently privileged users. Hence, user storage is reduced as compared to the NNL-SD scheme. Consequently, the scheme is stateful and the user keys have to be updated from time to time. User are assigned positions in the tree dynamically. This involves some re-keying cost.

A method to convert stateless key revocation schemes based on the subset cover framework to stateful schemes was proposed in [JKL06]. This work provided stateful variants of the SD scheme that would require less communication bandwidth as compared to the LKH scheme which is also stateful.

3.6 Combinatorial Works

Combinatorial Bounds for Broadcast Encryption; Luby, Staddon (Eurocrypt, 1998) [LS98]. In [LS98], it is assumed that in a BE system, each time the set of privileged users changes, the center enacts a protocol to establish a new broadcast key. This new key can be obtained only by the privileged users and and all subsequent transmissions are encrypted using it. The inherent trade-off between the user storage (in terms of the number of keys stored) and the communication overhead (the number of transmissions needed to establish the new broadcast key) is studied in this work. For a given upper bound on the user storage, a lower bound on the communication overhead is proved. These bounds are also shown to be tight.

Efficient Methods for Integrating Traceability and Broadcast Encryption; Gafni, Staddon, Yin (Crypto, 1999) [GSY99]. In [GSY99], general methods for integrating traceability and broadcasting capability were studied. The integration problem was studied from both directions. (1) The first method for adding any desired level of traceability to an arbitrary broadcast encryption scheme was developed. The central idea behind the method for adding traceability to broadcast encryption schemes is the use of *randomness* when allocating keys to users. This allows the users' key sets to be dispersed, and hence aids traceability. (2) A new method for adding any desired level of broadcasting capability to an arbitrary traceability scheme was also developed. The main idea behind this method uses the inherent broadcasting capability in the underlying traceability scheme. By making use of such inherent broadcasting *structure*, significant efficiency improvements could be achieved over the method in [SW98b]. New constructions of broadcast encryption schemes were proposed that were close to optimal in terms of the total number keys required. These new schemes were the first to be both maximally resilient and fully scalable.

Long-Lived Broadcast Encryption; Garay, Staddon, Wool (Crypto, 2000) [GSW00]. In a BE scheme, the user keys may become unusable due to expiry of subscription or because they were compromised and hence revoked. At some point, a user may not be left with any usable secret key. In [GSW00], the authors suggest that if required, the user keys of a BE scheme may be updated by the center. (For that, there should be a unique uncompromised key for each such user.) Every time a certain number of users are revoked for either of the above reasons, the center assumes the start of a new *epoch* (time interval). At the end of each epoch, the smart cards of the legitimate users out of the d users are re-keyed (or the cards may be replaced). They called these long-lived BE schemes and pointed out that due to the revocation with re-keying technique, these schemes offered more comprehensive solutions to piracy than traitor tracing schemes. Long-lived schemes were also argued to be more efficient in the long run as compared to revocation schemes through re-keying. These schemes were based on the idea of dividing the set of users into a *cover-free family* of subsets [GSW00].

One-Way-Chain Based Schemes; Jho et al. (Eurocrypt, 2005) [JHC+05, CJKY08]. Another interesting work on BE is [JHC+05, CJKY08]. It works on the idea of *"one key per punctured interval"* in which the worst case header length has been brought down to r (the number of revoked users) for the first time. This can also be decreased below r at the cost of increasing user storage. But, the method is more complicated than the SD scheme and the user storage requirement is rather high. For $n = 2^{28}$ and $r = 2^{10}$, the header length is below r at the cost of 3.4×10^8 times the storage of the SD scheme.

A Broadcast Encryption Scheme with Free-Riders but Unconditional Security; Adelsbach, Greveler (DRM-TICS, 2005) [AG05]. In [AG05] two schemes were proposed for efficient broadcast key establishment that enabled a sender to communicate to any subset of users by allowing a small ratio of *free-riders*. The schemes do not require stateful receivers. One of the schemes provided unconditional security. The free-riders would not however be able to learn from the past whether they might become free-riders for a certain transmission again in future. Hence, the number (or ratio) of free-riders (usually assumed to be 0) was introduced as a new parameter for controlling the efficiency trade-offs in BE schemes. The amount of free-riders could be varied to get varying communication overheads and user storages.

3.7 Public Key BE

To start with, in many scenarios, we may not want the sender to have the decryption keys [NNL01, NNL02]. Broadcasting may also be decentralized[2]. The asymmetric key model for broadcast encryption helps there. The group of privileged users will have a public key. Anybody can broadcast information to those privileged users. Although, we only concentrate on tree-based symmetric key BE schemes in this thesis, discussing the public-key based schemes is essential for the sake of completeness of the related works in this area.

According to [NNL01, NNL02, DF03], *a public key trace and revoke scheme* combines the functionality of broadcast encryption with the capability of traitor tracing. Specifically, (1) a *trusted* center publishes a single *public key file* (associating public keys with subsets) and distributes individual secret keys to the users of the system; (2) anybody can encrypt a message so that all but a specified subset of revoked users can decrypt the resulting ciphertext; and (3) if a (small) group of users combine their secret keys to produce a pirate decoder, the center can trace at least one of the traitors given access to this decoder.

Here we list several important works on public key broadcast encryption.

Asymmetric Fingerprinting and Trials of Traced Traitors; Pfitzmann (Info. Hiding, 1996) [Pfi96], Pfitzmann, Waidner (ACM CCS, 1997) [PW97]. Since traitor tracing was first formally introduced in [CFN94], all traitor tracing mechanisms were symmetric key based until [Pfi96]. It was argued in [Pfi96] that in a symmetric key based tracing scheme, the traitors could always claim that it was the center that leaked the keys. There would be no mathematical proof of their guilt. In other words, symmetric key based tracing schemes could never offer non-repudiation[3]. In [Pfi96], the first asymmetric traitor tracing schemes were defined. Using these schemes, the center when confronted with treachery, obtains information that he could not have produced on his own. That is therefore much better evidence.

A technique to convert fingerprinting and traitor tracing schemes based on random codes

[2] When there is no single center for broadcasting. Broadcast ciphertexts may come from a number of parties.

[3] Non-repudiation refers to the feature that would provide proof of the integrity and origin of a data. In other words, it is a mechanism to ensure authentication. In this context, a traitor will not be able to deny its role in the data leakage.

to asymmetric schemes was proposed in [PW97]. Effectively one could have asymmetric schemes that have the same collusion tolerance as the best symmetric schemes without introducing any new restrictions.

Both these works emphasized on the techniques to ensure that the traitors were convicted through mathematical proofs of their misdeeds. They separately identified the *tracing protocol* from the *trial protocol*. While the tracing protocol only intends to trace a traitor and outputs the identity of the traitor and a string *proof*, the trial protocol enables the center (information provider) to convince an arbitrary third party called the *judge* that the traced user is a traitor. For this, the center uses the string *proof* from the tracing algorithm. Furthermore, the judge would require the public key that uniquely identifies the accused user to give a verdict. Depending on how many of these three parties' inputs are involved in the computations done by the judge, these trials would be called 2-party or 3-party trials.

An Efficient Public Key Traitor Tracing Scheme. Boneh, Franklin (Crypto, 1999) [BF99]. A simple and efficient solution to the traitor tracing problem was proposed in [BF99]. The tracing algorithm was deterministic and *all* active traitors could be identified while *never* accusing innocent users. The scheme was partially black-box though. A minor modification to the scheme could make it resist an adaptive chosen ciphertext attack. Error correcting codes were applied to the discrete log representation problem to get the traitor tracing scheme.

All previous solutions to the traitor tracing problem [CFN94, NP98, Pfi96, PW97, SW98a] were combinatorial with probabilistic tracing techniques. In [BF99], the techniques used were algebraic and the tracing was deterministic. This approach being inherently public key, it was more efficient than the public key instantiations of the previous combinatorial constructions.

Additionally, three models of traitor tracing were considered possible: *non-black-box* tracing model, *single-key-black-box* tracing model, and *general-black-box* tracing model.

A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares; Tzeng, Tzeng (PKC, 2001) [TT01, TT05]. The trace and revoke scheme proposed in [TT01, TT05] used dynamic share and user revocation techniques. The header length depended on the collusion and revocation thresholds and not on the number of privileged users. Each receiver was required to store only one decryption key. The traitor tracing algorithm assumed that the pirate decoder was a black-box. The distinctive feature of this scheme was that when the traitors were found, their private keys could be revoked (up to some threshold z) without updating any private key of the remaining subscribers. Furthermore, the decryption privilege of a revoked private key could be restored later. In fact, the revocation capability could also be increased beyond z with dynamic assignment of shares through the header. This property made this scheme highly practical. Previously proposed public-key traitor tracing schemes had to update all existing private keys even when revoking one private key only.

The scheme in [TT01, TT05] was as efficient as the one in [BF99] in many aspects. One of them being that the traitor tracing scheme of [TT01, TT05] was fully k-resilient[4].

Public-Key Schemes Based on the NNL-SD Scheme; Naor, Naor, Lotspiech (Crypto, 2001) [NNL01, NNL02], Dodis, Fazio (DRM, 2002) [DF02]. In [NNL01, NNL02] it was shown how any subset cover revocation algorithm can be used in the public key mode. The *trusted* center would generate the private keys corresponding to each subset in $\mathcal{S}$. It would then provide each user with the secret keys of every subset it belongs to. The sender(s) who generate the ciphertext should only have access to "the public key file". The function E_{L_i} of the subset cover framework should be a public key cryptosystem whereas F_{K_s} may be as described before in Chapter 2. In principle, any public key encryption scheme with desirable security can be used for E_{L_i}. However, not all yield a system with a reasonable efficiency. A Diffie-Hellman type scheme best serves this mode.

The novelty of using a PRG for key assignment brought down the storage requirement of the NNL-SD scheme. One may recall from Section 2.1.1 that seeds were assigned to nodes in a full binary tree. A seed assigned to a node was further used to derive seeds for nodes below and hence the keys that are assigned to the subsets. These were symmetric keys and hence were shared between the users and the center. In the public key mode, the derived symmetric key for a subset will be used as the random string that will be used to generate a public-key-private-key pair. This mapping of the random bits with the key pairs has to be efficient. It turns out that Diffie-Hellman scheme efficiently establishes this association.

The natural extension of the symmetric key SD scheme [NNL01, NNL02] resulted in the following inefficiencies: (1) the public key for every subset had to be stored and as a consequence, the public key file would be too large; and (2) the secret keys for the subsets had to be generated from the random bits resulting in enormous increase of the decryption

[4] All traitors could be traced if their number was k or less.

time; or these secret keys would have to be stored at the user resulting in a huge increase in the storage requirement. In [DF02], this problem was solved by reducing the public key size to a constant while the user storage and communication overhead was the same as the symmetric key version. It used the concept of Hierarchical Identity Based Encryption that allows the derivation of decryption keys for a node from its ancestor.

A crucial point here is the assignment of identities to subsets. Starting from the root, for any node in $\mathcal{T}^0$, the edge to its left child is marked with 0 and the one to the right is labelled with 1. The identifier for a node i denoted as $\mathsf{ID}(i)$ is the string of 0's and 1's formed by concatenation of the labels of the edges on the path joining the root node to the node i. Given a descendant j of a node i in $\mathcal{T}^0$, $\mathsf{ID}(i)$ will be a prefix of $\mathsf{ID}(j)$. The notation $\mathsf{ID}(j) \setminus \mathsf{ID}(i)$ would denote the string formed by the concatenation of the labels on the path from i to j. In other words, it is the suffix of $\mathsf{ID}(j)$ that follows right after the prefix $\mathsf{ID}(i)$ in $\mathsf{ID}(j)$. Each subset $S_i \in \mathcal{S}$ would thus be identified by

$$\mathsf{HID}(S_{i,j}) = (\mathsf{ID}(i), [\mathsf{ID}(j) \setminus \mathsf{ID}(i)], \nu)$$

where ν is a terminator indicating the end of the string.

Similar techniques to get public-key versions of the NNL-CS and HS-LSD schemes were also described in [DF02].

Self Protecting Pirates and Black-Box Traitor Tracing; Kiayias, Yung (Crypto, 2001) [KY01]. A generic black-box traitor tracing model was proposed in [KY01] where the pirate-decoder employs self-protection techniques against tracing. It was proved that for black-box traitor tracing of self-protecting pirate decoders, if the number of traitor keys is super-logarithmic in the number of users, it is not possible to trace without the decoder noticing it, unless queries of a specific type are used. They fit BE schemes (like that of Boneh-Franklin [BF99]) into their model and showed that they are not traceable in the self-protecting traitor model, unless the efficiency features of these schemes are relinquished. However, the Chor-Fiat-Naor scheme [CFN94] was still traceable under this model.

Public Key Trace and Revoke Scheme Secure Against Adaptive Chosen Ciphertext Attack; Dodis, Fazio (PKC, 2003) [DF03]. The first chosen ciphertext (CCA2) secure trace and revoke scheme based on the DDH assumption was constructed in [DF03].

They were the first to provide a precise formalization of an appropriate notion of adaptive security for Broadcast Encryption. The adversary was allowed to corrupt players at any point during execution. Prior works (e.g., [NP00, TT01, TT05]) only achieved a very weak form of non-adaptive security even against chosen plaintext attacks.

Multi-Service Oriented Broadcast Encryption; Narayanan, Rangan, Kim (ACISP 2003), Jiang, Gong (ACISP, 2004) [NRK03, JG04]. A multi-service oriented BE (MOBE) scheme assumes that there are a set of services $\mathcal{V} = \{v_1, v_2, \ldots, v_\rho\}$ provided by the system and a user $u \in \mathcal{N}$ may avail any subset of these services in $\mathcal{V}$. Such a scheme will be called *fully flexible*. An example of such a system would be the Pay-TV system with the provision to avail different channel packs.

The first few works on schemes for Pay-TV include [MQ95], [Woo98], [Woo98, Woo00] and [MV01]. An overview of the conditional access system and the issues of copyright protection and authentication in a Pay-TV systems were described in [MQ95]. A description of the conditional access system was given and the need for the use of a *trusted third party* was demonstrated. The design of efficient copyright protection by watermarking images and image authentication by signatures were also briefly discussed.

In [Woo98, Woo00], the schemes described allow the broadcaster to offer a hierarchy of packages to the users. In [MV01] however, the focus has been on unsubscription process being totally transparent to the users. To achieve this they use techniques that associate multiple decryption keys with one encryption key.

An RSA-based scheme was presented in [NRK03], where there was an increase in the transmission overhead by a constant factor (and not by the number n of users or the number ρ of services). The scheme was stateless and hence user memory was not required to be updated. A session (called the *billing period* in this work) is assumed to be a period in which there are no changes in subscription status. The session keys provided to each user changes with each subscription status change of that user. This scheme was fully resilient to traitors.

However, as mentioned in [JG04], the amount of secure user memory required by the scheme in [NRK03] was linear in the number of services subscribed to by that user. Moreover, a service unsubscription required a unicast channel for each user that was still privileged. Such a unicast channel had to be secured from everyone other than the concerned user getting its session key updated. Hence, this scheme was not very suitable for systems with too many users or services with frequent subscription status updates (session changes).

In [JG04], the authors proposed the $\mathcal{M}$ framework for the MOBE problem. They achieved the multi-service functionality from the subset cover method. A user's key size in $\mathcal{M}$ was independent of the number of users or services in the system. The revoked users do not get involved in subscription process of users to a service. Furthermore, unsubscription is handled scalably in the number of services and users, making the system flexible. This framework is instantiated with the complete subtree scheme [NNL01, NNL02] and Asano's scheme [Asa02]. Finally, in order to evaluate the security of the framework, the notion of *dynamic security* was formally introduced. This captured threats from an adaptive adversary that might issue queries such as subscription, rekeying, corruption and new service provision. It showed the $\mathcal{M}$ framework to be secure under such a severe attack. Their proof was in the random oracle model[5].

Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys; Boneh, Gentry, Waters (Crypto, 2005) [BGW05]. Two new public key BE schemes for stateless receivers were proposed in [BGW05]. Both these systems were fully secure against any number of colluding users. These constructions used groups with an efficiently computable bilinear map. In the first construction both ciphertexts and private keys were of constant size for any privileged subset of users. Each user's private key was just a single group element while the ciphertext had only two group elements for any arbitrary set of privileged users. The public key size of this system was linear in the total number of receivers. The second system was a generalization of the first. It provided a tradeoff between ciphertext size and public key size. For example, a collusion resistant BE system could be instantiated for n users where both ciphertexts and public keys were of size $O(\sqrt{n})$ for any

[5] A random oracle is a black-box that is assumed to respond to every query with a random response chosen from its output domain. When hash functions used in a scheme cannot be proved to possess the mathematical properties required by the proof, they are assumed to be random oracles. The random oracle model assumes every hash function to be a random oracle.

Starting from [CGH04], there have been several criticisms of the random-oracle model. It has been argued that a scheme that is proven secure in the random-oracle model may have insecure implementations due to construction of the random-oracle using hash functions. However, these constructions of random oracles are of the type that in an extremely rare case, the bonafide user reveals the secret key. The Koblitz-Menezes riposte [KM15] is based on precisely this point. For practical systems, the secret key will not be given out in any circumstances and so the constructions provided to highlight the shortcomings of the random oracle model are artificial. Whether or not one has confidence in the random oracle model, the [CGH04] type constructions should not be used to discredit the random-oracle model for any practical cryptographic system.

subset of receivers. These systems could also be modified trivially to be used as group key management methods with short key update messages.

Scalable Public-Key Tracing and Revoking; Dodis, Fazio, Kiayias, Yung (Distributed Computing, 2005) [DFKY05]. In certain scenarios, both the user population and the set of content providers are *dynamic* (they may join or leave the system any time). Thus scalable user management and scalable provider management are crucial. In [DFKY05], the first public-key traitor tracing scheme for such a dynamic scenario was proposed. They proposed an efficient scalable public key traitor tracing scheme in which the populations of providers and users could change dynamically over time without incurring substantial penalty in terms of system performance and management complexity. A formal model for *scalable public key traitor tracing* was introduced and the first construction of such a scheme was presented. This model mandated deterministic traitor tracing and unlimited number of efficient provider and user management operations. As with other algebraic schemes, black-box traceability could not be satisfied efficiently in the construction of [DFKY05]. A formal adversarial model for the system was presented. The construction was proved to be secure against both adversaries (1) that attempt to cheat the provider and user management mechanism, and (2) adversaries that attempt to cheat the traitor tracing mechanism.

Privacy in Encrypted Content Distribution Using Private Broadcast Encryption; Barth, Boneh, Waters (Fin. Crypto, 2006) [BBW06]. It may be important to both restrict access of content to authorized users as well as to protect the identities of the users in content distribution schemes. For example, an encrypted file should hide who can access the content. Identity protection (also called recipient privacy) is achieved by introducing a notion called *private broadcast encryption* in [BBW06]. A private broadcast encryption scheme is used to encrypt a message to several recipients while hiding the identities of the recipients, even from each other. A private broadcast scheme is constructed in [BBW06], with a strong privacy guarantee against an active attacker, while being efficient in terms of ciphertext length and encryption-decryption time.

3.8 Miscellaneous

Renewable Traitor Tracing: a Trace-Revoke-Trace System for Anonymous Attack; Jin, Lotspiech (ESORICS, 2007) [JL07]. When traitors are identified, a *renewable* scheme can revoke and exclude the decryption keys used by the traitors during piracy. A renewable scheme is stateless by definition. In a BE system that uses hybrid encryption (as in the subset cover framework), the content encrypting (session) key or the content itself may get leaked. In that case the traitors remain *anonymous*, and hence it might not be possible to trace those traitors.

In [JL07], a renewable traitor tracing scheme was designed for this kind of *anonymous attack*. In this scheme, the revocation information included in a newly released (after a trace-revoke action) broadcast content will not only disallow traitors to playback the new content, but also provide new tracing information for continuous tracing. Such a system is therefore said to be a *trace-revoke-trace* system and hence [JL07] claim to have first proposed such a system for anonymous attack. In this scheme, the content owner would choose different points in the content and would encrypt these points differently and augment them to the otherwise same encryptions of the content, giving a new version each. Each version would also be differently watermarked. Each user gets one such unique version. Each device would be able to decrypt its own part from the augmented portions. This effectively creates different versions quite efficiently.

Unifying Broadcast Encryption and Traitor Tracing for Content Protection; Jin, Lotspiech (ACSAC, 2009) [JL09]. It must be evident by now that the features traceability and revocation demand different types of design. Schemes with both these features combined in it, usually leave one of these two aspects weak. Moreover, pirate attacks on these schemes may be through clone devices or through anonymous re-broadcasting. These two types of attacks were usually considered to be orthogonal to each other and hence have been tackled separately using two different trace-and-revoke schemes for each. In [JL09], the authors present a unified trace-and-revoke scheme that offered a very efficient solution for both revocation and traceability as well as simultaneously defending against the two types of attacks in a unified way. They also showed the equivalence of the clone attack with the anonymous re-broadcasting attack [JL09].

Chapter 4

The Complete Tree Subset Difference Scheme and its Analysis

4.1 Introduction

In Chapter 1, we listed a summary of our contributions included in this chapter that were published in [BS13]. We recollect them very briefly here.

Arbitrary Number of Users. The NNL-SD scheme described in Chapter 2 and all follow-up works [HS02, GST04, PB06, AK08, MMW09] assume the total number of users n to be a power of two. The actual number of users in real-life implementations may not be a power of two. Hence, the center has to assume the existence of dummy users to make the number of users a power of two. We relax this restriction to allow any arbitrary number of users in the system by introducing the Complete Tree Subset Difference (CTSD) scheme. When the number of users in the CTSD method is a power of two, it becomes exactly the same as the SD scheme. Inclusion of dummy users results in the expected header length of the SD scheme to be more than the CTSD scheme for practical values of n and r.

Combinatorial Analysis. We carry out a combinatorial study for the CTSD scheme and the results so obtained also apply to the SD scheme. A new approach is used for the detailed combinatorial analysis to count the number, $N(n, r, h)$, of ways that r out of n users can be revoked to get a header length of h in the CTSD scheme. This counting is formulated using two recurrences. Using these recurrences, a dynamic programming based algorithm is developed to compute $N(n, r, h)$ in polynomial time. Previous to our work, to compute $N(n, r, h)$ for the SD method, one would have to run the SD algorithm on the possibly exponentially many $\binom{n}{r}$ revocation patterns. As mentioned in Chapter 1, we obtain several interesting combinatorial results using these recurrences.

Probabilistic Analysis. We propose a simple and efficient algorithm for computing the expected header length for a given n and r in the CTSD and hence the SD method. The algorithm requires $O(r \log n)$ multiplications and $O(1)$ space. It can be used for all practical values of the parameters and hence it provides a useful tool to practitioners implementing either the SD or the CTSD method.

We show that the limiting upper bound on the expected header length is $1.25r$. The only previously known upper bound on the expected header length in the SD scheme for r revoked users was proved to be $1.38r$ in [NNL01, NNL02]. They also commented that experimental results indicated that the bound is probably $1.25r$. Our analysis of the expected header length shows that proving the precise limiting upper bound is more complicated than anticipated in [NNL01, NNL02].

4.2 The Complete Tree Subset Difference Method

The Subset Difference (SD) method described in Chapter 2 [NNL01, NNL02] and all follow-up work assumes the number n of users to be a power of two. We propose the Complete Tree Subset Difference (CTSD) algorithm that can accommodate any arbitrary number of users. Our algorithm considers a rooted complete binary tree $\mathcal{T}^0$ with n leaves. One may

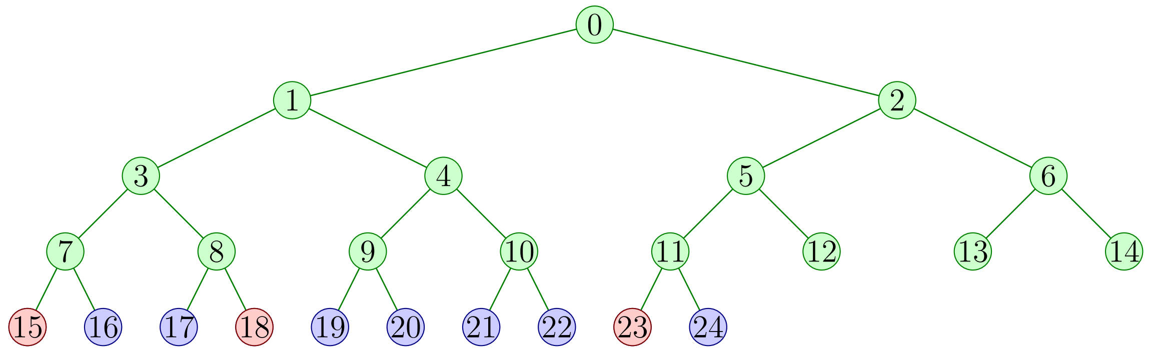


Figure 4.1: The *non-full complete* tree $\mathcal{T}^0$ with $n = 13$ users as its leaves. Privileged users are indicated in green and the revoked users are indicated in red. Here, $r = 3$. The tree $\mathcal{T}^1$ is a subtree of $\mathcal{T}^0$ and is a *full* subtree having 8 leaf nodes whereas the tree $\mathcal{T}^2$ is a *non-full complete* subtree of $\mathcal{T}^0$ with 5 leaf nodes.

note here that a *complete binary tree* has leaf nodes only at the bottom-most or last level and maybe also the last-but-one level. The leaves in the last level are filled from the left to the right in the tree. In a *full binary tree* of height ℓ there are 2^ℓ leaves, all at the last level. A full binary tree is also complete by definition. We will refer to trees that are complete but not full as *non-full*. Each user in $\mathcal{N}$ is associated with a leaf of the complete binary tree $\mathcal{T}^0$. There are a total of $2n-1$ nodes in $\mathcal{T}^0$. The root node of $\mathcal{T}^0$ is labeled as 0. All subsequent nodes are labeled as follows: the left child node of a node i is labeled as $2i+1$ and the right child is labeled as $2i+2$. Hence, nodes 0 to $n-2$ are the internal nodes and nodes $n-1$ to $2n-2$ are the leaf nodes. The subtree of $\mathcal{T}^0$ rooted at node i is denoted by $\mathcal{T}^i$. The number of leaf nodes in the subtree $\mathcal{T}^i$ is denoted by λ_i.

The collection $\mathcal{S}$ of subsets is defined as follows: The set $S_{i,j}$ is defined to contain users in the subtree $\mathcal{T}^i$ but *not* in $\mathcal{T}^j$. All subsets of users of the form $S_{i,j}(= \mathcal{T}^i \setminus \mathcal{T}^j)$, where node j is in the subtree $\mathcal{T}^i$ and hence a *descendant* of node i, is included in the collection $\mathcal{S}$. The set $\mathcal{N}$ of all users is also included in $\mathcal{S}$. Once this collection $\mathcal{S}$ has been created, each set $S_{i,j}$ in $\mathcal{S}$ has to be assigned a long-lived key $L_{i,j}$. We will look at the key assignment in Section 4.2.1.

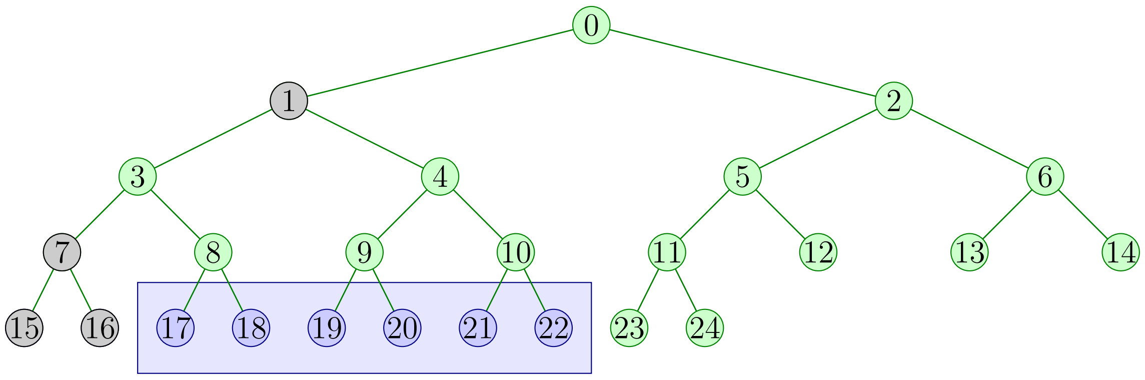


Figure 4.2: The *subset difference* subset $S_{1,7}$ which includes leaves in $\mathcal{T}^1$ but not in $\mathcal{T}^7$ i.e.; $S_{1,7} = \mathcal{T}^1 \setminus \mathcal{T}^7 = \{17, 18, 19, 20, 21, 22\}$.

During broadcast, the center will know the set $\mathcal{R}$ of revoked users and the message M to be broadcast. It has to find the subset cover $\mathcal{S}_c$ for $\mathcal{N} \setminus \mathcal{R}$. $\mathcal{S}_c$ contains pairwise disjoint sets $S_{i_1,j_1}, \ldots, S_{i_h,j_h}$ such that $\mathcal{N} \setminus \mathcal{R} = \bigcup_{k=1}^{h} \mathcal{S}_{i_k,j_k}$ where each S_{i_k,j_k} is taken from $\mathcal{S}$. If the set $\mathcal{R}$ is empty, then the only set in the cover $\mathcal{S}_c$ is $\mathcal{N}$. Otherwise, the following *cover-finding algorithm* is used: The center first constructs the Steiner Tree $\mathcal{ST}(\mathcal{R})$ induced by $\mathcal{R}$ on $\mathcal{T}^0$. The Steiner Tree $\mathcal{ST}(\mathcal{R})$ is a subgraph of $\mathcal{T}^0$ that only retains the nodes and edges on paths from the root node 0 to a revoked leaf node. All the other paths in $\mathcal{T}^0$ are deleted. The cover-finding algorithm runs iteratively by maintaining a tree $\mathcal{T}$ that is a sub-graph of $\mathcal{ST}(\mathcal{R})$. It starts by initializing $\mathcal{T}$ as a copy of $\mathcal{ST}(\mathcal{R})$. At every iteration, the algorithm keeps removing nodes from $\mathcal{T}$ while adding subsets to $\mathcal{S}_c$, until $\mathcal{T}$ has just one node left. At any point of time in the algorithm, a leaf node in $\mathcal{T}$ corresponds to either a leaf node in $\mathcal{T}^0$ or the root of a subtree in $\mathcal{T}^0$ all whose leaves have already been covered till that iteration. More precisely:

1. If there is only one leaf node in $\mathcal{T}$, jump to step 6.

2. Find two leaves j_1 and j_2 of $\mathcal{T}$ whose first common *ancestor* i does not have any other leaf node in its subtree in $\mathcal{T}$. Here, out of the many possible such pairs j_1 and j_2 one may choose the leftmost to have a specific algorithm.

3. Let i_1 (respectively i_2) be the immediate child node of i which is an ancestor of j_1 (respectively j_2) or is the node j_1 (respectively j_2) itself. If $i_1 \neq j_1$ then add the set S_{i_1,j_1} to the cover $\mathcal{S}_c$. Similarly, if $i_2 \neq j_2$ then add the set S_{i_2,j_2} to the cover $\mathcal{S}_c$.

4. Delete the paths joining j_1 and j_2 with their common ancestor i. Hence, node i becomes a leaf in $\mathcal{T}$.

5. If there are more than one leaves remaining in $\mathcal{T}$, go back to step 2.

6. If the only leaf node is the node 0, then there are no more subsets to be added to $\mathcal{S}_c$. Else, add the set $S_{0,j}$ to $\mathcal{S}_c$. Here j is the leaf node remaining in $\mathcal{T}$.

4.2.1 Key Assignment to each Subset $S_{i,j}$ in $\mathcal{S}$

Pseudo-Random Generator G. In order to assign keys to each subset in $\mathcal{S}$, the center assigns uniform random seeds to every non-leaf node in $\mathcal{T}^0$ and uses a *cryptographic pseudo-random generator* G. The pseudo-random generator G outputs a pseudo-random string that has three times the length of the input seed. The output string $G(seed)$ is divided into three equal parts $G_L(seed)$, $G_M(seed)$ and $G_R(seed)$. Hence, $G(seed) = G_L(seed) \parallel G_M(seed) \parallel G_R(seed)$. $G : \{0,1\}^k \to \{0,1\}^{3k}$ is a pseudo-random generator if no polynomial time adversary can distinguish between its output for a random seed from a truly random string of the same length. A hash function may be used in place of the PRG as was described in Chapter 2.

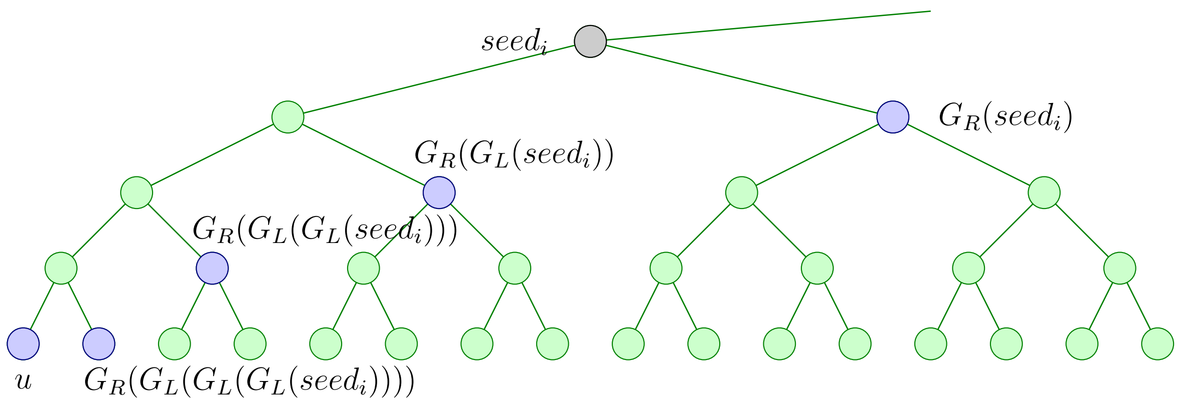


Figure 4.3: Secrets stored by u.

Seed Assignment to Nodes. Every non-leaf node i in $\mathcal{T}^0$ is assigned a uniform random string $seed_i$. Each non-root node j of $\mathcal{T}^0$ is assigned derived seeds from every ancestor i of j. The left child $2i+1$ of node i in $\mathcal{T}^0$ derives the seed $G_L(seed_i)$ from the random string $seed_i$ of i. All descendants of $2i+1$ further get derived seeds from this derived seed $G_L(seed_i)$ of $2i+1$. Similarly, the right child $2i+2$ of node i in $\mathcal{T}^0$ derives the seed $G_R(seed_i)$ from the random seed of i and all descendants of $2i+2$ get derived seeds from this derived seed $G_R(seed_i)$ of $2i+2$. We denote the seed for a node j derived from the random seed of node i as $seed_{i,j}$. Following such an assignment of random and derived seeds for nodes in $\mathcal{T}^0$, the long lived key $L_{i,j}$ assigned to the set $S_{i,j}$ is $G_M(seed_{i,j})$.

I_u **for each** $u \in \mathcal{N}$**.** Once the center is done with the assignment of random and derived seeds to nodes, it has to distribute the secret information I_u to each user $u \in \mathcal{N}$. The user associated with a leaf j of $\mathcal{T}^0$ must have been revoked when a set $S_{i,j}$ is in the cover $\mathcal{S}_c$. Hence, the user at leaf j should not be able to compute the $L_{i,j}$ for any of its predecessor i in $\mathcal{T}^0$. In fact, it should not be able to compute any $L_{i,k}$ where k, a descendant of i, is also one of its ancestors. In other words, a user at leaf j should be able to compute an $L_{i,k}$ if and only if i is an ancestor of j and k being a descendant of i, is not on the path joining j with i. In a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ to which a user at leaf j belongs, the node i has a random string $seed_i$. The user at j gets the seeds of all nodes adjacent to the path joining i and j that have been derived from $seed_i$ as shown in Figure 4.3. Say $i_1, \ldots, i_m$ are those nodes "falling off" from the path between node i and leaf j. The user at j will get the derived seeds $seed_{i,i_1}, seed_{i,i_2}, \ldots, seed_{i,i_m}$. To summarize, the I_u for a user u at leaf j consists of all derived seeds $seed_{i,k}$ such that i is a predecessor of j and k is adjacent to the path joining i and j. As derived in [NNL01, NNL02], the number of derived seeds in I_u is $\frac{1}{2}\log^2 n + \frac{1}{2}\log n + 1$ for n a power of two. For an arbitrary n, one has to consider the next higher power of two, say $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$. The number of derived seeds in I_u will be $\frac{1}{2}\ell_0^2 + \frac{1}{2}\ell_0 + 1$.

4.2.2 Dummy Users and the Associated Penalty

The CTSD scheme works with the actual number of users that are present in the system. It may be argued that even if n is not a power of two, the SD scheme can be applied by incorporating dummy users to make the total number of users to be a power of two. We argue that this impacts the size of the transmission overhead. For an actual broadcast, there are two ways to handle the dummy users – either consider all of them to be revoked or consider all of them to be privileged.

Suppose that the dummy users are considered to be distributed randomly among all the users. Then viewing them as revoked has very serious performance penalties. This is because the average header length is linear in the number of revoked users, as is proved later. Having a larger number of *randomly distributed* [1] revoked users leads to larger header size. If, on the other hand, the dummy users are viewed as privileged, then the performance penalty will be less.

Assuming the dummy users to be randomly distributed may not be fully justifiable. In an actual implementation, they may be considered to be one block. Suppose that $2^{\ell-1} < n < 2^{\ell}$ and that the users numbered $n+1, \ldots, 2^{\ell}$ are the dummy users and the real users are numbered 1 to n. The actual revoked users will be among the values 1 to n, whereas the users numbered $n+1, \ldots, 2^{\ell}$ will be considered to be either all revoked or all privileged.

We compare the expected header length of the CTSD method with the SD method in Table 4.1. These values are obtained by running the header generation algorithms on all possible (n, r)-revocation patterns. The SD algorithm is run assuming the dummy users to form a block at the right end of the tree. In separate cases, these dummy users are considered to be privileged and revoked as a group. Due to the exponentially many possible revocation patterns, the algorithm could be run only for small values of n. We, however, expect the

[1] The rationale behind the assumption that the revoked users are randomly distributed is the lack of any known distribution for revoking users. However, analysis of the schemes based on more realistic assumptions will be interesting and has been enlisted in the future directions of research in Section 8.4.

results to be indicative of the general behavior. For $17 \leq n \leq 24$ and $2 \leq r \leq 8$, the expected header length by the CTSD method is never more than that of the SD method and is almost always less.

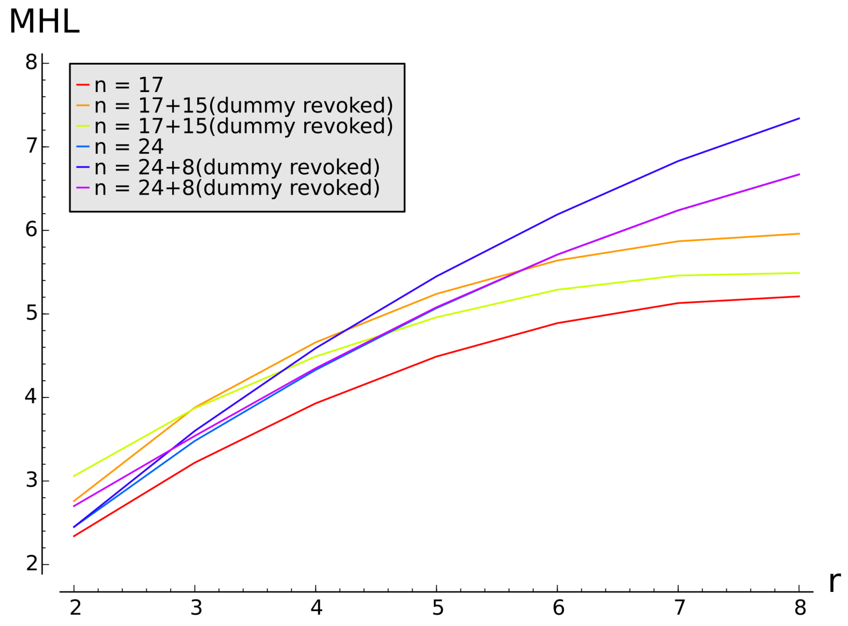


Figure 4.4: Plot showing how MHL varies with r in presence/absence of (privileged/revoked) dummy users.

4.3 Combinatorial Analysis of the SD and CTSD Methods

A given set of revoked users is called a *revocation pattern*. We denote a revocation pattern on n users where r are revoked, as an (n, r)-*revocation pattern*. The number of possible (n, r)-revocation patterns is $\binom{n}{r}$. In order to study the detailed combinatorial behavior of the CTSD and hence the SD algorithm, we find a method to count the number of (n, r)-revocation patterns that result in a header length of h.

Definition 3. *In a subtree $\mathcal{T}^j$ of $\mathcal{T}^0$ with λ_j users, $N(\lambda_j, r, h)$ is defined as the number of (λ_j, r)-revocation patterns that are covered by exactly h subsets. Similarly, for λ_j users in*

Table 4.1: Comparison of the expected header lengths for $17 \leq n \leq 24$ and $2 \leq r \leq 8$ in the CTSD method with the SD method working with dummy users forming a block at the right end. The dummy users may be privileged or revoked. It shows that the CTSD scheme always requires less bandwidth compared to the SD scheme with dummy users.

n	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$
17 (CTSD)	2.34	3.22	3.93	4.49	4.89	5.13	5.21
$17 + 15$ (dummy revoked)	3.06	3.87	4.49	4.96	5.29	5.46	5.49
$17 + 15$ (dummy privileged)	2.76	3.88	4.66	5.24	5.64	5.87	5.96
18 (CTSD)	2.36	3.29	4.05	4.67	5.14	5.45	5.60
$18 + 14$ (dummy revoked)	3.04	3.88	4.53	5.04	5.41	5.65	5.74
$18 + 14$ (dummy privileged)	2.67	3.76	4.53	5.09	5.51	5.78	5.92
19 (CTSD)	2.37	3.32	4.09	4.73	5.21	5.55	5.74
$19 + 13$ (dummy revoked)	3.12	4.01	4.72	5.27	5.69	5.97	6.11
$19 + 13$ (dummy privileged)	2.61	3.72	4.52	5.16	5.67	6.07	6.35
20 (CTSD)	2.39	3.38	4.19	4.86	5.39	5.77	6.02
$20 + 12$ (dummy revoked)	2.86	3.70	4.40	4.98	5.44	5.80	6.03
$20 + 12$ (dummy privileged)	2.56	3.66	4.48	5.15	5.69	6.12	6.44
21 (CTSD)	2.40	3.38	4.20	4.88	5.43	5.85	6.15
$21 + 11$ (dummy revoked)	3.69	4.44	5.07	5.60	6.02	6.35	6.56
$21 + 11$ (dummy privileged)	2.52	3.64	4.52	5.26	5.90	6.43	6.84
22 (CTSD)	2.42	3.43	4.27	4.98	5.58	6.06	6.42
$22 + 10$ (dummy revoked)	3.19	4.09	4.86	5.50	6.01	6.40	6.69
$22 + 10$ (dummy privileged)	2.49	3.62	4.53	5.31	5.99	6.56	7.03
23 (CTSD)	2.43	3.44	4.28	4.99	5.60	6.09	6.48
$23 + 9$ (dummy revoked)	3.27	4.20	5.01	5.68	6.23	6.66	6.98
$23 + 9$ (dummy privileged)	2.47	3.62	4.58	5.41	6.14	6.77	7.28
24 (CTSD)	2.45	3.48	4.33	5.07	5.71	6.24	6.67
$24 + 8$ (dummy revoked)	2.70	3.54	4.35	5.08	5.71	6.24	6.67
$24 + 8$ (dummy privileged)	2.45	3.60	4.59	5.45	6.19	6.83	7.34

$\mathcal{T}^j$, $T(\lambda_j, r, h)$ is defined as the number of (λ_j, r)-revocation patterns that are covered by h subsets such that there is at least one revoked user in both subtrees of $\mathcal{T}^j$.

Since the tree $\mathcal{T}^0$ has n $(= \lambda_0)$ leaves, $N(n, r, h) = N(\lambda_0, r, h)$ is the number of (n, r)-revocation patterns covered by a header length of h. We obtain recurrences for $N(n, r, h)$.

4.3.1 Some Notation

Level Number and Position of Nodes. Before we start deriving the expressions for $T(n, r, h)$ and $N(n, r, h)$, we fix a few notation for the ease of description. A level number of $\mathcal{T}^0$ is indicated by ℓ. In particular, the level of a node i is denoted by ℓ_i. The root node 0 is at the highest level ℓ_0. Hence, $\ell \in \{0, \ldots, \ell_0\}$. Since every subtree $\mathcal{T}^i$ is a complete binary tree, $2^{\ell_i - 1} < \lambda_i \le 2^{\ell_i}$. The number of nodes at level ℓ of $\mathcal{T}^0$ is denoted by q_ℓ. We see that the number of nodes at the last level is $q_0 = 2(n - 2^{\ell_1})$. For $\ell \in \{1, \ldots, \ell_0\}$, $q_\ell = 2^{\ell_0 - \ell}$. The position of a node at a level from the left is denoted by t where t ranges from 1 to q_ℓ. Hence, a node i is uniquely represented by the pair (ℓ_i, t_i) – the level ℓ_i of $\mathcal{T}^0$ to which it belongs and its position t_i from the left at that level. As an example, the root node 0 of $\mathcal{T}^0$ is represented by $(\ell_0, 1)$. We will interchangeably use both i and (ℓ_i, t_i) to denote a node.

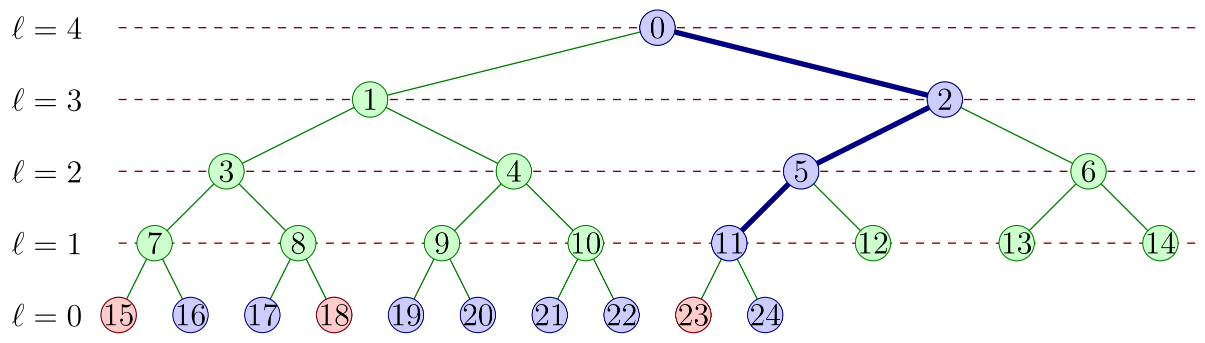


Figure 4.5: Level numbers in $\mathcal{T}^0$. The path $\mathcal{P}_0$ is marked with blue. Nodes colored blue are at position $t_\ell^{\mathcal{P}}$ for the respective level ℓ.

Non-Full Subtrees at each Level of $\mathcal{T}^0$. Let us take a closer look at the structure of the tree $\mathcal{T}^0$. In case $\mathcal{T}^0$ is full, all its subtrees are also full. In case $\mathcal{T}^0$ is non-full, we observe that every level $\ell > 0$ of $\mathcal{T}^0$ can have at most one non-full subtree. To identify these subtrees, we look at the path joining the root node 0 of $\mathcal{T}^0$ with node $n-2$ and denote it by $\mathcal{P}_0$. The node numbered $n-2$ is the last non-leaf node. There is exactly one node on $\mathcal{P}_0$ for every level $\ell > 0$ of $\mathcal{T}^0$. For level ℓ, the position of the node lying on the path $\mathcal{P}_0$ from the left, is denoted by $t_\ell^{\mathcal{P}}$. Let j be a node on $\mathcal{P}_0$, say the node represented by $(\ell, t_\ell^{\mathcal{P}})$. The part of the path $\mathcal{P}_0$ lying in the subtree $\mathcal{T}^j$ is denoted as $\mathcal{P}_j$. For the level ℓ, the subtree $\mathcal{T}^j$ rooted at node $(\ell, t_\ell^{\mathcal{P}})$ is the only possibly non-full subtree rooted at level ℓ. The subtrees to the left and right of node $t_\ell^{\mathcal{P}}$ at level ℓ are all full. The subtrees to the left (respectively right) of node $t_\ell^{\mathcal{P}}$ of level ℓ have 2^ℓ (respectively $2^{\ell-1}$) leaves. The number of leaves in the only possibly non-full subtree rooted at level ℓ is denoted by $\lambda^{\ell,\mathcal{P}}$. The root node of this subtree would be node $(\ell, t_\ell^{\mathcal{P}})$ of level ℓ. Hence, $2^{\ell-1} < \lambda^{\ell,\mathcal{P}} \leq 2^\ell$. More specifically, $\lambda^{\ell,\mathcal{P}} = n - ((t_\ell^{\mathcal{P}} - 1) \times 2^\ell) - ((2^{\ell_0-\ell} - t_\ell^{\mathcal{P}}) \times 2^{\ell-1})$. Also, $t_\ell^{\mathcal{P}} = \left\lceil \frac{q_0}{2^\ell} \right\rceil$. We define $t_\ell^{\mathcal{P}_j}$ for the path $\mathcal{P}_j$ as the position of the node at level ℓ on $\mathcal{P}_j$ from the left in the subtree $\mathcal{T}^j$. Hence, $t_\ell^{\mathcal{P}}$ is also denoted as $t_\ell^{\mathcal{P}_0}$. One can see that $t_\ell^{\mathcal{P}_j} = \left\lceil \frac{q_0 - (t_{\ell_j}^{\mathcal{P}} - 1) \times (2^{\ell_j})}{2^\ell} \right\rceil = \left\lceil \frac{q_0}{2^\ell} \right\rceil - (t_{\ell_j}^{\mathcal{P}} - 1) \times (2^{\ell_j - \ell})$.

4.3.2 Recurrences $N(n, r, h)$ and $T(n, r, h)$

Theorem 3. *For a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ with λ_i $(2^\ell < \lambda_i \leq 2^{\ell+1})$ leaves,*

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{j \in \mathsf{IN}(i)} T(\lambda_j, r_1, h_1 - 1), \tag{4.1}$$

where $\mathsf{IN}(i)$ is the set of all internal nodes in the subtree $\mathcal{T}^i$ excluding the node i.

Proof. We show that a revocation pattern is counted in $N(\lambda_i, r_1, h_1)$ if and only if it is counted in exactly one of $T(\lambda_i, r, h)$ or $T(\lambda_j, r, h-1)$ for some $j \in \mathsf{IN}(i)$. First we consider a (λ_i, r)-revocation pattern that is counted in $N(\lambda_i, r, h)$. There exists a minimal subtree $\mathcal{T}^j$, with $j \in \mathsf{IN}(i)$, of $\mathcal{T}^i$ that contains all the revoked leaves. If this subtree is rooted at i itself, then that revocation pattern is counted in $T(\lambda_i, r, h)$ and is covered by h subsets of $\mathcal{S}$. For any other node $j \neq i$, the revocation pattern is counted in $T(\lambda_j, r, h-1)$ and has to be covered by $h-1$ subsets of $\mathcal{S}$. The rest of the $\lambda_i - \lambda_j$ privileged users form one SD subset of the cover. The total cover size will hence be h. Since a set $\mathcal{R}$ of revoked users has a corresponding unique minimal subtree $\mathcal{T}^j$ of $\mathcal{T}^i$ containing all the users in $\mathcal{R}$, hence it is counted exactly once on the right side of (4.1).

Now, let us consider a (λ_i, r)-revocation pattern that has been counted in $T(\lambda_i, r, h)$. By the definitions of T and N, the (λ_i, r)-revocation patterns that are counted in $T(\lambda_i, r, h)$ are also counted in $N(\lambda_i, r, h)$. For some other revocation pattern, counted in $T(\lambda_j, r, h-1)$ for some $j \in \mathsf{IN}(i)$, both subtrees of $\mathcal{T}^j$ contain at least one revoked user in each. Hence, the minimal subtree of $\mathcal{T}^i$ containing the r revoked users for such a revocation pattern is $\mathcal{T}^j$. For the revocation patterns counted in $T(\lambda_j, r, h-1)$, the privileged users of the subtree $\mathcal{T}^j$ have been covered with $h-1$ SD subsets of $\mathcal{S}$. The rest of the $\lambda_i - \lambda_j$ users are all privileged and are covered by one more SD subset $S_{i,j}$. Hence, the corresponding (λ_i, r)-revocation pattern is counted in $N(\lambda_i, r, h)$. $\square$

Theorem 4. *For a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ with λ_i $(2^\ell < \lambda_i \le 2^{\ell+1})$ leaves,*

$$T(\lambda_i, r_1, h_1) = \sum_{r'=1}^{r_1-1} \sum_{h'=0}^{h_1} N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h'), \tag{4.2}$$

where λ_{2i+1} (respectively λ_{2i+2}) is the number of leaves in the left (respectively right) subtree of $\mathcal{T}^i$.

Proof. We show that a revocation pattern is counted in $T(\lambda_i, r_1, h_1)$ if and only if it is counted in the right hand side of (4.2). For a given λ_i, the number of leaves in the left and right subtrees get fixed to λ_{2i+1} and λ_{2i+2} respectively. When a (λ_i, r_1)-revocation pattern is counted in $T(\lambda_i, r_1, h_1)$, both the subtrees of $\mathcal{T}^i$ must have at least one revoked user. Assuming the left subtree of $\mathcal{T}^i$ has r' revoked users, the right subtree should have $r_1 - r'$ revoked users since the total number of revoked users is r_1. Similarly, assuming that the privileged users in this left subtree are covered by h' sets of $\mathcal{S}$, the privileged users in the right subtree should be covered by $h_1 - h'$ sets of $\mathcal{S}$. The number of (λ_{2i+1}, r')-revocation patterns in the left subtree covered by h' subsets is $N(\lambda_{2i+1}, r', h')$. Similarly, the number of $(\lambda_{2i+2}, r_1 - r')$-revocation patterns in the right subtree covered by $h_1 - h'$ subsets is $N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. Each such (λ_{2i+1}, r')-revocation pattern in the left subtree along with a $(\lambda_{2i+2}, r_1 - r')$-revocation pattern in the right subtree gives rise to a (λ_i, r)-revocation pattern in the tree $\mathcal{T}^i$ that is covered by h_1 subsets of $\mathcal{S}$. Hence, for all values of $r' \in \{1, \ldots, r_1 - 1\}$ and all values of $h' \in \{0, \ldots, h_1\}$, $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$ counts all the possible $T(\lambda_i, r_1, h_1)$.

Any (λ_i, r_1)-revocation pattern covered by h' subsets will be counted in the expression $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. The ones counted in $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$ for fixed values of r' and h' are counted exactly once in it. For other values of

Table 4.2: Boundary conditions on $T(n, r, h)$ and $N(n, r, h)$.

$T(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \leq r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 \geq 1$	0	0	0	from (4.2)	0	0
$N(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \leq r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 = 1$	0	1	n	from (4.1)	0	0
$h_1 > 1$	0	0	0	from (4.1)	0	0

r' and h', the corresponding (λ_i, r_1)-revocation patterns will be counted in the respective $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. Hence, a (λ_i, r_1)-revocation pattern is counted on the right hand side of (4.2) if and only if it is counted in $T(\lambda_i, r_1, h_1)$. $\square$

Boundary Conditions. The boundary conditions on $T(\lambda_i, r_1, h_1)$ and $N(\lambda_i, r_1, h_1)$ are given in Table 4.2. Other than the tabulated values, $N(\lambda_i, r_1, h_1) = 0$ for $\lambda_i \leq 0$ and $T(\lambda_i, r_1, h_1) = 0$ for $\lambda_i \leq 1$. From recurrences in Theorems 3 and 4 and the boundary conditions on these recurrences, one can find the value of $N(n, r, h)$ for any given n, r and h using dynamic programming.

4.3.3 Algorithms to Compute $N(n, r, h)$ and $T(n, r, h)$

Substituting for $j \in \mathsf{IN}(i)$. To use these recurrences as an algorithm, the nodes $j \in \mathsf{IN}(i)$ in (4.1) for a node i have to be explicitly identified and the corresponding λ_js have to be substituted. As described in Section 4.3.1 before, there are at most three types of subtrees rooted at a level ℓ_j of $\mathcal{T}^0$: full subtrees of height ℓ_i, full subtrees of height $\ell_i - 1$ and a non-full complete subtree of height ℓ_i.

(1) For a subtree $\mathcal{T}^i$ that is full and is of height ℓ_i and to the left of the node at position $t^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{\ell_j=1}^{\ell_i-1} (2^{\ell_i - \ell_j}) \times T(2^{\ell_j}, r_1, h_1 - 1). \tag{4.3}$$

(2) For a subtree $\mathcal{T}^i$ that is full and is of height $\ell_i - 1$ and to the right of the node at position $t^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{\ell_j=2}^{\ell_i-1} (2^{\ell_i-\ell_j}) \times T(2^{\ell_j-1}, r_1, h_1 - 1). \tag{4.4}$$

(3) For the only possibly non-full subtree $\mathcal{T}^i$ for $i = (\ell_i, t^{\mathcal{P}}_{\ell_i})$ of height ℓ_i and at position $t^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$\begin{aligned} N(\lambda_i, r_1, h_1) = {} & T(\lambda_i, r_1, h_1) \\ & + \sum_{\ell_j=2}^{\ell_i-1} [(t^{\mathcal{P}_i}_{\ell_j} - 1) \times T(2^{\ell_j}, r_1, h_1 - 1) + T(\lambda^{\ell_j,\mathcal{P}}, r_1, h_1 - 1) \\ & \qquad + (2^{\ell_i-\ell_j} - t^{\mathcal{P}_i}_{\ell_j}) \times T(2^{\ell_j-1}, r_1, h_1 - 1)]. \end{aligned} \tag{4.5}$$

Dynamic Programming. Computing $N(n, r, h)$ and $T(n, r, h)$ requires computing the values of $N(\lambda_i, r_1, h_1)$ and $T(\lambda_i, r_1, h_1)$ for some smaller λ_i, r_1 and h_1. We use dynamic programming technique where all values of $N(\lambda_i, r_1, h_1)$ and $T(\lambda_i, r_1, h_1)$ for smaller λ_i, r_1 and h_1 are pre-computed. The algorithm to compute $T(n, r, h)$ from these pre-computed values is obtained from (4.2) in a straightforward manner. The algorithm to compute $N(n, r, h)$ from these pre-computed values is obtained from (4.1). More specifically from either of (4.3) or (4.5). Level ℓ_i of $\mathcal{T}^0$ has $t^{\mathcal{P}}_{\ell_i} - 1$ full subtrees of height ℓ_i, $(2^{\ell_0-\ell_i}) - t^{\mathcal{P}}_{\ell_i}$ full subtrees of height $\ell_i - 1$ and one possibly non-full subtree. For every level in the tree $\mathcal{T}^0$, $T(\lambda_i, r, h - 1)$ is pre-computed once for each of the three types of nodes and used to compute $N(n, r, h)$.

Space and Time Complexity of the Algorithm. Using (4) to compute $T(n, r, h)$ from the pre-computed values of $N(\cdot, \cdot, \cdot)$ requires $O(rh)$ memory operations and multiplications. Equation (3) shows how $N(n, r, h)$ is related to pre-computed values of $T(\cdot, \cdot, \cdot)$. Actual computation is done using (4.3), (4.4) and (4.5). This requires $O(1)$ memory operations and a single addition for each of the $\lceil \log n \rceil$ levels of $\mathcal{T}^0$. Hence, the time complexity for computing $T(n, r, h)$ and then $N(n, r, h)$ *from pre-computed values* is $O(rh + \log n)$.

These pre-computed values in turn need to be computed. By the form of (4.3), (4.4) and (4.5) there are $\log n$ subtrees to be considered. For each such subtree, $O(rh)$ values need to be computed and the computation of these will be based on values computed earlier. A dynamic programming algorithm proceeds in a bottom-up fashion by computing the $O(rh)$ values corresponding to smaller sub-trees and then using these to compute the values for progressively larger sub-trees. This takes a total of $O(r^2h^2 \log n + rh \log^2 n)$ time. The space requirement is given by the number of pre-computed values that need to be stored to compute $N(n, r, h)$. For each of the $O(\log n)$ sub-trees, a total of $O(rh)$ values need to be stored and so the space complexity is $O(rh \log n)$.

The above time and space complexities are required for a single set of values of n, r and h. For a fixed n and r, it may be required to compute the values of $N(n, r, h)$ for all possible values of h. This would be a typical requirement for a broadcast center which will have a fixed number of users and for a particular transmission knows the number of revoked users. The corresponding time and space complexities can be obtained by substituting an appropriate value for h. In Lemma 5 of Section 4.3.4, we show that $h \leq 2r - 1$ which gives the expressions $O(r^4 \log n + r^2 \log n)$ and $O(r^2 \log^2 n)$ for time and space complexities respectively. For large n and moderate values of r, these are practical complexities.

Further, allowing r to range over all the $O(n)$ possible values leads to $O(n^4 \log n + n^2 \log^2 n)$ time and $O(n^2 \log n)$ space complexities respectively. If we are interested in computing $N(i, r, h)$ for all $2 \leq i \leq n$ and all possible values of r and h, then the time and space complexities are $O(n^5 + n^3 \log n)$ and $O(n^3)$ respectively.

As an example, using this dynamic programming algorithm, we find that for $n = 126$, $r = 63$ and $h = 37$, the floating point value of $N(n, r, h)$ is 7.44×10^{35}. Note that computing such a value would not be possible by direct enumeration. Attempting direct enumeration, would require considering $\binom{126}{63}$ possible revocation patterns which is way beyond the present computational capabilities.

4.3.4 Upper Bounds on the Header Length

The header length is an important efficiency parameter of a broadcast encryption scheme. So, upper bounds on the header length of the SD and CTSD schemes are of practical interest. A detailed combinatorial analysis of upper bounds on the header length is presented in this section.

The result below shows that the header length of the CTSD scheme is upper bounded by $2r - 1$.

Lemma 5. *$N(\lambda_i, r_1, h_1) = 0$ when $h_1 > 2r_1 - 1$. $T(\lambda_i, r_1, h_1) = 0$ when $h_1 \geq 2r_1 - 1$.*

Proof. First we show that $T(\lambda_i, r_1, h_1) = 0$ when $h_1 \geq 2r_1 - 1$ in (4.1). We prove this from (4.2) by induction on r_1. The boundary conditions have been listed in Table 4.2. We know that, $2^{\ell_i - 1} < \lambda_i \leq 2^{\ell_i}$. By induction hypothesis, when $h' > 2r' - 1$ and $1 \leq r' < r_1$, $N(\lambda_{2i+1}, r', h') = 0$. If $h' \leq 2r' - 1$, then $h_1 - h' > 2r_1 - 1 - h' \geq 2r_1 - 1 - 2r' + 1 = 2(r_1 - r')$. Then, again by induction hypothesis, $N(\lambda_{2i+2}, r_1 - r', h_1 - h') = 0$. Hence, when $h_1 \geq 2r_1 - 1$, $T(\lambda_i, r_1, h_1) = 0$.

Now, if $h_1 > 2r_1 - 1$, the other terms on the right hand side of (4.1) are $T(\lambda_i, r_1, h_1 - 1)$ where $h_1 - 1 \geq 2r_1 - 1$ for all terms and hence are all 0 as proved above. Hence, when $h_1 > 2r_1 - 1$, $N(\lambda_i, r_1, h_1) = 0$. □

We later show that for sufficiently large n, $N(n, r, 2r - 1)$ is positive and also characterize the minimum n for which this happens. Next, we show that $N(n, r, h)$ is monotonic on n for fixed r and h.

Lemma 6. *Let $n_1 \geq n_2$. If $N(n_2, r, h) \neq 0$ then $N(n_1, r, h) \neq 0$. If $T(n_2, r, h) \neq 0$ then $T(n_1, r, h) \neq 0$.*

Proof. Let $T(n_2, r, h) \neq 0$. From (4.2) we get:

$$T(n_2, r, h) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{h} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

Let $RH = \{(r_1, h_1) \ldots, (r_s, h_s)\}$ be such that both $N(\lambda_1, r', h')$ and $N(\lambda_2, r - r', h - h')$ are non-zero (and hence $N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h')$ is non-zero) when $(r', h') \in RH$. Hence, we can also write:

$$T(n_2, r, h) = \sum_{(r', h') \in RH} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

Since $\lambda_1 < n_2$ (by the structure of $\mathcal{T}^0$ with n_2 leaves), hence by induction hypothesis, for any $\lambda \geq \lambda_1$, $N(\lambda_1, r, h) \neq 0$ implies $N(\lambda, r, h) \neq 0$. Similarly, since $\lambda_2 < n_2$, hence by induction hypothesis, for any $\lambda \geq \lambda_2$, $N(\lambda_2, r, h) \neq 0$ implies $N(\lambda, r, h) \neq 0$. When there are n_1 leaves in the tree let there be λ'_1 leaves in the left subtree and λ'_2 leaves in the right subtree of the root node. Hence, by the construction of $\mathcal{T}^0$, we get $\lambda'_1 \geq \lambda_1$ and $\lambda'_2 \geq \lambda_2$. In the expression for $T(n_1, r, h)$, for $(r', h') \in RH$, by induction hypothesis, $N(\lambda'_1, r', h')$ and $N(\lambda'_2, r-r', h-h')$ are both non-zero. Hence, for at least $(r', h') \in RH$, $N(\lambda'_1, r', h') \times N(\lambda'_2, r-r', h-h')$ is non-zero. Thus, $T(n_1, r, h) \neq 0$.

Now, let $N(n_2, r, h) \neq 0$. From (4.1) we get:

$$N(n_2, r, h) = T(n_2, r, h) + \sum_{j=1}^{n_2-2} T(\lambda_j, r, h-1).$$

Let $I = \{i_1, \ldots, i_t\}$ be the nodes of $\mathcal{T}^0$ (with n_2 leaves) such that $T(\lambda_i, r, h) \neq 0$ for $i \in I$. By induction hypothesis, for any $\lambda_j < n_2$ and $\lambda_i > \lambda_j$, if $T(\lambda_j, r, h) \neq 0$ then $T(\lambda_i, r, h) \neq 0$. Hence, we can also write:

$$N(n_2, r, h) = T(n_2, r, h) + \sum_{i \in I} T(\lambda_i, r, h-1).$$

Here, $T(n_2, r, h) \neq 0$ implies $T(n_1, r, h) \neq 0$ by the first part of this proof. By the construction of the tree $\mathcal{T}^0$, $\lambda'_i \geq \lambda_i$ where λ'_i is the number of leaves in the subtree rooted at node i of the tree $\mathcal{T}^0$ for n_1 leaves. By induction hypothesis, at least for $i \in I$, since $T(\lambda_i, r, h-1) \neq 0$, hence $T(\lambda'_i, r, h-1) \neq 0$. Thus, $N(n_1, r, h) \neq 0$. □

Now, we prove that if r is not small compared to n, then $T(n, r, 2r-2) = 0$.

Lemma 7. *For $n \leq 2^{2k+1}$ and $r > 2^k$, $T(n, r, 2r-2) = 0$.*

Proof. For $T(n, r, 2r-2)$ in (4.2), let $h' < 2r'-1$, then $h-h' = 2r-2-h' > 2r-2-2r'+1 = 2(r-r')-1$. Hence by Lemma 5, $N(\lambda_2, r-r', h-h') = 0$. Similarly, if $h' > 2r'-1$, $N(\lambda_1, r', h') = 0$. So, in the expression for $T(n, r, 2r-2)$, the terms on the right hand side of (4.2) are 0 if $h' \neq 2r'-1$. Hence,

$$T(n, r, 2r-2) = \sum_{r'=1}^{r-1} N(\lambda_1, r', 2r'-1) \times N(\lambda_2, r-r', 2(r-r')-1). \qquad (4.6)$$

Now by induction on λ_i, we prove that $N(\lambda_1, r', 2r'-1) = 0$ and $N(\lambda_2, r-r', 2(r-r')-1) = 0$. The boundary conditions have been listed in Table 4.2. By induction hypothesis, for $\lambda_i \leq 2^{2m+1}$ where $m < k$ and $r' > 2^m$ let us assume $T(\lambda_i, r', 2r'-2) = 0$. In (4.6), let $r' \geq \frac{r}{2}$ which implies $r' > 2^{k-1}$. Hence, for $\lambda_i \leq 2^{2k-1}$, $T(\lambda_i, r', 2r'-2) = 0$ by the induction hypothesis. Also, by Lemma 5, $T(\lambda_1, r', 2r' - 1) = 0$. Putting these values in (4.1), we get $N(\lambda_1, r', 2r' - 1) = 0$. Similarly, for $r - r' \geq \frac{r}{2}$ which implies $r - r' > 2^{k-1}$, we get $N(\lambda_2, r - r', 2(r - r') - 1) = 0$. Hence, from (4.6) $T(n, r, 2r - 2) = 0$. □

Some Insight. Given a revocation pattern, if we revoke one more user from it, that can result in either increase, decrease or no change in the cover size. An increase in cover size mostly happens when the newly revoked user is not adjacent to any previously revoked user. The cover size remains unchanged or decreases when the newly revoked user is adjacent to a previously revoked user. Decrease in cover size happens when the user in a singleton subset of the cover is revoked. As the number of revoked users increase, the maximum possible cover size for that number of revoked users increases up to a certain point. After that the maximum possible cover size decreases. One may also observe that for $n > 2$, i.e., $\ell_1 \geq 1$, $q_0/2 = n - 2^{\ell_1}$. Since 2^{ℓ_1} is even for $\ell_1 \geq 1$, hence when n is even $q_0/2$ is even and when n is odd $q_0/2$ is odd.

Lemma 8. *The header length in the CTSD method for n users is at most $\left\lfloor \frac{n}{2} \right\rfloor$ irrespective of the number of revoked users.*

Proof. First, we show that $N(n, r, h) = 0$ for $h > \frac{n}{2}$ for any r. We prove this by induction on n. From (4.1) we have:

$$N(n, r, h) = T(n, r, h) + \sum_{i=1}^{n-2} T(\lambda_i, r, h - 1)$$

and hence, $T(n, r, h) \leq N(n, r, h)$. When $\lambda_i < n$ and $h - 1 \geq \left\lfloor \frac{n}{2} \right\rfloor$, $N(\lambda_i, r, h - 1) = 0$. Thus, $\sum_{i=1}^{n-2} T(\lambda_i, r, h - 1) = 0$. From (4.2) we get:

$$T(n, r, h) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{h} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

When $h' > \frac{\lambda_1}{2}$, $N(\lambda_1, r', h') = 0$ by induction hypothesis. When $h' \leq \frac{\lambda_1}{2}$, since $h > \frac{n}{2}$, $h - h' > \frac{n}{2} - \frac{\lambda_1}{2} = \frac{\lambda_2}{2}$. Therefore, $N(\lambda_2, r - r', h - h') = 0$ by induction hypothesis. Hence, $N(n, r, h) = 0$ for $h > \frac{n}{2}$ for any r.

Next, we show that the upper bound of $\left\lfloor \frac{n}{2} \right\rfloor$ is actually achieved. First let us assume that n is even and hence $q_0/2$ is even. We construct a revocation pattern such that none of the users are revoked initially. Now, let us form a revocation pattern by revoking one user from each of the $q_0/2$ subtrees rooted at level 1 with leaves at level 0 and one user each from subtrees rooted at level 2 with leaves at level 1. Since all the privileged users would form singleton subsets in the cover for this revocation pattern, hence the header length for the revocation pattern thus constructed is of size q_1 $(= \frac{n}{2})$. Now, if we attempt to revoke any other user, then by pigeonhole principle, one of the sets in the cover gets removed and hence the header length decreases. Hence, for even n, the maximum header length is $\frac{n}{2}$.

For odd n, $q_0/2$ is odd. We construct a revocation pattern similarly by revoking one user from each of the $q_0/2$ subtrees rooted at level q_1 with leaves at level q_0 and one user each from subtrees rooted at level 2 with leaves at level 1. Since $q_0/2$ is odd, there will be one subtree with leaves at both levels 0 and 1. This subtree is rooted at the node at position $t_2^{\mathcal{P}}$. For this subtree, only one out of the three users in it is revoked. All the privileged users other than the one generated from the above subtree would form singleton subsets. Hence the cover size for the revocation pattern thus constructed is of size q_1 $(= \lfloor \frac{n}{2} \rfloor)$. This is again the maximum header length by the same argument as above.

Hence, the maximum header length is $\lfloor \frac{n}{2} \rfloor$ for n users. □

In Lemma 5, it has been shown that the header length of the CTSD scheme is at most $2r - 1$. For the special case of the SD method, this bound was proved in [NNL01, NNL02]. This bound is made more specific in Theorem 9 below for the CTSD and hence the SD method.

Theorem 9. *The maximum header length in the CTSD method for n users is* $\min(2r - 1, \lfloor \frac{n}{2} \rfloor, n - r)$.

Proof. The bounds $2r - 1$ and $\lfloor n/2 \rfloor$ have already been shown. We show the bound of $n - r$ on the header size. The proof of this is similar to the first part of the proof of Lemma 8, i.e., we show that $N(n, r, h) = 0$ for $h > n - r$.

For $\lambda_i < n$, we have $h - 1 > n - 1 - r \geq \lambda_i - r$ and hence using induction, $N(\lambda_i, r, h - 1) = 0$ which implies that $T(\lambda_i, r, h - 1)$ is also zero. Again, consider the value of $T(n, r, h)$ and the recurrence expressing this in terms of $N(\lambda_1, r', h')$ and $N(\lambda_2, r - r', h - h')$, where $\lambda_1 + \lambda_2 = n$. If $h' > \lambda_1 - r'$, then using induction, $N(\lambda_1, r', h') = 0$. So, suppose that $h' \leq \lambda_1 - r'$. Using $h > n - r$, we have $h - h' > (n - \lambda_1) - (r - r') = \lambda_2 - (r - r')$ and again using induction, $N(\lambda_2, r - r', h - h') = 0$.

This shows that $T(n,r,h) = 0$ which combined with the fact that the other relevant values of $T(\cdot,\cdot,\cdot)$ are zero, shows that $N(n,r,h) = 0$ for $h > n - r$. $\square$

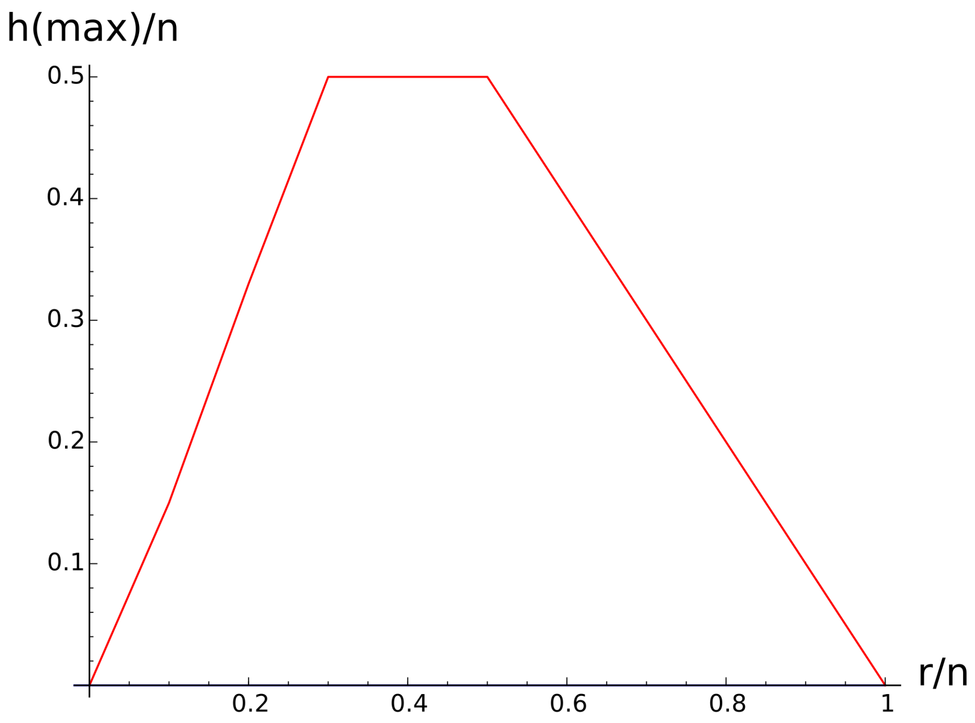


Figure 4.6: Plot showing the variation of the maximum header length with the ratio r/n.

The bound given by Theorem 9 gives a complete picture as portrayed in Figure 4.6. If $r \leq n/4$, then the bound $2r - 1$ is appropriate; if $n/4 < r \leq n/2$, then the bound $\lfloor n/2 \rfloor$ is appropriate; and for $r > n/2$, the bound $(n - r)$ is appropriate. The last bound has an important consequence. If the number of revoked users is greater than $n/2$, it may appear that using individual transmission to the privileged users would be better than using the CTSD method. But, The bound of $(n - r)$ on the header size shows that this is not true. Using the CTSD method is never worse than individual transmission to privileged users.

The bound of Theorem 9 holds for the SD scheme, i.e., for full trees. The only previously proved upper bound for the SD scheme is $2r - 1$. The other two bounds do not appear to have been reported with proofs in the literature. In fact, there does not seem to be an easy way to argue about these bounds without using the recurrences that we have derived.

The Value of n_r. Fix a value for r and denote by n_r the minimum value of n such that there exists an (n, r)-revocation pattern giving rise to a header of size $2r - 1$. Lemma 5 shows that the upper bound on the header length is $2r - 1$. By characterizing n_r we show that this upper bound on h is actually achieved.

Lemma 10. *In the CTSD method, $2^{t-1} < r \leq 2^t$ if and only if $2^{2t} < n_r \leq 2^{2t+1}$.*

Proof. We first prove that if $2^{t-1} < r \leq 2^t$, then $2^{2t} < n_r \leq 2^{2t+1}$ (by showing that $N(2^{2t}, r, 2r-1) = 0$ and $N(2^{2t+1}, r, 2r-1) \neq 0$). Although by Lemma 5, $T(2^{2t+1}, r, 2r-1) = 0$, we show that $T(2^{2t}, r, 2r-2) \neq 0$ and hence at least one of the terms on the right hand side of (4.1) is non-zero and hence $N(2^{2t+1}, r, 2r-1) \neq 0$. From (4.2) we get:

$$T(2^{2t}, r, 2r-2) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{2r-2} N(2^{2t-1}, r', h') \times N(2^{2t-1}, r - r', 2r - 2 - h').$$

When $h' > 2r'-1$, $N(2^{2t-1}, r', h') = 0$ by Lemma 5. Similarly, when $h' < 2r'-1$, $2r-2-h' > 2r - 2 - 2r' + 1 = 2(r - r') - 1$ and hence $N(2^{2t-1}, r - r', 2r - 2 - h') = 0$. Hence, we get

$$T(2^{2t}, r, 2r-2) = \sum_{r'=1}^{r-1} N(2^{2t-1}, r', 2r'-1) \times N(2^{2t-1}, r - r', 2(r - r') - 1).$$

When $r' = \lceil \frac{r}{2} \rceil$ $(2^{t-2} < r' \leq 2^{t-1})$ by induction hypothesis, $n_{r'} \leq 2^{2t-1}$ and hence by Lemma 6, both $N(2^{2t-1}, r', 2r'-1)$ and $N(2^{2t-1}, r - r', 2(r - r') - 1)$ are non-zero. Hence, $T(2^{2t}, r, 2r-2) \neq 0$ which implies $N(2^{2t+1}, r, 2r-1) \neq 0$. Since $T(n_r, r, 2r-1) = 0$ and $T(2^{2t-1}, r, 2r-2) = 0$ hence, $n_r < 2^{2t+1}$. Next, we show that $N(2^{2t}, r, 2r-1) = 0$. By Lemma 5, $T(2^{2t}, r, 2r-1) = 0$. By Lemma 7, for all $\lambda_i \leq 2^{2t-1}$ and $r > 2^{t-1}$, $T(\lambda_i, r, 2r-2) = 0$ and hence $N(2^{2t}, r, 2r-1) = 0$.

Next, we prove that for some $2^{2t} < n_r \leq 2^{2t+1}$, the corresponding r is such that $2^{t-1} < r \leq 2^t$. Let the corresponding r be such that $2^{t'-1} < r \leq 2^{t'}$ where $t \neq t'$. Then by the argument above, we know that $2^{2t'} < n_r \leq 2^{2t'+1}$ which is a contradiction since n_r is unique for a given r by definition. Hence the corresponding r is such that $2^{t-1} < r \leq 2^t$. □

Theorem 11 below characterizes n_r.

Theorem 11. *In the CTSD method, let $2^{t-1} < r \leq 2^t$. When $r \leq 2^{t-1} + 2^{t-2}$, let $r_1 = 2^{t-2}$ and $r_0 = r - 2^{t-2}$ and hence,*

$$n_r = n_{r_0} + 2^{2t-2} + 2^{2t-1}$$

Table 4.3: Listing a few values of r and their corresponding n_r.

r	1	2	3	4	5	6	7	8
n_r	2	6	18	22	66	70	82	86

and when $r > 2^{t-1} + 2^{t-2}$, *let* $r_0 = 2^{t-1}$ *and* $r_1 = r - 2^{t-1}$ *and hence,*

$$n_r = 2^{2t-1} + n_{r_1} + 2^{2t-1}.$$

Proof. From Lemma 10 we know that for $2^{t-1} < r \leq 2^t$, $2^{2t} < n_r \leq 2^{2t+1}$. For such an n_r, $\lambda_1 = n_r - 2^{2t-1}$ and $\lambda_2 = 2^{2t-1}$. From (4.1) we get

$$\begin{aligned} N(n_r, r, 2r-1) &= T(n_r, r, 2r-1) + T(n_r - 2^{2t-1}, r, 2r-2) + T(2^{2t-1}, r, 2r-2) \\ &+ \sum_{i=3}^{n_r-2} T(\lambda_i, r, 2r-2). \end{aligned} \tag{4.7}$$

From Lemma 5 we know that $T(n_r, r, 2r-1) = 0$. From Lemma 7 we know that when $r > 2^{t-1}$ and $\lambda_i \leq 2^{2t-1}$, $T(\lambda_i, r, 2r-2) = 0$. Hence the only non-zero component is $T(n_r - 2^{2t-1}, r, 2r-2)$. From (4.2) we get

$$N(n_r, r, 2r-1) = T(n_r - 2^{2t-1}, r, 2r-2) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{2r-2} N(\lambda_3, r', h') \times N(\lambda_4, r-r', 2r-2-h').$$

By an argument similar to the one used in the proof for Lemma 10, we get

$$N(n_r, r, 2r-1) = T(n_r - 2^{2t-1}, r, 2r-2) = \sum_{r'=1}^{r-1} N(\lambda_3, r', 2r'-1) \times N(\lambda_4, r-r', 2(r-r')-1).$$

By the construction of $\mathcal{T}^0$ and the fact that $\mathcal{T}^2$ does not have any revoked user, i.e. $T(2^{2t-1}, r, 2r-2) = 0$, it can be seen that $2^{2t-2} < \lambda_3 \leq 2^{2t-1}$ and $2^{2t-2} \leq \lambda_4 < 2^{2t-1}$.

When $r \leq 2^{t-1} + 2^{t-2}$, let $r' = r_0 = r - 2^{t-2}$ and $r - r' = r_1 = 2^{t-2}$. From the construction of the complete tree $\mathcal{T}^0$ for $(n_{r_0} + 2^{2t-2} + 2^{2t-1})$ users, it can be seen that $\lambda_3 = n_{r_0}$ and $\lambda_4 = 2^{2t-2}$. Hence, $N(\lambda_3, r', 2r'-1) = N(n_{r_0}, r_0, 2r_0 - 1) \neq 0$ by the definition of n_r. Also, from Lemma 6 and Lemma 10 we know that for $r = 2^t$ (consequently $n_r < 2^{2t+1}$) and $\lambda \geq 2^{2t+1}$, $N(\lambda, r, 2r-1) \neq 0$. So for $r_1 = r - r' = 2^{t-2}$ and $\lambda_4 = 2^{2(t-2)+2}$ we get, $N(\lambda_4, r-r', 2(r-r')-1) = N(2^{2t-2}, r_1, 2r_1-1) \neq 0$. Hence, for $r \leq 2^{t-1}+2^{t-2}$, $N(n_r, r, 2r-1) \neq 0$ where $n_r = n_{r_0} + 2^{2t-2} + 2^{2t-1}$.

Now, we show that for $2^{t-1} < r \leq 2^{t-1}+2^{t-2}$ ($r_0 = r-2^{t-2}$ and $r_1 = 2^{t-2}$), $N(n_r - 1, r, 2r-1) = 0$. In the tree $\mathcal{T}^0$ for $(n_{r_0}+2^{2t-2}+2^{2t-1})-1$ users, $\lambda_3 = n_{r_0}-1$ and $\lambda_4 = 2^{2t-2}$. Since there are $n_{r_0}-1$ users in $\mathcal{T}^3$, at most r_0-1 revoked users can be accommodated in $\mathcal{T}^3$ so that $N(\lambda_3, r', 2r'-1) \neq 0$ and hence $r' = r_0-1$ and $r-r' = 2^{t-2}+1$. By Lemma 10 for $r-r' > 2^{t-2}$, $n_{r-r'} > 2^{2t-2}$. But, $\lambda_4 = 2^{2t-2}$ and hence $N(\lambda_4, r-r', 2(r-r')-1) = 0$. Consequently, we get $N(n_r-1, r, 2r-1) = 0$.

When $r > 2^{t-1}+2^{t-2}$, let $r' = r_0 = 2^{t-1}$ and $r-r' = r_1 = r-2^{t-1}$. From the construction of the complete tree $\mathcal{T}^0$ for $(2^{2t-1}+n_{r_1}+2^{2t-1})$ users, it can be seen that $\lambda_3 = 2^{2t-1}$ and $\lambda_4 = n_{r_1}$. Hence, $N(\lambda_4, r-r', 2(r-r')-1) = N(n_{r_1}, r_1, 2r_1-1) \neq 0$ by the definition of n_r. From Lemma 6 and Lemma 10 we know that for $r = 2^t$ (consequently $n_r < 2^{2t+1}$) and $\lambda \geq 2^{2t+1}$, $N(\lambda, r, 2r-1) \neq 0$. So for $r_0 = r' = 2^{t-1}$ and $\lambda_3 = 2^{2(t-1)+1}$ we get, $N(\lambda_3, r', 2r'-1) = N(2^{2t-1}, r_0, 2r_0-1) \neq 0$. Hence, for $r > 2^{t-1}+2^{t-2}$, $N(n_r, r, 2r-1) \neq 0$ where $n_r = 2^{2t-1}+n_{r_1}+2^{2t-1}$.

Now, we show that for $r > 2^{t-1}+2^{t-2}$, i.e., $r_0 = 2^{t-1}$ and $r_1 = r-2^{t-1}$, $N(n_r-1, r, 2r-1) = 0$. In the tree $\mathcal{T}^0$ for $(2^{2t-1}+n_{r_1}+2^{2t-1})-1$ users, $\lambda_3 = 2^{2t-1}$ and $\lambda_4 = n_{r_1}-1$. Since there are $n_{r_1}-1$ users in $\mathcal{T}^4$, at most r_1-1 revoked users can be accommodated in $\mathcal{T}^4$ so that $N(\lambda_4, r-r', 2(r-r')-1) \neq 0$ and hence $r-r' = r_1-1$ and $r' = 2^{t-1}+1$. By Lemma 10 for $r' > 2^{t-1}$, $n_{r'} > 2^{2t-1}$. But, $\lambda_3 = 2^{2t-1}$ and hence $N(\lambda_3, r', 2r'-1) = 0$. Consequently, we get $N(n_r-1, r, 2r-1) = 0$. $\square$

From Theorem 11 it easily follows that for the SD method, for any r in the range $2^{t-1} < r \leq 2^t$, $n_r = 2^{2t+1}$. This has been earlier proved in [MMW09].

4.3.5 Generating Function

For the SD scheme the number of users is a power of 2. In this case, we show that the recurrences lead to a generating function for the sequence $N(n, r, h)$. Let the number of users be $n = 2^{\ell_0}$ and hence the tree $\mathcal{T}^0$ is full and of height ℓ_0. For a full tree $\mathcal{T}^0$, all subtrees $\mathcal{T}^i$ are full and at level ℓ, there are $2^{\ell_0-\ell}$ subtrees with 2^ℓ leaves in each.

We define $T_\ell(r,h) = T(2^\ell, r, h)$ and $N_\ell(r,h) = N(2^\ell, r, h)$. Then the recurrences (4.1) and (4.2) for counting the number of revocation patterns become.

$$N_{\ell_0}(r,h) = T_{\ell_0}(r,h) + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} \times T_\ell(r,h-1)\right). \tag{4.8}$$

$$T_{\ell_0}(r,h) = \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1). \tag{4.9}$$

The following result states the form of the generating function.

Theorem 12. *The generating function for the sequence $N_{\ell_0}(r,h)$ of numbers defined in (4.8) above, is given by $X_{\ell_0}(x,y)$ where*

$$\begin{aligned} X_{\ell_0}(x,y) &= \left(X_{\ell_0-1}(x,y) - xy^{2^{\ell_0-1}}\right)^2 + xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} \\ &\quad + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} xy^{2^{\ell_0}-2^\ell} \times \left(X_{\ell-1}(x,y) - xy^{2^{\ell-1}}\right)^2\right). \end{aligned} \tag{4.10}$$

Proof. Let $X_{\ell_0}(x,y)$ (respectively $Y_{\ell_0}(x,y)$) be the generating function for the sequence $N_{\ell_0}(2^{\ell_0}-r,h)$ (respectively $T_{\ell_0}(2^{\ell_0}-r,h)$).

$$X_{\ell_0}(x,y) = \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} N_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} \qquad Y_{\ell_0}(x,y) = \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} T_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} \tag{4.11}$$

By definition, when $\ell_0 = 0$, $Y_0(x,y) = 0$ and $X_0(x,y) = 1 + xy$ and when $\ell_0 = 1$, $Y_1(x,y) = 1$ and $X_1(x,y) = 1 + 2xy + xy^2$. Now, we note that:

$$\begin{aligned} X^2_{\ell_0-1}(x,y) &= \left(\sum_{r_1=0}^{2^{\ell_0-1}} \sum_{h_1=0}^{2^{\ell_0-1}-r_1} N_{\ell_0-1}(r_1,h_1)x^{h_1}y^{2^{\ell_0-1}-r_1}\right) \\ &\quad \times \left(\sum_{r_2=0}^{2^{\ell_0-1}} \sum_{h_2=0}^{2^{\ell_0-1}-r_2} N_{\ell_0-1}(r_2,h_2)x^{h_2}y^{2^{\ell_0-1}-r_2}\right) \\ &= \left(N_{\ell_0-1}(0,1)xy^{2^{\ell_0-1}} + \sum_{r_1=1}^{2^{\ell_0-1}} \sum_{h_1=0}^{2^{\ell_0-1}-r_1} N_{\ell_0-1}(r_1,h_1)x^{h_1}y^{2^{\ell_0-1}-r_1}\right) \end{aligned}$$

$$\times \left(N_{\ell_0-1}(0,1)xy^{2^{\ell_0-1}} + \sum_{r_2=1}^{2^{\ell_0-1}} \sum_{h_2=0}^{2^{\ell_0-1}-r_2} N_{\ell_0-1}(r_2,h_2)x^{h_2}y^{2^{\ell_0-1}-r_2} \right) \tag{4.12}$$

Putting $N_{\ell_0-1}(0,1) = 1$ in (4.12) we get:

$$\begin{aligned} X^2_{\ell_0-1}(x,y) &= \left(\sum_{r_1=1}^{2^{\ell_0-1}} \sum_{h_1=0}^{2^{\ell_0-1}-r_1} N_{\ell_0-1}(r_1,h_1)x^{h_1}y^{2^{\ell_0-1}-r_1} \right) \\ &\quad \times \left(\sum_{r_2=1}^{2^{\ell_0-1}} \sum_{h_2=0}^{2^{\ell_0-1}-r_2} N_{\ell_0-1}(r_2,h_2)x^{h_2}y^{2^{\ell_0-1}-r_2} \right) \\ &\quad + xy^{2^{\ell_0-1}} \left(\sum_{r_2=1}^{2^{\ell_0-1}} \sum_{h_2=0}^{2^{\ell_0-1}-r_2} N_{\ell_0-1}(r_2,h_2)x^{h_2}y^{2^{\ell_0-1}-r_2} \right) \\ &\quad + xy^{2^{\ell_0-1}} \left(\sum_{r_1=1}^{2^{\ell_0-1}} \sum_{h_1=0}^{2^{\ell_0-1}-r_1} N_{\ell_0-1}(r_1,h_1)x^{h_1}y^{2^{\ell_0-1}-r_1} \right) \\ &\quad + x^2y^{2^{\ell_0}} \end{aligned} \tag{4.13}$$

Let

$$\begin{aligned} C_{\ell_0}(x,y) &= \left(\sum_{r_1=1}^{2^{\ell_0-1}} \sum_{h_1=0}^{2^{\ell_0-1}-r_1} N_{\ell_0-1}(r_1,h_1)x^{h_1}y^{2^{\ell_0-1}-r_1} \right) \\ &\quad \times \left(\sum_{r_2=1}^{2^{\ell_0-1}} \sum_{h_2=0}^{2^{\ell_0-1}-r_2} N_{\ell_0-1}(r_2,h_2)x^{h_2}y^{2^{\ell_0-1}-r_2} \right) \\ &= \sum_{r=2}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1). \end{aligned} \tag{4.14}$$

Now we take a closer look at the generating function $Y_{\ell_0}(x,y)$ of (4.11):

$$Y_{\ell_0}(x,y) = \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} T_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r}$$

$$
\begin{aligned}
&= \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) \\
&= \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) \\
&= \sum_{r=2}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) \\
&\quad + \sum_{r=0}^{1} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) \\
&= \sum_{r=2}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) \qquad (4.15)
\end{aligned}
$$

In (4.15) above,

$$
\sum_{r=0}^{1} \sum_{h=0}^{2^{\ell_0}-r} x^h y^{2^{\ell_0}-r} \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1) = 0.
$$

The minimum value of r_1 or $r - r_1$ is 1. The maximum value for r_1 or $r - r_1$ such that $x^h y^{2^{\ell_0}-r}$ will have a non-zero coefficient $N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1)$ is 2^{ℓ_0-1}.

Hence, $C_{\ell_0}(x,y) = Y_{\ell_0}(x,y)$.

Let $A_{\ell_0-1}(x,y) = \left(\sum_{r=1}^{2^{\ell_0-1}} \sum_{h=0}^{2^{\ell_0-1}-r} N_{\ell_0-1}(r,h) x^h y^{2^{\ell_0-1}-r}\right)$. It can be easily seen that

$$
\begin{aligned}
X_{\ell_0-1}(x,y) &= \sum_{r=0}^{2^{\ell_0-1}} \sum_{h=0}^{2^{\ell_0-1}-r} N_{\ell_0-1}(r,h) x^h y^{2^{\ell_0-1}-r} \\
&= \sum_{h=0}^{2^{\ell_0-1}} N_{\ell_0-1}(0,h) x^h y^{2^{\ell_0-1}} + \sum_{r=1}^{2^{\ell_0-1}} \sum_{h=0}^{2^{\ell_0-1}-r} N_{\ell_0-1}(r,h) x^h y^{2^{\ell_0-1}-r} \\
&= xy^{2^{\ell_0-1}} + A_{\ell_0-1}(x,y) \qquad (4.16)
\end{aligned}
$$

Putting the value of $A_{\ell_0-1}(x,y)$ from (4.16) and the value of Y_{ℓ_0} from (4.15) into (4.13), we get:

$$
Y_{\ell_0}(x,y) = X_{\ell_0-1}^2(x,y) - 2xy^{2^{\ell_0-1}} X_{\ell_0-1}(x,y) - x^2 y^{2^{\ell_0}}
$$

$$= \left(X_{\ell_0-1}(x,y) - xy^{2^{\ell_0-1}}\right)^2. \tag{4.17}$$

Now, to find another relation between the generating functions $X_{\ell_0}(x,y)$ and $Y_{\ell_0}(x,y)$, we multiply both sides of (4.8) with $x^h y^{2^{\ell_0}-r}$ and sum both sides over $2 \le r \le 2^{\ell_0}$ and $0 \le h \le 2^{\ell_0}$:

$$\begin{aligned}\textstyle\sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} N_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} &= \sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} T_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} \\ &\quad + \sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} x^h y^{2^{\ell_0}-r} \times T_\ell(r,h-1)\right).\end{aligned} \tag{4.18}$$

Adding the values of $N_{\ell_0}(r,h)$ and $T_{\ell_0}(r,h)(=0)$ for $r < 2$ and $h \ge 1$ to both sides of (4.18) above, we get:

$$\begin{aligned}\textstyle\sum_{r=0}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} N_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} &= xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} + \sum_{r=0}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} T_{\ell_0}(r,h)x^h y^{2^{\ell_0}-r} \\ &\quad + \sum_{r=0}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} x^h y^{2^{\ell_0}-r} \times T_\ell(r,h-1)\right).\end{aligned} \tag{4.19}$$

Since for $h = 0$, $N_{\ell_0}(2^{\ell_0},0) = 1$ ($T_{\ell_0}(2^{\ell_0},0) = 1$) and for any $r < 2^{\ell_0}$, $N_{\ell_0}(r,0) = 0$ ($T_{\ell_0}(r,0) = 0$), from (4.19) above,

$$\begin{aligned}X_{\ell_0}(x,y) - 1 &= xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} + Y_{\ell_0}(x,y) - 1 \\ &\quad + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} \times \sum_{r=0}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} T_\ell(r,h-1)x^h y^{2^{\ell_0}-r}\right) \\ &= xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} + Y_{\ell_0}(x,y) - 1 \\ &\quad + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} xy^{2^{\ell_0}-2^\ell} \times \sum_{r=0}^{2^\ell} \sum_{h=0}^{2^{\ell_0}-r-1} T_\ell(r,h)x^h y^{2^\ell-r}\right).\end{aligned} \tag{4.20}$$

Since $2^{\ell_0} - r - 1 > 2^{\ell} - r$ for $1 \leq \ell \leq \ell_0 - 1$, hence from (4.20) we get:

$$X_{\ell_0}(x,y) \;= xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} + Y_{\ell_0}(x,y) + \textstyle\sum_{\ell=1}^{\ell_0-1}\left(2^{\ell_0-\ell}xy^{2^{\ell_0}-2^{\ell}} \times Y_{\ell}(x,y)\right). \quad (4.21)$$

Combining (4.17) and (4.21), we get:

$$\begin{aligned} X_{\ell_0}(x,y) &= \left(X_{\ell_0-1}(x,y) - xy^{2^{\ell_0-1}}\right)^2 + xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} \\ &\quad + \sum_{\ell=1}^{\ell_0-1}\left(2^{\ell_0-\ell}xy^{2^{\ell_0}-2^{\ell}} \times \left(X_{\ell-1}(x,y) - xy^{2^{\ell-1}}\right)^2\right). \end{aligned} \quad (4.22)$$

□

A similar generating function was found by Park and Blake in [PB06]. It was directly derived based on the structural properties of the tree. We have taken a different approach of first finding the recurrence relations for the sequence $N(n,r,h)$ and then deriving the generating function from it. (It is to be noted here that these generating functions are for the same sequences of $N(n,r,h)$ - only having different closed forms.)

4.4 Expected Header Length in the CTSD and SD Methods

In the previous section, we have studied upper bounds on the header length. In practice, however, it is of interest to know the average header length. This will provide a broadcast center with valuable information about the average communication bandwidth.

Given the number n of users such that $2^{\ell_0-1} < n \leq 2^{\ell_0}$, and the number r of revoked users, there are $\binom{n}{r}$ possible revocation patterns. Each such revocation pattern gives rise to a subset cover for the privileged users and hence a header in the ciphertext C. We now obtain an algorithm to compute the expected header length for a given n and r in the CTSD scheme. In particular this algorithm applies to the SD method and is of significant practical interest.

The Random Experiment.

We consider the random experiment where r out of the n initially un-revoked leaves of the tree $\mathcal{T}^0$ are chosen uniformly at random without replacement and revoked.

This gives rise to a random (n, r)-revocation pattern and hence a corresponding random subset cover $\mathcal{S}_c$ and its header length h. Let $X_{n,r}$ be the random variable taking the value of the header length h due to the (n, r)-revocation pattern of the above experiment. Next, we associate a random variable with each node of the tree $\mathcal{T}^0$. Let $X^i_{n,r} \in \{0, 1\}$ be a random variable associated with node i of $\mathcal{T}^0$. $X^i_{n,r} = 1$ denotes the event that the cover contains a subset $S_{i,j} = \mathcal{T}^i \setminus \mathcal{T}^j$ where j is some node in the subtree $\mathcal{T}^i$. In other words, when $X^i_{n,r} = 1$ we say that node i generates a subset for the cover. Similarly, $X^i_{n,r} = 0$ denotes the event that there is no subset $S_{i,j}$ in the cover. Since i is also represented by (ℓ_i, t_i), $X^i_{n,r}$ will also be written as $X^{\ell_i,t_i}_{n,r}$ whenever the nodes need to be viewed level-wise and is appropriate in the context.

The Expected Header Length. Since the header constitutes of subsets $S_{i,j}$, each rooted at a different node i, it is easy to see that, $X_{n,r} = X^0_{n,r} + X^1_{n,r} + \ldots + X^{n-2}_{n,r}$. By linearity of expectation:

$$E[X_{n,r}] = E[X^0_{n,r}] + E[X^1_{n,r}] + \ldots + E[X^{n-2}_{n,r}]. \tag{4.23}$$

Since all the random variables $X^t_{n,r}$ follow a *Bernoulli distribution with probability* $\Pr[X^t_{n,r} = 1]$, we get:

$$E[X_{n,r}] = \Pr[X^0_{n,r} = 1] + \Pr[X^1_{n,r} = 1] + \ldots + \Pr[X^{n-2}_{n,r} = 1]. \tag{4.24}$$

Calculating each of these $n-1$ probability terms individually would give the expected header length. However, the running time can be optimized. Recall that $\mathcal{P}^0$ is the unique path from the root to a leaf node which contains the nodes at which the non-full subtrees of $\mathcal{T}^0$ are rooted. As we had discussed before, the subtrees $\mathcal{T}^i$ for which i is not on $\mathcal{P}^0$ are full. For a level ℓ of $\mathcal{T}^0$ the subtrees to the left of $\mathcal{P}^0$ are all full and have equal number of leaves. Hence, $\Pr[X^i_{n,r} = 1]$ needs to be computed only once for every such node i to the left of $\mathcal{P}^0$ at level ℓ. Similarly for nodes to the right of $\mathcal{P}^0$. Hence, efficient computation of $E[X_{n,r}]$ using (4.24), boils down to finding $\Pr[X^j_{n,r} = 1]$ level-wise. There are q_ℓ internal nodes at all levels $\ell \geq 2$. At level 1, there are $n - 2^{\ell_1} = q_0/2$ internal nodes. The other $q_1 - (n - 2^{\ell_1})$

nodes at level 1 are leaves. Hence, (4.24) can also be written as:

$$E[X_{n,r}] = \sum_{\ell=2}^{\ell_0} \sum_{t=1}^{q_\ell} \Pr[X_{n,r}^{\ell,t} = 1] + \sum_{t=1}^{q_0/2} \Pr[X_{n,r}^{1,t} = 1]. \tag{4.25}$$

When $r = 0$, there is only one set $\mathcal{N}$ in the cover $\mathcal{S}_c$ and hence, $E[X_{n,0}] = 1$. Here on, we will consider $r \geq 1$.

$\Pr[X_{n,r}^{\ell_i,t_i} = 1]$ for the Node i of $\mathcal{T}^i$. The sibling subtree $\mathcal{T}^s$ of node i may be $\mathcal{T}^{i-1}$ on its left or $\mathcal{T}^{i+1}$ on its right. To find the probability that node i generates a subset $S_{i,j}$ for the cover, we observe that the event $X_{n,r}^i = 1$ occurs when the sibling subtree $\mathcal{T}^s$ of i has at least one revoked node and exactly one of the subtrees of i has at least one revoked user. We define the events R_{sb}^i, R_{lt}^i and R_{rt}^i for node i with respect to our random experiment. R_{sb}^i denotes the event that the number of revoked nodes in the sibling subtree of $\mathcal{T}^i$ is non-zero. R_{lt}^i (respectively R_{rt}^i) denotes the event that the number of revoked nodes in the left (respectively right) subtree $\mathcal{T}^{2i+1}$ (respectively $\mathcal{T}^{2i+2}$) is non-zero.

Lemma 13. *For an internal non-root node i in $\mathcal{T}^0$, the probability that the cover $\mathcal{S}_c$ contains a set of the form $\mathcal{T}^i \setminus \mathcal{T}^j$ where j is some node in the subtree $\mathcal{T}^i$, is given by* $\Pr[X_{n,r}^i = 1]$ *where*

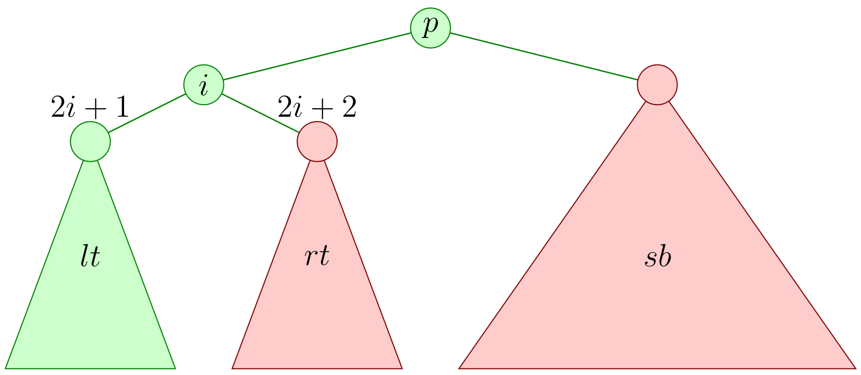
$$\Pr[X_{n,r}^i = 1] = \Pr[R_{sb}^i \wedge R_{rt}^i \wedge \overline{R_{lt}^i}] + \Pr[R_{sb}^i \wedge R_{lt}^i \wedge \overline{R_{rt}^i}].$$

For the root node 0*, this probability is given by* $\Pr[X_{n,r}^0 = 1]$ *where*

$$\Pr[X_{n,r}^0 = 1] = \Pr[\overline{R_{lt}^0}] + \Pr[\overline{R_{rt}^0}].$$

Proof. For a non-root node i, a subset $S_{i,j}$ occurs in the cover when there is at least one revoked user in exactly one of the subtrees $\mathcal{T}^{2i+1}$ or $\mathcal{T}^{2i+2}$ of i. The sibling subtree $\mathcal{T}^s$ should also have at least one revoked user. Hence the event $X_{n,r}^i = 1$ can be divided into two mutually exclusive and exhaustive events. First, when the sibling subtree and the right subtree of $\mathcal{T}^i$ have at least one revoked user in each and the left subtree does not have any: $(R_{sb}^i \wedge R_{rt}^i \wedge \overline{R_{lt}^i})$. Second, when the sibling subtree and the left subtree of $\mathcal{T}^i$ have at least one revoked user in each and the right subtree does not have any: $(R_{sb}^i \wedge R_{lt}^i \wedge \overline{R_{rt}^i})$.

The root node 0 does not have any sibling subtree. Hence the event $X_{n,r}^0 = 1$ occurs when all revoked users are either in the left or right subtree of 0. Hence the lemma. □



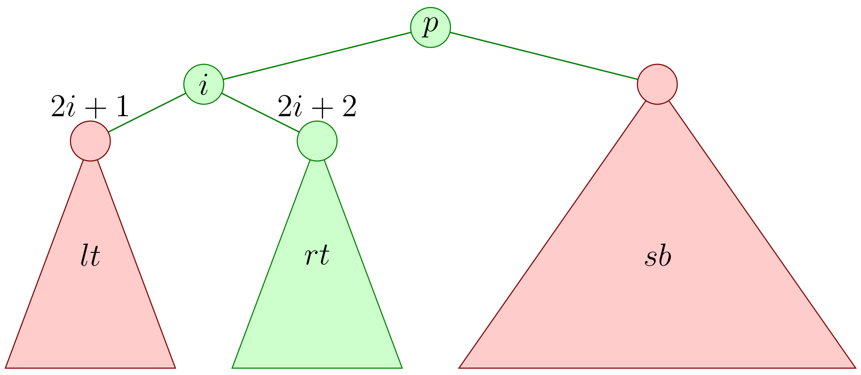


Figure 4.7: Figures demonstrating the events $R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}}$ and $R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}}$ respectively. The triangles represent subtrees rooted at the respective nodes. Green denotes that the subtree has no revoked user in it. Red denotes that the subtree has at least one revoked user in it. The sizes of the subtrees are not to the scale of the number of users in them.

To simplify the computation of these probabilities in Lemma 13, we define a new notation $\eta_r(\alpha, \beta)$ to indicate *the probability of choosing r elements from a set of α elements such that β out of these α elements are never chosen.* So, if $\beta \geq \alpha - r + 1$, then $\eta_r(\alpha, \beta) = 0$ by definition. Else, for $0 < \beta < \alpha - r + 1$,

$$\eta_r(\alpha, \beta) = \frac{\binom{\alpha-\beta}{r}}{\binom{\alpha}{r}} = \left(1 - \frac{\beta}{\alpha}\right)\left(1 - \frac{\beta}{\alpha-1}\right)\left(1 - \frac{\beta}{\alpha-2}\right)\dots\left(1 - \frac{\beta}{\alpha-r+1}\right). \tag{4.26}$$

Theorem 14. *For an internal non-root node i of $\mathcal{T}^0$ whose sibling subtree has λ_s leaves,*

$$\Pr[X^i_{n,r} = 1] = \eta_r(n, \lambda_{2i+1}) + \eta_r(n, \lambda_{2i+2}) - \eta_r(n, \lambda_s + \lambda_{2i+1}) - \eta_r(n, \lambda_s + \lambda_{2i+2})$$

$$- 2\eta_r(n, \lambda_{2i+1} + \lambda_{2i+2}) + 2\eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2}). \tag{4.27}$$

For the root node 0 *of* $\mathcal{T}^0$*,*

$$\Pr[X^0_{n,r} = 1] = \eta_r(n, \lambda_1) + \eta_r(n, \lambda_2). \tag{4.28}$$

Proof. The following two expressions can be obtained by usual probability arguments.

$$\left.\begin{array}{l} \Pr[R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}}] = \Pr[\overline{R^i_{lt}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}}] - \Pr[\overline{R^i_{rt}} \wedge \overline{R^i_{lt}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{rt}} \wedge \overline{R^i_{lt}}]; \\ \Pr[R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}}] = \Pr[\overline{R^i_{rt}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{rt}}] - \Pr[\overline{R^i_{lt}} \wedge \overline{R^i_{rt}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}]. \end{array}\right\} \tag{4.29}$$

Next, we deduce the expression for finding $\Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}]$ in terms of $\eta_r(\cdot, \cdot)$. This is the probability of choosing r elements from n such that none of the users in the subtrees $\mathcal{T}^{2i+1}$, $\mathcal{T}^{2i+2}$ or the sibling subtree $\mathcal{T}^s$ of i are chosen. Consequently, $\Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}] = \eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2})$. The other probabilities on the right hand sides of (4.29) can be found similarly by excluding the users in the respective subtrees. From Lemma 13, and substituting the probabilities on the right hand sides of (4.29) with their corresponding $\eta_r(\cdot, \cdot)$ equivalents, we get:

$$\begin{aligned} \Pr[X^i_{n,r} = 1] = {} & \eta_r(n, \lambda_{2i+1}) + \eta_r(n, \lambda_{2i+2}) - \eta_r(n, \lambda_s + \lambda_{2i+1}) - \eta_r(n, \lambda_s + \lambda_{2i+2}) \\ & - 2\eta_r(n, \lambda_{2i+1} + \lambda_{2i+2}) + 2\eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2}). \end{aligned} \tag{4.30}$$

For the root node, $\Pr[X^0_{n,r} = 1] = \Pr[\overline{R^0_{lt}}] + \Pr[\overline{R^0_{rt}}]$ where $\Pr[\overline{R^0_{lt}}] = \eta_r(n, \lambda_1)$ and $\Pr[\overline{R^0_{rt}}] = \eta_r(n, \lambda_2)$. Hence,

$$\Pr[X^0_{n,r} = 1] = \eta_r(n, \lambda_1) + \eta_r(n, \lambda_2). \tag{4.31}$$

□

The Algorithm for Computing $E[X_{n,r}]$. Now that we have the expressions to find $\Pr[X^i_{n,r} = 1]$ for all $i \in \{0, \ldots, n-2\}$ in Theorem 14, the values for λ_s, λ_{2i+1} and λ_{2i+2} for node i have to substituted appropriately in (4.30) and (4.31). By doing these substitutions for nodes at each level $\ell \in \{1, \ldots, \ell_0\}$ of $\mathcal{T}^0$, we get the complete algorithm. For level $\ell \in \{2, \ldots, \ell_0 - 1\}$, this computation is done in four steps: (1) for the node $t^{\mathcal{P}}_\ell$ of level ℓ,

ALGORITHM 1: Algorithm to compute $E[X_{n,r}]$

```
**Input**: $n, r$.
**Output**: $E[X_{n,r}]$.
$\ell_0 = \lceil \log n \rceil$;
$t_1^{\mathcal{P}} = n - 2^{\ell_0 - 1}$;
Compute $\Pr[X_{n,r}^{1,t} = 1]$ using (4.30);
$x = t_1^{\mathcal{P}} \times \Pr[X_{n,r}^{1,t} = 1]$;
**for** $\ell = 2$ *to* $\ell_0 - 1$ **do**
    $t_\ell^{\mathcal{P}} = \lceil (n - 2^{\ell_0 - 1})/2^{\ell - 1} \rceil$;
    Compute $\Pr[X_{n,r}^{\ell,t} = 1]$ for the node at $t_\ell^{\mathcal{P}}$, its sibling, all nodes on the left and
    right of $t_\ell^{\mathcal{P}}$ using (4.30);
    Add $\Pr[X_{n,r}^{\ell,t} = 1]$ for each node $(\ell, t)$ to $x$;
**end**
Compute $\Pr[X_{n,r}^{0} = 1]$ using (4.31);
$x = x + \Pr[X_{n,r}^{0} = 1]$;
$E[X_{n,r}] = x$;
```

(2) its sibling subtree, (3) all full subtrees to the left of the above two subtrees, and (4) all full subtrees to the right of the two subtrees in 1 and 2. The subtree at position $t_\ell^{\mathcal{P}}$ at level ℓ is the only possible non-full subtree for level ℓ and is of height ℓ. If $t_\ell^{\mathcal{P}}$ is odd, its sibling subtree is full and of height $\ell - 1$. If $t_\ell^{\mathcal{P}}$ is even, its sibling subtree is full and of height ℓ. The subtree at node $t_{\ell-1}^{\mathcal{P}}$ of level $\ell - 1$ is always a subtree of the tree rooted at node $t_\ell^{\mathcal{P}}$ of level ℓ. When $t_{\ell-1}^{\mathcal{P}}$ is odd, the right subtree of the tree rooted at node $t_\ell^{\mathcal{P}}$ of level ℓ is full. When $t_{\ell-1}^{\mathcal{P}}$ is even, the left subtree of the tree rooted at node $t_\ell^{\mathcal{P}}$ of level ℓ is full. For the root node 0 and the nodes at level 1, the substitutions are more simple. A pseudo-code for computing the expected header length is given as Algorithm 1.

To analyze the running time of the algorithm, we observe that each computation of $\eta_r(\alpha, \beta)$ involves $O(r)$ multiplications and there are a constant number of computations of $\eta_r(\alpha, \beta)$ for each level of the tree. Hence, the algorithm requires $O(r \log n)$ multiplications and $O(1)$ space.

Remarks.

Simulation method for estimating the expected header length: Suppose it is desired to obtain an idea of the average header length for n users of which r are revoked. One can choose m random revocation patterns. For each such pattern, the actual header generation algorithm is executed and the header size is obtained. The average header size over the m patterns provides an idea of the average header length. This method, however, is less efficient than our algorithm to compute the expected header length. For each of the m revocation patterns, the simulation will have to construct the Steiner Tree to compute the generated subsets. Each such run will require $\Omega(r^2 \log n)$ memory accesses and $O(n)$ space for finding the cover and hence the header length. In comparison, our algorithm requires $O(r \log n)$ multiplications and $O(1)$ space and finds the exact header length. Further, it is simpler to implement.

On the other hand, there is a situation where the simulation method may be useful. For the probability analysis, it is usual to assume that revocations take place uniformly. In practice, though, this may not be true. For non-uniform distributions, mathematical analysis may not be possible. For such situations, there is no other option but to use the simulation method to get an idea of the average header length. Additionally, simulations may provide more information about the probability distribution than just the average header length.

Approximation: In [PB06] a formula is given for the expected header length. However, they mentioned that their equations were "complex to compute and difficult to gain insight from". Consequently, they went forward to find *approximations* for the same. In contrast, our algorithm computes the exact value of the expected header length.

The Park-Blake approximations are quite close to the true values of the expected header lengths with the approximation factors varying over the different values of r and n. The exact algorithm that we provide is simpler to understand and implement. Also, [PB06] work only with the SD scheme and so their results do not apply when the number of users is not a power of two.

We have implemented our algorithm to compute the expected header length. Table 4.4 shows that as r goes above a certain minimum, the expected header length of the CTSD method is significantly better than the SD method. To summarize, the CTSD algorithm always gives better transmission efficiency and its cumulative improvement over many messages is significant on the bandwidth. Since replacing the SD algorithm with the CTSD scheme can be done with very little additional cost the CTSD algorithm should be the more efficient and practical choice.

Table 4.4: The expected header lengths for the SD and CTSD schemes for different n and r and the number of extra bytes needed per message of broadcast. Here we assume each session key is 128 bits long. The additional number of bytes required by the SD scheme is computed as 16 times the difference in header length of the two schemes.

r	$n < 2^{\ell_0}$ (CTSD)	CTSD $E[X_{n,r}]$	$n = 2^{\ell_0}$ (SD)	SD $E[X_{n,r}]$	Extra KBytes
10^2	$2^{19}+1$	124.49	2^{20}	124.50	0.001KB
10^2	$2^{19}+2^{18}$	124.49	2^{20}	124.50	0.001KB
10^3	$2^{19}+1$	1242.49	2^{20}	1243.80	0.021KB
10^3	$2^{19}+2^{18}$	1243.36	2^{20}	1243.80	0.007KB
5×10^3	$2^{19}+1$	6159.94	2^{20}	6192.74	0.525KB
5×10^3	$2^{19}+2^{18}$	6181.80	2^{20}	6192.74	0.175KB
10^4	$2^{19}+1$	12188.73	2^{20}	12319.86	2.098KB
10^4	$2^{19}+2^{18}$	12276.12	2^{20}	12319.86	0.700KB
10^5	$2^{19}+1$	98555.30	2^{20}	111451.58	206.340KB
10^5	$2^{19}+2^{18}$	107134.01	2^{20}	111451.58	69.081KB
10^5	$2^{23}+1$	122870.35	2^{24}	123690.49	13.122KB
10^5	$2^{23}+2^{22}$	123417.07	2^{24}	123690.49	4.375KB
10^6	$2^{23}+1$	1082115.11	2^{24}	1163305.89	1299.056KB
10^6	$2^{23}+2^{22}$	1136173.35	2^{24}	1163305.89	434.128KB

4.4.1 Asymptotic Analysis of the Expected Header Length for the SD Method

It is of interest to find the maximum possible value of the expected header length. We carry out this task for full binary trees. In this case, the CTSD method becomes the SD method.

For $n = 2^{\ell_0}$, for any internal node $i \in \{0, \dots, n-2\}$, $\lambda_{2i+1} = \lambda_{2i+2} = 2^{\ell_i - 1}$. For any node at level $\ell_i > 0$, $\lambda_s = 2^{\ell_i}$. Substituting these values for a node (ℓ, t), (4.30) becomes:

$$\Pr[X_{n,r}^{\ell,t} = 1] = 2[\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})]. \tag{4.32}$$

This probability is independent of t. In other words, the probability of generating a subset for the cover is equal for all nodes at level ℓ. Hence, we define the following:

Definition 4. $B_{n,r}^{(\ell)}$*: Let* ℓ $(1 \le \ell \le \ell_0)$ *be a level number of the tree* $\mathcal{T}^0$ *and* $n = 2^{\ell_0}$*.* $B_{n,r}^{(\ell)}$ *is defined as* $\Pr[X_{n,r}^{\ell,t} = 1]$ *of (4.32) for the node* (ℓ, t) *of* $\mathcal{T}^0$*. Hence,*

$$B_{n,r}^{(\ell)} = 2[\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})].$$

Note that by this definition, for the only node (the root node) at level ℓ_0, $B_{n,r}^{(\ell_0)} = 2\eta_r(n, 2^{\ell_0 - 1})$ which is consistent with (4.31) for $n = 2^{\ell_0}$. Hence, we define the following:

Definition 5. $H_{n,r}$*: For a given* $n = 2^{\ell_0}$ *and* r*, the expected header length* $H_{n,r}$ *due to the subset cover algorithm of the CSD scheme is defined as:*

$$H_{n,r} = E[X_{n,r}] = \sum_{\ell=1}^{\ell_0} 2^{\ell_0 - \ell} B_{n,r}^{(\ell)}.$$

Definition 6. $D_{n,r}$*: For a given* $n = 2^{\ell_0}$*, the difference between the expected header lengths for the number of revoked users being* r *and* $r-1$ *is defined as* $D_{n,r}$*. Hence,*

$$D_{n,r} = H_{n,r} - H_{n,r-1}.$$

We further observe that:

$$\begin{aligned} H_{n,r} &= H_{n,r-1} + D_{n,r} \\ &= H_{n,r-2} + D_{n,r} + D_{n,r-1} \end{aligned}$$

$$\begin{aligned}
&= H_{n,r-3} + D_{n,r} + D_{n,r-1} + D_{n,r-2} \\
&= \dots \\
&= H_1 + \sum_{i=2}^{r} D_{n,i} \\
&= 1 + \sum_{i=2}^{r} D_{n,i}. \qquad (4.33)
\end{aligned}$$

Using the definition of $B_{n,r}^{(\ell)}$ we also get:

$$\begin{aligned}
D_{n,r} &= H_{n,r} - H_{n,r-1} \\
&= \sum_{\ell=1}^{\ell_0} 2^{\ell_0-\ell} \left(B_{n,r}^{(\ell)} - B_{n,r-1}^{(\ell)} \right). \qquad (4.34)
\end{aligned}$$

In (4.34), $\eta_r(n,m) - \eta_{r-1}(n,m)$ can be rewritten as follows:

$$\begin{aligned}
\eta_r(n,m) - \eta_{r-1}(n,m) &= \frac{(n-m)_r}{(n)_r} - \frac{(n-m)_{r-1}}{(n)_{r-1}} \\
&= \frac{(n-m)(n-m-1)\dots(n-m-r+2)}{n(n-1)\dots(n-r+2)} \times \left(\frac{n-m-r+1}{n-r+1} - 1 \right) \\
&= \frac{(n-m)(n-m-1)\dots(n-m-r+2)}{n(n-1)\dots(n-r+2)} \times \frac{-m}{n-r+1} \\
&= \frac{(n-m)_{r-1}}{(n)_{r-1}} \times \frac{-m}{n-r+1} \\
&= -\eta_{r-1}(n,m) \times \frac{m}{n-r+1}. \qquad (4.35)
\end{aligned}$$

Hence from (4.34) and (4.35) we get:

$$\begin{aligned}
D_{n,r+1} &= \sum_{\ell=1}^{\ell_0} 2^{\ell_0-\ell} \left(B_{n,r+1}^{(\ell)} - B_{n,r}^{(\ell)} \right) \\
&= \frac{2n}{n-r} \sum_{\ell=1}^{\ell_0} \frac{1}{2^\ell} \Big(-2^{\ell-1}\eta_r(n, 2^{\ell-1}) + 2 \times 2^{\ell-1}\eta_r(n, 2 \times 2^{\ell-1}) \\
&\qquad + 3 \times 2^{\ell-1}\eta_r(n, 3 \times 2^{\ell-1}) - 4 \times 2^{\ell-1}\eta_r(n, 4 \times 2^{\ell-1}) \Big)
\end{aligned}$$

$$= \frac{n}{n-r}\left[-\eta_r(n,1)+\eta_r(n,2)+3\eta_r(n,3)-3\sum_{\ell=1}^{\ell_0-1}\left(\eta_r(n,2\times 2^\ell)-\eta_r(n,3\times 2^\ell)\right)\right]. \tag{4.36}$$

Here, we have made use of the fact that $\eta_r(\alpha,\beta)=0$ when $\beta \geq \alpha - r + 1$. From (4.36), we calculate the value of $H_{n,2}$ as follows:

$$\begin{aligned}
H_{n,2} &= H_{n,1} \\
&\quad + \frac{n}{n-1}\left[-\eta_1(n,1)+\eta_1(n,2)+3\eta_1(n,3)-3\sum_{\ell=1}^{\ell_0-1}\left(\eta_1(n,2\times 2^\ell)-\eta_1(n,3\times 2^\ell)\right)\right] \\
&= 1+\frac{n}{n-1}\left[\frac{3(n-3)}{n}+\frac{n-2}{n}-\frac{n-1}{n}-3\sum_{\ell=1}^{\ell_0-2}\left(\frac{n-2\times 2^\ell}{n}-\frac{n-3\times 2^\ell}{n}\right)\right] \\
&= 1+\frac{n}{n-1}\left[\frac{3(n-3)}{n}+\frac{n-2}{n}-\frac{n-1}{n}-\frac{3(n-2)}{2n}\right] \\
&= 1+\frac{n}{n-1}\left[\frac{3n-14}{2n}\right] \\
&= 1+\frac{3-\frac{14}{n}}{2(1-\frac{1}{n})}.
\end{aligned} \tag{4.37}$$

Note that $\lim_{n\to\infty} H_{n,2} = \frac{5}{2} = 1.25 \times 2$.

Now we analyze $D_{n,r+1}$ in (4.36) for $r > 2$. We use the notation $x \uparrow a$ to indicate that x increases to a and $x \downarrow a$ to indicate that x decreases to a.

Lemma 15. *$\eta_r(n,3) = \frac{(n-3)_r}{(n)_r} \uparrow 1$ as $n \uparrow \infty$.*

Proof. For any given n, $\frac{(n-3)_r}{(n)_r} < 1$.

$$\begin{aligned}
&\lim_{n\to\infty} \eta_r(n,3) \\
&= \lim_{n\to\infty} \frac{(n-3)_r}{(n)_r} \\
&= \lim_{n\to\infty} \frac{(n-3)(n-2)\ldots(n-3-r+1)}{n(n-1)\ldots(n-r+1)}
\end{aligned}$$

$$\begin{aligned}
&= \lim_{n\to\infty} \frac{(1-\frac{3}{n})(1-\frac{2}{n})\dots(1-\frac{r+2}{n})}{(1)(1-\frac{1}{n})\dots(1-\frac{r-1}{n})} \\
&= 1.
\end{aligned} \tag{4.38}$$

□

Hence, $3\eta_r(n,3) \uparrow 3$ as $n \uparrow \infty$.

Lemma 16. *$\eta_r(n,2) - \eta_r(n,1) \uparrow 0$ as $n \uparrow \infty$.*

Proof. For any given n, $\eta_r(n,2) - \eta_r(n,1) < 0$.

$$\begin{aligned}
&\lim_{n\to\infty} \eta_r(n,2) - \eta_r(n,1) \\
&= \lim_{n\to\infty} \left[\left(1-\frac{2}{n}\right)\dots\left(1-\frac{2}{n-r+1}\right) - \left(1-\frac{1}{n}\right)\dots\left(1-\frac{1}{n-r+1}\right)\right] \\
&= 0.
\end{aligned}$$

□

Hence, we claim that $(-\eta_r(n,1) + \eta_r(n,2) + 3\eta_r(n,3)) \uparrow 3$ as $n \uparrow \infty$. $\frac{n}{n-r} \uparrow 1$ as $n \uparrow \infty$. Finally, we look at $\sum_{\ell=1}^{\ell_0-2} \big(\eta_r(n, 2\times 2^\ell - \eta_r(n, 3\times 2^\ell)\big)$ to complete the analysis.

$$\begin{aligned}
\sum_{\ell=1}^{\ell_0-2} \big(\eta_r(n,2\times 2^\ell) - \eta_r(n,3\times 2^\ell)\big) &= \sum_{\ell=1}^{\ell_0-2} \left(\frac{(n-2\times 2^\ell)_r}{(n)_r} - \frac{(n-3\times 2^\ell)_r}{(n)_r}\right) \\
&\geq \frac{1}{(n)_r} \sum_{\ell=1}^{\ell_0-2} \left((n-r+1-2\times 2^\ell)^r - (n-3\times 2^\ell)^r\right) \\
&= \frac{1}{(n)_r} \sum_{\ell=1}^{\ell_0-2} \left((2^{\ell_0} - 2^{\ell+1} - r + 1)^r - (2^{\ell_0} - 3\times 2^\ell)^r\right) \\
&= \frac{1}{(n)_r} \sum_{\ell=2}^{\ell_0-1} \left((2^{\ell_0} - 2^{\ell_0-\ell+1} - r + 1)^r - (2^{\ell_0} - 3\times 2^\ell)^r\right) \\
&= \frac{1}{(n)_r} \sum_{\ell=2}^{\ell_0-1} \left(\left(\left(\frac{2^\ell-2}{2^\ell}\right)n - r + 1\right)^r - \left(\left(\frac{2^\ell-3}{2^\ell}\right)n\right)^r\right) \\
&\geq \frac{1}{n^r} \sum_{\ell=2}^{\ell_0-1} \left(\left(\left(\frac{2^\ell-2}{2^\ell}\right)n - r + 1\right)^r - \left(\left(\frac{2^\ell-3}{2^\ell}\right)n\right)^r\right)
\end{aligned}$$

$$= \sum_{\ell=2}^{\ell_0-1}\left(\Big(\frac{2^\ell-2}{2^\ell}-\frac{r-1}{n}\Big)^r-\Big(\frac{2^\ell-3}{2^\ell}\Big)^r\right). \tag{4.39}$$

We define K_r as follows:

Definition 7. *K_r:*

$$K_r = \lim_{n\to\infty}\sum_{\ell\geq 2}\left(\Big(\frac{2^\ell-2}{2^\ell}-\frac{r-1}{n}\Big)^r-\Big(\frac{2^\ell-3}{2^\ell}\Big)^r\right)$$

Hence,

$$\begin{aligned}
K_r &= \lim_{n\to\infty}\sum_{\ell\geq 2}\left(\Big(\frac{2^\ell-2}{2^\ell}-\frac{r-1}{n}\Big)^r-\Big(\frac{2^\ell-3}{2^\ell}\Big)^r\right)\\
&= \sum_{\ell\geq 2}\left(\Big(\frac{2^\ell-2}{2^\ell}\Big)^r-\Big(\frac{2^\ell-3}{2^\ell}\Big)^r\right)\\
&= \sum_{\ell\geq 2}\frac{1}{2^{r\ell}}\left(\Big(2^\ell-2\Big)^r-\Big(2^\ell-3\Big)^r\right)\\
&= \sum_{t=1}^{r}(-1)^t\binom{r}{t}(2^t-3^t)\sum_{\ell\geq 2}\frac{2^{\ell(r-t)}}{2^{r\ell}}\\
&= \sum_{t=1}^{r}(-1)^t\binom{r}{t}(2^t-3^t)\sum_{\ell\geq 2}\frac{1}{2^{\ell t}}.
\end{aligned} \tag{4.40}$$

Since $\sum_{\ell\geq 2}\frac{1}{2^{\ell t}} = \frac{1}{2^t(2^t-1)}$, we get

$$\begin{aligned}
K_r &= \sum_{t=1}^{r}(-1)^t\binom{r}{t}\frac{(2^t-3^t)}{2^t(2^t-1)}\\
&= \sum_{t=1}^{r}(-1)^t\binom{r}{t}\frac{(2^t-3^t)}{(2^t-1)}-\sum_{t=1}^{r}(-1)^t\binom{r}{t}\frac{(2^t-3^t)}{(2^t)}\\
&= \Big(-\frac{1}{2}\Big)^r+\sum_{t=1}^{r}(-1)^t\binom{r}{t}\frac{(2^t-3^t)}{(2^t-1)}.
\end{aligned} \tag{4.41}$$

We also define:

Table 4.5: Ratio $\frac{H_r}{r}$ for different values of r.

r	D_r	$\frac{H_r}{r}$
2	$\frac{3}{2}$	1.25
3	$\frac{5}{4}$	1.25
4	$\frac{69}{56}$	1.24553571
5	$\frac{417}{336}$	1.24464285
6	$\frac{25953}{20832}$	1.24483967

Definition 8. *H_r and D_r:*

$$H_r = \lim_{n\to\infty} H_{n,r}$$

$$D_r = \lim_{n\to\infty} D_{n,r}$$

The next result summarizes the above analysis.

Theorem 17. *For all $n \geq 1$, $r \geq 1$, the expected header length $H_{n,r} \uparrow H_r$, as n increases through powers of two, where*

$$H_r = 3r - 2 - 3 \times \sum_{i=1}^{r-1}\left(\left(-\frac{1}{2}\right)^i + \sum_{t=1}^{i}(-1)^t\binom{i}{t}\frac{(2^t-3^t)}{(2^t-1)}\right).$$

Proof. From (4.33), we get $H_r = 1 + \sum_{i=2}^{r} D_i$. Further, from (4.36), (4.39) and (4.41), we get $D_{r+1} = 3 - 3K_r$ where K_r is given by (4.41). □

Table 4.5 lists the values of D_r and $\frac{H_r}{r}$ for small values of r. This table shows the ratio $\frac{H_r}{r}$ is always less than $1.25r$.

In [NNL01, NNL02], a sketchy argument was given to show that $H_{n,r}$ is bounded above by $1.38r$. It was mentioned that simulation results showed a tighter upper bound of $1.25r$. Values computed using Theorem 17 explain this observation. On the other hand, Theorem 17 shows that the actual limiting value for the expected header length is much more complicated than the simple $1.25r$ that was suggested in [NNL01, NNL02]. Our experiments have shown that the convergence to this limiting value is quite fast. Further, the bound given by Theorem 17 can be computed in $O(r)$ time and $O(1)$ space.

Table 4.6: The expected header lengths for $n = 200$ and $n = 256$ for different r and the number of extra bytes needed per message of broadcast (assuming each session key is 128-bit long).

r	$n = 200$	$n = 256$	Extra Bytes
10	12	12	0
20	23	23	0
30	32	33	16
40	40	42	32
50	46	50	64

Table 4.7: The expected header lengths for $n = 1500$ and $n = 2048$ for different r and the number of extra bytes needed per message of broadcast (assuming each session key is 128-bit long).

r	$n = 1500$	$n = 2048$	Extra Bytes
50	61	61	0
100	116	118	32
150	167	172	80
200	213	223	160
250	255	270	240
300	293	314	336

4.4.2 Other Experimental Results

We return to the issue of comparing the CSD method to that of the SD method with dummy users. The situation where the dummy users form a block has been discussed in details in Section 4.2.2. Let us consider the situation where the dummy users are randomly distributed. If these are all considered to be revoked, then there is a large penalty on the transmission overhead. This is because the expected header length is linear in the number of revoked users. So, suppose that the randomly distributed dummy users are viewed as being privileged by the cover generation algorithm.

Running the algorithm to compute the expected header length for different values of n and r we compare the transmission efficiency of the CSD method with the SD method with dummy users. Additionally, we report other observations on the expected header length of

Table 4.8: The expected header lengths for $n = 10000$ and $n = 16384$ for different r and the number of extra bytes needed per message of broadcast (assuming each session key is 128-bit long).

r	$n = 10000$	$n = 16384$	Extra Bytes
500	589	602	208
1000	1109	1162	848
1500	1561	1680	1904
2000	1947	2157	3360
2500	2267	2593	5216
3000	2521	2988	7472

Table 4.9: $\frac{E[X_{n,r}]}{r}$ for $r = 2$, $16 \leq n < 32$ for the CSD scheme.

n	16	17	18	19	20	21	22	23
$\frac{E[X_{n,r}]}{2}$	1.167	1.169	1.180	1.184	1.195	1.200	1.210	1.215
n	24	25	26	27	28	29	30	31
$\frac{E[X_{n,r}]}{2}$	1.225	1.217	1.214	1.209	1.208	1.207	1.208	1.207

the CSD method.

1. For a fixed $n < 2^{\ell_0}$, as r goes above a certain minimum, the expected header length of the CSD method is significantly shorter than the corresponding instantiation of the SD method. As an example, for $n = 10000$, the expected header length is 1561 for $r = 1500$ while for the corresponding $n = 16384$ of the SD method, the expected header length is 1680 for the same r. Assuming the function F_K used for encrypting each block of digital data is AES-128, this difference of 119 in the expected header length causes an extra bandwidth consumption of 1904 $(= 119 \times 16)$ bytes per message on an average. Tables 4.6, 4.7 and 4.8 list the expected header lengths for $n = 200$, 1500 and 10000 and the corresponding next powers of two for different values of r.

2. For $n = 200$, by running the algorithm for computing the expected header length, we observe that the expected header lengths are better compared to $n = 256$ for all $r > 5$. Thus, CSD is more efficient in terms of the transmission overhead efficiency for all $r > 5$ for $n = 200$. Similarly, CSD gains over SD when $n = 1500$ for all $r > 7$ and when $n = 10000$, it gains for all $r > 28$. For real-time scenarios like Pay-TV, $n = 10000$ and

$r > 28$ are practical numbers. Thus, the CSD method will provide better transmission efficiency than SD for many practical purposes.

3. For full binary trees, we know from (4.37) that for $r = 2$, the limiting value of $\frac{E[X_{n,r}]}{2}$ is 1.25. By running our algorithm, we also observe that for n a power of two, the expected header length increases with increasing n for all $r \geq 2$.

4. For $r = 2$, as we keep increasing n from 2^ℓ to $2^{\ell+1} - 1$, the ratio $\frac{E[X_{n,r}]}{r}$ increases almost uniformly to reach a local maximum at $n = 2^\ell + 2^{\ell-1}$ and then decreases. The data in Table 4.9 demonstrates this behavior for $16 \leq n < 32$. For $32 \leq n < 64$, the maximum value of $\frac{E[X_{n,r}]}{r}$ is 1.225 observed at $n = 24$ and for $128 \leq n < 256$, the maximum value is 1.271 and is observed at $n = 192$. However, as r increases, the behavior of the above ratio changes, with local glitches disrupting the uniformity at most places.

4.5 Conclusion

In this chapter, we have proposed a new BE scheme which extends the tree-based NNL-SD scheme of Chapter 2 [NNL01, NNL02]. The new Complete Tree Subset Difference method is capable of accommodating any arbitrary number of users that may not be a power of two and hence subsumes the NNL-SD scheme of Chapter 2 [NNL01, NNL02]. Almost all results of the CTSD scheme that we subsequently prove are also new for the SD scheme.

Detailed combinatorial analysis of the CTSD scheme is done by finding two recurrences to count the number of ways r out of n users can be revoked to result in a subset cover size of h in the CTSD method. Using these recurrences, it is proved that the maximum possible header length for a given r is $2r - 1$. This is no worse than the SD scheme even though an arbitrary number of users are accommodated. The maximum header length for all r is $\left\lfloor \frac{n}{2} \right\rfloor$. The recurrences are the most efficient tool as per our knowledge to generate exhaustive data for the above count. Using the recurrences, we also find and prove the expression for the minimum number of users required to be in a system so that for a given r, the maximum cover size would reach $2r - 1$. For n a power of two, a generating function is found for generating the same sequence as the recurrences.

Probabilistic analysis of the revocation patterns in the CTSD scheme gives the most important result of this work: an efficient algorithm to compute the expected header length for a given n and r. Using this algorithm, it is shown that for practical values of n and r, the CTSD scheme provides better transmission efficiency as compared to the SD scheme. An asymptotic analysis is done using this algorithm that not only gives theoretical support to the empirical upper bound of $1.25r$ mentioned in [NNL01, NNL02], but also gives an expression to compute the maximum possible expected header length for a given r in the SD algorithm in $O(r)$ time.

Chapter 5

The (Layered) Complete Tree Subset Difference Scheme and its Analysis

5.1 Introduction

In Chapter 1, we gave a brief description of our contributions in this chapter. We recollect them very briefly here.

In this chapter, we work with the idea of layering described in Section 2.1.2 [HS02]. The Halevy-Shamir (HS) layering works for $n = 2^{\ell_0}$ users where ℓ_0 is a perfect square. This limits its usage to very specific number of users $(2^4, 2^9, 2^{16}, 2^{25})$. Two natural extensions of the HS layering strategy are provided. These extensions work for values of ℓ_0 that may not be a perfect square (and hence subsume the HS layering strategy).

We introduce the notion of storage minimal layering. For such a strategy, the user storage requirement is the minimum possible that can be obtained from 2-way splitting of SD subsets using layerings. An $O(\ell_0^3)$ time and $O(\ell_0^2)$ space dynamic programming algorithm is presented to compute storage minimal layerings. It is shown that making the root level non-special significantly improves the user storage while the effect on the average header length is negligible. We also propose the constrained minimization layering strategy where the user storage is reduced without affecting the header length for most practical values of r.

We describe an algorithm to compute the expected header length of the layering based SD schemes. This algorithm works for all possible values of the number of users (and not only those values which are powers of two). Assuming that r out of n users are revoked uniformly at random, our algorithm computes the expected header length in $O(r \log^2 n)$ time and $O(\log n)$ space.

The contents of this chapter were published in [BS14a].

5.2 General Layering Strategy

In general, a layering strategy $\boldsymbol{\ell}$ is denoted by the numbers of the special levels $\ell_0 > \ell_1 > ... > \ell_{e-1} > \ell_e = 0$. Let $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$. The layering strategy has $(e+1)$ special levels. It is sometimes more convenient to use another formulation to denote the layering. For $1 \leq i \leq e$, define $d_i = \ell_{i-1} - \ell_i$ so that d_i's are positive integers whose sum is ℓ_0. Conversely, given any sequence of positive integers $\mathbf{d} = (d_1, \ldots, d_e)$ whose sum is ℓ_0, it is possible to define a layering scheme where $\ell_i = \ell_0 - \sum_{j=1}^{i} d_j$.

The user storage for any such layering strategy $\boldsymbol{\ell}$ in general can be calculated as follows. Corresponding to each special level ℓ_i, a user has to store ℓ_i labels. Now consider the nodes in the layer bordered by ℓ_i and ℓ_{i+1}. Corresponding to any non-special level j in this layer a user has to store $j - \ell_{i+1}$ labels. So, the total number of labels that is required to be stored by a user considering both special and non-special levels is given by the following formula.

$$\begin{aligned}
\mathsf{storage}_0(\boldsymbol{\ell}) &= \sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=\ell_{i+1}+1}^{\ell_i - 1} (j - \ell_{i+1}) \\
&= \sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=1}^{\ell_i - \ell_{i+1} - 1} j \\
&= \sum_{i=0}^{e-1} \ell_i + \frac{1}{2} \sum_{i=0}^{e-1} (\ell_i - \ell_{i+1})(\ell_i - \ell_{i+1} - 1). \qquad (5.1)
\end{aligned}$$

A recursive description can be obtained as follows.

$$\begin{aligned}
\mathsf{storage}_0(\ell_0, \ell_1, \ldots, \ell_e) &= \ell_0 + \ell_1 + \cdots + \ell_e + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} \\
&\quad + \frac{(\ell_1 - \ell_2)(\ell_1 - \ell_2 - 1)}{2} \\
&\quad + \cdots + \frac{(\ell_{e-1} - \ell_e)(\ell_{e-1} - \ell_e - 1)}{2} \\
&= \ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \mathsf{storage}_0(\ell_1, \ldots, \ell_e). \qquad (5.2)
\end{aligned}$$

Equation (5.1) can be formulated in terms of the layer lengths $\mathbf{d} = (d_1, \ldots, d_e)$ as follows.

$$\mathsf{storage}_0(\boldsymbol{\ell}) \;=\; \ell_0(e+1) - \sum_{i=1}^{e}(e-i+1)d_i + \frac{1}{2}\sum_{i=1}^{e} d_i(d_i-1). \tag{5.3}$$

If all the d_i's are equal to d and $\ell_0 = e \times d$, then $\mathsf{storage}_0(\boldsymbol{\ell})$ is given by $\ell_0(e+d)/2$. This shows that the user storage using e layers of length d each is the same as the user storage using d layers of length e each. If all the layer lengths are equal, then the problem of minimizing the user storage is that of minimizing the sum $e+d$ subject to the constraint $ed = \ell_0$. From this it is easy to see that the minimum value is attained for $e = d = \sqrt{\ell_0}$ and the corresponding value of user storage is $\ell_0^{3/2}$. This justifies the choice made in [HS02] that was described in Section 2.1.2. Note that the minimization here is in the context of all the layer lengths being equal.

We look at some further combinatorial results on general layering strategies. It is easy to note that the layering strategy with each $d_i = 1$ or with $e = 1$ results in the SD scheme. In the following lemma, we look at two specific kinds of layerings that result in the same storage requirement.

Lemma 18. *Let $\ell_0 = d(e-1)+p$ with $1 \le p \le d$ and consider the layering strategies $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ whose layer lengths are respectively given by $(\underbrace{d,\ldots,d}_{e-1},p)$ and $(\underbrace{d,\ldots,d}_{e-d+p},\underbrace{d-1,\ldots,d-1}_{d-p})$. Then $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}')$.*

Proof. From (5.1)

$$\begin{aligned}
&\mathsf{storage}_0(\boldsymbol{\ell}) - \mathsf{storage}_0(\boldsymbol{\ell}') \\
&= (d-p) - \frac{(d-p)(d-p+1)}{2} \\
&\quad -\frac{d(d-1)}{2} + \frac{p(p-1)}{2} + (d-1)(d-p) \\
&= -\frac{(d-p)^2-(d-p)}{2} + \frac{(d-p)^2-(d-p)}{2} \\
&= 0.
\end{aligned}$$

□

We provide below some simple facts about storage.

1. Let $\mathbf{d} = (d_1, \ldots, d_e)$ and suppose that $d_i = d + \delta$ and $d_{e-j+1} = d$, i.e., the i-th layer length from the top is $d + \delta$ and the j-th layer length from the bottom is d. Suppose that $\mathbf{d}'$ is obtained from $\mathbf{d}$ by incrementing d_i (i.e., changing its value to $d + \delta + 1$) and decrementing d_{e-j+1} (i.e., changing its value to $d - 1$). Let $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ be the corresponding sequences of special levels. A simple calculation based on (5.3) shows that $\mathsf{storage}_0(\boldsymbol{\ell}) - \mathsf{storage}_0(\boldsymbol{\ell}') = (e - i - j - \delta)$. So, if $e > i + j + \delta$, then it is possible to reduce storage by incrementing d_i and decrementing d_{e-j+1}. This simple observation can be used to show that the storage requirement of a layering scheme with unequal layer lengths can be reduced below a layering scheme with equal layer lengths.

   Let ℓ_0 be a positive integer and assume that d divides ℓ_0 such that $\ell_0 = d \times e$. Consider the layering scheme with layer lengths $\mathbf{d} = (d, d, \ldots, d)$. Let $\theta \geq 1$ be such that $e > 2\theta$ and define
   $$\mathbf{d}' = (\underbrace{d+1, \ldots, d+1}_{\theta}, d, \ldots, d, \underbrace{d-1, \ldots, d-1}_{\theta}).$$
   Then $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}') + \theta(e - \theta - 1)$. The gap $\theta(e - \theta - 1)$ is positive.

2. Having a single layer of length d_e at the bottom of the tree is the same as having $d_e + 1$ layers of length 1 each at the bottom. A simple calculation based on (5.3) shows this.

3. Suppose $\mathbf{d} = (d_1, \ldots, d_e)$ with $d_1 \geq d_2 \geq \cdots \geq d_e$ and $\mathbf{d}' = (d_{\pi(1)}, \ldots, d_{\pi(e)})$ where π is a permutation of $\{1, \ldots, e\}$. Let $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ be the corresponding sequences of special levels. Then $\mathsf{storage}_0(\boldsymbol{\ell}) \leq \mathsf{storage}_0(\boldsymbol{\ell}')$. The quantity $\ell_0(e+1)$ and the quadratic terms in (5.3) are the same in both cases. A simple argument then shows the required inequality. As an example, suppose $\ell_0 = 12$ and fix $e = 8$. Then the scheme having $(d_1, d_2, \ldots, d_8) = (2, 2, 2, 2, 1, 1, 1, 1)$ requires a storage of 50 labels whereas the scheme having $(d_1, d_2, \ldots, d_8) = (1, 1, 1, 1, 2, 2, 2, 2)$ requires a storage of 66 labels.

5.2.1 The HS Layering with Residual Bottom Layer

Let ℓ_0 be any positive integer and $d \leq \ell_0$. We write $\ell_0 = d(e-1) + p$ where $1 \leq p \leq d$. Then the special levels are

$$\ell_0,\ \ell_0 - d,\ \ell_0 - 2d,\ \ldots,\ \ell - d(e-1),\ 0.$$

So, the tree will have a total of $e+1$ special levels (including the root level ℓ_0 and the leaf level 0) and e layers out of which $e-1$ layers are of length d each and the last layer is of length p. Note that the length p of the bottom-most layer can equal d which will lead to e layers each of length d. We find it convenient to always have level 0 (leaf level) as a special level as this does not have any effect on either the user storage or the header length. The Halevy-Shamir (HS) layering strategy is a special case where ℓ_0 is a perfect square with $d = \sqrt{\ell_0}$ and layer lengths $d, d, \ldots, d, p = d$.

5.2.2 The e-HS Layering Strategy

We now consider a layering strategy where the layer lengths are balanced. Write $\ell_0 = d(e-1)+p = (e-d+p)d + (d-p)(d-1)$ and define $\mathbf{d}' = (\underbrace{d, \ldots, d}_{e-d+p}, \underbrace{d-1, \ldots, d-1}_{d-p})$. Let $\boldsymbol{\ell}$ be the layering strategy with a residual bottom layer and $\boldsymbol{\ell}'$ be the balanced layering strategy. In Lemma 18, we have shown that $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}')$. So there is no difference between these two strategies in terms of user storage. Experimental results show that the average header lengths for both strategies are similar with that corresponding to the balanced strategy being slightly smaller. As an example, for $\ell_0 = 18$, $\mathbf{d}' = (5,5,4,4)$ yields less expected header lengths than $\mathbf{d} = (5,5,5,3)$ for all r between 256 and 16384 while the user storage 75 is the same for both. We call the balanced strategy to be the *extended HS* or *e-HS* layering strategy. This strategy coincides with the layering scheme given in Section 2.1.2 [HS02] for $n = 28$.

Using (5.3), it can be verified that storage requirement is $O(\log^{3/2} n)$ for both the e-HS and the residual bottom layer strategies.

5.2.3 Root at a Non-Special Level

In the HS layering described in Section 2.1.2 [HS02] as well as its extensions given in Section 5.2.1 and Section 5.2.2 above, the root level ℓ_0 is always taken as a special level. It is possible to obtain further reduction in user storage if we allow the root level to be a non-special level. Having the root as a special level contributes ℓ_0 labels to the user storage. If instead the root level is made non-special, then its contribution to the user storage will be $\ell_0 - \ell_1$ labels. Given a sequence of level numbers $\boldsymbol{\ell}$, let $\mathsf{storage}_1(\boldsymbol{\ell})$ be the number of labels required to be stored when the root (top-most) level is not special (and so, ℓ_1 is the first special level). Then the following relation holds.

$$\mathsf{storage}_1(\boldsymbol{\ell}) \quad = \quad \mathsf{storage}_0(\boldsymbol{\ell}) - \ell_1. \tag{5.4}$$

Combining this with (5.2) we get the following relation.

$$\mathsf{storage}_1(\ell_0, \ldots, \ell_e) \quad = \quad \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} + \mathsf{storage}_0(\ell_1, \ldots, \ell_e). \tag{5.5}$$

So, not having the root at a special level reduces the storage requirement by ℓ_1 labels. This can be quite significant as can be seen later from Table 5.3. Consider the e-HS layering strategy where $\ell_0 = d \times e$ and so $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_e)$ where $\ell_i - \ell_{i+1} = d$ for $0 \leq i < e$. In this case, $\mathsf{storage}_0(\boldsymbol{\ell}) = \ell_0^{3/2}$ and $\mathsf{storage}_1(\boldsymbol{\ell}) = \ell_0^{3/2} - (\ell_0 - \ell_0^{1/2})$.

It is important to understand the effect on the header length when the root level is not special. During the computation of the cover, suppose that the root generates an SD subset, i.e., the SD cover finding algorithm returns a subset of the form $S_{0,j}$. Since the root is not at a special level, this subset may be split into two if j is not in the first layer. We argue that for reasonable values of r (the number of revoked users), this effect is negligible. In fact, the argument is that the probability of the root generating an SD subset itself is small.

The root generates an SD subset only if exactly one of the two subtrees of the root node contains all the revoked users. Intuitively this probability is low even for moderate values of r. We provide some more justification. Suppose the revoked users are uniformly distributed, i.e., r users are uniformly sampled one-by-one without replacement and revoked. Then the probability that the left subtree does not have any revoked user (and consequently the right subtree contains all of them) is

$$\begin{aligned} &\left(1 - \frac{n/2}{n}\right)\left(1 - \frac{n/2}{n-1}\right) \cdots \left(1 - \frac{n/2}{n-r+1}\right) \\ = \; &\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{2\left(1 - \frac{1}{n}\right)}\right) \cdots \left(1 - \frac{1}{2\left(1 - \frac{r-1}{n}\right)}\right) \end{aligned}$$

The probability that the right subtree does not have any revoked user is also equal to this value. So, the total probability that the header generates a subset is twice this value. For practical applications of BE, the number n of users will usually be much larger than the number of revoked users r and so the ratio r/n will be small. Then the above expression can be approximated by 2^{-r}. This is negligible even for values of r as small as 20 or so. Consequently, for practical situations, there will be almost no effect on the header length if the root level is not made special.

5.2.4 Storage Minimal Layering

For a given value of ℓ_0, let $\mathrm{SML}_0(\ell_0)$ denote a layering strategy $\boldsymbol{\ell}$ (or equivalently is given by the sequence of differences $\mathbf{d}$), such that $\mathsf{storage}_0(\boldsymbol{\ell})$ takes the minimum value among all possible layering strategies for a tree with ℓ_0 levels and having the root as a special level. Let $\#\mathrm{SML}_0(\ell_0)$ denote $\mathsf{storage}_0(\boldsymbol{\ell})$ where $\boldsymbol{\ell}$ is a storage minimal layering strategy. Similarly define $\mathrm{SML}_1(\ell_0)$ and $\#\mathrm{SML}_1(\ell_0)$ that exclude the root level from being special.

We describe a dynamic programming based algorithm to compute $\mathrm{SML}_0(\ell_0)$ (and subsequently $\mathrm{SML}_1(\ell_0)$). The idea of the algorithm is explained as follows. For a fixed value of ℓ_0, the number e of layers can vary from 1 to ℓ_0. The cases $e = 1$ and $e = \ell_0$ correspond to the SD scheme and in these two cases the user storage is known to be equal to $\ell_0(\ell_0 + 1)/2$. Let $\mathrm{SML}_0(e, \ell_0)$ denote a storage minimal layering *using exactly e layers*. Clearly, the following relation holds.

$$\#\mathrm{SML}_0(\ell_0) = \min_{1 \le e \le \ell_0} \#\mathrm{SML}_0(e, \ell_0). \tag{5.6}$$

Also,

$$\#\mathrm{SML}_0(e, \ell_0) = \min_{(\ell_0, \ldots, \ell_e)} \mathsf{storage}_0(\ell_0, \ell_1, \ldots, \ell_e), \tag{5.7}$$

where the minimum is over all possible layering strategies $(\ell_0, \ell_1, \ldots, \ell_e)$. Using (5.2)

$$\#\mathrm{SML}_0(e, \ell_0) = \min_{1 \le \ell_1 < \ell_0} \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \#\mathrm{SML}_0(e - 1, \ell_1) \right). \tag{5.8}$$

This relation is the basis for the algorithm. Let Tab be an $\ell_0 \times \ell_0$ table such that $\mathsf{Tab}[e][\ell_0] = \#\mathrm{SML}_0(e, \ell_0)$. A simple $O(\ell_0^3)$ time dynamic programming algorithm can fill up this table as given in Algorithm 2.

ALGORITHM 2: Dynamic Programming Algorithm to find Tab

```
Input: $\ell_0$.
Output: An $\ell_0 \times \ell_0$ table Tab where $\mathsf{Tab}[e][\ell]$ contains the value of $\#\mathrm{SML}_0(e, \ell)$.
for $\ell = 1$ to $\ell_0$ do
    $\mathsf{Tab}[1][\ell] = \mathsf{Tab}[\ell][\ell] = \ell(\ell+1)/2$;
end
for $\ell = 2$ to $\ell_0$ do
    for $e = 2$ to $\ell - 1$ do
        $\mathsf{Tab}[e][\ell] = \min_{1 \le \ell_1 < \ell} \left( \ell + \frac{(\ell - \ell_1)(\ell - \ell_1 - 1)}{2} + \mathsf{Tab}[e-1][\ell_1] \right)$
    end
end
```

Using (5.6) provides $\#\mathrm{SML}_0(\ell_0)$ as the minimum value in column number ℓ_0 of Tab. Note that the minimum may occur for more than one possible value of e. These values of ℓ_1 are reported during the computation. Let $\Lambda(e, \ell_0)$ be the list of all possible values of ℓ_1 for which (5.8) holds. The above method can be extended to generate all possible layering strategies for which user storage is minimized.

An SML_0 layering strategy $\boldsymbol{\ell}$ can be generated as follows. Start with $\boldsymbol{\ell}$ as the list containing only ℓ_0 and keep on appending in the following manner to obtain the complete sequence. Let e be one of the possibilities for which $\mathsf{Tab}[e][\ell_0]$ takes the minimum value; choose ℓ_1 as any one value from $\Lambda(e, \ell_0)$ and append to $\boldsymbol{\ell}$; choose ℓ_2 as any one value from $\Lambda(e-1, \ell_1)$ and append to $\boldsymbol{\ell}$; continue until 0 is appended to the list. All SML_0 strategies can be generated by looping over all possible values of e, all possible values of ℓ_1, all possible values of ℓ_2 and so on.

Once Tab is prepared, computing $\#\mathrm{SML}_1(\ell_0)$ using (5.5) is easy.

$$\begin{aligned} \#\mathrm{SML}_1(\ell_0) &= \min_e \min_{\ell_1} \left(\#\mathrm{SML}_0(e-1, \ell_1) + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} \right) \\ &= \min_e \min_{\ell_1} \left(\mathsf{Tab}[e-1][\ell_1] + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} \right). \end{aligned} \tag{5.9}$$

The first minimization is over the number of layers and the second minimization is over the value of the first special level. The possible corresponding layering strategies can also be easily recovered. It is to be noted that the $\mathrm{SML}_1(\ell_0)$ layerings are due to the minimization of the user storage by assuming the root to be at a non-special level. It can be seen from

Table 5.1: The number of $\mathrm{SML}_0(\ell_0)$ and $\mathrm{SML}_1(\ell_0)$ layering strategies for various values of ℓ_0.

ℓ_0	no. of $\mathrm{SML}_0(\ell_0)$ layerings	no. of $\mathrm{SML}_1(\ell_0)$ layerings
12	10	10
16	6	15
20	6	1
24	35	35
25	35	21
28	1	8

Table 5.2: List of $\mathrm{SML}_0(\ell_0)$ and $\mathrm{SML}_1(\ell_0)$ layering strategies denoted by the special levels for $\ell_0 = 12$.

10 Special levels for $\mathrm{SML}_0(12)$	10 Special levels for $\mathrm{SML}_1(12)$
12,7,4,2,1,0	8,4,2,1,0
12,8,4,2,1,0	8,5,2,1,0
12,8,5,2,1,0	8,5,3,1,0
12,8,5,3,1,0	9,5,2,1,0
12,7,3,1,0	9,5,3,1,0
12,7,4,1,0	9,6,3,1,0
12,7,4,2,0	8,4,1,0
12,8,4,1,0	8,4,2,0
12,8,4,2,0	8,5,2,0
12,8,5,2,0	9,5,2,0

(5.8) and (5.9) that in an $\mathrm{SML}_0(\ell_0)$ layering, if the root is made non-special, it might not necessarily result in an $\mathrm{SML}_1(\ell_0)$ layering and vice versa.

Table 5.3 shows values of user storage for SML strategies for some ℓ_0. For comparison, we also show the storage requirements for the SD scheme and the e-HS layering strategy. Compared to the SD scheme, the e-HS layering strategy reduces the storage requirement very significantly (both asymptotically as well as in practical numbers). Compared to the e-HS scheme the value of $\#\mathrm{SML}_0(\ell_0)$ is slightly smaller and the value of $\#\mathrm{SML}_1(\ell_0)$ is about 18% to 24% lower for the newly suggested values of $\boldsymbol{\ell}$. So, given a value of ℓ_0, if the requirement is to minimize the user storage, then the SML strategies offer better alternatives. They also guarantee that using 2-way splitting of SD subsets with layering, further lowering of storage cannot be achieved.

The effect of $\mathrm{SML}_0(\ell_0)$ and $\mathrm{SML}_1(\ell_0)$ strategies on the average header length is also shown in Table 5.3. For computing the average header lengths, we have considered ten values of r equally spaced between $r_{\min}$ and $r_{\max}$. The reported values are the average header lengths of the different schemes normalized by the average header length of the SD scheme. As an example, the first value 1.69 corresponding to the row for e-HS and $\ell_0 = 28$ means that with $n = 2^{28}$ users out of which $r = 2^{10}$ are uniformly revoked, the average header length of the e-HS layering strategy is 1.69 times that of the SD scheme.

One may note the following points.

1. For a fixed ℓ_0, there may be more than one $\text{SML}_0(\ell_0)$ (resp. $\text{SML}_1(\ell_0)$) strategy which achieves storage of $\#\text{SML}_0(\ell_0)$ (resp. $\#\text{SML}_1(\ell_0)$). Table 5.1 gives the number of SML strategies for several values of ℓ_0. For $\ell_0 = 12$, Table 5.2 lists all possible $\text{SML}_0(\ell_0)$ and $\text{SML}_1(\ell_0)$ strategies for $\ell_0 = 12$. There, however, need not be a single layering strategy which minimizes expected header length for all possible values of r. Out of these, one would be interested in the layering that would give the minimum expected header length for most values of r under consideration. The SML strategies reported in Table 5.3 have this feature.

2. For $\ell_0 = 32$, Tab has been computed and reported in Table 5.4. It gives the values of the minimum storage for every $1 \leq \ell_0 \leq 32$ and $1 \leq e \leq \ell_0$. For a particular ℓ_0 and e, it also gives the values of ℓ_1 for which (5.8) holds. As an example, we see that for $\ell_0 = 32$ and $e = 8$, $\#\text{SML}_0(e, \ell_0) = 172$ and the values of ℓ_1 are 24 and 25. All possible $\text{SML}_0(\ell_0)$ strategies for $1 \leq \ell_0 \leq 32$ can be obtained from this table and the $\text{SML}_1(\ell_0)$ strategies can subsequently be found using (5.9).

3. As discussed earlier, if the root level is made non-special in an SML_0 strategy, it may not lead to an SML_1 strategy and vice versa. Table 5.2 shows that while the SML_0 strategy $\boldsymbol{\ell} = (12, 8, 4, 2, 1, 0)$ gives rise to an SML_1 strategy $\boldsymbol{\ell} = (8, 4, 2, 1, 0)$ by making the root level non-special, the SML_0 strategy $\boldsymbol{\ell} = (12, 7, 4, 2, 1, 0)$ does not. On the other hand, the SML_1 strategy $\boldsymbol{\ell} = (9, 5, 2, 1, 0)$ is not generated from an SML_0 strategy.

4. Extensive experimentation have shown that for practical values of r, there is no significant difference between the average header lengths of SML_0 and SML_1 strategies that differ at only the root being at a special level or not. For $\ell_0 = 12$ and 16, the reported SML_0 strategy with the root level made non-special turns out to be an SML_1 strategy (as reported in Table 5.3) with minimum expected header lengths. This supports the theoretical justification described before. However, for $\ell_0 = 20$, it turns out that making the root level of the SML_0 strategy non-special does not give rise to an SML_1

Table 5.3: Comparison of user storage and expected header lengths between e-HS LSD and SML. The tuples contain header lengths normalized with the SD header lengths corresponding to the values of r in $(r_{\min}, \ldots, r_{\max})$ respectively.

ℓ_0	$r_{\min}$	$r_{\max}$	scheme	special levels	storage	normalized header lengths for $(r_{\min}, \ldots, r_{\max})$
12	2^2	2^6	SD	12, 0	78	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			e-HS	12, 8, 4, 0	**42**	(**1.69, 1.59, 1.56, 1.56, 1.57, 1.57, 1.57, 1.56, 1.55**, 1.53, 1.52)
			SML_0	12, 8, 5, 3, 1, 0	40	(**1.68, 1.57, 1.54, 1.54, 1.54, 1.55, 1.55, 1.54, 1.54**, 1.53, 1.52)
			SML_1	8, 5, 3, 1, 0	**32**	(**1.68, 1.57, 1.54, 1.54, 1.54, 1.55, 1.55, 1.54, 1.54**, 1.53, 1.52)
16	2^3	2^8	SD	16, 0	136	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	16, 12, 8, 4, 0	**64**	(1.63, **1.65, 1.66, 1.64, 1.62, 1.60, 1.58, 1.57, 1.57, 1.56**)
			SML_0	16, 11, 7, 4, 2, 1, 0	61	(1.69, **1.60, 1.63**, 1.65, 1.65, 1.64, 1.63, 1.62, 1.60, 1.59)
			SML_1	12, 8, 5, 3, 1, 0	**50**	(1.63, **1.64, 1.65, 1.63, 1.60, 1.58, 1.57, 1.56, 1.55, 1.54**)
20	2^4	2^{10}	SD	20, 0	210	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			e-HS	20, 15, 10, 5, 0	**90**	(1.64, 1.72, 1.69, 1.66, **1.64, 1.62, 1.61, 1.61, 1.60, 1.60**)
			SML_0	20, 15, 10, 6, 3, 1, 0	85	(1.64, 1.72, 1.69, 1.66, **1.63, 1.62, 1.61, 1.60, 1.60, 1.60**)
			SML_1	15, 10, 6, 3, 1, 0	**70**	((1.64, 1.72, 1.69, 1.66, **1.63, 1.62, 1.61, 1.60, 1.60, 1.60**)
24	2^5	2^{12}	SD	24, 0	300	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			e-HS	24, 19, 14, 9, 4, 0	**116**	(1.62, 1.64, 1.62, 1.64, **1.67, 1.69, 1.71, 1.71, 1.72, 1.72**)
			SML_0	24, 18, 12, 7, 3, 1, 0	112	(1.65, 1.74, 1.70, 1.67, **1.65, 1.63, 1.63, 1.62, 1.62, 1.63**)
			SML_1	18, 12, 8, 5, 3, 1, 0	**94**	(1.65, 1.74, 1.69, 1.66, **1.63, 1.62, 1.61, 1.60, 1.60, 1.60**)
25	2^5	2^{12}	SD	24, 0	325	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	25, 20, 15, 10, 5, 0	**125**	(1.62, 1.64, 1.62, 1.64, **1.67, 1.69, 1.71, 1.71, 1.72, 1.72**)
			SML_0	25, 19, 13, 9, 6, 3, 1, 0	119	(1.65, 1.74, 1.69, 1.66, **1.63, 1.62, 1.61, 1.60, 1.60, 1.60**)
			SML_1	19, 13, 9, 6, 3, 1, 0	**100**	(1.65, 1.74, 1.69, 1.66, **1.63, 1.62, 1.61, 1.60, 1.60, 1.60**)
28	2^6	2^{14}	SD	28, 0	406	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			e-HS	28, 22, 16, 10, 5, 0	**146**	(1.64, 1.65, **1.63, 1.66, 1.69, 1.71, 1.73, 1.74, 1.75, 1.75**)
			SML_0	28, 21, 15, 10, 6, 3, 1, 0	140	(1.65, 1.70, 1.65, **1.63, 1.62, 1.63, 1.64, 1.66, 1.67, 1.68**)
			SML_1	22, 16, 11, 7, 4, 2, 0	**119**	(1.64, 1.65, **1.62, 1.64, 1.67, 1.69, 1.70, 1.71, 1.72, 1.72**)

strategy. For $\ell_0 = 24$ and 28, it is again true that making the root level of the reported SML_0 strategy non-special gives rise to an SML_1 strategy. But there are other SML_1 strategies that further reduce the expected header lengths and hence we report those strategies in Table 5.3.

5. In general, the header length of the e-HS scheme is smaller than that of SML_0 and SML_1. This is somewhat expected, since user storage in SML is smaller. On the other hand, the user storage is not the only determining factor. The actual layering strategy also plays a role and in some cases it turns out that the average header length in SML turns out to be smaller than that in e-HS. We do not have an analytical justification for this. Intuitively, it appears that for the number of revoked users that have been considered, the SML assigns keys to SD subsets which are more probable to occur in the header. As a result, in such cases, we see that *both* user storage and average header length are reduced. These are marked in bold and are particularly noticeable for $\ell_0 = 24$ and $\ell_0 = 28$. In the context of the [AAC] standard, SML_1 for $\ell_0 = 28$ is of particular significance.

5.2.5 Constrained Minimization of User Storage

From the viewpoint of minimizing communication bandwidth it is of interest to minimize the average header length. This is minimized when the number of keys is maximized which happens for the SD scheme, i.e., when all the levels are considered to be special levels or there is only a single layer. Taking the average header length for the SD scheme as a benchmark, one may ask the question as to how much the user storage can be reduced from that required by the SD scheme without significantly increasing the corresponding values for the average header length? The expression for the average header length (as can be derived from (5.11), (5.13) and Proposition 20 given later) is rather complicated and it appears quite impossible to have an analytical solution to this question. Instead, we use our average header length computation program (developed in Section 5.3.3) to study this behavior for concrete practical values of n, r and layering strategies $\boldsymbol{\ell}$. It turns out that it is indeed possible to significantly reduce the user storage values with minimal increase in the average header length values.

Our approach is the following. The increase in header length due to layering occurs because of the fact that certain SD subsets are split into two. If we can avoid making too

Table 5.4: $\#\mathrm{SML}_0(e,\ell_0)$ and $\Lambda(e,\ell_0)$ for $1 \leq \ell_0 \leq 32$ and $1 \leq e \leq \ell_0$.

ℓ / e	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1(0)	3(0)	6(0)	10(0)	15(0)	21(0)	28(0)	36(0)	45(0)	55(0)	66(0)	78(0)	91(0)	105(0)	120(0)	136(0)
2		3(1)	5(1)	8(1,2)	11(2)	15(2,3)	19(3)	24(3,4)	29(4)	35(4,5)	41(5)	48(5,6)	55(6)	63(6,7)	71(7)	80(7,8)
3			6(2)	8(2)	11(2,3)	14(3)	18(3,4)	22(4,5)	26(5)	31(5,6)	36(6,7)	41(7)	47(7,8)	53(8,9)	59(9)	66(9,10)
4				10(3)	12(3)	15(3,4)	18(4)	22(4,5)	26(5,6)	30(6)	35(6,7)	40(7,8)	45(8,9)	50(9)	56(9,10)	62(10,11)
5					15(4)	17(4)	20(4,5)	23(5)	27(5,6)	31(6,7)	35(7)	40(7,8)	45(8,9)	50(9,10)	55(10)	61(10,11)
6						21(5)	23(5)	26(5,6)	29(6)	33(6,7)	37(7,8)	41(8)	46(8,9)	51(9,10)	56(10,11)	61(11)
7							28(6)	30(6)	33(6,7)	36(7)	40(7,8)	44(8,9)	48(9)	53(9,10)	58(10,11)	63(11,12)
8								36(7)	38(7)	41(7,8)	44(8)	48(8,9)	52(9,10)	56(10)	61(10,11)	66(11,12)
9									45(8)	47(8)	50(8,9)	53(9)	57(9,10)	61(10,11)	65(11)	70(11,12)
10										55(9)	57(9)	60(9,10)	63(10)	67(10,11)	71(11,12)	75(12)
11											66(10)	68(10)	71(10,11)	74(11)	78(11,12)	82(12,13)
12												78(11)	80(11)	83(11,12)	86(12)	90(12,13)
13													91(12)	93(12)	96(12,13)	99(13)
14														105(13)	107(13)	110(13,14)
15															120(14)	122(14)
16																136(15)

ℓ / e	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	153(0)	171(0)	190(0)	210(0)	231(0)	253(0)	276(0)	300(0)	325(0)	351(0)	378(0)	406(0)	435(0)	465(0)	496(0)	528(0)
2	89(8)	99(8,9)	109(9)	120(9,10)	131(10)	143(10,11)	155(11)	168(11,12)	181(12)	195(12,13)	209(13)	224(13,14)	239(14)	255(14,15)	271(15)	288(15,16)
3	73(10,11)	80(11)	88(11,12)	96(12,13)	104(13)	113(13,14)	122(14,15)	131(15)	141(15,16)	151(16,17)	161(17)	172(17,18)	183(18,19)	194(19)	206(19,20)	218(20,21)
4	68(11,12)	74(12)	81(12,13)	88(13,14)	95(14,15)	102(15)	110(15,16)	118(16,17)	126(17,18)	134(18)	143(18,19)	152(19,20)	161(20,21)	170(21)	180(21,22)	190(22,23)
5	67(11,12)	73(12,13)	79(13,14)	85(14)	92(14,15)	99(15,16)	106(16,17)	113(17,18)	120(18)	128(18,19)	136(19,20)	144(20,21)	152(21,22)	160(22)	169(22,23)	178(23,24)
6	67(11,12)	73(12,13)	79(13,14)	85(14,15)	91(15)	98(15,16)	105(16,17)	112(17,18)	119(18,19)	126(19,20)	133(20)	141(20,21)	149(21,22)	157(22,23)	165(23,24)	173(24,25)
7	68(12)	74(12,13)	80(13,14)	86(14,15)	92(15,16)	98(16)	105(16,17)	112(17,18)	119(18,19)	126(19,20)	133(20,21)	140(21)	148(21,22)	156(22,23)	164(23,24)	172(24,25)
8	71(12,13)	76(13)	82(13,14)	88(14,15)	94(15,16)	100(16,17)	106(17)	113(17,18)	120(18,19)	127(19,20)	134(20,21)	141(21,22)	148(22)	156(22,23)	164(23,24)	172(24,25)
9	75(12,13)	80(13,14)	85(14)	91(14,15)	97(15,16)	103(16,17)	109(17,18)	115(18)	122(18,19)	129(19,20)	136(20,21)	143(21,22)	150(22,23)	157(23)	165(23,24)	173(24,25)
10	80(12,13)	85(13,14)	90(14,15)	95(15)	101(15,16)	107(16,17)	113(17,18)	119(18,19)	125(19)	132(19,20)	139(20,21)	146(21,22)	153(22,23)	160(23,24)	167(24)	175(24,25)
11	86(13)	91(13,14)	96(14,15)	101(15,16)	106(16)	112(16,17)	118(17,18)	124(18,19)	130(19,20)	136(20)	143(20,21)	150(21,22)	157(22,23)	164(23,24)	171(24,25)	178(25)
12	94(13,14)	98(14)	103(14,15)	108(15,16)	113(16,17)	118(17)	124(17,18)	130(18,19)	136(19,20)	142(20,21)	148(21)	155(21,22)	162(22,23)	169(23,24)	176(24,25)	183(25,26)
13	103(13,14)	107(14,15)	111(15)	116(15,16)	121(16,17)	126(17,18)	131(18)	137(18,19)	143(19,20)	149(20,21)	155(21,22)	161(22)	168(22,23)	175(23,24)	182(24,25)	189(25,26)
14	113(14)	117(14,15)	121(15,16)	125(16)	130(16,17)	135(17,18)	140(18,19)	145(19)	151(19,20)	157(20,21)	163(21,22)	169(22,23)	175(23)	182(23,24)	189(24,25)	196(25,26)
15	125(14,15)	128(15)	132(15,16)	136(16,17)	140(17)	145(17,18)	150(18,19)	155(19,20)	160(20)	166(20,21)	172(21,22)	178(22,23)	184(23,24)	190(24)	197(24,25)	204(25,26)
16	138(15)	141(15,16)	144(16)	148(16,17)	152(17,18)	156(18)	161(18,19)	166(19,20)	171(20,21)	176(21)	182(21,22)	188(22,23)	194(23,24)	200(24,25)	206(25)	213(25,26)
17	153(16)	155(16)	158(16,17)	161(17)	165(17,18)	169(18,19)	173(19)	178(19,20)	183(20,21)	188(21,22)	193(22)	199(22,23)	205(23,24)	211(24,25)	217(25,26)	223(26)
18		171(17)	173(17)	176(17,18)	179(18)	183(18,19)	187(19,20)	191(20)	196(20,21)	201(21,22)	206(22,23)	211(23)	217(23,24)	223(24,25)	229(25,26)	235(26,27)
19			190(18)	192(18)	195(18,19)	198(19)	202(19,20)	206(20,21)	210(21)	215(21,22)	220(22,23)	225(23,24)	230(24)	236(24,25)	242(25,26)	248(26,27)
20				210(19)	212(19)	215(19,20)	218(20)	222(20,21)	226(21,22)	230(22)	235(22,23)	240(23,24)	245(24,25)	250(25)	256(25,26)	262(26,27)
21					231(20)	233(20)	236(20,21)	239(21)	243(21,22)	247(22,23)	251(23)	256(23,24)	261(24,25)	266(25,26)	271(26)	277(26,27)
22						253(21)	255(21)	258(21,22)	261(22)	265(22,23)	269(23,24)	273(24)	278(24,25)	283(25,26)	288(26,27)	293(27)
23							276(22)	278(22)	281(22,23)	284(23)	288(23,24)	292(24,25)	296(25)	301(25,26)	306(26,27)	311(27,28)
24								300(23)	302(23)	305(23,24)	308(24)	312(24,25)	316(25,26)	320(26)	325(26,27)	330(27,28)
25									325(24)	327(24)	330(24,25)	333(25)	337(25,26)	341(26,27)	345(27)	350(27,28)
26										351(25)	353(25)	356(25,26)	359(26)	363(26,27)	367(27,28)	371(28)
27											378(26)	380(26)	383(26,27)	386(27)	390(27,28)	394(28,29)
28												406(27)	408(27)	411(27,28)	414(28)	418(28,29)
29													435(28)	437(28)	440(28,29)	443(29)
30														465(29)	467(29)	470(29,30)
31															496(30)	498(30)
32																528(31)

many splits, then we can ensure that the header length does not increase by too much in comparison to the SD scheme. Consider an SD subset of the form $S_{i,j}$ where node i is at level ℓ. We say that this subset is generated from the node i. Now, consider the expected number of SD subsets that will be generated from all the nodes at level ℓ. If this number is 'large', then we make the level ℓ special. This ensures that SD subsets originating level ℓ will not be split. Overall, the strategy is to ensure that SD subsets originating from levels which contribute most to the header are not split. This mitigates the effect of splits.

Suppose there are n users and r of them are revoked. In Section 4.4 [BS13] it has been shown that the probability that a particular node at level ℓ generates a subset in the header is $2(\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1}))$ where $\eta_r(n, x) = (1 - x/n)(1 - x/(n-1)) \cdots (1 - x/(n - r + 1))$ if $n > r - 1$ else 0. Since there are $2^{\ell_0 - \ell}$ nodes at level ℓ, the expected number of subsets arising from all nodes at level ℓ is

$$2^{\ell_0-\ell+1}(\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})). \qquad (5.10)$$

This expression gives the expected contribution of a level to the header size for a given r.

For a fixed n and r, one can consider the problem of finding ℓ for which (5.10) is maximized. Analytically, this seems to be very difficult to do. Instead we have done extensive experimentation. Empirical values suggest that the maximum occurs for some level $\ell \leq \ell_0 - \lfloor \log_2 r \rfloor$. Also, for $\ell > \ell_0 - \lfloor \log_2 r \rfloor$, the value of (5.10) is quite small.

Based on this empirical evidence we suggest the following layering strategy.

- Make level $\ell_0 - \lfloor \log_2 r \rfloor$ special. Level 0 is also special.
- No level $0 < \ell < \ell_0 - \lfloor \log_2 r \rfloor$ is made special. In terms of user storage and expected header length this is equivalent to making all levels $\ell < \ell_0 - \lfloor \log_2 r \rfloor$ to be special.
- The root level is not made special.
- At most one level that is midway between ℓ_0 and $\ell_0 - \lfloor \log_2 r \rfloor$ is made special. While this does not significantly affect header size, it can reduce the storage requirement.

We call this the *constrained minimization layering (CML)* strategy. This strategy will ensure that if $\ell \leq \ell_0 - \lfloor \log_2 r \rfloor$, then no SD subset generated from level ℓ or below will be split. Splits will occur only for SD subsets originating from levels above ℓ. But, the expected number of such subsets is small and so, splits will occur only for a small number of SD subsets.

One issue with this strategy is that the value of r will not be known a priori while the layering scheme will have to be decided upon during the design phase itself. A way out is to make an assumption about the minimum number of revoked users that will occur in the steady state operation of the BE scheme. For example, in AACS with 2^{28} users one may assume that in the steady state at least 2^{10} users will be revoked due to equipment piracy problems.

Suppose that $r_{\min}$ is the minimum number of users that will be revoked during each broadcast. The above layering strategy is used with $r_{\min}$. Suppose now that during a broadcast, the number r of users that is actually revoked is greater than $r_{\min}$. Then from our empirical evidence the level for which the average header length is maximized will be $\ell_0 - \lfloor \log_2 r \rfloor$. Since this value is less than $\ell_0 - \lfloor \log_2 r_{\min} \rfloor$, none of the subsets generated from this level will be split. So, the feature of not splitting a large number of SD subsets is still retained.

Table 5.5 shows a comparison between the SD scheme, the e-HS layering scheme and a constrained minimization layering scheme as described above, in terms of both their user storage requirement and the expected header length normalized with respect to the SD scheme. The average header length depends on the number r of revoked users. So, for a given $n = 2^{\ell_0}$, we computed the expected header lengths for 10 equispaced values of r between and including $r_{\min}$ and $r_{\max}$. The values in the table illustrate the point that compared to the SD scheme, the constrained minimization layering scheme substantially reduces the user storage with a small increase in the average header length.

The layering scheme is designed assuming that the number of revoked users is at least $r_{\min}$. What happens if the number of revoked users in an actual broadcast is smaller than $r_{\min}$? Clearly, we cannot expect the average header length to still be almost equal to that of the SD scheme. This effect is shown for some values of r in Table 5.6. Again the values of the average header length are normalized by that of the corresponding SD scheme. For comparison, we have also provided the average header lengths of the e-HS layering strategy. It is to be noted that the expected header lengths of the CML scheme are mostly better than the e-HS scheme. As an example, for $n = 2^{24}$, for $r > 6$, the CML strategy gives smaller expected header lengths than the e-HS layering strategy. Table 5.6 shows that for any value of n, the CML strategy leads to smaller expected header lengths for all $r > 15$.

To summarize, the constrained minimization layering strategy requires significantly less user storage than the SD scheme. In terms of the expected header length, it is as good as the SD scheme for $r \geq r_{\min}$. If $r < r_{\min}$, then it is better than e-HS layering but inferior to the SD scheme. It is to be noted that if r is small, then the absolute size of the header itself is not too large. As a result, the effective transmission overhead of the scheme will never be too high compared to the actual body of the message.

Table 5.5: Comparison of user storage and average header length for SD, e-HS LSD and the constrained minimization layering. The tuples contain header lengths normalized with the SD header lengths corresponding to the values of r in $(r_{\min}, \ldots, r_{\max})$ respectively.

ℓ_0	$r_{\min}$	$r_{\max}$	scheme	special levels	storage	normalized header lengths for $(r_{\min}, \ldots, r_{\max})$
12	2^2	2^6	SD	$12, 0$	78	$(1, \ldots, 1)$
			e-HS	$12, 8, 4, 0$	42	$(1.69, 1.59, 1.56, 1.56, 1.57, 1.57, 1.57, 1.56, 1.55, 1.53, 1.52)$
			CML	$10, 0$	58	$(1.15, 1.01, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
16	2^6	2^8	SD	$16, 0$	136	$(1, \ldots, 1)$
			HS	$16, 12, 8, 4, 0$	64	$(1.66, 1.64, 1.62, 1.61, 1.59, 1.58, 1.58, 1.57, 1.57, 1.56)$
			CML	$10, 0$	76	$(1.14, 1.08, 1.05, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$
20	2^8	2^{10}	SD	$20, 0$	210	$(1, \ldots, 1)$
			e-HS	$20, 15, 10, 5, 0$	90	$(1.68, 1.66, 1.64, 1.63, 1.62, 1.61, 1.61, 1.60, 1.60, 1.60)$
			CML	$16, 12, 0$	110	$(1.14, 1.08, 1.04, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$
24	2^{10}	2^{12}	SD	$24, 0$	300	$(1, \ldots, 1)$
			e-HS	$24, 19, 14, 9, 4, 0$	116	$(1.63, 1.64, 1.66, 1.68, 1.69, 1.71, 1.71, 1.72, 1.72, 1.72)$
			CML	$19, 14, 0$	149	$(1.14, 1.08, 1.04, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$
25	2^{10}	2^{12}	SD	$25, 0$	325	$(1, \ldots, 1)$
			e-HS	$25, 20, 15, 10, 5, 0$	125	$(1.63, 1.64, 1.66, 1.68, 1.69, 1.71, 1.71, 1.72, 1.72, 1.72)$
			CML	$20, 15, 0$	165	$(1.14, 1.08, 1.04, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$
28	2^{10}	2^{14}	SD	$28, 0$	406	$(1, \ldots, 1)$
			e-HS	$28, 22, 16, 10, 5, 0$	146	$(1.69, 1.63, 1.64, 1.67, 1.69, 1.72, 1.73, 1.74, 1.75, 1.75)$
			CML	$23, 18, 0$	219	$(1.14, 1.08, 1.04, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$

Table 5.6: Comparison of average header length for $r < r_{\min}$ between e-HS layering strategy and the constrained minimization layering strategy.

ℓ_0	$r_{\min}$	scheme	special levels	storage	header lengths normalized with the SD scheme
					$r = (1, \mathbf{2}, 3, 4)$
12	2^2	e-HS	$12, 8, 4, 0$	42	$(1.00, \mathbf{1.74}, 1.72, 1.69)$
		CML	$10, 0$	58	$(2.00, \mathbf{1.50}, 1.26, 1.15)$
					$r = (2, 4, 6, 8, 10, \mathbf{12}, 14, 16, 18, 20)$
16	2^6	HS	$12, 8, 4, 0$	64	$(1.75, 1.70, 1.66, 1.63, 1.61, \mathbf{1.60}, 1.60, 1.60, 1.60, 1.61)$
		CML	$10, 0$	76	$(1.78, 1.74, 1.70, 1.66, 1.63, \mathbf{1.59}, 1.56, 1.53, 1.50, 1.47)$
					$r = (2, \mathbf{4}, 6, 8, 10, 12, 14, 16, 18, 20)$
20	2^8	e-HS	$20, 15, 10, 5, 0$	90	$(1.77, \mathbf{1.75}, 1.72, 1.70, 1.68, 1.66, 1.65, 1.64, 1.63, 1.63)$
		CML	$16, 12, 0$	110	$(1.77, \mathbf{1.69}, 1.64, 1.61, 1.59, 1.57, 1.56, 1.56, 1.56, 1.56)$
					$r = (2, 4, 6, \mathbf{8}, 10, 12, 14, 16, 18, 20)$
24	2^{10}	e-HS	$24, 19, 14, 9, 4, 0$	116	$(1.77, 1.75, 1.72, \mathbf{1.70}, 1.68, 1.66, 1.65, 1.64, 1.63, 1.63)$
		CML	$19, 14, 0$	149	$(1.79, 1.75, 1.72, \mathbf{1.69}, 1.67, 1.65, 1.64, 1.63, 1.62, 1.61)$
					$r = (2, 4, 6, \mathbf{8}, 10, 12, 14, 16, 18, 20)$
25	2^{10}	e-HS	$25, 20, 15, 10, 5, 0$	125	$(1.77, 1.75, 1.72, \mathbf{1.70}, 1.68, 1.66, 1.65, 1.64, 1.63, 1.63)$
		CML	$20, 15, 0$	165	$(1.79, 1.75, 1.72, \mathbf{1.69}, 1.67, 1.65, 1.64, 1.63, 1.62, 1.61)$
					$r = (2, \mathbf{4}, 6, 8, 10, 12, 14, 16, 18, 20)$
28	2^{10}	e-HS	$28, 22, 16, 10, 5, 0$	146	$(1.79, \mathbf{1.78}, 1.76, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69, 1.68)$
		CML	$23, 18, 0$	219	$(1.79, \mathbf{1.75}, 1.72, 1.69, 1.67, 1.65, 1.64, 1.63, 1.62, 1.61)$

5.3 Header Length

The main point of the discussion in this section is to obtain an efficient algorithm for computing the expected header length for the layered SD schemes including the LSD scheme. The algorithm we obtain works for all possible values of the number of users. To ensure this, we first need to extend the scheme to handle an arbitrary number of users. For the SD scheme, this was done in Section 4.2 [BS13] by using the notion of complete binary trees. Here, we extend the scheme of Section 4.2 [BS13] to handle layering as well.

5.3.1 Tackling Arbitrary Number of Users

In the NNL-SD and HS-LSD schemes described in Chapter 2, the number of users has been taken to be a power of two, i.e., $n = 2^{\ell_0}$. One has to consider dummy users in the system to make the number of users a power of two. The inclusion of dummy users (considered revoked or privileged) increase the expected header length in the system. Hence, this is not always convenient as has been argued in details in Section 4.2.2 [BS13].

By modifying the structure of the tree, it is possible to handle an arbitrary number of users. This modification is based on the notion of complete binary trees. These are trees where the leaf nodes are at the last and maybe the second last levels. The last level has all its nodes to the left side. An example of a complete subtree accommodating 13 users is shown in Figure 5.1. In this case $\ell_0 = 4$ and choosing $d = 2$ gives two layers and three special levels as shown in the figure. When the number of users is a power of two, the corresponding tree is called a full binary tree. This difference in terminology between full and complete has been taken from the literature on data structures. We explain some terminology with respect to Figure 5.1. The left and the right subtrees of node 3 are the subtrees rooted at nodes 7 and 8 respectively. The sibling subtree of node 3 is the subtree rooted at node 4. The only non-full subtrees are those rooted at nodes 0, 2 and 5. We call the path labelled by the nodes 0, 2 and 5 to be the *dividing path*.

In general given n with $2^{\ell_0-1} < n \leq 2^{\ell_0}$, it is possible to accommodate n users as the leaves of a complete binary tree with n leaves. The root node is at level ℓ_0. The leaves and hence the users are either at level 0 or at level 1. Suppose the sequence of special levels is $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$. For users at level 0, the storage requirement is $\mathsf{storage}_0(\boldsymbol{\ell})$ while for users at level 1, the storage requirement is $\mathsf{storage}_0(\boldsymbol{\ell}) - (e + p - 2)$ where p is the number of levels in the bottom-most layer. This reduction is due to the fact that these users need to store one less label for each special level above it and for each level in its last layer. The distribution of labels using the PRG is done as usual.

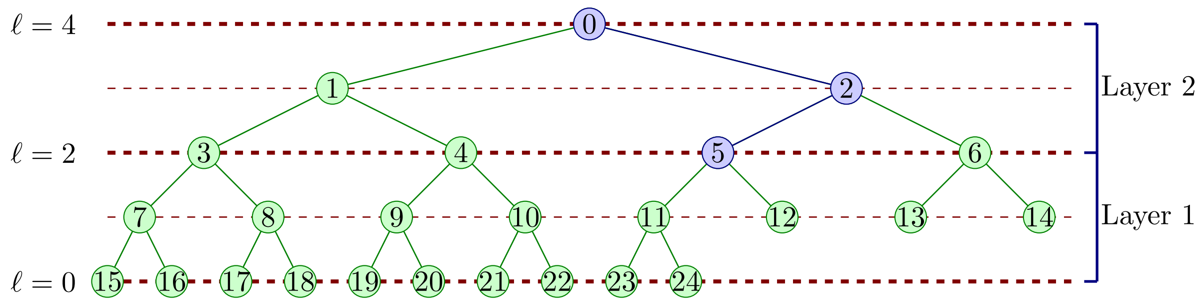


Figure 5.1: A complete tree with 13 leaf nodes. The levels 0, 2 and 4 are special levels and hence there are two layers. The nodes 0, 2 and 5 are roots of non-full complete subtrees and hence they lie on the dividing path.

During a broadcast, the actual header generation is done in much the same way. First, as in the SD scheme, the set of non-revoked users is covered exactly by subsets of the form $S_{i,j}$ where i is a node in the tree and j is a node in the subtree rooted at i. If i is at a non-special level and j is not in the same layer as i, then this set is further split into $(S_{i,k}) \cup (S_{k,j})$ where k is the first node appearing at a special level on the path from i to j.

Complications for complete but non-full trees arise due to the following reason. For the internal nodes lying on the dividing path, the subtree rooted at it may not be full. A node not on the dividing path and at level ℓ is the root of a subtree having either 2^{ℓ} leaves or $2^{\ell-1}$ leaves accordingly as whether the node is to the left or to the right of the dividing path. As an example, in Figure 5.1, nodes 3, 4, 5 and 6 are at level 2. Node 5 is on the dividing path and the subtree rooted at node 5 is non-full; nodes 3 and 4 are to the left of 5 and are the roots of subtrees having $2^2 = 4$ leaves; node 6 is to the right of node 5 and the subtree rooted at 6 has 2 leaves.

The LSD scheme is based on full binary trees and this extension to complete binary trees gives rise to the *complete tree layered subset difference (CTLSD) scheme*. The LSD scheme had improved upon the SD scheme by reducing the user storage at the cost of almost double the transmission overhead. The CTLSD scheme subsumes all these schemes by accommodating an arbitrary number of users and allowing appropriate choices of the layering strategy $\boldsymbol{\ell}$ for specific applications.

5.3.2 Maximum Header Length

Before considering the expected header length, we state the following bound on the worst case header length.

Proposition 19. *The maximum header length in the CTLSD scheme for n users out of which r are revoked is* $\min\left(4r-2, \left\lceil \frac{n}{2} \right\rceil, n-r\right)$*. If the root is a special level, then the bound is* $\min\left(4r-3, \left\lceil \frac{n}{2} \right\rceil, n-r\right)$*.*

Proof. The bound is independent of the actual layering strategy. The upper bound of $2r-1$ for the SD scheme was already given in [NNL01, NNL02] and in Chapter 4 [BS13] it was shown that this also holds for the CTSD scheme. Using the layering strategy, each subset returned by the SD algorithm can split into at most two subsets. So, if the number of SD subsets is at most $2r-1$, then there are at most $4r-2$ subsets.

Suppose the header consists of h subsets out of which h_1 are singleton sets and h_2 sets have 2 or more elements each. For each node in a singleton privileged set, its sibling (if there is one) must be a revoked user. Among all these leaves, there is only one which may not have a sibling that is also a leaf node (and this is the first privileged user from the left at level 1, for odd n). So, for the h_1 privileged users, there are at least h_1-1 other revoked users. This accounts for at least $h_1+h_1-1+2h_2 = 2h-1$ users. It is now easy to argue that if $h > \lceil n/2 \rceil$, then $2h-1$ is greater than n. Since the total number users is n, this cannot happen. So $h \leq \lceil n/2 \rceil$.

Since each subset in the subset cover will have at least one privileged user, the maximum number of subsets in the header is equal to the number of non-revoked users which is equal to $n-r$.

The bound of $4r-2$ holds for both the cases when the root is or is not a special level. If the root is a special level the bound of $4r-2$ can be improved to $4r-3$. We first provide a short argument to justify that in the SD scheme if the header length is $2r-1$, then there is a subset of the form $S_{0,j}$ in the header. As mentioned earlier, such a subset is added to the header if and only if exactly one of the subtrees of the root node do not contain any revoked user. So, if such a subset is not in the header, then both the subtrees of the root node contain at least one revoked user. Suppose the number of revoked users in these two subtrees are r_1 and r_2 where $r = r_1 + r_2$. Applying the bound on the maximum header length, we have the header to be of maximum length $2r_1-1+2r_2-1 = 2r-2$. So, if the header length is $2r-1$, then there must be a subset of the type $S_{0,j}$ in the header. Using the layering strategy, each subset returned by the SD algorithm can split into at most two subsets. So, if the number of SD subsets is at most $2r-2$, then there are at most $4r-4$ subsets. On the other hand, if the number of SD subsets is equal to $2r-1$, then as argued above there must an SD subset of the form $S_{0,j}$ in the header. Since the root node 0 is considered to be a special node, this subset will not split while all other subsets may split into two. As a result, there can be at most $4r-3$ subsets in the header. □

5.3.3 Expected Header Length

Assume that the layering strategy is given by $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_e)$. Additionally, the information as to whether the root level is or is not special is also provided as a bit β. If $\beta = 0$, then the root node is special and if $\beta = 1$, the root node is not special. So, $(\boldsymbol{\ell}, \beta)$ provides complete information about the layering strategy. For compactness, we denote this as $\boldsymbol{\ell}_\beta$.

The expected header length is computed under the same random experiment that was stated in Section 4.4, where out of n users, a set of r users are chosen uniformly at random and are revoked. The corresponding header length is then a random variable and let $Y_{n,r}$ denote this header length. We are interested in $E[Y_{n,r}]$. Due to the random revocation of the users, for each internal node i, three possibilities arise: $S_{i,j}$ is added to the header; $(S_{i,k}) \cup (S_{k,j})$ is added to the header; or nothing is added to the header. So, corresponding to node i, either 0 or 1 or 2 subsets are added to the header. Denote this number by $Y^i_{n,r}$. Then $Y_{n,r} = \sum Y^i_{n,r}$ where the sum is taken over all internal nodes i.

Computing this directly is not convenient. So, we simplify it further. Let $X^i_{n,r}$ be a binary valued random variable which takes the value 1 if and only if there is at least one subset generated from i and let $Z^i_{n,r}$ be another binary valued random variable which takes the value 1 if and only if there are exactly two subsets generated from i. (Note that if i is at a special level, then the probability $Z^i_{n,r} = 1$ is 0.) Then it follows that $Y^i_{n,r} = X^i_{n,r} + Z^i_{n,r}$. The reasoning is as follows. If i generates no subset, then both sides are zero; if exactly one subset is generated, then $Y^i_{n,r}$ and $X^i_{n,r}$ are both 1 but, $Z^i_{n,r}$ is 0; if exactly two subsets are generated then $Y^i_{n,r}$ is 2 and both $X^i_{n,r}$ and $Z^i_{n,r}$ are 1. By linearity of expectation, we have

$$\begin{aligned} E[Y_{n,r}] &= E\left[\sum Y^i_{n,r}\right] = \sum E\left[X^i_{n,r} + Z^i_{n,r}\right] \\ &= \sum E\left[X^i_{n,r}\right] + \sum E\left[Z^i_{n,r}\right]. \end{aligned} \tag{5.11}$$

The sum is over all internal nodes i of the tree. The quantity $\sum X^i_{n,r}$ is exactly the expected header length obtained using the SD algorithm. This is because i generates at least one subset if and only if the SD algorithm results in i generating a subset. Let $X_{n,r} = \sum X^i_{n,r}$ and $Z_{n,r} = \sum Z^i_{n,r}$. So,

$$E\left[Y_{n,r}\right] = E\left[X_{n,r}\right] + E\left[Z_{n,r}\right]. \tag{5.12}$$

Algorithm 1 for computing $E[X_{n,r}]$ has been already developed in Section 4.4 [BS13]. So, it only remains to determine $E[Z_{n,r}]$.

Given n and a layering sequence $\boldsymbol{\ell}_\beta$ we define the set $\mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$ to consist of pairs of nodes (i, j) such that i is not at a special level and j is in the subtree rooted at i but not in the same layer as i. So, whenever an SD subset $S_{i,j}$ is such that $(i, j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$, it is split into two subsets. If i is at level ℓ, then there are at most $\ell - 1$ values of level for j such that (i, j) is in $\mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$.

Let i be at a non-special level and let j be not in the same layer as i. Define the binary valued random variable $W^{i,j}_{n,r}$ to take the value 1 if and only if the SD algorithm returns the subset $S_{i,j}$ to the header, in which case the LSD algorithm will split this subset into two sets. So, we have $Z^i_{n,r} = \sum_{(i,j)\in\mathsf{SubsetsForSplit}(n,\boldsymbol{\ell}_\beta)} W^{i,j}_{n,r}$. Again by linearity of expectation, the task reduces to computing $E[W^{i,j}_{n,r}]$. Since this is a binary valued random variable, $E[W^{i,j}_{n,r}] = \Pr[W^{i,j}_{n,r} = 1]$. So,

$$E[Z_{n,r}] = \sum_i E[Z^i_{n,r}] = \sum_i \sum_{(i,j)\in\mathsf{SubsetsForSplit}(n,\boldsymbol{\ell}_\beta)} \Pr[W^{i,j}_{n,r} = 1]. \tag{5.13}$$

Here the first sum is over all nodes i at non-special levels. For a fixed i and j, we show how to compute $\Pr[W^{i,j}_{n,r} = 1]$. To do this, we need to characterize the event $W^{i,j}_{n,r} = 1$ for a pair $(i, j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$. This event occurs if and only if the following conditions hold.

- Node i is either the root (in which case it does not have any sibling tree) or the sibling tree of i has at least one revoked user among its leaves.
- Either j is a leaf and is revoked or both subtrees of j have at least one revoked user among its leaves.
- There are no revoked users in the set $S_{i,j}$.

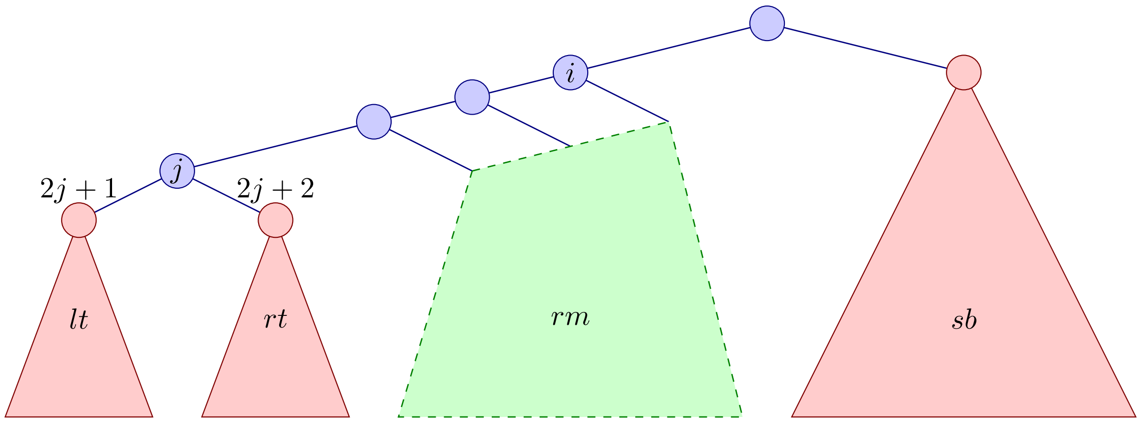


Figure 5.2: Figure demonstrating the event $R^i_{sb} \wedge \overline{R^{i,j}_{rm}} \wedge R^j_{lt} \wedge R^j_{rt}$. The triangles represent subtrees rooted at the respective nodes. The quadrilateral represents the union of all subtrees in $\mathcal{T}^i \setminus \mathcal{T}^j$ that contain the users in $S_{i,j}$. Green denotes that the portion of the tree has no revoked user in it. Red denotes that the subtree has at least one revoked user in it. The sizes of the subtrees are not to the scale of the number of users in them.

Define the following events:

1. R^j_{lt}: there is at least one revoked user in the left subtree of j;
2. R^j_{rt}: there is at least one revoked user in the right subtree of j;
3. R^i_{sb}: there is at least one revoked user in the sibling subtree of i;
4. $R^{i,j}_{rm}$: there is at least one revoked user in the set $S_{i,j}$.

Let $(i,j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$. Suppose i is not the root. If j is not a leaf node, the event $W^{i,j}_{n,r} = 1$ is equivalent to the event $R^i_{sb} \wedge \overline{R^{i,j}_{rm}} \wedge R^j_{lt} \wedge R^j_{rt}$. If j is a leaf node, the event $W^{i,j}_{n,r} = 1$ is equivalent to the event $R^i_{sb} \wedge \overline{R^{i,j}_{rm}}$. Now suppose i is the root and is not special (i.e., $\beta = 1$). If j is not a leaf, then the event $W^{i,j}_{n,r} = 1$ is equivalent to $\overline{R^{i,j}_{rm}} \wedge R^j_{lt} \wedge R^j_{rt}$. If j is a leaf, then this can happen only if there is a single revoked user. So, for $r = 1$, the probability of $W^{i,j}_{n,r} = 1$ is 1 and for $r \geq 2$, the probability of $W^{i,j}_{n,r} = 1$ is 0.

Let λ_i (resp. λ_j; λ_s) be the number of leaves in the subtree rooted at i (resp. j; the sibling subtree of i). Similarly, let λ_{2j+1} and λ_{2j+2} respectively be the number of leaves in the left and right subtrees of j. So, $\lambda_j = \lambda_{2j+1} + \lambda_{2j+2}$. The number of leaves in the set $S_{i,j}$ is $\lambda_i - \lambda_j$. Note that since we are dealing with an arbitrary number of users, the subtrees that are being considered are not necessarily full. So, the values of the λ's are not necessarily powers of two.

Fix t users and consider the probability $\eta_r(n,t)$ that was defined in Section 4.4 with respect to the random experiment where none of the t users have been chosen. Recall that the random experiment is to choose r users uniformly and without replacement from the set of n users. As discussed earlier

$$\eta_r(n,t) = \left(1-\frac{t}{n}\right)\left(1-\frac{t}{n-1}\right)\cdots\left(1-\frac{t}{n-r+1}\right).$$

This makes it convenient to express the probability that none among a set of users of certain size is revoked. For example, the probability of $\overline{R_{lt}^{j}}$ is $\eta_r(n,\lambda_{2j+1})$. Similarly, the probability of the event $\overline{R_{lt}^{j}} \wedge \overline{R_{rm}^{i,j}}$ is $\eta_r(n,\lambda_{2j+1}+\lambda_i-\lambda_j) = \eta_r(n,\lambda_i-\lambda_{2j+2})$. Such calculations will be used in what follows.

Proposition 20. *Let i and j be nodes such that $(i,j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$.*

- *If i is the root and j is a leaf, then $\Pr[W_{n,r}^{i,j}=1]=1$ if $r=1$ and $\Pr[W_{n,r}^{i,j}=1]=0$ if $r\geq 2$.*
- *If i is the root and j is not a leaf, then*

$$\begin{aligned}\Pr[W_{n,r}^{i,j}=1] = {} & \eta_r(n,\lambda_i-\lambda_j) - \eta_r(n,\lambda_{2j+1}+\lambda_i-\lambda_j)\\ & -\eta_r(n,\lambda_{2j+2}+\lambda_i-\lambda_j)\\ & +\eta_r(n,\lambda_{2j+1}+\lambda_{2j+2}+\lambda_i-\lambda_j). \qquad (5.14)\end{aligned}$$

- *If i is not the root and j is a leaf, then*

$$\Pr[W_{n,r}^{i,j}=1] = \eta_r(n,\lambda_i-\lambda_j) - \eta_r(n,\lambda_s+\lambda_i-\lambda_j). \qquad (5.15)$$

- *If i is not the root and j is not a leaf, then*

$$\begin{aligned}\Pr[W_{n,r}^{i,j}=1] = {} & \eta_r(n,\lambda_i-\lambda_j) - \eta_r(n,\lambda_s+\lambda_i-\lambda_j)\\ & -\eta_r(n,\lambda_{2j+1}+\lambda_i-\lambda_j)\\ & -\eta_r(n,\lambda_{2j+2}+\lambda_i-\lambda_j)\\ & +\eta_r(n,\lambda_s+\lambda_{2j+1}+\lambda_i-\lambda_j)\end{aligned}$$

$$
\begin{aligned}
&+\eta_r(n, \lambda_s + \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&+\eta_r(n, \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&-\eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j).
\end{aligned}
\tag{5.16}
$$

Proof. We consider the case when i is not the root and j is not a leaf. The other cases are similar. When i is not the root and j is not a leaf, the event $W_{n,r}^{i,j} = 1$ is equivalent to the event $R_{sb}^{i,j} \wedge \overline{R_{rm}^{i,j}} \wedge R_{lt}^{i,j} \wedge R_{rt}^{i,j}$. We now compute as follows.

$$
\begin{aligned}
&\Pr[R_{sb}^{i,j} \wedge \overline{R_{rm}^{i,j}} \wedge R_{lt}^{i,j} \wedge R_{rt}^{i,j}] \\
&= \Pr[R_{sb}^{i,j} \wedge R_{lt}^{i,j} \wedge R_{rt}^{i,j} | \overline{R_{rm}^{i,j}}] \times \Pr[\overline{R_{rm}^{i,j}}] \\
&= \Big(1 - \Pr[\overline{R_{sb}^{i,j} \wedge R_{lt}^{i,j} \wedge R_{rt}^{i,j}} | \overline{R_{rm}^{i,j}}]\Big) \times \Pr[\overline{R_{rm}^{i,j}}] \\
&= (1 - \Pr[\overline{R_{sb}^{i,j}} | \overline{R_{rm}^{i,j}}] - \Pr[\overline{R_{lt}^{i,j}} | \overline{R_{rm}^{i,j}}] - \Pr[\overline{R_{rt}^{i,j}} | \overline{R_{rm}^{i,j}}] \\
&\quad + \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{lt}^{i,j}} | \overline{R_{rm}^{i,j}}] + \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{rt}^{i,j}} | \overline{R_{rm}^{i,j}}] \\
&\quad + \Pr[\overline{R_{lt}^{i,j}} \wedge \overline{R_{rt}^{i,j}} | \overline{R_{rm}^{i,j}}] \\
&\quad - \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{lt}^{i,j}} \wedge \overline{R_{rt}^{i,j}} | \overline{R_{rm}^{i,j}}]) \times \Pr[\overline{R_{rm}^{i,j}}] \\
&= (\Pr[\overline{R_{rm}^{i,j}}] - \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] - \Pr[\overline{R_{lt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] \\
&\quad - \Pr[\overline{R_{rt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] + \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{lt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] \\
&\quad + \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{rt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] + \Pr[\overline{R_{lt}^{i,j}} \wedge \overline{R_{rt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}] \\
&\quad - \Pr[\overline{R_{sb}^{i,j}} \wedge \overline{R_{lt}^{i,j}} \wedge \overline{R_{rt}^{i,j}} \wedge \overline{R_{rm}^{i,j}}]) \\
&= \eta_r(n, \lambda_i - \lambda_j) \\
&\quad -\eta_r(n, \lambda_s + \lambda_i - \lambda_j) \\
&\quad -\eta_r(n, \lambda_{2j+1} + \lambda_i - \lambda_j) \\
&\quad -\eta_r(n, \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&\quad +\eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_i - \lambda_j) \\
&\quad +\eta_r(n, \lambda_s + \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&\quad +\eta_r(n, \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&\quad -\eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j).
\end{aligned}
\tag{5.17}
$$

The above expression is obtained by conditioning on the event $\overline{R_{rm}^{i,j}}$ and so for the computation to go through one needs to assume that the probability of this event is positive. In the case where this probability is zero, one can directly verify that the probabilities on both sides are zero. □

Algorithm to compute $Z_{n,r}$: For any fixed $(i, j) \in \mathsf{SubsetsForSplits}(n, \boldsymbol{\ell}_\beta)$, Proposition 20 provides a method for computing $\Pr[W^{i,j}_{n,r} = 1]$. Each of the η expressions can be computed using r multiplications and since there are a constant number of η's, the value of $\Pr[W^{i,j}_{n,r} = 1]$ can be computed using $O(r)$ multiplications. Using (5.13) this immediately gives a method for computing $Z_{n,r}$. Doing this directly, however, is not very efficient. The first sum in (5.13) is over all possible nodes i and the second sum is over the relevant j which are paired with i. Since the number of nodes is $O(n)$, a direct computation will lead to an algorithm whose running time is $O(rn^2)$.

This can be significantly improved. To explain the idea, first consider n to be a power of two so that the tree is a full binary tree. Fix a non-special node i and consider all possible j for which the second sum in (5.13) has to be evaluated. From the expression for $\Pr[W^{i,j}_{n,r} = 1]$ it is easy to note that for a fixed (n and r and) i, the value of $\Pr[W^{i,j}_{n,r} = 1]$ is determined only by the number of leaves in the subtree rooted at j and consequently the number of leaves in the left and the right subtrees of j. Since the tree is full, these values depend only on the value of the level of node j. So, for each appropriate level below i, one can compute the value of $\Pr[W^{i,j}_{n,r} = 1]$ for one particular j at that level and then multiply by the number of nodes in the subtree rooted at i at the level of j. As a result, the second sum in (5.13) can be computed in $O(r \log \lambda_i)$ time where λ_i is the number of leaves in the subtree rooted at i so that $\log \lambda_i$ is the level number of i. Since $\lambda_i \leq n$, the second sum in (5.13) can be computed using $O(r \log n)$ time.

Consider now the first sum in (5.13) (and still assume that n is a power of two). Again, it is easy to note that the value of $E[Z^i_{n,r}]$ is determined by the value of the level number of i. So, for each appropriate level, one can compute $E[Z^i_{n,r}]$ for one i and then multiply by the number of nodes at that level. As a result, computing $E[Z_{n,r}]$ requires a total of $O(r \log^2 n)$ multiplications.

If n is not a power of two, then the tree is a complete but, non-full tree and we need to revise the above description. The idea that all nodes at the same level contribute the same value does not hold any more. This is because the number of leaves in the subtrees rooted at nodes at the same level can be different. There is however, a way out which is based on the idea of the dividing path. One may recollect that the dividing path joins all nodes that are roots of non-full subtrees. All nodes at the same level and on the same side of the dividing path have the same number of leaf nodes. So, for each level, we compute separately for three cases: for nodes to the left of the dividing path; for the node on the dividing path; and for nodes to the right of the dividing path. For nodes at the same level and on the same side of the dividing path, we compute $\Pr[W_{n,r}^{i,j} = 1]$ once and multiply by the number of nodes satisfying this condition. Similarly the computation of $E[Z_{n,r}^{i}]$ is carried out. The level-wise computations of $E[Z_{n,r}^{i}]$ along with that of $E[X_{n,r}^{i}]$ in Algorithm 1 gives us the algorithm to compute the expected header length. Overall, the complexity of the algorithm is still $O(r \log^2 n)$.

There is one complication that we have not explained. This is the problem of characterizing the dividing path and counting the number of nodes at the same level and on the same side of the dividing path. It turns out that given the value of n, this can always be done. The details are provided in Section 4.4 [BS13] and so are omitted here. We have incorporated these in our implementation of the algorithm to compute expected header length given any value of n and r.

The expected header length of the CTLSD method is $E[Y_{n,r}]$. As given in (5.12), this quantity is equal to the sum of $E[X_{n,r}]$ and $E[Z_{n,r}]$. We have shown that $E[Z_{n,r}]$ can be computed in $O(r \log^2 n)$ time. The quantity $E[X_{n,r}]$ is the expected header length of the CTSD scheme and can be computed in $O(r \log n)$ time as has been described in Section 4.4 [BS13]. So, the overall complexity of the algorithm is $O(r \log^2 n)$.

Table 5.7 provides some examples of running the algorithm for computing expected header length for non-full trees using the CTSD and the CTLSD schemes. The chosen values of r are 10 equispaced values between $r_{\min}$ and $r_{\max}$ for the respective n. The CTLSD method is run by adopting the constrained minimization layering strategy where all levels including and below $\ell_0 - \lfloor \log_2 r_{\min} \rfloor$ are considered to be in one layer. The expected header length of the CTLSD method is almost similar to the CTSD scheme while the user storage requirement is a little more than half of the CTSD scheme. Hence, with an assumption on the minimum number of revoked users, the CTLSD scheme with the constrained minimization layering strategy would be the more practical choice.

Since the CTLSD scheme subsumes the HS LSD and the e-HS LSD schemes, this algorithm computes the expected header length for these schemes too. In [HS02], it was mentioned that the expected header length for their layering scheme, i.e; HS layering is around

$2r$. As we have seen earlier, by suitably placing the special levels, this can be brought down significantly to about the expected header length of the SD scheme. On the other hand, for the (e-)HS scheme, the expected header length can also be somewhat larger than $2r$. For example, for $l_0 = 28$ and $r = 2$, the expected header length is $2.23r$.

5.4 Conclusion

In this chapter, we have suggested new layering strategies for the SD scheme. At one end we have shown that it is possible to decrease the user storage below that obtained by Halevy and Shamir [HS02]. At the other end, we have shown that it is possible to attain header length very close to that of the SD scheme while still requiring a significantly smaller number of keys. The LSD scheme is extended to handle an arbitrary number of users leading to the CTLSD scheme. We have obtained an efficient algorithm to compute the expected header length in the CTLSD scheme. Our analysis of different scenarios is made possible by using this algorithm.

Table 5.7: Comparison of the storage and the expected header lengths for the CTSD and the CTLSD (with constrained minimization layering) schemes.

n	scheme	special layers	storage	$r_{\min}$	$r_{\max}$	header length normalized by CTSD
10^3	CTSD	10,0	55	2^2	2^5	$(1,\ldots,1)$
	CTLSD	8,0	**39**	2^2	2^5	$(1.09, 1.02, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^4	CTSD	14,0	105	2^4	2^7	$(1,\ldots,1)$
	CTLSD	10,0	**65**	2^4	2^7	$(1.04, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^5	CTSD	17,0	153	2^6	2^8	$(1,\ldots,1)$
	CTLSD	11,0	**87**	2^6	2^8	$(1.08, 1.04, 1.02, 1.01, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^6	CTSD	20,0	210	2^8	2^{10}	$(1,\ldots,1)$
	CTLSD	16,12,0	**110**	2^8	2^{10}	$(1.13, 1.07, 1.04, 1.02, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$
10^7	CTSD	24,0	300	2^{10}	2^{12}	$(1,\ldots,1)$
	CTLSD	19,14,0	**149**	2^{10}	2^{12}	$(1.04, 1.02, 1.01, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^8	CTSD	27,0	378	2^{10}	2^{13}	$(1,\ldots,1)$
	CTLSD	22,17,0	**200**	2^{10}	2^{13}	$(1.08, 1.04, 1.02, 1.01, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^9	CTSD	30,0	465	2^{10}	2^{15}	$(1,\ldots,1)$
	CTLSD	25,20,0	**260**	2^{10}	2^{15}	$(1.12, 1.07, 1.04, 1.02, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$

Chapter 6

Generalization of the Subset Difference Scheme Using Trees of Higher Arity

6.1 Introduction

In Chapter 1, we pointed out that as the number of sets in the collection $\mathcal{S}$ grows, the header length may come down at the cost of increasing the user storage. With this understanding, this chapter extends the idea of SD sets introduced for binary trees in Chapter 2 [NNL01, NNL02] to k-ary trees for any $k \geq 2$. (We start off assuming n to be a power of k and later extend it using the idea of complete trees instead of full trees.) Our treatment is general and unified for all values of k.

As the arity k of the underlying tree increases, the number of subsets in $\mathcal{S}$ also increases and as a result the header length generally decreases (with exceptions that have been discussed later). Working out the details of the scheme and the resulting analysis shows up a rich complexity of behavior which is not apparent at the outset. We provide an extensive analysis of the scheme covering the following points.

The Generalized Scheme. We propose a hierarchy of BE schemes parameterized by the arity k of the underlying tree. For a fixed value of k, we get a BE scheme in this hierarchy. A single cover generation algorithm which works for all k is developed. Putting $k = 2$ yields the NNL-SD scheme of Chapter 2 [NNL01, NNL02]. The work of NNL provides a clever way to use a pseudo-random generator so that user storage consists of $1 + \lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)/2$ seeds. The direct combination of this idea with the SD sets of a k-ary tree makes the user storage to be $1 + (2^{k-1} - 1)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds. We show that a modification based on the use of cyclotomic cosets modulo $2^k - 1$ reduces the user storage to $1 + (\chi_k - 2)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds, where χ_k is the number of cyclotomic cosets modulo $2^k - 1$.

Traitor Tracing. The NNL paper [NNL01, NNL02] provides a mechanism for tracing traitors. With some modification, this idea also fits the k-ary BE scheme. It turns out that compared to binary trees, for $k \geq 3$, tracing traitors can be done with fewer queries.

Header Length. For k-ary trees with n users, the maximum header length of a transmission with r revoked users is shown to be $\min(2r-1, n-r, \lceil n/k \rceil)$. Using the technique developed in Chapter 4, we devise an algorithm to compute the expected header length of these schemes for given values of k, n and r. The expression for the expected header length can be computed in $O(r \log n)$ time and $O(1)$ space. Using our implementation of this algorithm we provide representative values to show the average header lengths for different values of k.

Layering. The idea of layering is extended for the k-ary tree generalization of the SD scheme. The choice of the layering strategy determines the user storage of the layered version of the scheme. A dynamic programming algorithm is proposed to compute the layering strategies for which the user storage is minimum. This generalizes the algorithm for $k = 2$ which was given in Chapter 5 [BS14a].

The contents of this chapter are based on the paper [BS15].

6.2 The k-ary Tree Subset Difference Scheme

The description of the scheme is given in two parts – initiation and the cover generation algorithm.

6.2.1 Initiation

Fix the arity of the underlying tree to be a positive integer $k \geq 2$ and let the number n of users to be a power of k, say $n = k^{\ell_0}$. (Later, we describe how to handle the case when n is not a power of k.) Let $\mathcal{T}^0$ be a full k-ary tree having $n = k^{\ell_0}$ leaf nodes. There are $\ell_0 + 1$ levels in $\mathcal{T}^0$. The root node is considered to be at level ℓ_0 while the leaf nodes are considered to be at level 0. The total number of nodes in $\mathcal{T}^0$ is

$$1 + k + k^2 + \ldots + k^{\ell_0} = \frac{nk-1}{k-1}.$$

The users are assumed to be at the leaf nodes of $\mathcal{T}^0$. So, the set $\mathcal{N}$ of all users consists of the leaf nodes of $\mathcal{T}^0$.

Numbering of the Nodes in $\mathcal{T}^0$. The nodes in $\mathcal{T}^0$ are numbered as follows: the root is numbered 0; the k children of an internal node i are numbered from left to right as $ki+1$, $ki+2$, …, $ki+k$. The nodes in $\mathcal{T}^0$ are identified by their numbers. So, the parent of any node i is $\lfloor \frac{i-1}{k} \rfloor$. For a node i, we denote by $\mathcal{T}^i$ the subtree of $\mathcal{T}^0$ rooted at i.

The Collection $\mathcal{S}$. Let i be an internal node and suppose J is a set of nodes in $\mathcal{T}^i$ which have a common parent (and so the nodes in J are siblings) such that $1 \leq |J| < k$. Let $S_{i,J}$ be the set of leaf nodes in the subgraph

$$\mathcal{T}^i \setminus \bigcup_{j \in J} \mathcal{T}^j.$$

$S_{i,J}$ is a subset of the leaf nodes of $\mathcal{T}^0$ and so a subset of the set of all users $\mathcal{N}$. Define $\mathcal{S}$ to be the collection of all possible $S_{i,J}$ and also $\mathcal{N}$. Keys are assigned only to the subsets of users in $\mathcal{S}$ and to no other subsets of $\mathcal{N}$.

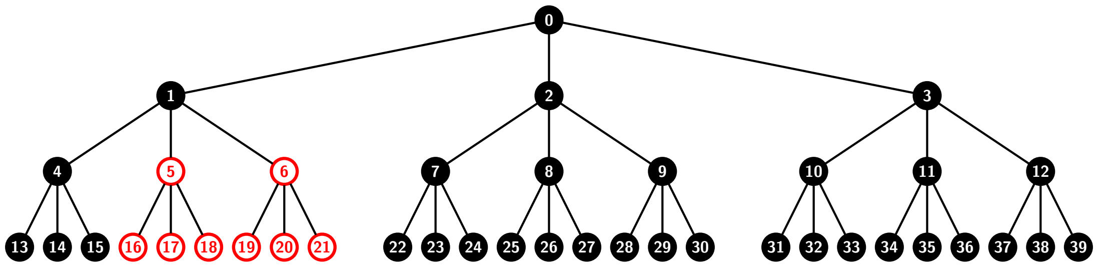


Figure 6.1: The k-ary tree $\mathcal{T}^0$ with $k = 3$ and $n = 27$ users. The subset $S_{0,\{5,6\}}$ contains all users (leaves) in the subtree $\mathcal{T}^0$ but not in $\mathcal{T}^5$ or $\mathcal{T}^6$. Hence, $S_{0,\{5,6\}} = \{13, 14, 15, 22, 23, \ldots, 39\}$.

Number of Subsets in the Collection $\mathcal{S}$. We count the number of subsets $S_{i,J}$ in $\mathcal{S}$. Fix an internal node i and suppose it is at level ℓ. Let us now consider the number of subsets of nodes J such that $S_{i,J}$ is in $\mathcal{S}$. There are two conditions on J: all nodes in J have the same parent and $1 \leq |J| \leq k-1$. The common parent of the nodes in J is an internal node in $\mathcal{T}^i$. So an internal node in $\mathcal{T}^i$ gives rise to $2^k - 2$ possible subsets of nodes J. The number of internal nodes in $\mathcal{T}^i$ is $1 + k + k^2 + \ldots + k^{\ell-1} = (k^\ell - 1)/(k-1)$. So for a fixed node i at level ℓ, there are a total of $(2^k - 2)((k^\ell - 1)/(k-1))$ possible subsets $S_{i,J}$.

In $\mathcal{T}^0$, there are $k^{\ell_0 - \ell}$ internal nodes at level ℓ. Therefore, the number of subsets generated by all the nodes at level ℓ is $(k^\ell - 1)/(k-1) \times (2^k - 2) \times k^{\ell_0 - \ell}$. Summing this by varying ℓ from 0 to ℓ_0 we get the total number of subsets $S_{i,J}$ in $\mathcal{S}$. Additionally, we have to count the set $\mathcal{N}$ of all users. Hence, the total number of subsets in $\mathcal{S}$ is

$$\begin{aligned} |\mathcal{S}| &= 1 + (2^k - 2) \sum_{\ell=1}^{\ell_0} \frac{k^\ell - 1}{k-1} (k^{\ell_0 - \ell}) \\ &= 1 + \frac{2^k - 2}{k-1} \sum_{\ell=0}^{\ell_0 - 1} (n - k^\ell) \\ &= 1 + \frac{2^k - 2}{k-1} \left(n\ell_0 + \frac{n-1}{k-1} \right). \end{aligned} \tag{6.1}$$

For $n = 16$, and $k = 2$, the number of subsets in $\mathcal{S}$ is 159. For $n = 16$, and $k = 4$, the number of subsets in $\mathcal{S}$ is 323. We observe that for a fixed n, the number of subsets in the collection increases with increasing k. Intuitively, it seems that increasing the number of subsets in $\mathcal{S}$ by increasing k should decrease the header length. This, however, is not always true. Later, we make a detailed analysis of both the maximum and the average header length of the scheme.

Key Assignment to Subsets in $\mathcal{S}$. Given an m-bit string, we need to obtain $2^k - 1$ m-bit strings. This is achieved as follows:

Let $G : \{0, \ldots, 2^k - 2\} \times \{0,1\}^m \to \{0,1\}^m$ be a cryptographic hash function. Define $G_\sigma(\mathit{seed}) \triangleq G(\sigma, \mathit{seed})$. This defines $G_\sigma(\mathit{seed})$ for an m-bit string *seed* and $0 \leq \sigma \leq 2^k - 2$. One advantage of this method is that given σ it allows directly "jumping" to a particular $G_\sigma(\mathit{seed})$. We note though that the description of the key assignment method given below does not depend on the particular manner in which G_σ has been defined.

The key assigned to a subset $S_{i,J}$ is defined indirectly. The procedure is described as follows.

1. Every internal node i is assigned an independent and uniform random seed L_i.
2. Every node $j \neq i$ in the subtree $\mathcal{T}^i$ is assigned a seed $L_{i,\{j\}}$ derived from L_i using G in the following manner.
   (a) Suppose j is an immediate child of i and write j as $j = ki + s + 1$ for some $0 \leq s \leq k-1$. Define $L_{i,\{j\}} = G_{2^s}(L_i)$.
   (b) If j is not an immediate child of i, then let $i = t_0, \ldots, t_p = j$ be a sequence of nodes from i to j. Let $t_q = kt_{q-1} + s_q + 1$ where $0 \leq s_q \leq k-1$ for $0 \leq q \leq p$. Define $L_{i,\{j\}} = G_{2^{s_p}}(G_{2^{s_{p-1}}}(\cdots G_{2^{s_1}}(L_i)))$.
3. Let j (possibly equal to i) be an internal node in $\mathcal{T}^i$ and let $J \subset \{kj+1, kj+2, \ldots, kj+k\}$ with $2 \leq |J| \leq k-1$. The previous step has already defined the seed $L_{i,\{j\}}$. Let s be the unique integer in $\{0, \ldots, 2^k - 2\}$ such that the k-bit binary representation of s encodes J, i.e., the bth bit of this binary representation is 1 if and only if $kj + b$ is in J. Define $L_{i,J} = G_s(L_{i,\{j\}})$.
4. For each possible subset $S_{i,J}$, the above procedure defines the seed $L_{i,J}$. The key assigned to the subset $S_{i,J}$ is $G_0(L_{i,J})$.

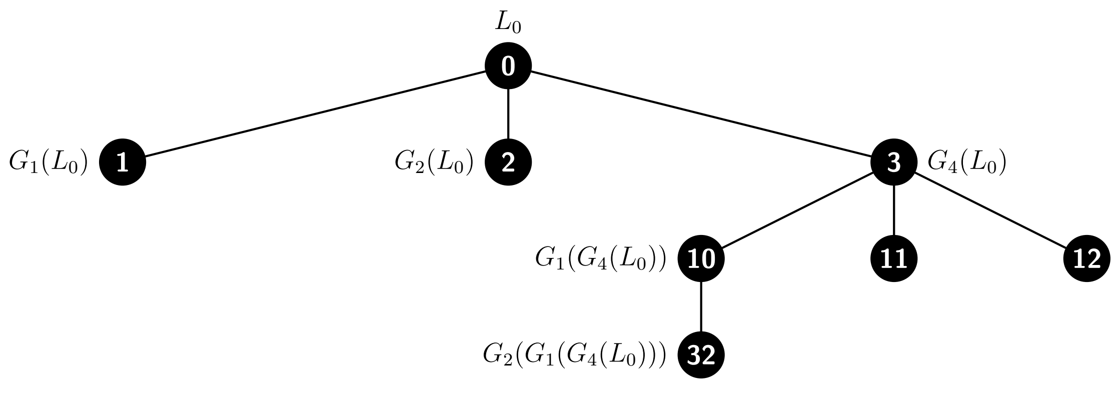


Figure 6.2: Seeds derived by node 32 and its ancestors 3 and 10 from L_0. $L_{0,\{32\}} = G_2(G_1(G_4(L_0)))$.

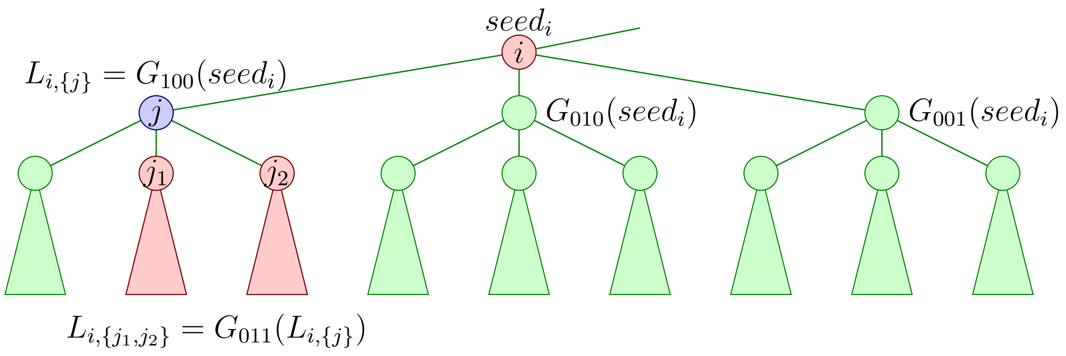

$seed_i$
i
$L_{i,\{j\}} = G_{100}(seed_i)$
j
$G_{010}(seed_i)$
$G_{001}(seed_i)$
j_1
j_2
$L_{i,\{j_1,j_2\}} = G_{011}(L_{i,\{j\}})$

Figure 6.3: Key of $S_{i,\{j_1,j_2\}}$ is $G_{000}(L_{i,\{j_1,j_2\}}) = G_{000}(G_{011}(G_{100}(seed_i)))$.

To illustrate the assignment of seeds, let us consider the tree $\mathcal{T}^0$ with $k = 3$ and $n = 27$ users as shown in Figure 6.1. The internal nodes $0, \ldots, 12$ get uniform random seeds $L_0, \ldots, L_{12}$ respectively. The seeds derived from L_0 by nodes at levels 2 and 1 are as follows:

For level 2:

$L_{0,\{1\}} = G_1(L_0)$, $L_{0,\{2\}} = G_2(L_0)$, $L_{0,\{3\}} = G_4(L_0)$,
$L_{0,\{1,2\}} = G_3(L_0)$, $L_{0,\{2,3\}} = G_6(L_0)$, $L_{0,\{1,3\}} = G_5(L_0)$.

For level 1:

$L_{0,\{4\}} = G_1(G_1(L_0))$, $L_{0,\{5\}} = G_2(G_1(L_0))$, $L_{0,\{6\}} = G_4(G_1(L_0))$,
$L_{0,\{4,5\}} = G_3(G_1(L_0))$, $L_{0,\{5,6\}} = G_6(G_1(L_0))$, $L_{0,\{4,6\}} = G_5(G_1(L_0))$,
$L_{0,\{7\}} = G_1(G_2(L_0))$, $L_{0,\{8\}} = G_2(G_2(L_0))$, $L_{0,\{9\}} = G_4(G_2(L_0))$,
$L_{0,\{7,8\}} = G_3(G_2(L_0))$, $L_{0,\{8,9\}} = G_6(G_2(L_0))$, $L_{0,\{7,9\}} = G_5(G_2(L_0))$,
$L_{0,\{10\}} = G_1(G_4(L_0))$, $L_{0,\{11\}} = G_2(G_4(L_0))$, $L_{0,\{12\}} = G_4(G_4(L_0))$,
$L_{0,\{10,11\}} = G_3(G_4(L_0))$, $L_{0,\{11,12\}} = G_6(G_4(L_0))$, $L_{0,\{10,12\}} = G_5(G_4(L_0))$.

Similarly, the seeds derived from L_0 for subtrees at level 0 and their combinations, can be determined.

Figure 6.2 shows how node 32 of Figure 6.1 gets its derived seed from the uniform random seed L_0. There will be seeds derived from every such L_i. An example of key assignment to a subset $S_{i,J}$ where $2 \leq |J| \leq k-1$ is shown in Figure 6.3.

Storage Per User. During initiation, a user will receive information (a set of derived seeds), from which it can derive the keys of all subsets it belongs to and no more. A user is associated to a leaf node in $\mathcal{T}^0$. The SD subsets $S_{i,J}$ the user will belong to, will be rooted at some ancestor node i of that leaf. However, none of the nodes in J will be an ancestor of that leaf. Thus, a user belongs to all subsets $S_{i,\{j_1,\ldots,j_s\}}$ for which

- i is an ancestor of the user leaf, and
- none of the nodes $j_1, \ldots, j_s$ are on the path joining the root node and the user leaf.

A user has to receive seeds such that it can generate the keys of all such subsets. We have already seen how keys for subsets are derived from seeds assigned to nodes in $\mathcal{T}^0$. Out of these seeds, a user gets the derived seeds from which it can generate the keys of subsets to which it belongs and no more.

The general strategy for assignment of seeds to users is as follows. Let us consider the path joining the user leaf and the root node in $\mathcal{T}^0$. Let i be a node on this path and hence an ancestor of that user. The key for a subset $S_{i,J}$ to which the user belongs will be derived from L_i. None of the nodes in J are on the path joining the user leaf and i (a part of the path with the root). Hence, the nodes in J are siblings that are either directly attached to this path or are in a subtree attached to this path. *The user gets the seeds derived from L_i of all nodes and their combinations that are directly attached with (or "falling-off" from) this path.* Using these seeds and G, the user can derive the keys of every subset $S_{i,J}$ to which it belongs and no more.

To illustrate the assignment of seeds to the users, let us again consider the tree in Figure 6.1. As an example, we look at the information that has to be given to the user at leaf 13. For that, we first identify the subsets to which the user at 13 belongs. The user at leaf 13 has three ancestor nodes $4, 1$ and 0. Hence, it belongs to subsets of the form $S_{0,J}$, $S_{1,J}$ and $S_{4,J}$ where nodes in the respective subsets J are not ancestors of the leaf 13. If the user at leaf 13 gets the derived seed $L_{0,\{2\}}$, it can derive using G, the key for any subset $S_{0,J}$ where nodes in J are in the subtree rooted at node 2. Similarly, with the derived seed $L_{0,\{3\}}$, the user can derive the key for any subset $S_{0,J}$ where nodes in J are in the subtree rooted at node 3. Additionally, it needs the key for the subset $S_{0,\{2,3\}}$. We know that, if the user at leaf 13 gets the derived seeds $L_{i,J}$ for every ancestor node i and the set J has a node (or a combination of nodes) directly attached to the path joining the leaf 13 to the root node 0, it can derive the key for any subset it belongs to.

Using this strategy, the user at leaf node 13 gets the seeds for $S_{0,J}$ for the following J: $\{2\}$, $\{3\}$, $\{2,3\}$, $\{5\}$, $\{6\}$, $\{5,6\}$, $\{14\}$, $\{15\}$, $\{14,15\}$. It gets the seeds for $S_{1,J}$ for the following J: $\{5\}$, $\{6\}$, $\{5,6\}$, $\{14\}$, $\{15\}$, $\{14,15\}$. It gets the seeds for $S_{4,J}$ for the following J: $\{14\}$, $\{15\}$, $\{14,15\}$. Hence, the user at leaf node 13 gets the following seeds:

derived from L_0:
$G_2(L_0)$, $G_4(L_0)$, $G_6(L_0)$,
$G_2(G_1(L_0))$, $G_4(G_1(L_0))$, $G_6(G_1(L_0))$,
$G_2(G_1(G_1(L_0)))$, $G_4(G_1(G_1(L_0)))$, $G_6(G_1(G_1(L_0)))$.
derived from L_1:
$G_2(L_1)$, $G_4(L_1)$, $G_6(L_1)$,
$G_2(G_1(L_1))$, $G_4(G_1(L_1))$, $G_6(G_1(L_1))$.
derived from L_4:
$G_2(L_4)$, $G_4(L_4)$, $G_6(L_4)$.

Next we compute the number of seeds that the user will have to store. The number of seeds derived from seed L_i of an ancestor node i at level ℓ, will be $2^{k-1}-1$ for each level below ℓ. Thus, the total number of derived seeds due to node i will be $(2^{k-1}-1)\ell$. Since there are ℓ_0 such ancestor nodes of the user at each level $1,\ldots,\ell_0$, the total number of seeds to be stored by the user will be

$$1+(2^{k-1}-1)\sum_{\ell=1}^{\ell_0}\ell = 1+\frac{\ell_0(\ell_0+1)}{2}(2^{k-1}-1). \tag{6.2}$$

The addition of 1 in the above expression is due to the key that is assigned to the set $\mathcal{N}$ of all users. Each user will be required to store this key to decrypt a message that is broadcast to all the users, i.e., when there are no revoked users. The factor $(2^{k-1}-1)$ in (6.2) can be reduced using a modified method of distributing secret information to the users. We describe how to do this in Section 6.5.

Full Resilience Against Colluding Users. Full resilience of a broadcast encryption scheme is ensured if the collusion of all revoked users does not result in the correct decryption of the encrypted message. This holds for the NNL-SD scheme and also holds for the current scheme in a similar manner. The cryptographic assumption that is required is that for any

$seed$, complete information about $G_0(seed), \ldots, G_{2^k-2}(seed)$ does not reveal any information about $seed$. Starting from this assumption, it is possible to argue in a manner similar to that done in Section 2.1 [NNL01, NNL02], that the scheme achieves full resilience.

6.2.2 Cover Finding Algorithm

Once the initiation is over and users have been given their secret information, the center can start broadcasting encrypted messages to the set of privileged users. If there is no revoked user, the only set for which the messages are encrypted is the set $\mathcal{N}$ of all users. Otherwise, for a given set of revoked users, the center finds the subset cover using the iterative algorithm outlined below. The subset cover contains subsets of the form $S_{i,J}$ where all nodes in J are siblings and hence are at the same level.

The algorithm runs on a list $\mathcal{L}$ of nodes in $\mathcal{T}^0$ that lie on the paths joining revoked leaf nodes to the root node. To start with, the list $\mathcal{L}$ consists of all revoked leaves from left to right in $\mathcal{T}^0$. In the course of the algorithm, $\mathcal{L}$ is appended with all nodes on the paths joining the revoked leaves to the root. This is done as follows. Once $\mathcal{L}$ is populated with the revoked leaves, the algorithm runs iteratively from left to right on $\mathcal{L}$. In iteration t, it considers the t^{th} node j from the left in $\mathcal{L}$. If j is not the root, the parent i of j is appended to $\mathcal{L}$, if it is not already present there. Hence $\mathcal{L}$ keeps growing on the right, with nodes at higher levels on the tree (up to the root) getting added to its right end. The root node eventually gets appended to $\mathcal{L}$. The algorithm terminates after working on the root node.

For each node j in $\mathcal{L}$, a summary of requisite information about the subtree $\mathcal{T}^j$ is maintained in $\mathcal{L}$ along with the node j. Each node j in $\mathcal{L}$ has an associated set $\mathsf{SDnodes}[j]$. The set $\mathsf{SDnodes}[j]$ contains roots of all subtrees that will be subtracted from $\mathcal{T}^j$, in case an SD subset is generated from node j. The cover finding algorithm ensures that all nodes in $\mathsf{SDnodes}[j]$ for any node j are at the same level in $\mathcal{T}^0$. For each leaf node j in the initial list $\mathcal{L}$, $\mathsf{SDnodes}[j] = \{j\}$. If a node j gives rise to a subset $S_{j,J}$ in the algorithm, then $J = \mathsf{SDnodes}[j]$. In the course of the algorithm, each such node j from which a subset $S_{j,\mathsf{SDnodes}[j]}$ should be generated, have to be identified. To that end, each node in $\mathcal{L}$ gets marked as "intermediate" or "covered" depending upon its position in the tree. Every iteration of the algorithm works on a particular node and based on the mark of that node and its siblings, subsets for the cover may or may not be generated. Let the t^{th} node from the left in $\mathcal{L}$ be denoted by $\mathcal{L}[t]$. The node $\mathcal{L}[t]$ is processed in the t^{th} iteration.

The Algorithm. Takes as input the set $\mathcal{R}$ of revoked users and outputs the subset cover $\mathcal{S}_c$.

1. Initialize list $\mathcal{L}$ with the r revoked leaf nodes of $\mathcal{T}^0$ in the same left-to-right order as in the tree. Mark each node $j \in \mathcal{L}$ as covered and set $\mathsf{SDnodes}[j] = \{j\}$.
2. Process the nodes in $\mathcal{L}$ iteratively from left to right as follows. At the t^{th} iteration:
   (a) If $\mathcal{L}[t]$ is the root node, go to step 3. If $\mathcal{L}[t+1]$ has the same parent as $\mathcal{L}[t]$, skip step 2-(b) below.
   (b) Let i be the parent of $\mathcal{L}[t]$. Append i to $\mathcal{L}$. Let $\{j_1, \ldots, j_c\}$ be the children of i in $\mathcal{L}$. The following mutually exclusive cases occur:
      i. Case when all nodes $j_1, \ldots, j_c$ are covered:
         A. If $c < k$, mark i as intermediate and set $\mathsf{SDnodes}[i] = \{j_1, \ldots, j_c\}$.
         B. For $c = k$, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.
      ii. Case when $c = 1$ and j_1 is intermediate:
         Mark i as intermediate and copy $\mathsf{SDnodes}[j_1]$ to $\mathsf{SDnodes}[i]$.
      iii. Case when $c > 1$ and there is at least one intermediate node in $\{j_1, \ldots, j_c\}$:
         For $j \in \{j_1, \ldots, j_c\}$ that is intermediate, add $S_{j,\mathsf{SDnodes}[j]}$ to the cover $\mathcal{S}_c$ and mark j as covered.
         A. If $c < k$, mark i as intermediate and set $\mathsf{SDnodes}[i] = \{j_1, \ldots, j_c\}$.
         B. For $c = k$, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.

   Continue step 2 for the next iteration with $t = t + 1$.
3. If the root node is marked as intermediate, add $S_{0,\mathsf{SDnodes}[0]}$ to the cover $\mathcal{S}_c$.

During the iterations of step 2 in the above algorithm, all the k child nodes of the root may eventually get marked as covered. In that case, the root node will already been marked as covered before the algorithm reaches step 3. It implies that all privileged users have been covered and hence no more SD subsets are added in step 3. The subset cover $\mathcal{S}_c$ output by the algorithm is a collection of subsets of the form $S_{i,\mathsf{SDnodes}[i]}$.

The performance of this generalised algorithm for the k-ary tree SD scheme in terms of speed, memory required and accessed is asymptotically same as that of the NNL-SD cover-finding algorithm. It may be noted here that for $k = 2$, this algorithm will check some redundant conditions that are not required for the NNL-SD scheme. Hence, an implementation of this algorithm for $k > 2$ will require more instructions and will be slower. Nonetheless, the output of this algorithm for $k = 2$ will be the NNL-SD subset cover and an implementation optimised for $k = 2$ will be precisely the same as that of the NNL-SD scheme. As the value of k increases, the height of the underlying tree would decrease for the same number of users. Hence, the number of nodes required to be stored and accessed by the algorithm should decrease. But we store $\mathsf{SDnodes}$ for each node in our algorithm. So, the memory usage and access for higher values of k would increase compared to the lower values. Our experience in executing the implementations show that for $n > 10^8, r > 0.4n$ and $k > 8$, the memory requirements go beyond that of a PC with 4 GBytes of RAM.

The understanding and hence the pseudo-code for our algorithm is cleaner compared to the one for the NNL-SD scheme. This is because our algorithm does not involve the arbitration of Steiner Trees that have been used in all papers related to the NNL-SD scheme in the existing literature to the best of our knowledge. Our algorithm views the underlying tree as an array and hence makes it closer to actual implementation and simple to visualize without the need for the understanding of any extraneous structure.

Algorithm Demonstration. To demonstrate the above algorithm, let us consider the revocation pattern $\mathcal{R} = \{14, 15, 22\}$ in the tree $\mathcal{T}^0$ with 27 users in Figure 6.4. The list $\mathcal{L}$ that is operated on iteratively in the algorithm, is eventually populated with the nodes $\{14, 15, 22, 4, 7, 1, 2, 0\}$. These are nodes that lie on the paths joining revoked users with the root node. Nodes $14, 15$ and 22 are initially $\mathsf{covered}$. The parent 4 of 14 and 15 is appended to the list and marked as $\mathsf{intermediate}$ with $\mathsf{SDnodes}[4] = \{14, 15\}$. Similarly, 7 is appended to the list and marked as $\mathsf{intermediate}$ with $\mathsf{SDnodes}[7] = \{22\}$. Next 1 is appended and marked as $\mathsf{intermediate}$ with $\mathsf{SDnodes}[4] = \{14, 15\}$ copied to $\mathsf{SDnodes}[1]$. Then 2 is appended and marked as $\mathsf{intermediate}$ with $\mathsf{SDnodes}[7] = \{22\}$ copied to $\mathsf{SDnodes}[2]$. Finally, 0 is appended to the list. Since 0 has two children in the list which are not $\mathsf{covered}$, the subsets $S_{1,\mathsf{SDnodes}[1]}$ and $S_{2,\mathsf{SDnodes}[2]}$ are added to the cover $\mathcal{S}_c$. Node 0 is marked as $\mathsf{intermediate}$ with $\mathsf{SDnodes}[0] = \{1, 2\}$. Finally, the subset $S_{0,\mathsf{SDnodes}[0]}$ is added to $\mathcal{S}_c$. Hence, for $\mathcal{R} = \{14, 15, 22\}$, $\mathcal{S}_c = \{S_{1,\{14,15\}}, S_{2,\{22\}}, S_{0,\{1,2\}}\}$.

Once the subset cover $\mathcal{S}_c$ has been constructed, the message M is encrypted using a random session key, which in turn is encrypted for each set in the cover. These encryptions of the session key are sent along with the encrypted message as the header part of the cipher-text. The number of sets in the cover, also called the header length, is the parameter determining the transmission overhead of the scheme.

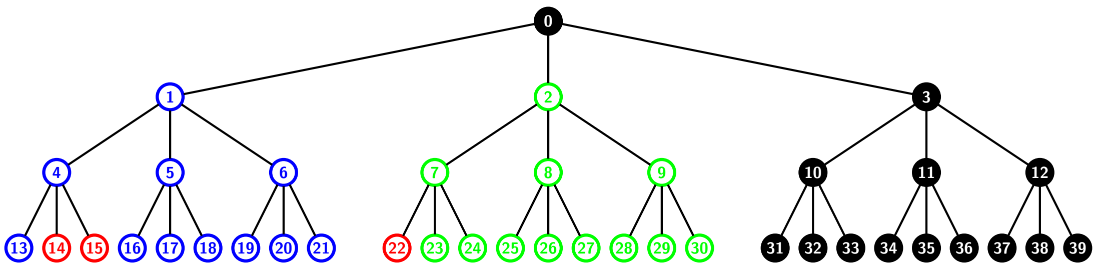


Figure 6.4: The subset cover $\mathcal{S}_c$ for $\mathcal{R} = \{14, 15, 22\}$ will contain the SD subsets $S_{1,\{14,15\}}$, $S_{2,\{22\}}$ and $S_{0,\{1,2\}}$.

Nodes that Generate a Subset. Nodes in $\mathcal{L}$ are the only nodes of $\mathcal{T}^0$ that are processed in the cover-finding algorithm. Hence, subsets in the cover are generated from nodes in $\mathcal{L}$ only. In other words, if $S_{j,\mathsf{SDnodes}[j]}$ is in the cover, then $j \in \mathcal{L}$ and $\mathsf{SDnodes}[j] \subset \mathcal{L}$. The following Lemma 21 identifies the properties of the node j and the set of nodes $\mathsf{SDnodes}[j]$.

Lemma 21. *Suppose $S_{j,\mathsf{SDnodes}[j]}$ is in the subset cover. Node j and the nodes in $\mathsf{SDnodes}[j]$ have the following properties:*
(1-a) Not all k children of j are in $\mathcal{L}$, and
(1-b) j is either the root or an internal node with a sibling in $\mathcal{L}$.
(2-a) If $\mathsf{SDnodes}[j] = \{v\}$, then v is either a leaf node or an internal node with all its children in $\mathcal{L}$.
(2-b) If $|\mathsf{SDnodes}[j]| > 1$, then all nodes of $\mathsf{SDnodes}[j]$ are siblings.
(2-c) For any node j, $|\mathsf{SDnodes}[j]| < k$.

Proof. First we show that j is either the root or an internal node with a sibling in $\mathcal{L}$. At step 3 of the cover finding algorithm described above, we see that SD subsets of the form $S_{0,\mathsf{SDnodes}[0]}$ may be generated. Hence, node j can be the root. If node j is not the root, then the only other way a subset may be generated is in step 2-b-iii of the algorithm. In this step, the algorithm considers a node i in $\mathcal{L}$ with a set $\{j_1, \ldots, j_c\}$ of its children in $\mathcal{L}$ where $c > 1$. Every $j \in \{j_1, \ldots, j_c\}$ that is marked as $\mathsf{intermediate}$ at that point, generates a subset. Hence, a non-root node j that generates a subset, must have a sibling in $\mathcal{L}$.

Next, we show that not all k children of j are in $\mathcal{L}$. The root node generates a subset in step 3 of the algorithm only if it is marked as $\mathsf{intermediate}$. A node j that generates a subset in step 2-b-iii, is marked as $\mathsf{intermediate}$ until that point. Hence j is not marked $\mathsf{covered}$ until the subset $S_{j,\mathsf{SDnodes}[j]}$ is generated from it. This implies that in previous iterations, when the children of j in $\mathcal{L}$ were being processed, it was not marked as $\mathsf{covered}$. A node may be marked as $\mathsf{covered}$ in either (1) step 2-b-i-B or 2-b-iii-B when all its children are in $\mathcal{L}$, or (2) step 2-b-iii after it has generated a subset. If $j = \mathcal{L}[t]$, then until the t^{th} iteration, j remains marked as $\mathsf{intermediate}$ if the number of children of j in $\mathcal{L}$ is smaller than k. Thus, not all k children of j are in $\mathcal{L}$.

In the cover finding algorithm, step 1 and the three mutually exclusive steps within step 2-b are the only places from where a set $\mathsf{SDnodes}[j]$ may arise. From step 1 of the algorithm, we see that for each leaf node j, $\mathsf{SDnodes}[j]$ is a singleton. From steps 2-b-i-B and 2-b-iii-B, we see that if all k children of an internal node j are in $\mathcal{L}$, then $\mathsf{SDnodes}[j]$ is a singleton. These are the only two ways in which $\mathsf{SDnodes}[j]$ for a node j can be a singleton.

A set $\mathsf{SDnodes}[j]$ with more than one node is created only in steps 2-b-i-A and 2-b-iii-A. Clearly, the nodes in $\mathsf{SDnodes}[j]$ have a common parent i and hence are siblings of each other. Hence, if $|\mathsf{SDnodes}[j]| > 1$, then all nodes in $\mathsf{SDnodes}[j]$ are siblings. It is also clear from steps 2-b-i-A and 2-b-iii-A that $\mathsf{SDnodes}[j]$ can have at most k nodes. Hence, $|\mathsf{SDnodes}[j]| < k$ for any j. $\square$

Correctness of the Algorithm. We prove that the algorithm described above, generates subsets that were assigned keys during initiation as has been described in Section 6.2.1. We also show that all privileged users are in some subset in $\mathcal{S}_c$ and no revoked user is included in any of the subsets.

Theorem 22. *A subset $S_{j,\mathsf{SDnodes}[j]}$ generated by the algorithm is such that*

1. $1 \leq |\mathsf{SDnodes}[j]| < k$,
2. *all nodes in* $\mathsf{SDnodes}[j]$ *are siblings of each other, and*
3. *j is their ancestor.*

The union of the subsets in $\mathcal{S}_c$ include all privileged leaves and no revoked leaves.

Proof. From Lemma 21 we know that all nodes in $\mathsf{SDnodes}[j]$ are siblings of each other and $1 \leq |\mathsf{SDnodes}[j]| < k$. It can be seen from steps 1 and 2-b of the algorithm that a node j_1 gets inserted into a set $\mathsf{SDnodes}[j]$ only if j is an ancestor of j_1.

A subset $S_{i,\mathsf{SDnodes}[i]}$ output by the algorithm, represents all leaf nodes in the induced subgraph

$$\mathcal{T}^i \setminus \cup_{j\in\mathsf{SDnodes}[i]}\mathcal{T}^j.$$

In other words, the subset $S_{i,\mathsf{SDnodes}[i]}$ has leaves in $\mathcal{T}^i$ that are not in the subtrees in $\mathsf{SDnodes}[i]$. It can be seen from steps 2-b-i and 2-b-iii that a $\mathsf{covered}$ node in $\mathcal{L}$ is always within some subtree in the set $\mathsf{SDnodes}$ of its parent and ancestors thereon. Hence, once marked $\mathsf{covered}$, a node is not in any set $S_{i,\mathsf{SDnodes}[i]}$ in $\mathcal{S}_c$ that is included thereafter. From steps 1 and 2-b-1 of the cover finding algorithm, we know that each revoked leaf j is in $\mathsf{SDnodes}[j]$ and hence in some subtree in $\mathsf{SDnodes}[i]$ for every ancestor i of j. This implies that a revoked leaf can not be in any subset in $\mathcal{S}_c$.

We next show that any privileged leaf is in a subset in $\mathcal{S}_c$. Let us consider the path joining a privileged leaf to the root in $\mathcal{T}^0$. Since the root node is in $\mathcal{L}$, hence there will be at least one node on this path that is in $\mathcal{L}$. Let j_1 be the node on this path that is in $\mathcal{L}$ and is nearest to the privileged leaf. All subsequent nodes above j_1 are in $\mathcal{L}$. Again, since the root node is on the path $(j_1, \ldots, 0)$, at least one of the nodes on this path generate a subset in $\mathcal{S}_c$. Let the node in $(j_1, \ldots, 0)$ that is nearest to the privileged leaf and generates a subset be j. Either $j = j_1$ or j is an ancestor of j_1 on the path $(j_1, \ldots, 0)$. The subset $S_{j,\mathsf{SDnodes}[j]}$ generated from node j has all leaves in $\mathcal{T}^j$ but not in any of the subtrees in $\mathsf{SDnodes}[j]$. The privileged leaf is in $\mathcal{T}^j$. We show that it is not in any of the subtrees in $\mathsf{SDnodes}[j]$. From steps 1 and 2-b in the algorithm, we know that $\mathsf{SDnodes}[j]$ has nodes that have been $\mathsf{covered}$. Nodes between the privileged leaf and before j_1 are not in $\mathcal{L}$ and hence cannot be $\mathsf{covered}$. Since j is the only node on the path $(j_1, \ldots, j)$ that generates a subset, all nodes on this path are $\mathsf{intermediate}$ until the subset $S_{j,\mathsf{SDnodes}[j]}$ is generated from j. Consequently, $\mathsf{SDnodes}[j]$ does not have any of the nodes on the path joining the privileged leaf and j. Hence, the privileged leaf is not in any of the subtrees rooted at nodes in $\cup_{j'\in\mathsf{SDnodes}[j]}\mathcal{T}^{j'}$. Hence, the privileged user is in the subset $S_{j,\mathsf{SDnodes}[j]}$. □

Relation to the NNL-SD Scheme. The case $k = 2$ of the above scheme is exactly the NNL-SD scheme described in Chapter 2 [NNL01, NNL02]. So, the new scheme is a generalization of the NNL-SD scheme and subsumes it. One important advantage of the new scheme is a uniform description of the cover generation algorithm irrespective of the value of k. We note that this description is simpler than the description for $k = 2$ given in [NNL01, NNL02] which is phrased in terms of Steiner trees. It turns out that the more elementary description of the cover generation algorithm is cleaner which leads to an easier implementation.

Relation to the Ternary Tree Scheme in [FKTS08]. For $k = 3$, the collection $\mathcal{S}$ that we consider is the same as that in [FKTS08]. However, the method for assigning keys to these subsets is different. The work [FKTS08] uses a hash chain method and mentions that it does not extend to k-ary trees for $k \geq 4$. On the other hand, our method of distributing secret keys to the users is general and works for all k. In Section 6.5, we describe a modified method of distributing secret keys which further lowers the user storage requirement.

6.2.3 Traitor Tracing

We would like to recollect here the definition of the bifurcation property [NNL01, NNL02] that was described in Chapter 2. The bifurcation property states that *given any subset that is in the collection $\mathcal{S}$ and hence has been assigned a key, it is possible to partition the set into two (or a constant number of) almost equal subsets from $\mathcal{S}$*. The *bifurcation value* is defined to be the ratio of the size of the largest subset to that of the set itself.

For the k-ary tree SD scheme, a subset $S_{i,J}$ in its collection $\mathcal{S}$ is such that all nodes in the set J are siblings and are in the subtree $\mathcal{T}^i$. If the parent of the nodes in J is i, then the subset $S_{i,J}$ is split into equal sized subsets $\mathcal{T}^j$ where j is a child of i and $j \notin J$. Thus, each child subtree of i whose root is not in J forms a subset in the split. All the subsets formed by splitting $S_{i,J}$ will be of equal size and hence the bifurcation value in this case is $1/(k - |J|)$. The worst case (maximum bifurcation value) occurs when there are two child subtrees of i that are not in J (and are hence privileged). The maximum bifurcation value is $1/2$. If the parent of nodes in J is a descendant of i, then the subset $S_{i,J}$ will be split into exactly k subsets each formed from a child subtree of node i. There will be one subset formed from the child subtree of i that contains the nodes in J. This subset will be smaller than the rest of the $k - 1$ equal-sized subsets. The bifurcation value in this case will be $1/k$. All subsets in the collection $\mathcal{S}$ of the layered k-ary tree SD scheme belong to the collection of subsets that are assigned keys in the k-ary tree SD scheme. Hence, the bifurcation property also holds for those subsets. Thus, a traitor tracing mechanism can be devised for the scheme introduced in this work in a manner similar to the one described in [NNL01, NNL02].

The number of queries required by the traitor tracing algorithm depends on the bifurcation value. At every step of the traitor tracing algorithm, a subset S of users that contains a traitor is divided into subsets $S_1, \ldots, S_t$ using the bifurcation property as mentioned above. Each subset S_t is tested for containment of a traitor. The ratio $|S_t|/|S|$ is at most the bifurcation value. The size of the remaining subset from which the traitors have to be traced, reduces with the bifurcation value. Hence, the traitor tracing algorithm will be more efficient. The bifurcation value of the NNL-SD scheme is $2/3$. The bifurcation value of the k-ary tree SD scheme is $1/2$ for $k \geq 3$. Hence, the traitor tracing mechanism for the k-ary tree SD scheme will be more efficient than the NNL-SD scheme.

6.3 Header Length Analysis

When there are no revoked users, the header length is 1. Henceforth, we assume the set $\mathcal{R}$ of revoked users to be non-empty.

Theorem 23. *Fix $k \geq 2$, $n \geq 1$ and $1 \leq r \leq n$. Then the maximum header length that can be achieved is* $\min(2r-1, n-r, n/k)$.

Note: When r is small, the bound $2r-1$ applies. For $k=2$, the upper bound of $2r-1$ was given in [NNL01, NNL02]. The more general form of the bound for binary trees was mentioned in Section 4.3.4 [BS13]. For $k=3$, it has been shown that the scheme in [FKTS08] has an upper bound of $\min(2r-1, n/3)$.

Proof. The bound $n-r$ on the header length arises since each of the $n-r$ privileged users can be covered by singleton subsets in the header.

The bound n/k is obtained as follows. Suppose the header consists of h subsets. Write $h = h_1 + \cdots + h_{k-1} + h_k$ where for $1 \leq i \leq k-1$, h_i is the number of subsets in the header having exactly i privileged users and h_k is the number of subsets in the header having at least k privileged users.

Suppose S is a subset counted in h_i for some i in $[1, k-1]$. From the cover finding algorithm, it necessarily follows that the leaf nodes in S are siblings and the other siblings of the nodes in S are revoked. So, to each subset S counted in h_i, there corresponds a total of k users (i users in S and the other $k - |S|$ revoked siblings of the users in S). As a result, the total number of users accounted for by $h_1, \ldots, h_{k-1}$ is $k(h_1 + \cdots + h_{k-1})$. Since each subset counted in h_k has at least k users, the total number of users n is at least $k(h_1 + \cdots + h_{k-1}) + kh_k = k(h_1 + \cdots + h_{k-1} + h_k) = kh$. From this it follows that $h \leq n/k$.

Now, we turn to the bound $2r - 1$. The subtree $\mathcal{T}^j$ may be written as a union of all its child subtrees. Hence,

$$\mathcal{T}^j = \bigcup_{j' \in \{kj+1,\ldots,kj+k\}} \mathcal{T}^{j'}.$$

Thus, the node j can be replaced by $\{kj + 1, \ldots, kj + k\}$. For a subset $S_{i,J} \in \mathcal{S}_c$, if $j \in J$ such that all k children of j are in $\mathcal{L}$, we replace j with $\{kj + 1, \ldots, kj + k\}$ in J to get J'.

$$J' = (J \setminus \{j\}) \cup \{kj + 1, \ldots, kj + k\}.$$

We keep replacing nodes in J' having k children in $\mathcal{L}$ by their children until all nodes in J' have less than k children in $\mathcal{L}$. Some nodes in J' may have no children (leaf nodes) in $\mathcal{L}$. These are revoked leaf nodes of $\mathcal{T}^0$ that were inserted in $\mathcal{L}$ in step 1 of the algorithm. The new representation $\mathcal{S}'_c$ of the subset cover $\mathcal{S}_c$ will have

$$\mathcal{S}'_c = (\mathcal{S}_c \setminus S_{i,J}) \cup S_{i,J'}.$$

However, the privileged users in $S_{i,J'}$ are exactly the privileged users in $S_{i,J}$. We do this for all subsets in $\mathcal{S}_c$ to complete the new representation $\mathcal{S}'_c$ of $\mathcal{S}_c$.

We first show that all internal nodes in J' generate a subset each. From Lemma 21 we know that for $S_{i,J} \in \mathcal{S}_c$, all nodes in J are in $\mathcal{L}$ and are siblings. During the transformation, a node $j \in J$ is replaced by k nodes which are also in $\mathcal{L}$ and are siblings of each other. Hence, each node $j \in J'$ has a sibling in $\mathcal{L}$. Since j is in $\mathcal{L}$ and has less than k children, hence from step 2-b of the algorithm we know that it is marked as intermediate unless a subset is generated from it. From step 2-b-iii we know that an intermediate node having a sibling in $\mathcal{L}$, generates a subset. Hence, a node in J' is either a revoked leaf node or an internal node that generates a subset.

We construct a graph Υ such that for each subset $S_{i,J'}$ in $\mathcal{S}'_c$, node i and all nodes in J' are in Υ. For every subset $S_{i,J'}$ in $\mathcal{S}'_c$, there is an edge (i, j) for each $j \in J'$ in Υ. A node $j \in J'$ that is an internal node in $\mathcal{T}^0$ generates a subset and hence is an internal node in Υ.

A leaf node in J' is a leaf node in Υ.

We first show that Υ is a forest with one or more component trees. Once a subset $S_{i,J}$ is included in $\mathcal{S}_c$ at step 2-b-iii of the algorithm, i is marked as $\mathsf{covered}$ and $\mathsf{SDnodes}[i] = i$. Hence for an ancestor i_1 of i, any descendant of i is not in $\mathsf{SDnodes}[i_1]$. If S_{i_1,J_1} is included in the cover, it may have i in J_1. Since i generates a subset in the cover, the transformation of J_1 to J_1' will not reach any descendant of i. Hence, J_1' will not have any descendant of i. Consequently, there will be an edge (i, j) in Υ for each $j \in J'$ but there will be no other edge between j and any other ancestor of j. Since this is true for any node $j \in \Upsilon$, it is an acyclic graph. Additionally, the cover might not have a subset generated from the root. Thus, components of Υ may not be connected. Hence, it is a forest with one or more component trees.

The nodes in Υ are either internal nodes in $\mathcal{T}^0$ that generate a subset each, or revoked leaf nodes. Hence, the number of internal nodes in Υ is the number of subsets in the subset cover. For a subset $S_{i,J} \in \mathcal{S}_c$ if $|J| = 1$, then by Lemma 21 the node in J is either an internal node with all its k children in $\mathcal{L}$ or a leaf node. In the corresponding $S_{i,J'} \in \mathcal{S}_c'$, an internal node in $j \in J$ with all its k children in $\mathcal{L}$ has been replaced by its child nodes. Hence, if a subset $S_{i,J'} \in \mathcal{S}_c'$ is such that $|J'| = 1$, then the set J' has a single leaf node. Hence, if an internal node i in Υ has only one child, it will be a leaf. Any other internal node in Υ will have at least two children in Υ.

The transformation ensures that each of the r revoked leaves of $\mathcal{T}^0$ is a leaf in Υ. Hence, there can be at most r internal nodes that have leaf nodes amongst their children in Υ. The graph Υ is reduced to Υ' by merging an internal node having a single leaf child, with its child. Consequently, Υ' is a forest with at most r leaves and internal nodes in Υ' have at least two children each. Hence, there are at most $r - 1$ internal nodes in Υ'. Thus, the maximum number of internal nodes in Υ is $r + r - 1 = 2r - 1$. Hence, there can be at most $2r - 1$ subsets in the subset cover. $\square$

This upper bound of $2r - 1$ on the maximum header length can be achieved for a given r and any fixed value of k provided n can be made as large as required. For $k = 2$, this bound has been shown to be tight in Section 4.3.4 [BS13]. For $k = 3$, let us consider the tree $\mathcal{T}^0$ of Figure 6.5 where the set of revoked users is $\mathcal{R} = \{13, 16, 22, 25\}$. The nodes in $\mathcal{P}$ are $\{13, 16, 22, 25, 1, 2, 0\}$. The subsets in the cover are $S_{4,\{13\}}$, $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$. It can be seen from this figure that for higher arities (> 3), additional subtrees are added to all the internal nodes. Assuming that the revoked users remain the same as marked in the figure, we notice the following. The subset $S_{4,\{13\}}$ gets additional users that are attached to the node 4. Similarly, each of the subsets $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$ get the additional users attached to the nodes $5, 7, 8, 1, 2$ and 0 respectively. Hence, this upper bound is tight for any arity k in general provided n can be chosen to be large.

For a given k and $n = k^{\ell_0}$ this maximum header length of $2r - 1$ is achieved for r given by Lemma 24.

Lemma 24. *For a given k and $n = k^{\ell_0}$, the maximum header length of $2r - 1$ is achieved for $r = 2^{\ell_0 - 1}$.*

Proof. We prove this by induction on ℓ_0. For $\ell_0 = 1$, $n = k$ and $r = 1$. Let j_1 be the only revoked leaf. The only subset in the cover is $S_{0,\{j_1\}}$ and hence the header length is $2r-1 = 1$. We assume that the maximum header length that can be achieved for $2^{\ell_0 - 2}$ revoked users in a full tree of arity k and with $k^{\ell_0 - 1}$ users is $2^{\ell_0 - 1} - 1$. Let us consider a tree with $n = k^{\ell_0}$ users and $r = 2^{\ell_0 - 1}$ revoked users such that two of the subtrees (out of k) of the root node, which are rooted at nodes j_1 and j_2, have $r/2 = 2^{\ell_0 - 2}$ revoked users in each and the rest of the subtrees of the root node do not have any revoked user in them. Since each of these two subtrees have $k^{\ell_0 - 1}$ users in each, hence by assumption they give rise to $2^{\ell_0 - 1} - 1$ subsets each in the cover. Additionally, there will be a subset $S_{0,\{j_1,j_2\}}$ in the cover. Hence, the total number of subsets generated by this construction for $r = 2^{\ell_0 - 1}$ is $2 \times (2^{\ell_0 - 1} - 1) + 1 = 2r - 1$.

We need to show that these subsets do not combine to reduce the header length. Two SD subsets $S_{i_1,\mathsf{SDnodes}[i_1]}$ and $S_{i_2,\mathsf{SDnodes}[i_2]}$ can be combined into one SD subset if (1) $\mathsf{SDnodes}[i_1] = \{i_2\}$ or $\mathsf{SDnodes}[i_2] = \{i_1\}$ or (2) if nodes in $\mathsf{SDnodes}[i_1]$ are siblings of nodes in $\mathsf{SDnodes}[i_2]$. Any two SD subsets $S_{i_1,\mathsf{SDnodes}[i_1]}$ and $S_{i_2,\mathsf{SDnodes}[i_2]}$ from the subtrees rooted at j_1 and j_2 respectively cannot satisfy either of the above two conditions. For the same reason, the subset $S_{0,\{j_1,j_2\}}$ cannot be combined with any of the SD subsets in these two subtrees. Hence, none of these subsets combine to reduce the total number of subsets $2r - 1$ in the subset cover. Hence, for $r = 2^{\ell_0 - 1}$, the maximum header length of $2r - 1$ is achieved for a fixed k and $n = k^{\ell_0}$. □

Effect of k on the Header Length. In the Subset-Cover framework, the subsets in the collection $\mathcal{S}$ are used to cover the privileged users. It seems intuitive that if the number of

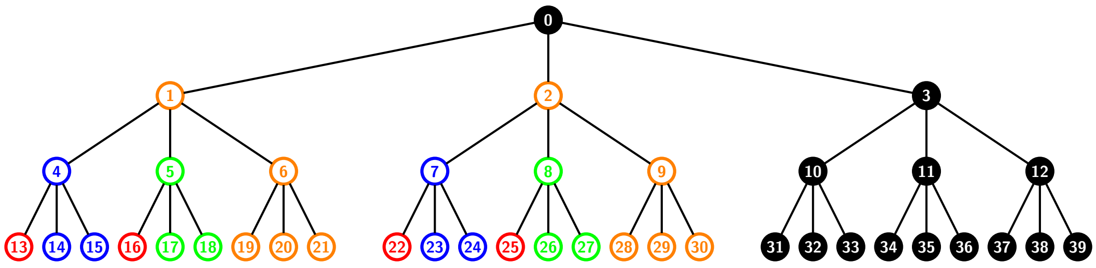


Figure 6.5: Example showing that the upper bound of $2r-1$ on the header length is tight for $k=3$. The subset cover for $\mathcal{R}=\{13,16,22,25\}$ in the tree $\mathcal{T}^0$ with $k=3$, will contain the SD subsets $S_{4,\{13\}}$, $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$.

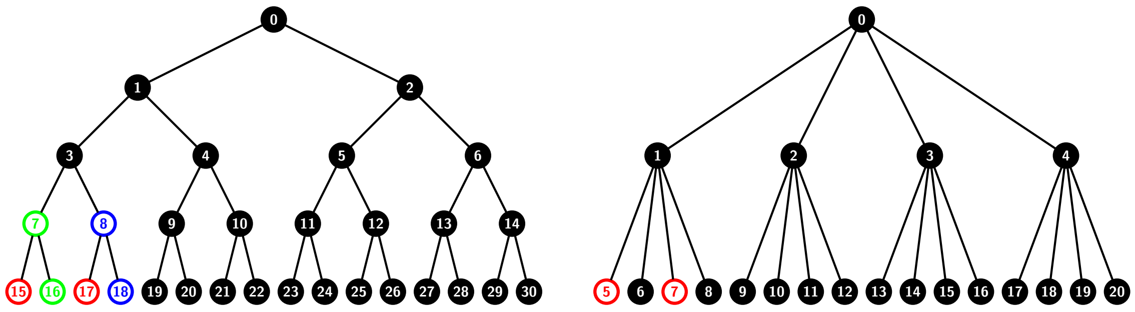


Figure 6.6: Example where the header length of 4-ary is better than 2-ary: For $n=16$ users, the header length for $\mathcal{R}=\{15,17\}$ in the 2-ary tree is more for $k=2$ than for $k=4$. For $k=2$, the subset cover $\mathcal{S}_c=\{S_{7,\{15\}},S_{8,\{17\}},S_{0,\{3\}}\}$. For $k=4$, the subset cover $\mathcal{S}_c=\{S_{0,\{5,7\}}\}$.

subsets in the collection increases, the number of subsets required to form a cover would decrease. As mentioned earlier, for $n=16$, the number of subsets in the collection for $k=4$ is more than that for $k=2$. More subsets in the collection should reduce the header length. In Figure 6.6 we see that the header length for the specific revocation pattern is smaller for the 4-ary tree as compared to the 2-ary one. However, this may not always happen. In Figure 6.7, the revocation pattern is such that the header length is smaller for the 2-ary tree as compared to the 4-ary one. Hence, we see that the header length may not always reduce by increasing the arity. By Theorem 23, the maximum header length is independent of the underlying arity k. The expected header lengths for different values of k will give a better idea about the effect of arity on the overall communication overhead.

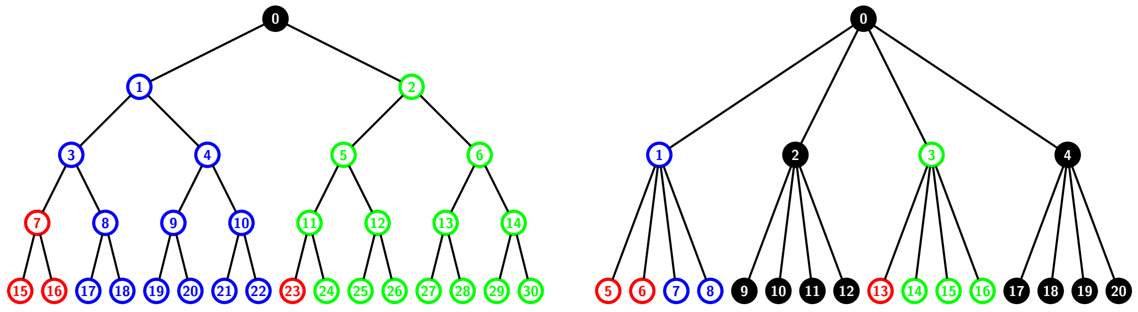


Figure 6.7: Example where the header length of 2-ary is better than 4-ary: For $n = 16$ users, the header length for $\mathcal{R} = \{15, 16, 23\}$ in the 2-ary tree is more for $k = 4$ than for $k = 2$. For $k = 2$, the subset cover $\mathcal{S}_c = \{S_{1,\{7\}}, S_{2,\{23\}}\}$. For $k = 4$, the subset cover $\mathcal{S}_c = \{S_{1,\{5,6\}}, S_{3,\{13\}}, S_{0,\{1,3\}}\}$.

6.3.1 Expected Header Length

Fix k, n and r. Consider the same random experiment as described in Section 4.4, where one has to randomly choose r out of the n users uniformly at random one-by-one and without replacement. Consider the selected set of r users to be revoked. The expected header length under this random experiment is given by the following result.

Theorem 25. *Fix $k \geq 2$, $n = k^{\ell_0} \geq 1$ and $1 \leq r \leq n$. The expected header length in the k-ary tree SD scheme is given by*

$$\sum_{c=1}^{k-1} \binom{k}{c} \left(\gamma_{\ell_0,c} + \sum_{\ell=1}^{\ell_0-1} (k^{\ell_0-\ell}) \gamma_{\ell,c} \right)$$

where

$$\begin{aligned} \gamma_{\ell,c} &= \eta_r(n, k^{\ell} - ck^{\ell-1}) - \eta_r(n, k^{\ell+1} - ck^{\ell-1}) \\ &\quad - \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell} - (c-t)k^{\ell-1}) \\ &\quad + \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell+1} - (c-t)k^{\ell-1}) \qquad \text{for } 1 \leq \ell \leq \ell_0 - 1 \end{aligned} \tag{6.3}$$

and

$$\gamma_{\ell_0,c} = \eta_r(n, k^{\ell_0} - ck^{\ell_0-1}) - \sum_{t=1}^{c}(-1)^{t+1}\binom{c}{t}\eta_r(n, k^{\ell_0} - (c-t)k^{\ell_0-1}). \tag{6.4}$$

Proof. Let $X_{n,r}$ be the random variable taking the value of the header length. The expected header length also depends on k and so strictly speaking we should be using the notation $X_{n,r,k}$ to denote this dependence. We have chosen the simpler notation $X_{n,r}$ since for a particular implementation, k will be fixed and so clear from the context.

Each subset in the cover $\mathcal{S}_c$ is rooted at some internal node i of $\mathcal{T}^0$. Each such node in the tree contributes at most one subset to the cover. Let $X^i_{n,r}$ be the random variable associated with node i, that denotes its contribution to the header length. Hence, $X^i_{n,r} \in \{0, 1\}$. The event $X^i_{n,r} = 1$ occurs when there is a subset $S_{i,J}$ in the cover and $X^i_{n,r} = 0$ otherwise. It can be seen from the cover finding algorithm described in Section 6.2.2 that the leaf nodes of $\mathcal{T}^0$ do not generate SD subsets. Hence, SD subsets are generated only from the internal nodes in $\mathcal{T}^0$. It follows that

$$X_{n,r} = X^0_{n,r} + X^1_{n,r} + \cdots + X^{i_f}_{n,r}, \tag{6.5}$$

where $i_f = \frac{nk-1}{k-1} - n - 1$ is the last internal node as per the labeling of the tree $\mathcal{T}^0$. By linearity of expectation,

$$E[X_{n,r}] = E[X^0_{n,r}] + E[X^1_{n,r}] + \cdots + E[X^{i_f}_{n,r}]. \tag{6.6}$$

To find the expected header length, one needs to compute the values of $E[X^i_{n,r}]$ for each $i \in \{0, 1, \ldots, i_f\}$. Since $X^i_{n,r} \in \{0, 1\}$, the random variable $X^i_{n,r}$ follows *Bernoulli distribution*. Hence, $E[X^i_{n,r} = 1] = \Pr[X^i_{n,r} = 1]$. Thus, to compute the expected header length $E[X_{n,r}]$ for a random revocation pattern, the probability $\Pr[X^i_{n,r} = 1]$ that a node i generates a subset has to be computed.

Let I be the set of all child nodes of i and let p be the parent node of i in $\mathcal{T}^0$. Hence, $I = \{ki + 1, \ldots, ki + k\}$ and $p = \lfloor (i-1)/k \rfloor$. Let $J \subset I$ be a non-empty subset of child nodes of i. The event $X^i_{n,r} = 1$ occurs when a subset $S_{i,J}$ is in the cover. For a subset $S_{i,J}$ to be in the cover, the following have to be true with respect to node i in $\mathcal{T}^0$:

- At least one but not all child subtrees of i would contain some revoked nodes; and

- If $i \neq 0$ (for a non-root internal node), at least one sibling subtree of i would contain a revoked node.

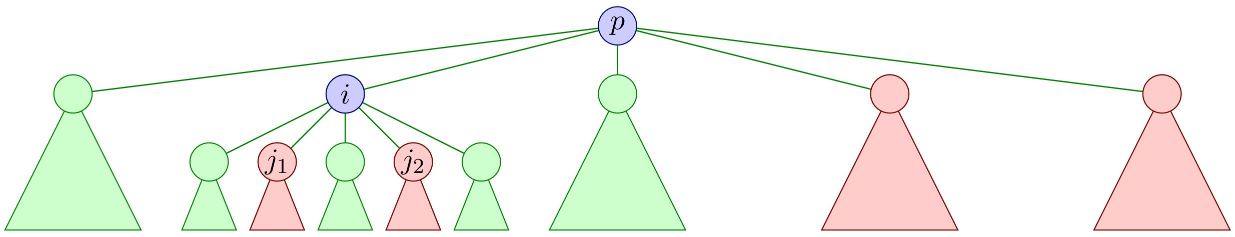


Figure 6.8: Example of a scenario for the event $X^i_{n,r} = 1$ where a set $S_{i,J}$ occurs in the subset cover. Child subtrees of i that are red in color, contain at least one revoked user in each. Hence their root nodes j_1 and j_2 are in the set J. The other child subtrees of i (green in color) do not contain any revoked user. Hence they are in the set $I \setminus J$. Sibling subtrees of i that are red in color, contain at least one revoked user in each. Hence, at least one child subtree of i (not all) has revoked users and at least one sibling subtree of i has revoked users. Consequently, a subset rooted at node i is generated.

In order to formulate these conditions when a subset $S_{i,J}$ rooted at node i occurs in the subset cover, we define some additional events with respect to the node i and a non-empty subset of its child nodes $J \subset I$. The event R^i_j (where $j \in J$) is defined to occur when for a revocation pattern, the subtree rooted at node j contains at least one revoked user. Hence, event $\overline{R^i_j}$ occurs when the subtree rooted at node j does not contain any revoked user. Let $R^i_J = \bigwedge_{j \in J} R^i_j$ be the event where each subtree rooted at nodes in J has at least one revoked user. Hence, $\overline{R^i_J}$ is the event where none of the subtrees rooted at nodes in J have any revoked user. For $i \neq 0$, event R^i_{sb} is defined to occur when the union of all sibling subtrees of i (children of p other than i) contains at least one revoked node. Hence, a subset $S_{i,J}$ is in the subset cover when the following condition is true

$$\left(\bigwedge_{j \in J} R^i_j \right) \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}} = R^i_J \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}}.$$

Since a subset $S_{i,J}$ may occur for any non-empty set $J \subset I$, hence the event $X^i_{n,r} = 1$ can also be written as

$$\bigvee_{J \subset I, J \neq \phi} \left(R^i_J \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}} \right).$$

These events $(R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}})$ are mutually exclusive and they exhaustively form the event

$X^i_{n,r} = 1$. Hence, we can write

$$\Pr[X^i_{n,r} = 1] = \sum_{J \subset I; J \neq \phi} \Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}}]. \tag{6.7}$$

It can be similarly seen that for the root node with children in $\{1, \ldots, k\}$

$$\Pr[X^0_{n,r}] = \sum_{J \subset \{1,\ldots,k\}; J \neq \phi} \Pr[R^i_J \wedge \overline{R^i_{I \setminus J}}]. \tag{6.8}$$

The probability $\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}}]$ can be written as

$$\begin{aligned}
\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}}] &= \Pr[R^i_{sb} \wedge R^i_J | \overline{R^i_{I \setminus J}}] \times \Pr[\overline{R^i_{I \setminus J}}] \\
&= (1 - \Pr[\overline{R^i_{sb} \wedge R^i_J} | \overline{R^i_{I \setminus J}}]) \times \Pr[\overline{R^i_{I \setminus J}}] \\
&= (1 - \Pr[\overline{R^i_{sb}} | \overline{R^i_{I \setminus J}}] - \Pr[\overline{R^i_J} | \overline{R^i_{I \setminus J}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} | \overline{R^i_{I \setminus J}}]) \\
&\quad \times \Pr[\overline{R^i_{I \setminus J}}] \\
&= \Pr[\overline{R^i_{I \setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I \setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I \setminus J}}] \\
&\quad + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} \wedge \overline{R^i_{I \setminus J}}])
\end{aligned} \tag{6.9}$$

and for the root node, $\Pr[R^i_J \wedge \overline{R^i_{I \setminus J}}]$ can be written as

$$\Pr[R^i_J \wedge \overline{R^i_{I \setminus J}}] \quad = \Pr[\overline{R^i_{I \setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I \setminus J}}]. \tag{6.10}$$

For the computation of (6.9) and (6.10) above, we observe that the event $\overline{R^i_J}$ occurs when at least one $j \in J$ does not contain any revoked user. Now let us consider two events A and B in general such that the event A occurs when all the sub-events $A_1, \ldots, A_c$ occur. Hence, $A = A_1 \wedge \ldots \wedge A_c$.

$$\begin{aligned}
\Pr[\overline{A} \wedge \overline{B}] &= \Pr[\overline{A_1 \wedge \ldots \wedge A_c} | \overline{B}] \Pr[\overline{B}] \\
&= \Pr[\overline{A_1} \vee \ldots \vee \overline{A_c} | \overline{B}] \Pr[\overline{B}] \\
&= \Pr[\overline{A_1} \wedge \overline{B}] + \ldots + \Pr[\overline{A_c} \wedge \overline{B}] - \Pr[\overline{A_1} \wedge \overline{A_2} \wedge \overline{B}] \\
&\quad - \ldots - \Pr[\overline{A_{c-1}} \wedge \overline{A_c} \wedge \overline{B}] \\
&\quad + \ldots + (-1)^{c+1} \left(\Pr[\overline{A_1} \wedge \ldots \wedge \overline{A_c} \wedge \overline{B}]\right).
\end{aligned} \tag{6.11}$$

Now, when $A = R^i_J$ where $A_t = R^i_{j_t}$ and $B = R^i_{I\setminus J}$, we get

$$\begin{aligned}
\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] &= \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}}] + \ldots + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}}] - \ldots - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \ldots + (-1)^{c+1}\left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}]\right). \qquad (6.12)
\end{aligned}$$

Similarly, for the expression $\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}]$ in (6.9) we get the following using the result in (6.11)

$$\begin{aligned}
\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] &= \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad + \ldots + (-1)^{c+1}\left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}]\right). \qquad (6.13)
\end{aligned}$$

From (6.9), (6.12) and (6.13) we get

$$\begin{aligned}
\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] &= \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] \\
&= \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}}] - \ldots - \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}}] + \ldots + \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \ldots + (-1)^{c+2}\left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}]\right) \\
&\quad + \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad + \ldots + (-1)^{c+1}\left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}]\right). \qquad (6.14)
\end{aligned}$$

To find these probabilities, we define $\eta_r(n, x)$ to be the probability that r out of n elements are chosen uniformly at random without replacement but x out of these n elements never get chosen. In other words,

$$\eta_r(n, x) = \frac{\binom{n-x}{r}}{\binom{n}{r}}. \qquad (6.15)$$

Let the number of users in the subtree rooted at node i be λ^i. Hence, the number of users in all the sibling subtrees of i is $\lambda^p - \lambda^i$. Hence, the sum of the number of users in the subtrees rooted at nodes in $I \setminus J$ is $\lambda^i - \sum_{j\in J}\lambda^j$. From (6.14) and (6.15) we get

$$\begin{aligned}
\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] &= \eta_r(n, \lambda^i - \sum_{j\in J}\lambda^j) - \eta_r(n, \lambda^p - \sum_{j\in J}\lambda^j)\\
&\quad -\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1\}}\lambda^j) - \ldots - \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_c\}}\lambda^j)\\
&\quad +\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1,j_2\}}\lambda^j) + \ldots + \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_{c-1},j_c\}}\lambda^j)\\
&\quad - \ldots + (-1)^{c+2}\eta_r(n, \lambda_i)\\
&\quad +\eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_1\}}\lambda^j) + \ldots + \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_c\}}\lambda^j)\\
&\quad -\eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_1,j_2\}}\lambda^j) - \ldots - \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_{c-1},j_c\}}\lambda^j)\\
&\quad + \ldots + (-1)^{c+1}\eta_r(n, \lambda_p).
\end{aligned} \tag{6.16}$$

Similarly for the root node, from (6.10), (6.12) and (6.15) we get

$$\begin{aligned}
\Pr[R^i_J \wedge \overline{R^i_{I\setminus J}}] &= \eta_r(n, \lambda^i - \sum_{j\in J}\lambda^j)\\
&\quad -\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1\}}\lambda^j) - \ldots - \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_c\}}\lambda^j)\\
&\quad +\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1,j_2\}}\lambda^j) + \ldots + \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_{c-1},j_c\}}\lambda^j)\\
&\quad - \ldots + (-1)^{c+2}\eta_r(n, \lambda_i).
\end{aligned} \tag{6.17}$$

From (6.6), (6.7), (6.8), (6.16) and (6.17) we get the algorithm for computing $E[X_{n,r}]$. In Section 6.4, we discuss that this scheme may be further extended for an arbitrary number of users (instead of a power of k). In such a case, the underlying tree may be assumed to be a complete tree (with users at the last two levels of the tree) instead of a full tree. All the above expressions for computing probabilities are also valid for complete trees where the number of users may not be a power of k.

However, for the k-ary tree SD scheme as it has been described here, the number of users is assumed to be a power of k. Hence, $n = k^{\ell_0}$. Let $c = |J|$ be the cardinality of the set J of some child nodes of i. Hence, $0 < c < k$. There are ℓ_0 levels with internal nodes in the tree $\mathcal{T}^0$. All subtrees rooted at level ℓ have the same number of leaf nodes k^ℓ. For a non-root i at level ℓ and a corresponding set of child nodes J ($|J| = c$), the contribution to the header can be computed from (6.16) as

$$\begin{aligned}\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] &= \eta_r(n, k^\ell - ck^{\ell-1}) - \eta_r(n, k^{\ell+1} - ck^{\ell-1}) \\ &\quad - \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^\ell - (c-t)k^{\ell-1}) \\ &\quad + \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell+1} - (c-t)k^{\ell-1}). \end{aligned} \tag{6.18}$$

Let $\gamma_{\ell,c}$ be the value of this probability given by (6.18) for a non-root node i at level ℓ ($1 \le \ell \le \ell_0 - 1$) and $|J| = c$. Since there are $k^{\ell_0-\ell}$ nodes at level ℓ in the tree $\mathcal{T}^0$, hence the contribution of all the nodes at level ℓ and a fixed value of c is $(k^{\ell_0-\ell})\gamma_{\ell,c}$. For the root node 0 at level ℓ_0 and a corresponding set of child nodes $J \subset \{1, \ldots, k\}$ ($|J| = c$), the contribution to the header can be computed from (6.17) as

$$\begin{aligned}\Pr[R^0_J \wedge \overline{R^0_{I\setminus J}}] &= \eta_r(n, k^{\ell_0} - ck^{\ell_0-1}) \\ &\quad - \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell_0} - (c-t)k^{\ell_0-1}). \end{aligned} \tag{6.19}$$

The value of this probability given by (6.19) for the root node at level ℓ_0 and $|J| = c$ is denoted by $\gamma_{\ell_0,c}$. Hence, the expected header length for a given k, ℓ_0 and r is given by

$$E[X_{n,r}] = \sum_{c=1}^{k-1} \binom{k}{c} \left(\gamma_{\ell_0,c} + \sum_{\ell=1}^{\ell_0-1} (k^{\ell_0-\ell}) \gamma_{\ell,c} \right). \tag{6.20}$$

$\square$

Algorithm to Compute the Expected Header Length. The result in Theorem 25 can be converted into an algorithm to compute the expected header length. The algorithm takes as input the values of k, n and r. Here $n = k^{\ell_0}$ for some $\ell_0 \ge 1$ and $1 \le r \le n$. The algorithm computes the values of $\gamma_{\ell,c}$ for each level ℓ in the tree $\mathcal{T}^0$ and each value of c. For fixed values of ℓ and c, computing $\gamma_{\ell,c}$ requires computing a fixed number of $\eta_r(\cdot,\cdot)$. One

Table 6.1: Table showing the results of the algorithm for computing the expected header length. For each k, we have chosen n to be k^a and k^b which are the two closest powers of k to 10^8. The column MHL_k/r gives the ratio of the mean header length MHL_k for k-ary tree to the number r of revoked users.

k	n	r	MHL_k/r	k	n	r	MHL_k/r
2	$(2^{26}, 2^{27})$	10^5	$(1.24, 1.24)$	3	$(3^{16}, 3^{17})$	10^5	$(1.48, 1.48)$
		10^6	$(1.23, 1.24)$			10^6	$(1.43, 1.46)$
		10^7	$(1.23, 1.24)$			10^7	$(1.00, 1.31)$
4	$(4^{13}, 4^{14})$	10^5	$(1.49, 1.50)$	5	$(5^{11}, 5^{12})$	10^5	$(1.47, 1.48)$
		10^6	$(1.45, 1.49)$			10^6	$(1.40, 1.46)$
		10^7	$(1.08, 1.38)$			10^7	$(0.83, 1.32)$
6	$(6^{10}, 6^{11})$	10^5	$(1.45, 1.46)$	7	$(7^9, 7^{10})$	10^5	$(1.42, 1.43)$
		10^6	$(1.38, 1.45)$			10^6	$(1.30, 1.42)$
		10^7	$(0.82, 1.32)$			10^7	$(0.55, 1.24)$
8	$(8^8, 8^9)$	10^5	$(1.38, 1.42)$				
		10^6	$(1.05, 1.37)$				
		10^7	$(0.21, 0.97)$				

computation of $\eta_r(\cdot,\cdot)$ requires $O(r)$ multiplications. Hence, computing $\gamma_{\ell,c}$ also requires $O(r)$ multiplications. Since there are $\log_k n + 1$ levels in the tree, hence computing the expected header length requires $O(r \log n)$ multiplications. The algorithm requires constant amount of space. Hence, we have an algorithm requiring $O(r \log n)$ time and $O(1)$ space to compute the expected header length in the k-ary tree SD scheme for given values of k, n and r.

We have implemented the algorithm. Table 6.1 provides examples of outputs of the algorithm for different values of k, n and r.

6.4 Tackling Arbitrary Number of Users

In the description of the scheme so far, we have assumed that n is a power of k. This may turn out to be restrictive in practice. Here we describe how to modify the scheme so as to be able to handle an arbitrary number of users. When n is a power of k, the underlying structure is a *full* k-ary tree. In the more general case where n is not a power of k, we work with a *complete* k-ary tree. This is an analogue of complete binary trees used in data structures to describe heap algorithms.

The structure of a complete k-ary tree can be described as follows. Let $\ell_0 = \lceil \log_k n \rceil$ and $k^{\ell_0 - 1} < n \leq k^{\ell_0}$. By an abuse of notation, we denote by $\mathcal{T}^0$ the complete k-ary tree with n leaf nodes. The leaf nodes are at levels 0 and 1. Suppose that there are n_1 leaf nodes at level 0 and n_2 leaf nodes are at level 1. Let $n = k^{\ell_0 - 1} + i$ with $1 \leq i \leq k^{\ell_0} - k^{\ell_0 - 1}$. Then a simple calculation shows that $n_2 = k^{\ell_0 - 1} - \lceil i/(k-1) \rceil$ and $n_1 = n - n_2$. In $\mathcal{T}^0$, consider the path joining the root node 0 to the right-most internal node at level 1. Clearly, all subtrees rooted at nodes that are *not* on this path are full k-ary trees. In particular, subtrees rooted at nodes at level ℓ that are to the left (respectively right) of this path are of height ℓ (respectively $\ell - 1$). This path is consequently called the *dividing path*. If n is not a power of k, then subtrees rooted on the dividing path may not be full k-ary trees.

Definition of the Collection $\mathcal{S}$. This remains unchanged, i.e., $\mathcal{S}$ still consists of $\mathcal{N}$ and subsets $S_{i,J}$ where i is an internal node in $\mathcal{T}^0$ and J is a subset of nodes with a common parent in the subtree of $\mathcal{T}^0$ rooted at i. The method for assigning keys to the subsets also remain unchanged. (In Section 6.5 later, we provide a different method for assigning keys.)

User Storage. The actual number of seeds that a user will require depends on whether the user corresponds to a leaf at level 0 or a leaf at level 1. This number is at least $(2^{k-1}-1)\frac{\ell_0(\ell_0-1)}{2}$ and at most $(2^{k-1}-1)\frac{\ell_0(\ell_0+1)}{2}$. All users are attached to some node of the dividing path. Users to the left of the dividing path and attached to it at a level greater than 1, get $(2^{k-1}-1)\frac{\ell_0(\ell_0+1)}{2}$ (maximum number of) seeds; users to the right of the dividing path and attached to it at a level greater than 1, get $(2^{k-1}-1)\frac{\ell_0(\ell_0-1)}{2}$ (minimum number of) seeds. The number of seeds assigned to users attached to the dividing path at level 1 (to the rightmost internal node), is in the above mentioned range and can be easily calculated based upon the number of children of the last two nodes of the dividing path.

Cover Generation Algorithm. The algorithm remains by and large the same. It is only at the initial stage that some modification is required. The cover generation algorithm for full k-ary trees progresses by processing the nodes in the list $\mathcal{L}$ one by one. This list is maintained as a queue and is initialized by inserting all the revoked users into it in the left-to-right order. All the users and so all the revoked users are necessarily at level 0.

In the case of complete trees, some of the users may be at level 1. So, the initialization of $\mathcal{L}$ is done by inserting all the revoked nodes at level 0 in the left-to-right order. At this point, the users at level 1 are not inserted into the list. The nodes in $\mathcal{L}$ are now processed one-by-one as in the cover generation algorithm. As part of this processing, the parents of these nodes get appended to $\mathcal{L}$. These parents are (internal) nodes at level 1. When the processing of the last revoked node at level 0 which is in $\mathcal{L}$ is completed, all nodes at level 1 which have at least one revoked child have been added to $\mathcal{L}$ in the left-to-right order. Now, all nodes corresponding to revoked users at level 1 are inserted into $\mathcal{L}$. From this point onwards there is no further change in the cover generation algorithm. It proceeds exactly as in the case of full trees and generates the cover. It is not difficult to argue that the algorithm correctly generates the cover. We have implemented this cover generation algorithm and used it in the analysis of average header length.

Header Length Analysis. Moving from a full to a complete k-ary tree does not affect the upper bound on the header length of the algorithm. It is $\min(2r-1, n-r, \lceil n/k \rceil)$. For expected header length, as in the case of full k-ary trees, in theory, it is possible to develop an algorithm to compute the expected header length for the complete k-ary tree SD scheme.

As before let $X^i_{n,r}$ be the binary valued random variable which takes the value 1 if and only if the node i gives rise to a subset in the header. Hence, $\Pr[X^i_{n,r} = 1]$ has to be computed using (6.16) and (6.17). To that end, the number of nodes under the subtree rooted at a node i has to be calculated and substituted appropriately in the equations. Note that the subtrees rooted at nodes on the dividing path may or may not be full. Thus, in order to compute $\Pr[X^i_{n,r} = 1]$ for a node i using (6.16) and (6.17) it is required to consider a large number of cases depending on the relative position of a node with respect to the dividing path. While this can be done, the resulting algorithm becomes quite complicated and becomes difficult to implement.

In view of this difficulty, we have chosen not to implement the exact algorithm for finding the expected header length for complete k-ary trees. Instead, we have opted for a simulation study of the expected header length. For given values of k, n and r, we generate random revocation patterns using Floyd's algorithm [BF87] to sample r users from the set of n users. For each such random revocation pattern, the cover generation algorithm finds the exact cover and hence we get the header length for a particular revocation pattern. Taking the average of the header lengths obtained on different runs gives a statistical estimate of the expected header length. The number of iterations is chosen so that the average value of the header length stabilizes.

We have implemented this method. The result has been checked for accuracy in the following manner. In the case when n is a power of k, we have developed and implemented the algorithm to find the actual value of the expected header length. For such values of n, the results of the simulation study has been compared to the output of the exact algorithm and has been found to have tallied very well. Later, we report comparative performance analysis based on the simulation study of the expected header length.

6.5 Reducing User Storage

Given a $k \geq 2$, let n (not necessarily a power of k) be the number of users. By $\mathsf{us}_k(n)$ we denote the maximum number of m-bit seeds required to be stored by any of the n users in the system such that a user is able to generate the key associated to any subset in $\mathcal{S}$ of which it is a member. From (6.2), it appears that $\mathsf{us}_k(n)$ is $1 + (2^{k-1} - 1)\ell_0(\ell_0 + 1)/2$ where $\ell_0 = \lceil \log_k n \rceil$. In comparison to the case $k = 2$, for $k > 2$, the factor $(2^{k-1} - 1)$ contributes to the blow-up in the key size. In this section, we describe methods by which this blow-up can be somewhat mitigated leading to values of us_k which are lower than that given by (6.2). For small values of k, in comparison to (6.2), the decrease in user storage that is attained is significant.

The reduction in user storage described in this section is achieved by deriving the seeds in a different fashion. The collection $\mathcal{S}$ remains unaltered and so the cover generation algorithm does not change. Also, the header length analysis (both maximum and expected), remains unaltered.

6.5.1 The Case $k = 3$

To explain the basic idea, we start by considering the case of $k = 3$. In this case, from (6.2) the maximum number of seeds required to be stored by any user is $1 + 3\ell_0(\ell_0 + 1)/2$, where $\ell_0 = \lceil \log_3 n \rceil$. We show that this can be reduced to $1 + \ell_0(\ell_0 + 1)$.

Consider the tree $\mathcal{T}^0$ where each internal node has (at most) 3 children. (For ease of understanding, one may initially assume $\mathcal{T}^0$ to be a full 3-ary tree.) Let j be an internal node of $\mathcal{T}^0$ and its children are nodes numbered $3j + 1, 3j + 2, 3j + 3$. Users in $\mathcal{T}^j$ get seeds derived from the seeds associated to j. There are two kinds of seeds associated to j: the uniform random seed L_j and the derived seed $L_{i,\{j\}}$ where i is some ancestor of j. For any such seed L, there are seven seeds $L_\sigma = G_\sigma(L)$, $0 \leq \sigma \leq 6$ which are derived from L. If L is of the form $L_{i,\{j\}}$, then L_0 is the key associated to the subset $S_{i,\{j\}}$, while all the other L_σ's are distributed to the users in $\mathcal{T}^j$ in the following manner. (We identify σ with their 3-bit binary representations.)

Users in $\mathcal{T}^{3j+1}$ get: $L_{011}, L_{010}, L_{001}$.
Users in $\mathcal{T}^{3j+2}$ get: $L_{101}, L_{100}, L_{001}$.
Users in $\mathcal{T}^{3j+3}$ get: $L_{110}, L_{100}, L_{010}$.

Hence, corresponding to the label L associated to j, each user in $\mathcal{T}^j$ gets three seeds. We show that by adopting a different strategy for generating the L_σ's, it is possible to provide each user in $\mathcal{T}^j$ with two seeds, from which it can generate the three required seeds.

The idea is based on replacing G by another cryptographic hash function $H : \{0,1,2\} \times \{0,1\}^m \to \{0,1\}^m$. For $b = 0,1,2$, we denote $H(b, \mathit{seed})$ as $H_b(\mathit{seed})$. We define $G_0(\mathit{seed})$ to be $H_2(\mathit{seed})$. Suppose σ is a t-bit string $b_1 \cdots b_t$. Then H_σ is defined to be

$$H_{b_t}(\cdots(H_{b_1}(\mathit{seed})\cdots).$$

Consider again a seed L associated with the internal node j from which seeds for users in $\mathcal{T}^j$ are to be derived. For a t-bit binary string σ with $t \geq 1$, define $\widehat{L}_\sigma$ to be equal to $H_\sigma(L)$. A simple way of viewing this is the following. Consider an auxiliary full binary tree structure (independent of $\mathcal{T}^0$) of height t. Each path from the root to a leaf in this tree is of length t and is encoded as follows. Moving to the left child from a node is encoded by 0 and moving to the right child is encoded by 1. Then any node in the tree is encoded by a binary string σ which represents the path from the root to that node. The seed L is associated to the root node of the auxiliary tree. The action of H on the seed of a node to derive the seeds of its children, results in the association of the seed $\widehat{L}_\sigma$ to the node encoded by σ.

The different L's to be distributed to the users in $\mathcal{T}^j$ are defined from the $\widehat{L}$'s by a suitable permutation on the set of all 3-bit strings. More concretely, we define

$$L_{001} = \widehat{L}_{000},\ L_{011} = \widehat{L}_{001},\ L_{010} = \widehat{L}_{010},\ L_{110} = \widehat{L}_{011},$$
$$L_{100} = \widehat{L}_{100},\ L_{101} = \widehat{L}_{101},\ L_{000} = \widehat{L}_{110},\ L_{111} = \widehat{L}_{111}.$$

The new assignment of seeds to users in $\mathcal{T}^j$ is as follows:

Users in $\mathcal{T}^{3j+1}$ get: $\widehat{L}_{00}, \widehat{L}_{010}$.
Users in $\mathcal{T}^{3j+2}$ get: $\widehat{L}_{10}, \widehat{L}_{000}$.
Users in $\mathcal{T}^{3j+3}$ get: $\widehat{L}_{01}, \widehat{L}_{100}$.

Since $L_{001} = \widehat{L}_{000} = H_0(\widehat{L}_{00})$, $L_{011} = \widehat{L}_{001} = H_1(\widehat{L}_{00})$ and $L_{010} = \widehat{L}_{010}$, users in $\mathcal{T}^{3j+1}$ can generate the required seeds. Similarly, the users in $\mathcal{T}^{3j+2}$ and $\mathcal{T}^{3j+3}$ can generate the seeds required by them.

The above method shows that for any seed associated to any internal node j, the number of derived seeds to be stored by users in $\mathcal{T}^j$ reduces to 2 from 3. As a result, the number of seeds required to be stored by any user is (at most) $1 + 2 \times \ell_0(\ell_0 + 1)/2 = 1 + \ell_0(\ell_0 + 1)$. This is summarized in the following result.

Proposition 26. *Suppose $k = 3$ and there are n users with $\ell_0 = \lceil \log_3 n \rceil$. Then the maximum number of m-bit seeds required to be stored by any user is $\mathsf{us}_3(n) = 1 + \ell_0(\ell_0 + 1)$.*

Given n, the expressions for us_2 and us_3 are as follows:

$$\mathsf{us}_2(n) = 1 + \frac{1}{2} \times \lceil \log_2 n \rceil \left(\lceil \log_2 n \rceil + 1 \right) \quad \text{and} \quad \mathsf{us}_3(n) = 1 + \lceil \log_3 n \rceil \left(\lceil \log_3 n \rceil + 1 \right). \tag{6.21}$$

It is interesting to form a comparative study of us_2 and us_3. This is done using the following sequence of results.

Lemma 27. *Let $\ell \geq 1$. Let s be the least positive integer such that $2^s > 3^\ell$. Then $3^\ell + 1 \leq 2^s < 2^{s+2} < 3^{\ell+2} + 1$. In other words, there are at least three powers of two between $3^\ell + 1$ and $3^{\ell+2} + 1$.*

Proof. Let $2^s = 3^\ell + x$ for some $x \geq 1$. Since s is the least positive integer such that $2^s > 3^\ell$, it follows that $2^{s-1} < 3^\ell$. From this, we get $3^\ell/2 + x/2 < 3^\ell$ so that $x < 3^\ell$. Now, $2^{s+2} < 3^{\ell+2}$ if $4 \times 3^\ell + 4x < 3^{\ell+2}$ if $x < (5/4)3^\ell$. Since we already have $x < 3^\ell$, the result follows. □

Lemma 28. *Let ℓ be a positive integer and s be the least positive integer such that $2^s > 3^\ell + 1$. Then the following holds.*

1. *If ℓ is even then $3s \geq 4(\ell + 1)$. Further, the inequality is strict for even $\ell \geq 4$.*

2. *If $\ell \geq 5$ is odd then $3s \geq 4(\ell+1)$. Further, the inequality is strict for odd $\ell \geq 7$.*

Proof. We prove (1), the proof of (2) being similar. The proof is by induction on even $\ell \geq 2$. The base case is for $\ell = 2$ and then $s = 4$ and so the result holds. For the induction step, we first note that by Lemma 27, there are at least 3 powers of 2 between $3^\ell + 1$ and $3^{\ell+2}+1$. So the least power of 2 which is greater than $3^{\ell+2}+1$ is at least 2^{s+3}. By induction hypothesis, we have $3s > 4(\ell+1)$ and so $3(s+3) = 3s+9 > 4(\ell+1)+8 = 4(\ell+3)$. This shows the induction step. For $\ell = 4$, the inequality is strict and by the induction step, it follows that the inequality is strict for all even $\ell \geq 4$. □

Lemma 29. *Let $\ell \geq 4$ and s be the least positive integer such that $2^s > 3^\ell + 1$. Then for any n with $2^s + 1 \leq n \leq 3^{\ell+1}$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$.*

Proof. From the range of n it follows that $\mathsf{us}_3(n) = (\ell+1)(\ell+2)$. In the given range for n, $\mathsf{us}_2(n) \geq (s+1)(s+2)/2$.

We first prove the result by induction on even $\ell \geq 4$. For $\ell = 4$, $s = 7$ and the result holds. For the induction step, suppose the result holds for ℓ, i.e., $(s+1)(s+2)/2 > (\ell+1)(\ell+2)$. Also, by Lemma 28, we have $3s \geq 4(\ell+1)$. Consider the case for $\ell+2$. By Lemma 27, the least power of 2 which is greater than $3^{\ell+2}+1$ is at least 2^{s+3} and we have to consider n in the range $2^{s+3}+1 \leq n \leq 3^{\ell+3}$. In this range $\mathsf{us}_3(n) = (\ell+3)(\ell+4)$ and $\mathsf{us}_2(n) = (s+4)(s+5)/2$. The following computation shows the inductive step.

$$\begin{aligned}(s+4)(s+5)/2 &= 3s+6+(s+1)(s+2)/2 \\ &> 4(\ell+1)+6+(\ell+1)(\ell+2) \\ &= (\ell+3)(\ell+4).\end{aligned}$$

A similar argument by induction on odd $\ell \geq 5$ shows the result. □

Lemma 30. *Let $\ell \geq 7$ and s be the least positive integer such that $2^s > 3^\ell + 1$. Then for any n with $3^\ell + 1 \leq n \leq 2^s$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$.*

Proof. In the given range, $\mathsf{us}_3(n) = (\ell+2)(\ell+3)$ and $\mathsf{us}_2(n) = s(s+1)/2$. The induction is by separate induction for odd $\ell \geq 7$ (with corresponding $s = 12$) and even $\ell \geq 8$ (with corresponding $s = 13$). The base cases can be directly verified. The separate induction steps follow by an argument similar to that for Lemma 29. □

We finally get the following result.

Proposition 31. *Define I to be the following set of integers.*

$$I = \{3\} \cup [2^2+1, 3^2] \cup [2^4+1, 3^3] \cup [2^5+1, 3^4] \cup [2^7+1, 3^5] \cup [2^8+1, 3^6] \cup [2^{10}+1, \infty].$$

For $n \in I$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$ and for $n \in \mathbb{Z} \setminus I$, $\mathsf{us}_2(n) < \mathsf{us}_3(n)$.

Proof. For $n \leq 1024$, the result can be seen by direct computations. (Some of the cases also follow from Lemma 29.) For $n > 1024$, the combined effect of Lemma 29 and Lemma 30 shows the result. □

The above provides the complete comparison of the user storages for $k = 2$ and $k = 3$ and precisely proves that for $n > 1024$ the user storage required by the ternary tree based scheme is smaller than the user storage required by the binary tree based scheme. A similar observation with less refinement and without proof was made in [FKTS08].

6.5.2 The Case $k = 4$

As in the case for $k = 3$, let j be an internal node and L be a seed associated with j. Users in $\mathcal{T}^j$ obtain seeds L_σ derived from L using G_σ where in this case σ is a 4-bit string. More precisely, users in $\mathcal{T}^{4j+b}$, $1 \leq b \leq 4$, get seeds L_σ such that σ is a non-zero 4-bit string whose bth position from the left is zero. So, for the label L, each user in $\mathcal{T}^j$ gets 7 seeds.

In a manner similar to that of $k = 3$, it is possible to provide each user in $\mathcal{T}^j$ with 4 seeds such that from these seeds all the required 7 seeds can be derived. The idea is again based on using the function H to define certain seeds $\widehat{L}$'s and then define the L's in terms of the $\widehat{L}$'s. The definition of the $\widehat{L}$'s using H is the same as that in the case for $k = 3$. So, all we need to provide is the definition of L's in terms of the $\widehat{L}$'s. This is done as follows:

$$\begin{array}{l}
L_{0111} = \widehat{L}_{0000},\ L_{0110} = \widehat{L}_{0001},\ L_{0100} = \widehat{L}_{0010},\ L_{0101} = \widehat{L}_{0011},\\
L_{1011} = \widehat{L}_{0100},\ L_{0011} = \widehat{L}_{0101},\ L_{0010} = \widehat{L}_{0110},\ L_{1101} = \widehat{L}_{1001},\\
L_{1001} = \widehat{L}_{1010},\ L_{0001} = \widehat{L}_{1011},\ L_{1110} = \widehat{L}_{1100},\ L_{1100} = \widehat{L}_{1101},\\
L_{1000} = \widehat{L}_{1110},\ L_{1010} = \widehat{L}_{1111}.
\end{array}$$

The distribution of seeds to the users in $\mathcal{T}^j$ is the following.

Users in $\mathcal{T}^{4j+1}$ get: $\widehat{L}_{00}, \widehat{L}_{0110}, \widehat{L}_{0101}, \widehat{L}_{1011}$.
Users in $\mathcal{T}^{4j+2}$ get: $\widehat{L}_{01}, \widehat{L}_{101}, \widehat{L}_{111}$.
Users in $\mathcal{T}^{4j+3}$ get: $\widehat{L}_{001}, \widehat{L}_{10}, \widehat{L}_{1101}, \widehat{L}_{1110}$.
Users in $\mathcal{T}^{4j+4}$ get: $\widehat{L}_{0001}, \widehat{L}_{0010}, \widehat{L}_{0110}, \widehat{L}_{11}$.

Using these seeds, each user can create the L's that it is supposed to get. For example, users in $\mathcal{T}^{4j+1}$ should be able to create $L_{0001}, L_{0010}, L_{0011}, L_{0100}, L_{0101}, L_{0110}, L_{0111}$. These can be obtained from the seeds obtained by the users in $\mathcal{T}^{4j+1}$ in the following manner.

$$L_{0111} = \widehat{L}_{0000} = H_0(H_0(\widehat{L}_{00})); \; L_{0110} = \widehat{L}_{0001} = H_1(H_0(\widehat{L}_{00}));$$
$$L_{0100} = \widehat{L}_{0010} = H_0(H_1(\widehat{L}_{00})); \; L_{0101} = \widehat{L}_{0011} = H_1(H_1(\widehat{L}_{00}));$$
$$L_{0011} = \widehat{L}_{0101}; \; L_{0010} = \widehat{L}_{0110}; \; L_{0001} = \widehat{L}_{1011}.$$

In a similar manner, users in the other subtrees of $\mathcal{T}^j$ can create the seeds required by them. Corresponding to the seed L associated with node j, the number of seeds to be stored by users in the subtree $\mathcal{T}^{4j+2}$ is 3, while users in all the other subtrees require to store 4 seeds. From this we get the following result.

Proposition 32. *Suppose $k = 4$ and there are n users with $\ell_0 = \lceil \log_4 n \rceil$. Then the maximum number of m-bit seeds required to be stored by any user is $\mathsf{us}_4(n) = 1 + 2\ell_0(\ell_0 + 1)$.*

Note that this is a significant improvement over the requirement of storing $1+3.5\ell_0(\ell_0+1)$ seeds as indicated by (6.2). The value of $\mathsf{us}_4(n)$, however, is greater than $\mathsf{us}_2(n)$ for $n \geq 4$. So, the user storage for binary trees is less than that for 4-ary trees.

6.5.3 The Technique for General k

Let $k \geq 3$ and consider the k-ary tree $\mathcal{T}^0$. As before, for any internal node j, there are two kinds of seeds associated with it: one uniform random label L_j and several labels $L_{i,\{j\}}$ each derived from some ancestor i of j. Let L be any such seed. Users in the tree $\mathcal{T}^j$ get seeds derived from L with users in the subtree $\mathcal{T}^{kj+b}$ getting all seeds $L_\sigma = G_\sigma(L)$ where σ is any non-zero k-bit string having a zero at position b from the left.

In the cases of $k = 3$ and $k = 4$, we have seen alternative ways of deriving L_σ. The idea has been to derive the $\widehat{L}_\sigma$'s using H and then define the L_σ's in terms of the $\widehat{L}_\sigma$'s. For the case of general k, it is still possible to derive the $\widehat{L}_\sigma$'s using H. The problem, however, is in defining the L's in terms of the $\widehat{L}_\sigma$'s. For $k = 3$ and $k = 4$, this has been done in a somewhat ad-hoc fashion and does not extend to the case for general k. The technique is however clear - we try to arrange the $\widehat{L}_\sigma$'s such that σ's with the same prefix are grouped together. Below we describe a more systematic method of deriving the L_σ's from L.

The idea is based on the notion of cyclotomic cosets. There are several equivalent ways of viewing cyclotomic cosets. The description that we give below is primarily based on cyclic shifts of bit strings. This is equivalent to the more conventional description [MS78] as we point out later.

The intuition behind using cyclotomic cosets viewed as cyclic shifts of bit strings is as follows. A matrix with elements from each coset forming a row would have at least one column such that a particular bit position is 0 for all elements in that column. Each of these elements in the matrix can be viewed as bit strings each representing a subset. If the elements of the matrix are generated row-wise in lexicographic ordering or the coset leaders, the columns of the matrix not only have one bit position as zero, the subsequent bit positions on the cyclic right also have the 0's and 1's grouped together in a recursive fashion. This leads us to creating a tree structure so that we get a general algorithm for finding a way to derive keys with lesser number of keys to be stored by each user.

Let σ be a k-bit string. Then the cyclotomic coset containing σ is the set of all k-bit strings that can be obtained by one or more circular left shifts of σ. Clearly there can be at most k elements in any cyclotomic coset and further, the number of elements in a cyclotomic coset is necessarily a divisor of k. So, if k is a prime, then the number of elements in any cyclotomic coset is either 1 or k. The all-zero string forms a cyclotomic coset by itself as does the all-one string. These are the only two cyclotomic cosets consisting of single elements. Given k, let χ_k denote the total number of cyclotomic cosets defined from k-bit strings.

The above can be described in terms of modulo arithmetic as follows. Let s be an integer in $[0, 2^k - 2]$. Then $2s \bmod 2^k - 1$ corresponds to a cyclic left shift of the k-bit binary representation of s. So, the cyclotomic coset containing the k-bit binary representation of s is essentially also the set of integers $s, 2s \bmod 2^k - 1, \ldots$.

If α is a generator of the field $GF(2^k)$, then α raised to the powers of elements (seen as integers) of one cyclotomic coset form the roots of one irreducible polynomial. Using this correspondence, the number $I(m)$ of irreducible polynomials of degree m over $GF(2)$ is given as [MS78]

$$I(m) = \frac{1}{m} \sum_{d|m} \mu(d) 2^{m/d}, \tag{6.22}$$

where $\mu()$ is the Möbius function. The factorization of $x^{2^k} - x$ consists of all irreducible polynomials whose degrees divide k. The number of such polynomials is the number χ_k of cyclotomic cosets of k-bit strings and is obtained by summing $I(m)$ over all m which divides k. Using elementary results on the Möbius function, this turns out to be the following expression

$$\chi_k = \frac{1}{k} \sum_{t|k} \phi(t) 2^{k/t}, \tag{6.23}$$

where $\phi()$ is the Euler totient function.

Given two binary strings σ and τ of the same length, we define $\sigma \prec \tau$ if the integer represented by σ is smaller than the one represented by τ. In the following, we will assume that the elements of any cyclotomic coset are ordered from left-to-right based on $\prec$ and the first element will be called the coset representative. Further, we assume that the cyclotomic cosets are themselves ordered based on their coset representatives.

Let $C_0, \ldots, C_{\chi_k - 1}$ be the ordering of the cyclotomic cosets. Then C_0 is the coset containing the all-zero string and C_{χ_k-1} is the coset containing the all-one string. We will consider only the cosets $C_1, \ldots, C_{\chi_k-2}$. These are ordered in a matrix fashion with the ith row of the matrix consisting of the elements of C_i. Examples of the matrix for $k = 3$, $k = 4$ and $k = 5$ are given in Table 6.2. If k is prime, each row of the matrix will have k strings and if k is composite, the number of elements in the rows will be divisors of k.

Let us denote the matrix for k by $M^{(k)}$. Let the columns of $M^{(k)}$ be denoted by $V_1^{(k)}, \ldots, V_k^{(k)}$. Note that if k is composite, some of the $V_b^{(k)}$ will have blanks (the empty string) in their components. The non-empty strings in any column $V_b^{(k)}$ are obtained by a circular left shift of the corresponding elements in the column $V_{b-1}^{(k)}$. Extending this, the non-empty strings in $V_b^{(k)}$ are obtained as circular left shifts by b places of the corresponding elements in the column $V_1^{(k)}$. By construction, the first bit of each entry of $V_1^{(k)}$ is 0. By the left shift property, the bth bit position of each non-empty string in $V_b^{(k)}$ is 0.

Based on the matrix $M^{(k)}$ we define an auxiliary tree $T^{(k)}$. This tree is not a sub-tree

Table 6.2: Examples of $M^{(k)}$ for $k = 3$, $k = 4$ and $k = 5$.

$k = 3$			$k = 4$				$k = 5$				
							00001	00010	00100	01000	10000
			0001	0010	0100	1000	00011	00110	01100	11000	10001
001	010	100	0011	0110	1100	1001	00101	01010	10100	01001	10010
011	110	101	0101	1010			00111	01110	11100	11001	10011
			0111	1110	1101	1011	01011	10110	01101	11010	10101
							01111	11110	11101	11011	10111

of $\mathcal{T}^0$. (Note the difference in the notation between $\mathcal{T}^0$ and $T^{(k)}$.) Its role is to define the L_σ's in a manner such that the number of seeds required to be stored by a user reduces from the number $(2^{k-1} - 1)$ given by (6.2). There are a total of k levels in $T^{(k)}$ with the root at level 0 and the level numbers increasing as we move down the tree. (Note that this level numbering is opposite to the one used in $\mathcal{T}^0$. This is for notational convenience.) The root note of $T^{(k)}$ has k children. By $T_b^{(k)}$, $b = 1, \ldots, k$, we denote the k subtrees rooted at these k nodes. Each $T_b^{(k)}$ is a binary tree having k levels numbered 1 to $k-1$ and the the number of leaf nodes in $T_b^{(k)}$ is the number of non-empty strings in the column $V_b^{(k)}$ of $M^{(k)}$. The root node of $T_b^{(k)}$ is labelled by $(b, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda}$ is the empty string. (For simplicity, we sometimes write b instead of $(b, \boldsymbol{\lambda})$.) The other nodes of $T_b^{(k)}$ are labelled by a pair (b, τ), where τ is a binary string which encodes the path from the root of $T_b^{(k)}$ to the node. The tree $T_b^{(k)}$ is not balanced. The construction of $T_b^{(k)}$ based on $V_b^{(k)}$ is described as follows.

1. The bth bit of each non-empty string in $V_b^{(k)}$ is 0. This bit position corresponds to the root node $(b, \boldsymbol{\lambda})$ of $T_b^{(k)}$ at level 1. Starting from the bth bit, we cyclically move right over the bit positions in the non-empty strings of $V_b^{(k)}$. Apart from bit position b, there are $k-1$ other positions in the non-empty strings in $V_b^{(k)}$. To these positions correspond the levels numbered 2 to k of $T_b^{(k)}$.

2. There are two nodes at level numbered 2 of $T_b^{(k)}$ and these are labelled as $(b, 0)$ and $(b, 1)$. These nodes have binary trees rooted at them. All strings in $V_b^{(k)}$ whose $(b+1)$st bit position is 0 form the leaf nodes of the tree rooted at $(b, 0)$. Similarly, all strings in $V_b^{(k)}$ whose $(b+1)$st bit position is 1 form the leaf nodes of the tree rooted at $(b, 1)$.

3. Continuing the above, suppose the tree $T_b^{(k)}$ has been constructed up to level $l < k-1$. To construct the nodes at level $l+1$, we look at the $(b+l+1)$th bit position (cyclically

from the right) of the strings in $V_b^{(k)}$. Let (b, τ) be a node at level l of $T_b^{(k)}$, so that τ is an $(l-1)$-bit string. Then 0τ is a substring in bit positions b to $b+l$ in one of the strings in $V_b^{(k)}$. Considering bit position $b+l+1$, the string 0τ is extended in two possible ways: $0\tau0$ and $0\tau1$. This gives rise to two children of (b, τ) labelled as $(b, \tau0)$ and $(b, \tau1)$.

As a consequence of this construction, to the leaf nodes of $T_b^{(k)}$ are associated the seeds L_σ where σ ranges over the non-empty strings in $V_b^{(k)}$. The top-to-bottom order in $V_b^{(k)}$ corresponds to the left-to-right order of the leaf nodes in $T_b^{(k)}$. The structure of $T^{(k)}$ for $k = 3, 4$ and 5 and the associated seeds L_σ's are shown in Figures 6.9, 6.10 and 6.11 respectively.

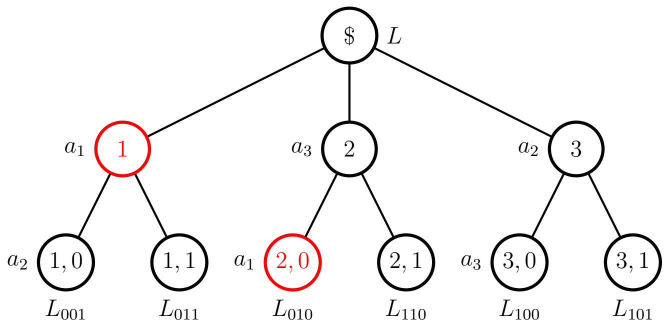


Figure 6.9: The structure of tree $T^{(3)}$. The seeds of the nodes marked with a_c are assigned to all users in $\mathcal{T}^{3j+c}$.

Consider again an internal node j of $\mathcal{T}^0$ and a seed L associated with the node j from which seeds L_σ's for the users in the subtree $\mathcal{T}^j$ are to be derived. The derivation of these seeds is done with the structure of $T^{(k)}$ and two hash functions $F : [1, t] \times \{0, 1\}^m \to \{0, 1\}^m$ and $H : \{0, 1, 2\} \times \{0, 1\}^m \to \{0, 1\}^m$. The function H is as used in Sections 6.5.1 and 6.5.2 while the function F is new. As before, we will use the notation $F_b(\cdot)$ and $H_c(\cdot)$ to denote the functions $F(b, \cdot)$ and $H(c, \cdot)$ respectively. For a binary string τ, the notation H_τ is as defined earlier.

The functions F and H together replace the function G used in Section 6.2.1 in the following manner. For any *seed*, the corresponding key is defined to be $H_2(\textit{seed})$ which in Section 6.2.1 was defined as $G_0(\textit{seed})$. The bth child of $T^{(k)}$ is given the seed $\widehat{L}_b = F_b(L)$. For any other node of $T_b^{(k)}$ labelled by a pair (b, τ), we associate the seed $\widehat{L}_{b,\tau} = H_\tau(F_b(L)) = H_\tau(L_b)$. Any leaf node of $T_b^{(k)}$ is labelled by a pair (b, τ) and has an associated L_σ. We define $L_\sigma = \widehat{L}_{b,\tau}$. This provides the definition of all the L_σ's that are required to be distributed to the users in the subtree $\mathcal{T}^j$.

We next look at the assignment of seeds to users. Each user in $\mathcal{T}^{kj+b}$ should be given a set of seeds such that it is able to generate all L_σ such that σ is a non-zero k-bit string whose bth position from the left is 0; also, it should not be able to generate any other seed. This is achieved by giving each user a subset of the $\widehat{L}$'s.

The seeds distributed to the users in $\mathcal{T}^{(kj+b)}$ are $\widehat{L}_{c,\tau}$ such that the following condition holds. In the subtree of $T^{(k)}$ rooted at the parent of the node labelled (c,τ) there is at least one leaf which is labelled by L_σ where the bth bit from the left in σ is 1.

For $k=5$, the assignment using $T^{(5)}$ shown in Figure 6.11 is the following.

Users in $\mathcal{T}^{5j+1}$ get: $\widehat{L}_{1,\boldsymbol{\lambda}}$, $\widehat{L}_{2,0}$, $\widehat{L}_{3,00}$, $\widehat{L}_{3,10}$, $\widehat{L}_{4,000}$, $\widehat{L}_{4,010}$.
Users in $\mathcal{T}^{5j+2}$ get: $\widehat{L}_{5,\boldsymbol{\lambda}}$, $\widehat{L}_{1,0}$, $\widehat{L}_{2,00}$, $\widehat{L}_{2,10}$, $\widehat{L}_{3,000}$, $\widehat{L}_{3,010}$.
Users in $\mathcal{T}^{5j+3}$ get: $\widehat{L}_{4,\boldsymbol{\lambda}}$, $\widehat{L}_{5,0}$, $\widehat{L}_{1,00}$, $\widehat{L}_{1,10}$, $\widehat{L}_{2,000}$, $\widehat{L}_{2,010}$.
Users in $\mathcal{T}^{5j+4}$ get: $\widehat{L}_{3,\boldsymbol{\lambda}}$, $\widehat{L}_{4,0}$, $\widehat{L}_{5,00}$, $\widehat{L}_{5,10}$, $\widehat{L}_{1,000}$, $\widehat{L}_{1,010}$.
Users in $\mathcal{T}^{5j+5}$ get: $\widehat{L}_{2,\boldsymbol{\lambda}}$, $\widehat{L}_{3,0}$, $\widehat{L}_{4,00}$, $\widehat{L}_{4,10}$, $\widehat{L}_{5,000}$, $\widehat{L}_{5,010}$.

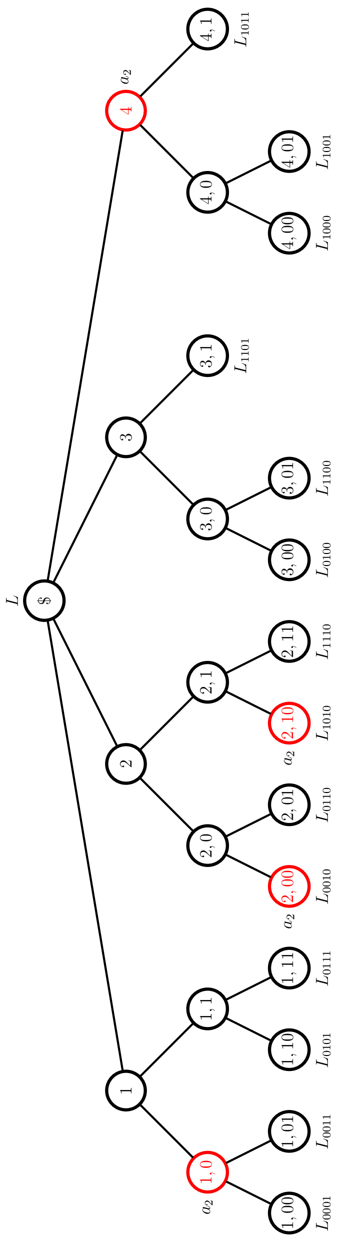


Figure 6.10: The structure of $T^{(4)}$.

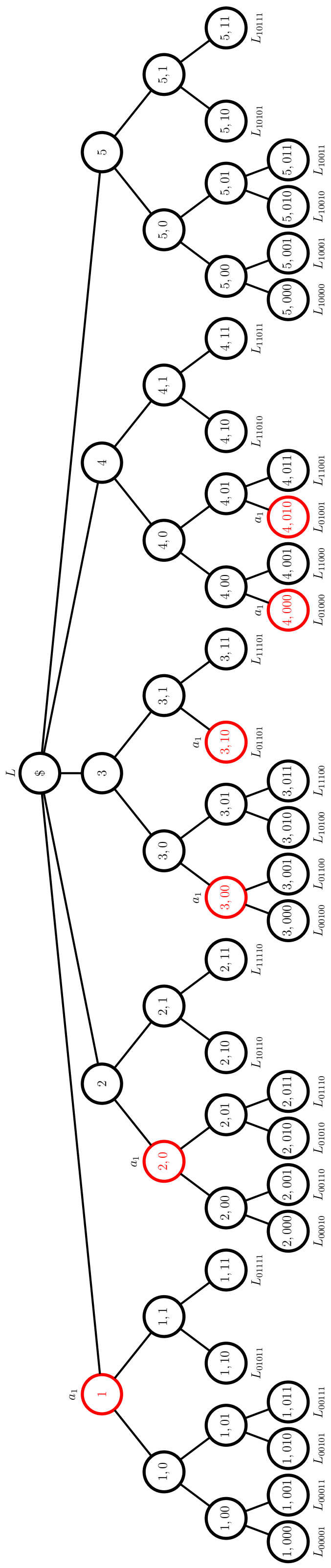


Figure 6.11: The structure of $T^{(5)}$.

The total number of seeds assigned to any user is given by the following result.

Proposition 33. *Let $k \geq 3$, $n \geq 1$ and $\ell_0 = \lceil \log_k n \rceil$. Then* $\mathsf{us}_k(n) = (\chi_k - 2)(\ell_0(\ell_0 + 1))/2$.

Proof. Consider the tree $T^{(k)}$. The root node has k children $T_1^{(k)}, \ldots, T_k^{(k)}$. Seeds of the form $\widehat{L}_{b,\tau}$ associated with the nodes of these trees are assigned to the different users. The leaf nodes of $T_b^{(k)}$ are also labelled by the seeds L_σ's which are elements of $V_b^{(k)}$, the bth column of the matrix $M^{(k)}$. Recall that the non-empty strings in the bth column of $M^{(k)}$ are obtained by a cyclic left shift of the corresponding strings in the $(b-1)$th column of $M^{(k)}$.

As a result, the σ's corresponding to the labels L_σ's of the leaf nodes of $T_b^{(k)}$ are obtained by a cyclic left shift of the respective ζ's corresponding to the labels L_ζ's of the leaf nodes of $T_{b-1}^{(k)}$. Due to this, the following symmetry property holds. For $b > 1$, if the users in $\mathcal{T}^{kj+b}$ get x seeds of the form $\widehat{L}_{1,\tau}$, then the users in $\mathcal{T}^{kj+1}$ get (at most) x seeds of the form $\widehat{L}_{b,\tau}$.

A consequence of this symmetry property is that the number of seeds given to the users in $\mathcal{T}^{kj+1}$ is equal to the number of seeds of the form $\widehat{L}_{1,\tau}$ which are assigned to all the users. By construction, if τ ends with a 0, then the corresponding $\widehat{L}_{1,\tau}$ is assigned to some user. So, the only seeds of the form $\widehat{L}_{1,\tau}$ which are not assigned to any user are those ending with a 1. These seeds correspond exactly to the nodes of $T_1^{(k)}$ which are the right children of some node.

The number of leaf nodes of $T_1^{(k)}$ is the number of strings in $V_1^{(k)}$ which in turn is equal to $\chi_k - 2$. Since $T_1^{(k)}$ is a binary tree, the number of internal nodes of $T_1^{(k)}$ is equal to $\chi_k - 3$. So, the total number of nodes of $T_1^{(k)}$ is $2\chi_k - 5$. Each internal node has exactly one child node and so the number of nodes which are right children is equal to the number of internal nodes of $T_1^{(k)}$ which is $\chi_k - 3$. As a result, the number of nodes of $T_1^{(k)}$ which are labelled by $(1, \tau)$ with τ ending with 0 is equal to $2\chi_k - 5 - (\chi_k - 3) = \chi_k - 2$. □

Note that $\chi_3 = 4$ and $\chi_4 = 6$ and so the user storage given by this result agrees with the user storage given in Propositions 26 and 32 respectively. The methods of deriving the seeds, however, are different.

In Table 6.3, we provide a comparison of the user storage given by Proposition 33 to that given by (6.2). In concrete terms, the reduction is quite significant. Comparing to the user storage for $k = 2$, the increase is only a few times. This can be seen from the values of $\mathsf{us}_2(n)$ and $\mathsf{us}_k(n)$ for $k > 2$ and different values of n as given in Table 6.5.

Table 6.3: Reduction of user storage achieved by Proposition 33 in comparison to 6.2. In each case, $\ell_0 = \lceil \log_k n \rceil$.

	$k = 3$	$k = 4$	$k = 5$
Eqn. (6.2)	$1 + 1.5\ell_0(\ell_0 + 1)$	$1 + 3.5\ell_0(\ell_0 + 1)$	$1 + 7.5\ell_0(\ell_0 + 1)$
us_k	$1 + \ell_0(\ell_0 + 1)$	$1 + 2\ell_0(\ell_0 + 1)$	$1 + 3\ell_0(\ell_0 + 1)$

	$k = 6$	$k = 7$	$k = 8$
Eqn. (6.2)	$1 + 15.5\ell_0(\ell_0 + 1)$	$1 + 31.5\ell_0(\ell_0 + 1)$	$1 + 63.5\ell_0(\ell_0 + 1)$
us_k	$1 + 6\ell_0(\ell_0 + 1)$	$1 + 9\ell_0(\ell_0 + 1)$	$1 + 18.5\ell_0(\ell_0 + 1)$

6.6 The Layered k-ary Tree Subset Difference Scheme

The idea of layering the levels of the underlying binary tree $\mathcal{T}^0$ of the NNL-SD scheme in order to reduce the user storage was introduced in Section 2.1.2 followed by a detailed description and analysis in Chapter 5 [HS02]. Here we apply the same technique to reduce the storage of the k-ary tree generalization of the SD scheme.

As before, we work with an underlying full k-ary tree with $n = k^{\ell_0}$ leaf nodes. Nodes at equal distances from the root are said to be at the same level. There are $\ell_0 = \log_k n$ levels in the tree $\mathcal{T}^0$. Some of these levels are marked as *special*. A *layer* is defined to be the levels in between and including two consecutive special levels. Hence, a layering strategy $\boldsymbol{\ell}$ is defined by the numbers of the special levels $\ell_0 > \ell_1 > \ldots > \ell_{e-1} > \ell_e = 0$.

Let $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{e-1}, \ell_e)$ be a layering strategy. There are $e+1$ special levels in $\boldsymbol{\ell}$. An alternate representation of the layering strategy is by the length of each layer. For $1 \leq i \leq e$, we define $d_i = \ell_{i-1} - \ell_i$ so that d_i's are positive integers whose sum is ℓ_0. At the same time, given any sequence of positive integers $\boldsymbol{d} = (d_1, \ldots, d_e)$ whose sum is ℓ_0, it is possible to define a layering scheme where $\ell_i = \ell_0 - \sum_{j=1}^{i} d_j$.

The Collection $\mathcal{S}$ and Key Assignment. Let j be an internal node in $\mathcal{T}^i$ having k children in $\mathcal{T}^0$ namely $\{kj+1, \ldots, kj+k\}$. Let $J \subset \{kj+1, \ldots, kj+k\}$ such that $0 < |J| < k$. Subsets in the collection $\mathcal{S}$ are of the form $S_{i,J}$ as has been described in Section 6.2.1 for the k-ary tree SD scheme. Each internal node i is assigned a uniform random seed L_i as before. However, unlike the k-ary tree SD scheme, not all subsets of the form $S_{i,J}$ are assigned keys. With the introduction of layering, only certain subsets of the form $S_{i,J}$ are assigned keys.

These subsets are of the following type:

- If i is at a special level, then J can be any set of nodes that are siblings in the subtree $\mathcal{T}^i$.
- If i is not at a special level, then J will be a set of nodes that are siblings in the subtree $\mathcal{T}^i$ and in the same layer as i.

We have seen in Section 6.2 and Section 6.5 two different techniques to derive the key $L_{i,J}$ for a set $S_{i,J}$. (One may recollect here that the collection $\mathcal{S}$ of subsets remains unchanged for both key assignment techniques.) For the layered version of the scheme, we assume that the second technique of Section 6.5 that requires less user storage, is used to assign keys to subsets.

User Storage. Given a layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{e-1}, \ell_e)$ in a tree with $n = k^{\ell_0}$ leaves, we compute the number of seeds that a user needs to store. For an ancestor i of the user that is at a special level ℓ, the user has to store $(\chi_k - 2)$ seeds derived by sets of nodes that are directly attached to (or "falling off from") the path between the user leaf and i. Hence the total number of seeds to be stored for an ancestor at a special level is $(\chi_k - 2)\ell$. Similarly, for an ancestor of the user that is at a non-special level ℓ which is between two special levels ℓ_{i-1} and ℓ_i $(\ell_{i-1} < \ell < \ell_i)$ the user has to store $(\chi_k - 2)(\ell - \ell_i)$ seeds. Hence the user storage for the layering strategy $\boldsymbol{\ell}$ is

$$\mathsf{storage}_0^k(\boldsymbol{\ell}) = (\chi_k - 2) \times \left(\sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=\ell_{i+1}+1}^{\ell_i - 1} (j - \ell_{i+1}) \right), \tag{6.24}$$

where $\ell_0 = \lceil \log_k n \rceil$. The expression to compute $\mathsf{storage}_0^k(\boldsymbol{\ell})$ in (6.24) for general k is derived by a similar logic as used in Chapter 5 [BS14a] for $k = 2$. The storage requirement derived for $k = 2$ from (6.24), is exactly the same as found in Chapter 5 [BS14a].

6.6.1 Storage Minimal Layering

Now, let us consider two extreme layering strategies and find their storage requirement. The first layering strategy has only the top-most and bottom-most levels as special and hence

$\boldsymbol{\ell} = (\ell_0, 0)$. It can be easily seen that this scheme is the same as the k-ary tree SD scheme and hence has the same storage requirement as that of the k-ary tree SD scheme.

As more special levels are introduced in between these two levels, the user storage should go down. This is because, the number of seeds derived from the nodes at non-special levels above the bottom-most layer, reduces in the user storage.

However, we see that as we continue marking more and more levels as special, we finally get the layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_0 - 1, \ldots, 1, 0)$ where all the levels are marked as special. The resultant scheme is again exactly the same as the k-ary tree SD scheme. Hence, as for binary trees in Chapter 5 [BS14a], there should exist a layering strategy of the k-ary trees that results in minimum storage.

For a given k and ℓ_0, let $\mathrm{SML}_0^k(\ell_0)$ denote a layering strategy $\boldsymbol{\ell}$ (or equivalently given by the sequence of differences $\boldsymbol{d}$), such that $\mathsf{storage}_0^k(\boldsymbol{\ell})$ takes the minimum value among all possible layering strategies. Let $\#\mathrm{SML}_0^k(\ell_0)$ denote the storage requirement $\mathsf{storage}_0^k(\boldsymbol{\ell})$ for the storage minimal layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_e)$.

The storage minimal layering strategy $\mathrm{SML}_0^k(\ell_0)$ can be found using a dynamic programming algorithm as follows. We first fix the number e of layers in a layering strategy. Out of all the storage requirements of these layering strategies one will be minimum. Let $\mathrm{SML}_0^k(e, \ell_0)$ denote a layering strategy that requires minimum storage amongst all layerings with e layers. The number of layers e can be at least 1 and at most ℓ_0. Hence, $\mathrm{SML}_0^k(\ell_0)$ will be the minimum of all these layering strategies over all values of e. So we get

$$\#\mathrm{SML}_0^k(\ell_0) = \min_{1 \le e \le \ell_0} \#\mathrm{SML}_0^k(e, \ell_0). \tag{6.25}$$

Similarly, $\#\mathrm{SML}_0^k(e, \ell_0)$ is the minimum storage requirement amongst all the layering strategies for a given number of layers e. So we get

$$\#\mathrm{SML}_0^k(e, \ell_0) = \min_{(\ell_0, \ell_1, \ldots, \ell_e)} \mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e). \tag{6.26}$$

We write the expression to compute $\mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e)$ on the right hand side of (6.24) in a recursive fashion as follows

$$\mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e) = (\chi_k - 2) \times \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2}\right) + \mathsf{storage}_0^k(\ell_1, \ldots, \ell_e). \tag{6.27}$$

Table 6.4: Ranges of n ($< 2^{30}$) such that $\#\text{SML}_0^2 > \#\text{SML}_0^3$.

Range of n	$(\#\text{SML}_0^2, \#\text{SML}_0^3)$	Range of n	$(\#\text{SML}_0^2, \#\text{SML}_0^3)$
$\{2^4+1,\ldots,3^3\}$	$(11,10)$	$\{2^6+1,\ldots,3^4\}$	$(18,16)$
$\{2^7+1,\ldots,3^5\}$	$(22,22)$	$\{2^9+1,\ldots,3^6\}$	$(30,28)$
$\{2^{11}+1,\ldots,3^7\}$	$(40,36)$	$\{2^{12}+1,\ldots,3^8\}$	$(45,44)$
$\{2^{14}+1,\ldots,3^9\}$	$(55,52)$	$\{2^{15}+1,\ldots,3^{10}\}$	$(61,60)$
$\{2^{17}+1,\ldots,3^{11}\}$	$(73,70)$	$\{2^{19}+1,\ldots,3^{12}\}$	$(85,80)$
$\{2^{20}+1,\ldots,3^{13}\}$	$(91,90)$	$\{2^{22}+1,\ldots,3^{14}\}$	$(105,100)$
$\{2^{23}+1,\ldots,3^{15}\}$	$(112,110)$	$\{2^{25}+1,\ldots,3^{16}\}$	$(126,122)$
$\{2^{28}+1,\ldots,3^{18}\}$	$(148,146)$		

Using (6.27) and (6.26), we get a recursive definition of $\#\text{SML}_0^k(e,\ell_0)$ in terms of $\#\text{SML}_0^k(e-1,\ell_1)$ as follows

$$\#\text{SML}_0^k(e,\ell_0) = \min_{1\le\ell_1<\ell_0}\left((\chi_k-2)\times\left(\ell_0+\frac{(\ell_0-\ell_1)(\ell_0-\ell_1-1)}{2}\right)+\#\text{SML}_0^k(e-1,\ell_1)\right). \tag{6.28}$$

This recursive definition of (6.28) is the basis for our dynamic programming algorithm. A similar dynamic programming algorithm to compute the $\text{SML}_0^2(\ell_0)$ for layering in binary trees has been proposed in Section 5.2.4 [BS14a]. The above algorithm is a generalization using k-ary trees of that algorithm using binary trees.

Empirical Analysis. In Proposition 6.5.2 we have seen that the storage requirement of the k-ary tree SD scheme for $k=3$ is less than that of $k=2$ (the binary tree case) for $n\ge 2^{10}$. We know from Section 5.2 [HS02, BS14a] that the user storage of a subset difference based scheme can be reduced using different layering strategies. Hence, it is of interest to check the effect of layering on the k-ary tree SD scheme.

We have implemented the dynamic programming algorithm for finding the storage minimal layering in the k-ary tree SD scheme. Executing this algorithm for computing the storage minimal layering for $k=3$ for different values of n and comparing with the case when $k=2$, we find the range of n where the storage due to $k=3$ is less than the storage due to $k=2$. Table 6.4 lists those ranges for $n<2^{30}$ and the corresponding storage requirements.

6.7 A Comparative Study

Table 6.5 provides a comparative study of the mean header length MHL_k and the user storage us_k as k varies from 2 to 8. For the study, we have varied n from 10^3 to 10^8. Since n is not a power of k, the complete tree extension of the scheme described in Section 6.4 has been used. The reported results for MHL_k has been done using the simulation program. (Earlier, in Table 6.1 we have provided results based on running the algorithm for computing the expected header length when n is a power of k.) User storage is obtained from Proposition 33. We observe the following from Table 6.5:

- For small values of r/n, $\mathsf{MHL}_k/r > \mathsf{MHL}_2/r$ while for larger values of r/n, $\mathsf{MHL}_k/r < \mathsf{MHL}_2/r$. This indicates that for a given $k > 2$, there is a threshold value $\delta_k \in (0, 1)$ such that for $r/n > \delta_k$, the mean header length of the k-ary tree SD scheme is smaller than that for $k = 2$.

- For a fixed k, the values of MHL_k/r are (almost) the same for a given ratio r/n for any arbitrary n. This behavior is captured in Table 6.6 and the corresponding plot of its data is given in Figure 6.12. The almost straight red line in Figure 6.12 shows the behavior for $k = 2$. For other values of k, the points where the respective curves intersect this straight line correspond to $r/n = \delta_k$. These approximate values of δ_k are shown in Table 6.7. We see that as k increases the value of δ_k decreases and consequently, the performance of the k-ary tree SD scheme is better than $k = 2$ for a larger range of values of r.

A Practical Consideration. An important application of broadcast encryption is pay-per-view of cable TV and DTH services. In cable TV systems, a set of basic channels are free to air and are not scrambled. Hence, everyone with a cable TV connection can view these channels. All other channels are encrypted. For paid channels or pay-per-view programs, it is quite likely that the number of users subscribing to the channel/program is substantially less than the total number of customers n of the cable company. Hence, the number of revoked users is of the magnitude of n. (Such an assumption may not be true of other applications of broadcast encryption such as DRM in audio/video players.) As an example, consider a Pay-TV application with $n = 10^8$ users. From Table 6.5, it can be seen that for $r = 0.4n$ and assuming 128-bit keys, the bandwidth savings of $k = 8$ over $k = 2$ for that

Table 6.5: User storage and mean header lengths in the complete k-ary tree scheme for values of k between 2 and 8. For a fixed n, we report MHL_k/r for three different choices of r namely, $r = (0.1n, 0.2n, 0.4n)$.

n	k	us_k	MHL_k/r	n	k	us_k	MHL_k/r
	2	55	$(1.10, 0.98, 0.72)$		2	105	$(1.11, 0.97, 0.71)$
	3	56	$(1.27, 1.06, 0.72)$		3	90	$(1.26, 1.07, 0.72)$
	4	60	$(1.21, 0.96, 0.59)$		4	112	$(1.20, 0.96, 0.59)$
10^3	5	90	$(1.11, 0.84, 0.50)$	10^4	5	126	$(1.11, 0.84, 0.49)$
	6	120	$(1.03, 0.73, 0.42)$		6	252	$(1.02, 0.73, 0.41)$
	7	180	$(0.95, 0.65, 0.36)$		7	270	$(0.94, 0.65, 0.36)$
	8	340	$(0.86, 0.58, 0.32)$		8	510	$(0.86, 0.58, 0.31)$
	2	153	$(1.11, 0.97, 0.71)$		2	210	$(1.11, 0.97, 0.71)$
	3	132	$(1.27, 1.06, 0.72)$		3	182	$(1.27, 1.07, 0.72)$
	4	180	$(1.20, 0.96, 0.59)$		4	220	$(1.20, 0.96, 0.59)$
10^5	5	216	$(1.11, 0.84, 0.49)$	10^6	5	270	$(1.11, 0.84, 0.49)$
	6	336	$(1.02, 0.73, 0.41)$		6	432	$(1.02, 0.73, 0.41)$
	7	378	$(0.94, 0.65, 0.36)$		7	648	$(0.94, 0.65, 0.36)$
	8	714	$(0.87, 0.58, 0.31)$		8	952	$(0.87, 0.58, 0.31)$
	2	300	$(1.11, 0.97, 0.71)$		2	378	$(1.11, 0.97, 0.71)$
	3	240	$(1.27, 1.06, 0.72)$		3	306	$(1.27, 1.06, 0.72)$
	4	312	$(1.20, 0.96, 0.59)$		4	420	$(1.20, 0.96, 0.59)$
10^7	5	396	$(1.11, 0.84, 0.49)$	10^8	5	468	$(1.11, 0.84, 0.49)$
	6	540	$(1.02, 0.73, 0.41)$		6	792	$(1.02, 0.73, 0.41)$
	7	810	$(0.94, 0.65, 0.36)$		7	990	$(0.94, 0.65, 0.36)$
	8	1224	$(0.87, 0.58, 0.31)$		8	1530	$(0.87, 0.58, 0.31)$

Table 6.6: List of values of the ratio MHL_k/r (for any n) corresponding to the varying ratio r/n for each k. For a given $k > 2$, the values in bold indicate the minimum value of r/n from where the scheme performs better than that for $k = 2$.

r/n \ k	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
2	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
3	(1.46,	1.37,	1.27,	1.06,	0.88,	0.72,	**0.57**,	0.43,	0.31,	0.20,	0.10,	0.00)
4	(1.47,	1.35,	1.20,	**0.96**,	0.76,	0.59,	0.47,	0.36,	0.27,	0.18,	0.10,	0.00)
5	(1.44,	1.28,	1.11,	**0.84**,	0.63,	0.49,	0.39,	0.31,	0.24,	0.17,	0.09,	0.00)
6	(1.41,	1.22,	**1.02**,	0.73,	0.54,	0.41,	0.33,	0.27,	0.21,	0.15,	0.09,	0.00)
7	(1.38,	**1.16**,	0.94,	0.65,	0.47,	0.36,	0.28,	0.23,	0.19,	0.14,	0.08,	0.00)
8	(1.34,	**1.11**,	0.87,	0.58,	0.41,	0.31,	0.25,	0.21,	0.17,	0.13,	0.08,	0.00)
16	(**1.22**,	0.78,	0.55,	0.31,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

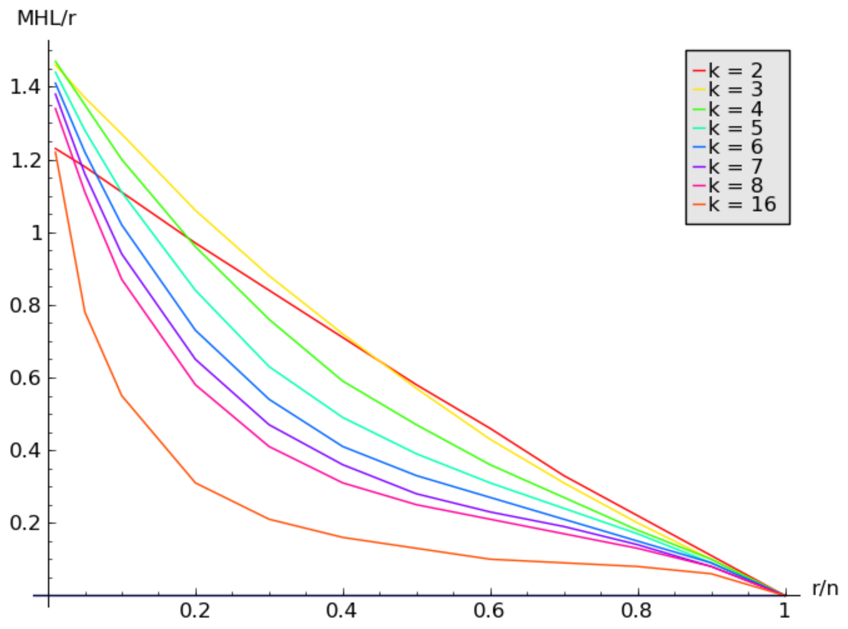


Figure 6.12: Plot showing how MHL_k/r varies with r/n.

Table 6.7: Values of the threshold δ_k.

k	3	4	5	6	7	8	16
δ_k	0.44	0.19	0.11	0.07	0.05	0.04	< 0.01

channel/program is 244 Mbyte per session. The user storage, on the other hand, increases from 5.9 Kbyte to 23.9 Kbyte. Due to steadily decreasing memory prices, the cumulative benefit of savings in communication bandwidth over a period of time is likely to outweigh the cost of extra memory.

6.8 Conclusion

The most popular BE scheme is the NNL-SD scheme described in Chapter 2 [NNL01, NNL02] that is defined on a binary tree structure. We present a generalization of the scheme which works with a k-ary tree for any $k \geq 2$. As a result, our work subsumes the NNL-SD scheme. We present detailed analysis of the user storage and the header length, the two important efficiency parameters of a BE scheme. This shows that if the number of revoked users is of the order of the number of total users, then using a k greater than 2 results in lower communication overhead at the cost of increased user storage. For applications where the increase in user storage can be tolerated, our work provides a wider variety of trade-off options between user storage and bandwidth.

Chapter 7

The Augmented Binary Tree Subset Difference Scheme

7.1 Introduction

Like in Chapter 6, our goal in this chapter is to explore methods to reduce the communication bandwidth in the NNL-SD scheme. We have already seen that the k-ary tree SD scheme reduces the communication overhead at the cost of increased storage. However, this reduction happens only for certain values of r such that the ratio r/n is greater than a threshold. In this chapter we propose a new scheme that reduces the expected header length for *all* values of r.

In Chapter 1 we discussed the basic combinatorial intuition behind reducing the header length of a BE scheme. If we can somehow manage to increase the number of subsets in $\mathcal{S}$, then it may become easier to cover the privileged users using a smaller number of subsets. We follow up on this intuition. In Chapter 6 header lengths were in general reduced by altering the structure of the underlying tree. In the schemes with larger arity of the underlying tree resulting in smaller header lengths, all subsets for a smaller arity were not necessarily included. Our goal in this chapter is to explore methods to include all subsets from a scheme while increasing the number of subsets in $\mathcal{S}$ in order to reduce the header length.

The new scheme that we introduce in this chapter uses the same underlying binary tree $\mathcal{T}^0$ as in the NNL-SD scheme. Additionally, we use small trees of height a rooted at internal nodes of $\mathcal{T}^0$ to identify additional subsets which are to be assigned keys. In the scheme, a is a parameter whose value is greater than or equal to 1. Accordingly, the new scheme is called as the a-augmented binary tree subset difference (a-ABTSD) scheme. For $a = 1$, the new scheme is the same as the NNL-SD scheme. For $a > 1$, the flexibility of having additional subsets arises. As a result, the new scheme is a proper generalization of the NNL-SD scheme.

For a scheme with n users, the user storage is still $O(\log^2 n)$. The difference with the NNL-SD scheme is that the constant in the big-oh notation is proportional to 2^{k-1} where

$k = 2^a$. So, for a fixed n, the a-ABTSD scheme is meaningful only if a is small. The worst case header length of the a-ABTSD scheme is $2r - 1$ (irrespective of the value of a) as in the case of the NNL-SD scheme. It has been shown though that for any particular set of revoked users, the header size of the new scheme is never more than that of the NNL-SD scheme.

The main gain in using the a-ABTSD scheme is the reduction in the average header length. It turns out that for all values of r, the average header length of the new scheme for $a > 1$ is lower than that of the NNL-SD scheme. The lowering effect of the header length becomes more pronounced as either r increases or as a increases. Our results show that in scenarios where reducing communication bandwidth is a major concern, the new scheme provides an attractive alternative to the NNL-SD scheme.

This work is under submission. The draft of the submitted version is available online at [BS14b].

7.1.1 Some Notation

We know that under the subset cover framework described in Chapter 2 [NNL01, NNL02], for a user u, $\mathcal{S}_u$ denotes the subsets in $\mathcal{S}$ which contain u, i.e., $\mathcal{S}_u = \{S : S \in \mathcal{S} \text{ and } u \in S\}$. For each broadcast session, the center knows the set of revoked users $\mathcal{R}$. It forms a *partition* $\mathcal{S}_c$ of the set of privileged users $\mathcal{N} \setminus \mathcal{R}$ using subsets in $\mathcal{S}$, i.e., $\mathcal{S}_c \subseteq \mathcal{S}$; for $S_1, S_2 \in \mathcal{S}_c$, $S_1 \cap S_2 = \emptyset$; and $\cup_{S \in \mathcal{S}_c} S = \mathcal{N} \setminus \mathcal{R}$. This set of subsets $\mathcal{S}_c$ is called the *subset cover* and the algorithm to find $\mathcal{S}_c$ is called the *cover generation or cover finding algorithm.*

A *full binary tree* $\mathcal{T}^0$ of height ℓ_0 forms the underlying structure for the NNL-SD scheme that has been described in Chapter 2. We recollect here that each user is associated with a unique leaf of $\mathcal{T}^0$. There are a total of $\ell_0 + 1$ levels in the tree $\mathcal{T}^0$. The leaf nodes are at level 0; any internal node is at level $\ell + 1$ if its children are at level ℓ. So, the root node is at level ℓ_0. By $\mathsf{level}(i)$ we denote the level number of the node i in the tree $\mathcal{T}^0$. If J is a set of nodes all of which are at the same level, we will denote this common level by $\mathsf{level}(J)$.

The Collection NNL-$\mathcal{S}$. For the NNL-SD scheme, let us denote the collection of subsets which are assigned keys by NNL-$\mathcal{S}$. Then

$$\mathsf{NNL}\text{-}\mathcal{S} = \{\mathcal{N}\} \cup \{S_{i,j} : i \text{ is a non-leaf node of } \mathcal{T}^0 \text{ and } j \text{ is a non-root node of } \mathcal{T}^i\}. \tag{7.1}$$

The size of the collection $\mathsf{NNL}\text{-}\mathcal{S}$ is $1 + \ell_0 2^{\ell_0+1} - 2^{\ell_0} + 1 = 2 + \ell_0 2^{\ell_0+1} - 2^{\ell_0}$.

Key Assignment to Subsets in $\mathsf{NNL}\text{-}\mathcal{S}$. A key K_0 is assigned to the subset $\mathcal{N}$. For key assignment to the other subsets in $\mathcal{S}$, a cryptographic hash function

$$G : \{0, 1, 2\} \times \{0, 1\}^m \to \{0, 1\}^m \tag{7.2}$$

is chosen by the center and is made available to all users in the system. Here m is the key-size of the underlying symmetric cipher. For $t = 0, 1, 2$, let $G_t(\cdot) \triangleq G(t, \cdot)$. Each subset $S_{i,j} \in \mathcal{S}$ is assigned a key as follows.

- Every internal node i in $\mathcal{T}^0$ is assigned a uniform random m-bit seed L_i.
- All non-root nodes j in the subtree $\mathcal{T}^i$ derive seeds from L_i in the following manner. Let $j = t_0, \ldots, t_p = i$ be the sequence of nodes in the path from j to i. Then for $\imath = p - 1, \ldots, 0$, $t_\imath = 2t_{\imath+1} + s_\imath$ where $s_\imath \in \{1, 2\}$. Define the label $L_{i,j}$ associated to $S_{i,j}$ to be $L_{i,j} \triangleq G_{s_0}(\cdots G_{s_{p-2}}(G_{s_{p-1}}(L_i)) \cdots)$.
- The key $K_{i,j}$ associated to the subset $S_{i,j}$ is defined to be $K_{i,j} \triangleq G_0(L_{i,j})$.

The Set I_u for a User u. For a user u consider the set $\mathsf{NNL}\text{-}\mathcal{S}_u$ of subsets in $\mathsf{NNL}\text{-}\mathcal{S}$ which contain u. If $S_{i,j}$ is such a subset, then i is an ancestor of the leaf node u and j is not an ancestor of u. The user u should be able to generate the keys of all such subsets and no more. User u is at level 0 and suppose i is at level ℓ. Further suppose $u = i_0, i_1, \ldots, i_\ell = i$ be the path from u to i. Let $j_1, \ldots, j_\ell$ be the siblings of $i_1, \ldots, i_\ell$ respectively. Corresponding to the ancestor i at level ℓ, user u is given the ℓ seeds $L_{i,j_1}, \ldots, L_{i,j_\ell}$. Since u has ℓ_0 ancestors, the total number of seeds given to u is $\ell_0(\ell_0 + 1)/2$ plus the key K_0 assigned to the set $\mathcal{N}$. This assignment of seeds to u was earlier explained in details in Section 2.1.1. Denote the set of all seeds given to u by $\mathsf{NNL}\text{-}I_u$, i.e.,

$$\begin{aligned}
&\mathsf{NNL}\text{-}I_u \\
&= \{K_0\} \cup \\
&\quad \{L_{i,j} : i \text{ is an ancestor of } u \\
&\qquad \text{and } j \text{ is the sibling of some node in the path from } u \text{ to } i\}.
\end{aligned} \tag{7.3}$$

It can be seen that from the seeds that u gets, it can derive the keys for all subsets to which it belongs and no more.

7.2 The a-Augmented Binary Tree Subset Difference Scheme

The a-Augmented Binary Tree Subset Difference (a-ABTSD) scheme is a generalization of the NNL-SD scheme. It assumes an underlying full binary tree $\mathcal{T}^0$ as in the case of the NNL-SD scheme and imposes additional structure on this tree. The size of the structure is determined by a parameter a. For $a = 1$, the scheme turns out to be the same as the NNL-SD scheme.

Underlying Structure. As in the case of the NNL-SD scheme, there are $n = 2^{\ell_0}$ users associated with the leaves of the underlying full binary tree $\mathcal{T}^0$. The nodes and levels are also numbered as in the NNL-SD scheme.

For ease of later description, we introduce a few notions. Suppose J_1 and J_2 are two sets of nodes of $\mathcal{T}^0$ such that there is a node $j \in J_1$ and nodes $j_1, j_2 \in J_2$ such that $J_1 \setminus \{j\} = J_2 \setminus \{j_1, j_2\}$ and j_1, j_2 are the two children of j. Then the set J_2 can be thought of as being obtained from J_1 by replacing $\{j_1, j_2\}$ by j. Call the operation of replacing j_1, j_2 by their parent j to be a *moving-up* step.

Given a set J, it is possible to repeatedly apply the moving-up operation to get a set J' such that the moving-up operation can no longer be applied on J. We call J' to be a *reduced set*. Given a set J, there is a unique reduced set which can be obtained by repeatedly applying the moving-up step.

Let $\mathcal{T}$ be a full binary tree and J be a non-empty subset of the leaf nodes of $\mathcal{T}$. If J is either singleton, or, J can be reduced to a singleton set using moving-up operation, then J is called a *simple* subset of $\mathcal{T}$; otherwise, J is called a *non-simple* subset of $\mathcal{T}$. Figure 7.1 and Figure 7.2 show examples of simple and non-simple subsets respectively. By $\mathcal{J}_s(\mathcal{T})$ we denote the set of all simple subsets of $\mathcal{T}$. Similarly, $\mathcal{J}_{ns}(\mathcal{T})$ denotes the set of all non-simple subsets of $\mathcal{T}$. Note that both $\mathcal{J}_s(\mathcal{T})$ and $\mathcal{J}_{ns}(\mathcal{T})$ consist of subsets of the set of leaf nodes of $\mathcal{T}$.

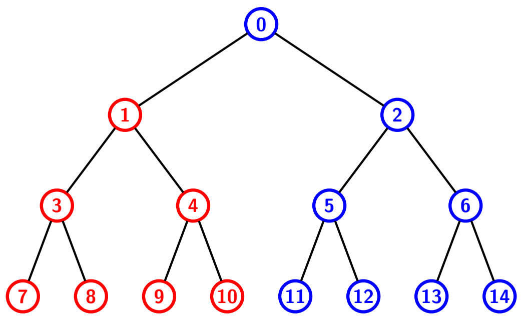


Figure 7.1: A full binary tree $\mathcal{T}$ with the set $J_1 = \{7, 8, 9, 10\}$ of leaf nodes that can be reduced to a singleton set $J_1' = \{1\}$. Hence, J_1 is a simple subset of $\mathcal{T}$.

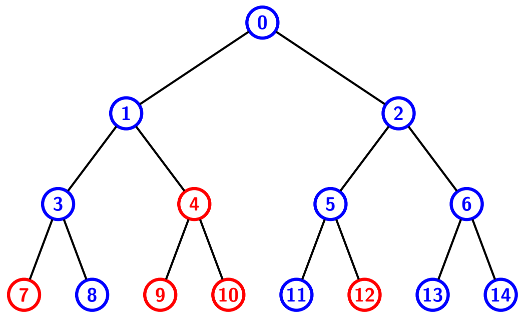


Figure 7.2: A full binary tree $\mathcal{T}$ where the set $J_2 = \{7, 9, 10, 12\}$ of leaf nodes may be reduced to $J_2' = \{7, 4, 12\}$ which is not singleton. Hence, J_2 is a non-simple subset of $\mathcal{T}$.

For the new scheme, $\mathcal{T}^0$ is endowed with an additional structure in the following manner. Define an *a-tree* $\mathcal{A}_a^j$ to be a subgraph of $\mathcal{T}^0$ which is the full binary tree rooted at node j and of height a. So, the number of nodes in an a-tree is $1 + 2 + \ldots + 2^a = 2^{a+1} - 1$. The scheme is parameterized by the number a.

We provide an example to illustrate this notion. In Figure 7.3 where $a = 2$, the subtree rooted at node 4 is the a-tree $\mathcal{A}_2^4$ containing the nodes $\{4, 9, 10, 19, 20, 21, 22\}$. Another a-tree $\mathcal{A}_2^1$ is the subgraph containing the nodes $\{1, 3, 4, 7, 8, 9, 10\}$.

For a fixed value of a in $\mathcal{T}^0$, each a-tree is uniquely identified by its root node. Alternatively, suppose J is a non-empty subset of leaf nodes of an a-tree $\mathcal{A}_a^j$ such that the nodes in J are at level ℓ (of $\mathcal{T}^0$). Then the root j is the unique ancestor at level $\ell + a$ of the nodes in J. So, given J, the node j is uniquely determined and we will call j to be the *a-pivot* of J.

The level number of the root node of any a-tree in $\mathcal{T}^0$ is at least a. Hence, for a full binary tree with $n = 2^{\ell_0}$ leaves, the number of distinct a-trees is the number of internal nodes at levels between ℓ_0 and a. Since there are $2^{\ell_0 - \ell}$ nodes at level ℓ in $\mathcal{T}^0$, hence the number of a-trees is

$$1 + 2 + \ldots + 2^{\ell_0 - a} = 2^{\ell_0 - a + 1} - 1.$$

For any internal node i of $\mathcal{T}^0$ and any non-root node j in $\mathcal{T}^i$, $\mathcal{T}^i \setminus \mathcal{T}^j$ is the subgraph of $\mathcal{T}^i$ obtained by taking away $\mathcal{T}^j$. We generalize this notion in the following manner. As before, let i be a non-leaf node in $\mathcal{T}^0$ and let $J = \{j_1, \ldots, j_c\}$ be a non-empty subset of non-root nodes in $\mathcal{T}^i$. Define $\mathcal{T}_{i,J}$ to be the subgraph of $\mathcal{T}^i$ formed by taking away all of

$\mathcal{T}^{j_1}, \ldots, \mathcal{T}^{j_c}$ from $\mathcal{T}^i$. In other words,

$$\mathcal{T}_{i,J} = \mathcal{T}^i \setminus \left(\mathcal{T}^{j_1} \cup \cdots \cup \mathcal{T}^{j_c}\right).$$

Let $S_{i,J}$ denote the set of leaf nodes of the subgraph $\mathcal{T}_{i,J}$.

Suppose J_1 and J_2 are two sets of nodes in $\mathcal{T}^i$ such that J_2 is obtained from J_1 by a moving-up step. Then it is easy to see that the set of leaf nodes of $\mathcal{T}_{i,J_1}$ is the same as the set of leaf nodes of $\mathcal{T}_{i,J_2}$ and so $S_{i,J_1} = S_{i,J_2}$. We say (i, J_1) and (i, J_2) are two representations of the set $S_{i,J_1} = S_{i,J_2}$. If J' is a reduced set obtained by successively applying the moving-up operation to a set J, then $S_{i,J} = S_{i,J'}$. By an extension of terminology, we will call the representation (i, J') to be the reduced form representation of the set $S_{i,J}$.

The Collection $\mathcal{S}$. Let i be an internal node of $\mathcal{T}^0$ and J be a non-simple subset of $\mathcal{A}_a^j$ where j is a node of $\mathcal{T}^i$. We call such a pair (i, J) to be *allowed*.

Suppose (i, J) is an allowed pair where the nodes in J are at level ℓ. Then the level of the a-pivot j of J is $\ell + a$ and so the level of i is at least $\ell + a$. This shows that there cannot be an allowed pair (i, J) where the level of i is less than a.

The collection $\mathcal{S}$ consists of the following subsets:

- all NNL-SD subsets $S_{i,j}$; and
- $S_{i,J}$ for all allowed pairs (i, J).

In other words,

$$\mathcal{S} \;=\; \mathsf{NNL}\text{-}\mathcal{S} \cup \mathcal{A}\text{-}\mathcal{S}, \tag{7.4}$$

where $\mathcal{A}\text{-}\mathcal{S} \stackrel{\Delta}{=} \{S_{i,J} : (i, J) \text{ is allowed}\}$.

For $S_{i,J} \in \mathcal{A}\text{-}\mathcal{S}$, J is non-simple and so J cannot be reduced to a singleton set using moving-up operations. As a result, $S_{i,J}$ is not equal to any NNL-SD subset. So, the collections $\mathsf{NNL}\text{-}\mathcal{S}$ and $\mathcal{A}\text{-}\mathcal{S}$ are disjoint.

If $a = 1$, then any J which is a non-empty subset of the leaf nodes of an a-tree is necessarily simple. So, there are no allowed pairs (i, J) showing that $\mathcal{A}\text{-}\mathcal{S} = \emptyset$. As a consequence, in this case, the a-ABTSD scheme collapses to the NNL-SD scheme.

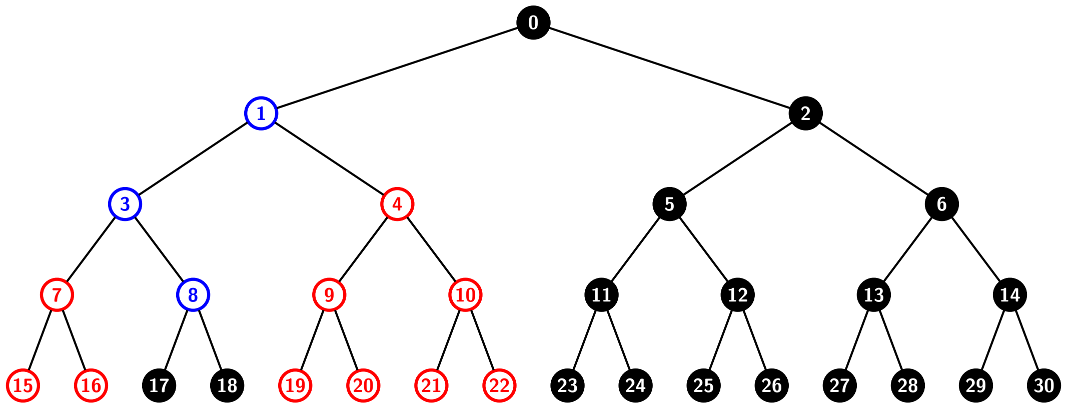


Figure 7.3: The binary tree $\mathcal{T}^0$ that is the underlying structure of the a-ABTSD scheme for $n = 16$ users is shown here. The red leaf nodes denote revoked users while the black ones denote privileged users. Here we assume $a = 2$. The subset $S_{0,\{7,9,10\}} = \{17, 18, 23, 24, \ldots, 30\}$ from the collection $\mathcal{S}$ ($\mathcal{A}$-$\mathcal{S}$ in particular) is also shown. It has all users in the subtree $\mathcal{T}^0$ but not in $\mathcal{T}^7 \cup \mathcal{T}^9 \cup \mathcal{T}^{10}$. Since $J = \{7, 9, 10\}$ is a non-simple subset of the a-tree $\mathcal{A}_2^1$, $(1, J)$ is an allowed pair. Using the moving up operation, the subset J may also be represented as $S_{0,\{7,4\}}$.

As an example, let us consider the tree $\mathcal{T}^0$ in Figure 7.3 with 16 users. It shows the subset that has been formed by excluding the users in $\mathcal{T}^7$, $\mathcal{T}^9$ and $\mathcal{T}^{10}$ from the users in $\mathcal{T}^0$. The subset is denoted as $S_{0,\{7,9,10\}}$. Nodes $\{7, 9, 10\}$ are leaves of the a-tree $\mathcal{A}_2^1$. Note that the set $\{7, 4\}$ can be obtained from the set $\{7, 9, 10\}$ by a moving-up operation. So, $S_{0,\{7,9,10\}} = S_{0,\{7,4\}}$.

Key Assignment to Subsets in $\mathcal{S}$. The key assignment strategy is an extension of the strategy for the NNL-SD scheme. The collection $\mathcal{S}$ consists of two sub-collections NNL-$\mathcal{S}$ and A-$\mathcal{S}$. We assume as in the case of the NNL-SD scheme that each internal node i of $\mathcal{T}^0$ is assigned an independent and uniform random m-bit seed L_i. Further, for any non-root j in $\mathcal{T}^i$, the seed $L_{i,j}$ is also defined using G_t as in the NNL-SD scheme and the key for the NNL-SD subset $S_{i,j}$ is $K_{i,j} = G_0(L_{i,j})$. In other words, keys to the subsets in NNL-$\mathcal{S}$ are assigned as in the NNL-SD scheme. For convenience of notation, we define $L_{i,i} \triangleq L_i$.

Let $\mathcal{T}$ be a full binary tree of height a and as defined earlier $\mathcal{J}_{ns}(\mathcal{T})$ is the set of all non-simple subsets of $\mathcal{T}$. We define a cryptographic hash function

$$H[\mathcal{T}] : \mathcal{J}_{ns}(\mathcal{T}) \times \{0,1\}^m \to \{0,1\}^m. \tag{7.5}$$

Keys to the subsets in $\mathcal{A}$-$\mathcal{S}$ are defined using the hash function H. Note that H is defined with respect to the tree $\mathcal{T}$. This is because the domain of H depends on $\mathcal{T}$. On the other hand, we expect H to act on any full binary tree of height a in the same manner. So, when $\mathcal{T}$ is clear from the context, we will write H instead of $H[\mathcal{T}]$.

Let $k = 2^a$ which is the number of leaf nodes in any a-tree. Suppose $S_{i,J}$ is in the collection $\mathcal{A}$-$\mathcal{S}$. Then (i, J) is an allowed pair and suppose the a-pivot of J is j. Then J is necessarily a non-simple subset of $\mathcal{A}_a^j$, i.e., $J \in \mathcal{J}_{ns}(\mathcal{A}_a^j)$. *The key $K_{i,J}$ assigned to $S_{i,J}$ is*

$$K_{i,J} \triangleq H[\mathcal{A}_a^j](J, L_{i,j}). \tag{7.6}$$

Note that j can be equal to i and in that case $L_{i,i}$ is simply L_i.

Number of Subsets in the Collection. As mentioned earlier, the count of the number of NNL-SD subsets is $2 + \ell_0 2^{\ell_0+1} - 2^{\ell_0}$. We now consider the number of subsets in $\mathcal{A}$-$\mathcal{S}$. The following result gives the number of simple and non-simple subsets of a full binary tree of height a.

Lemma 34. *Let $\mathcal{T}$ be a full binary tree of height a and $k = 2^a$. Then the number of simple subsets of $\mathcal{T}$, i.e. $|\mathcal{J}_s(\mathcal{T})|$ equals $2k - 1$. Consequently, the number of non-simple subsets of $\mathcal{T}$, i.e. $|\mathcal{J}_{ns}(\mathcal{T})|$, equals $2^k - 2k$.*

Proof. $\mathcal{T}$ has $k = 2^a$ leaf nodes and a total of $2k - 1$ nodes. If J is a simple subset of $\mathcal{T}$, then J is either a singleton subset of the set of leaf nodes of $\mathcal{T}$ or J can be reduced to one of the internal nodes of $\mathcal{T}$. So, the number of simple nodes of $\mathcal{T}$ is $2k - 1$. The total number of non-empty subsets of the leaf nodes of $\mathcal{T}$ is $2^k - 1$. Out of these $2k - 1$ are simple subsets. As a result, there are $2^k - 2k$ non-simple subsets of $\mathcal{T}$. $\square$

Fix a node i of $\mathcal{T}^0$ with $\mathsf{level}(i) = \ell$. Out of the $2^{\ell+1} - 1$ nodes in $\mathcal{T}^i$, $2^{\ell-a+1} + \ldots + 2^{\ell}$ nodes are at the bottom-most a levels. These nodes cannot be the a-pivot for any set J such that the pair (i, J) is allowed. Each of the remaining $2^{\ell-a+1} - 1$ nodes in $\mathcal{T}^i$ will be the root of an a-tree that generate subsets. For a node i, each such a-tree will generate $2^k - 2k$

subsets of the form $S_{i,J}$ where J is non-simple. Thus, the total number of subsets of the form $S_{i,J}$ in $\mathcal{A}$-$\mathcal{S}$ is

$$\sum_{\ell=a}^{\ell_0} 2^{\ell_0-\ell}(2^{\ell-a+1}-1)(2^k-2k-2) = (2^k-2k)((\ell_0-a)2^{\ell_0-a+1}-2^{\ell_0-a+1}+1).$$

Hence, the total number of subsets in the collection $\mathcal{S}$ is

$$\begin{aligned} |\mathcal{S}| &= |\mathsf{NNL}\text{-}\mathcal{S}| + |\mathcal{A}\text{-}\mathcal{S}| \\ &= 2+\ell_0 2^{\ell_0+1}-2^{\ell_0}+(2^k-2k)((\ell_0-a)2^{\ell_0-a+1}-2^{\ell_0-a+1}+1). \end{aligned} \quad (7.7)$$

I_u per User u. Let u be a user, i.e. a leaf node of $\mathcal{T}^0$. The information provided to u consists of two disjoint subsets which we call $I_u^{(1)}$ and $I_u^{(2)}$.

The Subset $I_u^{(1)}$. The first part is the same as that in the NNL-SD scheme, i.e., $I_u^{(1)} = \mathsf{NNL}\text{-}I_u$. Recall that $\mathsf{NNL}\text{-}I_u$ consists of seeds $L_{i,j}$ where i is an ancestor of u and j is the sibling of some node in the path from u to i. As mentioned earlier, the number of m-bit seeds in $I_u^{(1)}$ is $|I_u^{(1)}| = 1+\ell_0(\ell_0+1)/2$. From the seeds in $I_u^{(1)}$, u can derive keys of the following type:

- key $K_{i,j}$ corresponding to any NNL-SD subset $S_{i,j}$ containing u;
- key $K_{i,J}$ corresponding to any subset $S_{i,J}$ containing u such that the a-pivot of J is in the subtree rooted at the sibling of some node in the path from u to i.

The seeds in $I_u^{(1)}$ are not actual keys for subsets. These actual keys have to be derived from the seeds by one or more applications of the hash functions G and/or H.

The Subset $I_u^{(2)}$. Let $\mathcal{T}$ be a full binary tree of height a and v be a leaf node of $\mathcal{T}$. Let $\mathcal{J}_{ns,v}(\mathcal{T})$ denote the set of all non-simple sets of $\mathcal{T}$ not containing v. In other words, J is in $\mathcal{J}_{ns,v}(\mathcal{T})$ if J is a non-empty subset of the leaf nodes of $\mathcal{T}$, J cannot be reduced to singleton subset and $v \notin J$.

Lemma 35. *Let $\mathcal{T}$ be a full binary tree of height a and v be a leaf node of $\mathcal{T}$. Then* $|\mathcal{J}_{ns,v}(\mathcal{T})| = 2^{k-1}-2k+a+1$.

Proof. Consider a non-empty subset of the leaf nodes of $\mathcal{T}$ not containing v. Since $\mathcal{T}$ has k leaf nodes, there are a total of $2^{k-1} - 1$ possibilities for J. Further J cannot be reduced to any of the ancestors of v in $\mathcal{T}$. □

Define $\mathcal{S}_u^{(2)}$ to be collection of subsets $S_{i,J}$ in $\mathcal{A}$-$\mathcal{S}$ satisfying the following conditions:

- i is an ancestor of u and the a-pivot j of J is also an ancestor of u;
- the ancestor v of u at $\mathsf{level}(J)$ is not in J.

Define

$$I_u^{(2)} \quad = \quad \{K_{i,J} : S_{i,J} \text{ is in } \mathcal{S}_u^{(2)}\}. \tag{7.8}$$

The size of $I_u^{(2)}$ is calculated as follows. If i is at level ℓ, then the possible levels for the a-pivot j of J are $a, a+1, \ldots, \ell$. Fix a level ℓ' of j. We now need to find the number of non-simple subsets J satisfying the above conditions. There are $k = 2^a$ leaf nodes of $\mathcal{A}_a^j$. The ancestor v of u at level ℓ' is a leaf node of $\mathcal{A}_a^j$. By the above condition, v should not be in J and so there are $k-1$ leaf nodes of $\mathcal{A}_a^j$ which can be in J. Any subset J' of the leaf nodes of $\mathcal{A}_a^j$ which does not contain v cannot be reduced to any of the singleton nodes in the path from v to j (both inclusive). There are a total of $(2k-1)-(a+1)$ nodes in $\mathcal{A}_a^j$ to which it may be possible to reduce J' by applying moving-up operations. So, the number of J satisfying the required conditions is $2^{k-1} - 1 - (2k - a - 2)$. For a node i at level ℓ, there are $(\ell - a + 1)$ possible choices for j and for each j there are $2^{k-1} - 2k + a + 1$ choices for J. So, the number of keys in $I_u^{(2)}$ is

$$\begin{aligned} |I_u^{(2)}| &= \sum_{\ell=a}^{\ell_0} (\ell - a + 1)(2^{k-1} - 2k + a + 1) \\ &= \frac{1}{2} \times (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1). \end{aligned} \tag{7.9}$$

Recall that for a user u, $\mathcal{S}_u$ denotes the collection of subsets in $\mathcal{S}$ which contain u. Also, NNL-$\mathcal{S}_u$ denotes the collection of all NNL-SD subsets which contain u. Define $\mathcal{A}$-$\mathcal{S}_u$ to be the collection of all subsets from $\mathcal{A}$-$\mathcal{S}$ which contain u. Then $\mathcal{S}_u$ is the disjoint union of NNL-$\mathcal{S}_u$ and $\mathcal{A}$-$\mathcal{S}$. The set $I_u^{(1)}$ provides u with information to generate keys for any subset in NNL-$\mathcal{S}_u$. Similarly, the set $I_u^{(2)}$ provides u with information to generate keys for any subset in $\mathcal{A}$-$\mathcal{S}$. Further, the two sets $I_u^{(1)}$ and $I_u^{(2)}$ are disjoint and their union is the set I_u which provides u with information to generate keys for any subset in $\mathcal{S}_u$. The total number of m-bit seeds that u needs to store is the cardinality of I_u and is given by the following.

$$\begin{aligned} |I_u| &= |I_u^{(1)}| + |I_u^{(2)}| \\ &= 1 + \frac{\ell_0(\ell_0+1)}{2} + \frac{(2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \end{aligned} \tag{7.10}$$

For a fixed k and as n grows, the expression in (7.10) is $O(\log^2 n)$ which is the same as that of the NNL-SD scheme. This is much better than the number of keys being proportional to n. On the other hand, for a fixed n as k increases, the number of keys also increases. The set $I_u^{(2)}$ consists of actual keys for the subsets in $\mathcal{S}_u^{(2)}$. Later we show how to define the hash function H such that the definition of $I_u^{(2)}$ can be altered to provide information using which seeds in $\mathcal{S}_u^{(2)}$ can be derived. This results in decreasing the factor $(2^{k-1} - 2k + a + 1)$ in the above expression.

7.3 Cover Finding Algorithm

The algorithm takes as input the set $\mathcal{R}$ of revoked users and outputs the subset cover $\mathcal{S}_c$. If $\mathcal{R} = \emptyset$ then the only set in the subset cover is the set $\mathcal{N}$ of all users. If $\mathcal{R} \neq \emptyset$, then the subset cover consists of NNL-SD subsets $S_{i,j}$ or $S_{i,J}$ for allowed pairs (i, J). The subset cover algorithm that we describe below identifies NNL-SD subsets $S_{i,j}$ with $S_{i,\{j\}}$. For any allowed pair (i, J), the algorithm obtains $S_{i,J'}$ where J' is the reduced form of J.

The algorithm runs iteratively and maintains a list $\mathcal{L}$ of nodes on the paths joining revoked leaf nodes with the root. The list $\mathcal{L}$ is initially populated with the revoked leaf nodes, all marked as covered. The algorithm runs from left to right on this list and keeps adding the parent nodes of each node in the list until the root. Each node j in the list has an associated list $\mathsf{SDnodes}[j]$ of its descendant nodes. For a node j at level $\mathsf{level}(j) \geq a$, the nodes in $\mathsf{SDnodes}[j]$ are in an a-tree rooted at j or at some descendant of j. For a node j at level $\mathsf{level}(j) < a$, the list $\mathsf{SDnodes}[j]$ will have nodes from the subtree $\mathcal{T}^j$. While investigating the child nodes of i in the list, $\mathsf{SDnodes}[i]$ and the status of i are updated. The algorithm works as follows.

Algorithm $\mathcal{C}$. Takes as input the set $\mathcal{R} \neq \emptyset$ of revoked users and outputs the subset cover $\mathcal{S}_c$. Each subset in $\mathcal{S}_c$ is in reduced form.

1. Form the initial list $\mathcal{L}$ with all revoked leaf nodes of $\mathcal{T}^0$. Mark each node j as $\mathsf{covered}$ and set $\mathsf{SDnodes}[j] = \{j\}$. Set $\mathcal{S}_c$ to be the empty set.
2. Process nodes in $\mathcal{L}$ from left to right. Let $\mathcal{L}[t]$ be the node that is processed at the t^{th} iteration. If $\mathcal{L}[t]$ is the root node, go to step 3. Let i be the parent of $\mathcal{L}[t]$. At the t^{th} iteration:
   (a) If $\mathcal{L}[t]$ and $\mathcal{L}[t+1]$ have the same parent, proceed to the next iteration for $\mathcal{L}[t+1]$.
   (b) Else, append i to $\mathcal{L}$. Node i can have at most two children in $\mathcal{L}$. Let the children of i in $\mathcal{L}$ be $\{j_1, j_c\}$ where $(1 \leq c \leq 2)$. The following mutually exclusive cases occur:
      i. Case when all c children of i are $\mathsf{covered}$:
         A. If $c = 1$, mark i as $\mathsf{intermediate}$ and set $\mathsf{SDnodes}[i] = \{j_1\}$.
         B. For $c = 2$, mark i as $\mathsf{covered}$ and set $\mathsf{SDnodes}[i] = \{i\}$.
      ii. Case when $c = 1$ and j_1 is $\mathsf{intermediate}$:
         Mark i as $\mathsf{intermediate}$ and copy $\mathsf{SDnodes}[j_1]$ to $\mathsf{SDnodes}[i]$.
      iii. Case when $c = 2$ and at least one node in $\{j_1, j_2\}$ is $\mathsf{intermediate}$:
         A. If for some $j \in \{j_1, j_2\}$, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\mathsf{level}(j) - \mathsf{level}(j') \geq a$, then for each $j \in \{j_1, j_2\}$ that is marked as $\mathsf{intermediate}$, add $S_{j,\mathsf{SDnodes}[j]}$ to $\mathcal{S}_c$. Subsequently, mark i as $\mathsf{covered}$ and set $\mathsf{SDnodes}[i] = \{i\}$.
         B. Otherwise, mark i as $\mathsf{intermediate}$ and set $\mathsf{SDnodes}[i]$ to $\mathsf{SDnodes}[j_1] \cup \mathsf{SDnodes}[j_2]$.
3. If the root node is marked as $\mathsf{intermediate}$, add $S_{0,\mathsf{SDnodes}[0]}$ to the cover $\mathcal{S}_c$.

The subset cover $\mathcal{S}_c$ output by the algorithm is a collection of subsets of the form $S_{i,\mathsf{SDnodes}[i]}$.

Figure 7.4 shows an example where $a = 2$, $n = 32$ and $\mathcal{R} = \{31, 33, 39, 43\}$. Hence, the list $\mathcal{L}$ eventually gets populated with the nodes $\{31, 33, 39, 43, 15, 16, 19, 21, 7, 9, 10, 3, 4, 1, 0\}$ that lie on the paths joining the revoked leaves with the root node. The subsets generated by the algorithm working on the above list are $S_{9,\{39\}}$, $S_{10,\{43\}}$, $S_{3,\{31,33\}}$ and $S_{0,\{1\}}$.

The cover generation algorithm outputs sets of the type $S_{j,\mathsf{SDnodes}[j]}$. To show the correctness of the algorithm we need to argue two things.

1. Each subset produced by Algorithm $\mathcal{C}$ is in $\mathcal{S}$.
2. The subsets that are produced form a partition of the set of privileged users.

Lemma 36. *If Algorithm $\mathcal{C}$ produces a subset $S_{i,J}$, then every element of J has been marked* covered.

Proof. J is of the form $\mathsf{SDnodes}[j]$ for some node j. Further, all nodes in $\mathsf{SDnodes}[j]$ are marked covered. This can be seen from the manner in which the $\mathsf{SDnodes}[j]$ is built up. Nodes enter $\mathsf{SDnodes}[j]$ either in Step 1 or in Step 2(b)(i) and in both cases they are marked covered; the set $\mathsf{SDnodes}[j]$ grows in Step 2(b)(iii)(B) through the union of two other sets of the same type and hence the property of having only covered nodes is preserved. □

Lemma 37. *If a subset $S_{i,J}$ is produced by Algorithm $\mathcal{C}$, then J is a reduced set.*

Proof. All nodes in J are marked covered. Let if possible j_1 and j_2 be siblings in J and i is their parent. Then both j_1 and j_2 are marked as covered. When the node i is considered in Step 2(b), then c is 2 and Step 2(b)(i)(B) is executed which results in $\mathsf{SDnodes}[i]$ being set to $\{i\}$ and j_1, j_2 do not enter any $\mathsf{SDnode}[i]$. So, they cannot be members of any J such that $S_{i,J}$ is produced by Algorithm $\mathcal{C}$ at a later point of time. □

Lemma 38. *For any set $\mathsf{SDnodes}[j]$, if $i_1, i_2 \in \mathsf{SDnodes}[j]$, then $\mathsf{level}(i_1) - \mathsf{level}(i_2) < a$. Further, all nodes of $\mathsf{SDnodes}[j]$ belong to some a-tree.*

Proof. Let $J = \mathsf{SDnodes}[j]$. If J is a singleton set, then this is clearly true; if J contains more than one element, then J must have been formed by the merger of two $\mathsf{SDnodes}$ set in Step 2(b)(iii)(B). Such merger can take place only if the maximum of the differences in the levels of the nodes in the resulting set is less than a.

For the last statement, again it is easy to see this if J is a singleton set. On the other hand, if J has been formed by merger (one or more times), then each such merger is a union of the $\mathsf{SDnodes}$ of two siblings. Consequently, this corresponds to a moving-up operation within the same a-tree. □

Lemma 39. *Any subset produced by Algorithm $\mathcal{C}$ is in the collection $\mathcal{S}$.*

Proof. Suppose $S_{j,\mathsf{SDnodes}[j]}$ is produced. Then all the nodes in $J = \mathsf{SDnodes}[j]$ are in the subtree rooted at j. By Lemma 38, the nodes in J are in some a-tree and by the previous statement, the root of this a-tree is also in $\mathcal{T}^j$. So, $S_{j,J}$ is in $\mathcal{S}$. □

Lemma 40. *If u is a leaf node corresponding to a revoked user, then Algorithm $\mathcal{C}$ visits all ancestors of u.*

Proof. Whenever a node i is processed by Algorithm $\mathcal{C}$, its parent is added to $\mathcal{L}$. Further, every node in $\mathcal{L}$ is processed before the algorithm terminates. Since the initial list $\mathcal{L}$ contains the node u, every ancestor of u is processed by Algorithm $\mathcal{C}$. □

Lemma 41. *Any privileged (i.e., non-revoked) user is in one of the subsets produced by Algorithm $\mathcal{C}$.*

Proof. Let v be a privileged user. Since there is at least one revoked user, there is a minimal subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ which contains both v and some revoked user u. Let j_1 and j_2 be the two children of i and suppose v is a leaf node of $\mathcal{T}^{j_2}$. By the minimality of $\mathcal{T}^i$, it follows that u is necessarily in $\mathcal{T}^{j_1}$ and further all leaf nodes of $\mathcal{T}^{j_2}$ are privileged.

Since i is an ancestor of the revoked node u, by the previous lemma, Algorithm $\mathcal{C}$ will process both nodes i_1 and i. The node i is added to $\mathcal{L}$ when node i_1 is processed. Since all nodes in $\mathcal{T}^{i_2}$ are privileged, node i_2 does not enter $\mathcal{L}$. So, i has exactly one child in $\mathcal{L}$ and either by Step 2(b)(i)(A) or by Step 2(b)(ii), i is marked $\mathsf{intermediate}$ and $\mathsf{SDnodes}[i]$ is set to either $\{j_1\}$ or to $\mathsf{SDnodes}[j_1]$. In both cases, v is in $S_{i,\mathsf{SDnodes}[i]}$. From this point onwards, Algorithm $\mathcal{C}$ ensures the following. If i' is an ancestor of i, then either the set $S_{i',\mathsf{SDnodes}[i']}$ is produced, or, $S_{i',\mathsf{SDnodes}[i']}$ contains v. Since, the second case cannot continue indefinitely, at some point of time, Algorithm $\mathcal{C}$ will produce a set $S_{i',\mathsf{SDnodes}[i']}$ for some ancestor i' of i and so v will be in this subset. □

From Lemmas 40 and 41, we get the following result on the correctness of Algorithm $\mathcal{C}$.

Theorem 42. *Algorithm $\mathcal{C}$ produces a sub-collection of subsets of $\mathcal{S}$ which form a partition of the set of privileged users.*

The complexity of Algorithm $\mathcal{C}$ is given by the following result.

Theorem 43. *Algorithm $\mathcal{C}$ requires $O(r \log n)$ time where r is the number of revoked nodes.*

Proof. As proved in Lemma 40, the algorithm processes every ancestor of any revoked node. There are $O(\log n)$ such ancestors and so the total time taken by the algorithm is proportional to $r \log n$. $\square$

It has already been remarked that for $a = 1$, the a-ABTSD scheme collapses to the NNL-SD scheme. The following result shows that for $a > 1$ and any revocation pattern, the header length of the a-ABTSD scheme is never more than that of the NNL-SD scheme.

Theorem 44. *For a given $\mathcal{R}$ (revocation pattern) the header length due to the NNL-SD scheme is at least as large as that of the a-ABTSD scheme.*

Proof. For a given value of a, let $\mathcal{J}_a$ be the collection of all nodes j in $\mathcal{T}^0$ such that $S_{j,\mathsf{SDnodes}[j]} \in \mathcal{S}_c$. Let us consider a node i in $\mathcal{T}^0$ that have both children $\{j_1, j_2\}$ in $\mathcal{L}$ and at least one of them is marked as $\mathsf{intermediate}$. When $a = 1$, for every $\mathsf{intermediate}$ child j of i, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\ell_j - \ell_{j'} \geq 1$. Hence, $S_{j,\mathsf{SDnodes}[j]} \in \mathcal{S}_c$ and hence $j \in \mathcal{J}_{a=1}$. For $a > 1$, if for some $j \in \{j_1, j_2\}$, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\ell_j - \ell_{j'} \geq a$, only then all intermediate children of i generate SD subsets. Otherwise, i is marked as $\mathsf{intermediate}$ and $\mathsf{SDnodes}[j]$ is included in $\mathsf{SDnodes}[i]$ and is carried upwards. Hence, $\mathcal{J}_{a=1} \subseteq \mathcal{J}_{a>1}$. Thus, the header length due to a revocation pattern for the a-ABTSD scheme will be at most that of the NNL-SD scheme. $\square$

It follows from Theorem 44 above that the worst case header length for the a-ABTSD scheme will be less than or equal to that of the NNL-SD scheme. From [NNL01, NNL02] we know that for a given r, the worst case header length of the NNL-SD scheme is $2r - 1$. Hence we get the following theorem.

Theorem 45. *For a given r in the a-ABTSD scheme, the maximum header length that can be achieved for any n, is $2r - 1$.*

To show that this upper bound is tight, we consider the a-ABTSD scheme with $a = 2$ for $n = 32$ users in Figure 7.5 where $\mathcal{R} = \{31, 39\}$. The subset cover for this revocation pattern is $\mathcal{S}_c = \{S_{3,\{31\}}, S_{4,\{39\}}, S_{0,\{1\}}\}$. Hence, the header length is $2|\mathcal{R}| - 1 = 3$. A similar example can be constructed to show the tightness of this upper bound for any general value of a with larger values of n. The subtrees rooted at nodes $3, 4, 5$ and 6 in Figure 7.5 where $a = 2$, are of height $a + 1 = 3$ each. For any general a, these subtrees should be full subtrees of height $a + 1$ each. It is to be noted that the tree $\mathcal{T}^0$ in such a case will be of height $a + 3$ and the total number of users will be 2^{a+3}. There will be two revoked users, one in each of the subtrees rooted at nodes 3 and 4. The subset cover will have three subsets. Two of these subsets will be rooted at nodes 3 and 4. The third subset will be $S_{0,\{3,4\}} = S_{0,\{1\}}$. Hence, the upper bound given by Theorem 45 is tight for any $a \geq 1$.

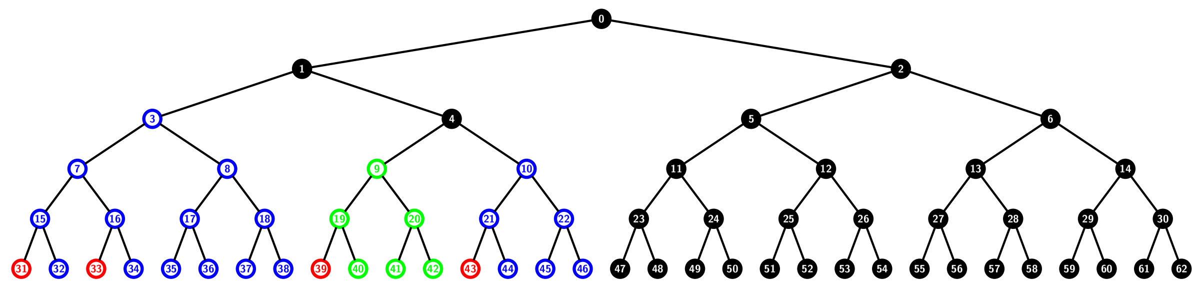


Figure 7.4: Example of a subset cover for $\mathcal{R} = \{31, 33, 39, 43\}$ in the a-ABTSD scheme with $a = 2$ and $n = 32$ users. The subsets in the cover are $S_{3,\{31,33\}}$, $S_{9,\{39\}}$, $S_{10,\{43\}}$ and $S_{0,\{1\}}$.

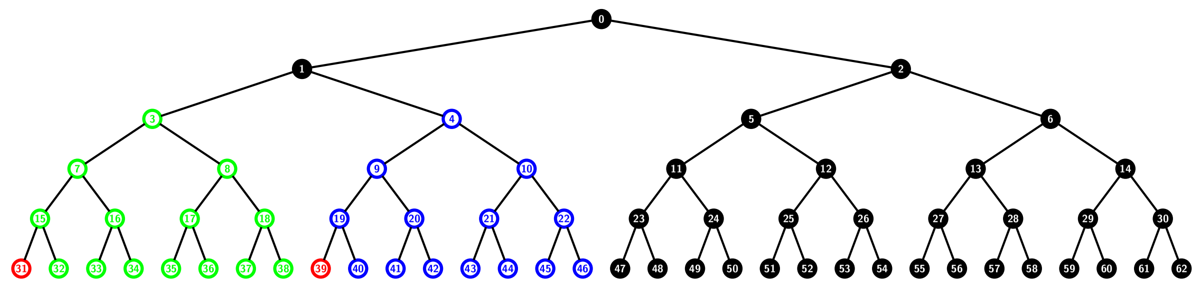


Figure 7.5: Example to show that the upper bound $2r - 1$ of the header length in the a-ABTSD scheme with $a = 2$ is tight. The subset cover for $\mathcal{R} = \{31, 39\}$ in the binary tree $\mathcal{T}^0$ with $n = 32$ users contains the subsets $S_{3,\{31\}}$, $S_{4,\{39\}}$ and $S_{0,\{1\}}$.

7.4 Other Issues

In this section, we consider two issues. The first one is the ability to extend the scheme to handle arbitrary number of users and the second one is the issue of traitor tracing.

7.4.1 Accommodating an Arbitrary Number of Users

We know from Chapter 2 that the NNL-SD [NNL01, NNL02] scheme assumes the number n of users to be a power of two. The a-ABTSD scheme retains this assumption and hence assumes an underlying full binary tree. In practice this may be restrictive. We extend the a-ABTSD scheme for an arbitrary number of users by assuming a *complete* binary tree instead of full. A complete binary tree with $2^{\ell_0-1} < n \leq 2^{\ell_0}$ leaves is formed by adding child nodes to the leaf nodes of a full tree with 2^{ℓ_0-1} leaf nodes, starting from the left. These newly added leaves are said to be at level 0. The old leaves are at level 1. The newly constructed complete tree has n leaves, some of which are filled from the left of level 0 and the others (if $2^{\ell_0-1} < n < 2^{\ell_0}$) are on the right at level 1.

Since the underlying tree $\mathcal{T}^0$ is a complete tree (that may not be full) and hence an a-tree may also be a non-full complete binary tree. Thus, an a-tree $\mathcal{A}_a^i$ is a complete tree rooted at node i in $\mathcal{T}^0$ and is of height a. Let us call the path joining the root node and the right-most internal node at level 1 to be the *dividing path*. Any subtree of $\mathcal{T}^0$ rooted at a node other than the dividing path, is full. Hence, only the a-tree rooted at the node on the dividing path at level a may be non-full. The subsets that are included in the collection $\mathcal{S}$ are formed as before. A subset $S_{i,J} \in \mathcal{S}$ is such that all nodes in J are within a single (possibly non-full but complete) a-tree.

The user storage requirement of the a-ABTSD scheme assuming $n = 2^{\ell_0}$ is given by (7.10) where ℓ_0 is the height of the underlying tree. Let us denote this storage requirement as $\mathsf{us}_a(2^{\ell_0})$. Then the user storage of the scheme assuming the complete tree structure will be at least $\mathsf{us}_a(2^{\ell_0-1})$ and at most $\mathsf{us}_a(2^{\ell_0})$, depending on where a user is placed in the tree with respect to the dividing path. All users are attached to some node on the dividing path. Users that are to the left (respectively right) of the dividing path and are attached to it at nodes on or above level a, receive $\mathsf{us}_a(2^{\ell_0})$ seeds (respectively $\mathsf{us}_a(2^{\ell_0-1})$ seeds). For the users that are attached to the dividing path at a level less than a, the number of seeds can be easily calculated from the number of users attached to the dividing path at those levels.

The cover generation algorithm for the complete tree version of the scheme would have an additional pre-processing step for the leaf nodes at level 0. First, all the revoked leaf nodes at level 0 are inserted into the list $\mathcal{L}$ in left-to-right order. These nodes are processed one after another as in the cover generation algorithm. The parent of each leaf in $\mathcal{L}$ gets appended to it and their respective data structures are appropriately updated. Once all revoked leaves at level 0 have been processed, all their parents at level 1 are in the list. The remaining revoked leaf nodes that are at level 1 in $\mathcal{T}^0$, are then appended to $\mathcal{L}$. Then onwards, the cover generation algorithm proceeds exactly as it did for full trees. The worst-case header length remains $2r - 1$ for the complete tree version of the scheme. We have implemented this algorithm and results are reported later.

7.4.2 Traitor Tracing

From the discussion on traitor tracing of Chapter 2 we know that the bifurcation property states that *given any subset that is in the collection $\mathcal{S}$ and hence has been assigned a key, it is possible to partition the set into two (or a constant number of) almost equal subsets from $\mathcal{S}$*. The *bifurcation value* is defined to be the ratio of the size of the largest subset to that of the set itself.

For the a-ABTSD scheme that we have proposed in this work, keys are assigned to subsets that are in general different from those in the NNL-SD scheme. Hence, the traitor tracing for these schemes do not directly follow from the NNL-SD traitor tracing algorithm. However, the subsets of this scheme do follow the bifurcation property. Here we state very briefly how these subsets can be split into roughly equal sized subsets from their respective collection $\mathcal{S}$.

In the a-ABTSD scheme, the subsets in the collection $\mathcal{S}$ are of the forms $S_{i,j}$ or $S_{i,J}$. Any subset of the form $S_{i,j}$ can also be written as $S_{i,J}$ where J is a simple subset of $\mathcal{A}_a^j$. Assume that all subsets in $\mathcal{S}$ are of the form $S_{i,J}$ where J is a non-empty subset of the leaf nodes of $\mathcal{A}_a^j$ for some j in the subtree rooted at i. Subsets where $J = \{j\}$ is a singleton set are split into two as was done in Chapter 2 for the NNL-SD scheme. The node j will be in either of the two subtrees rooted at $2i + 1$ or $2i + 2$. If j is in $\mathcal{T}^{2i+1}$, the subsets after split will be $S_{2i+1,j}$ and $S_{i,2i+1}$. If j is in $\mathcal{T}^{2i+2}$, the subsets after split will be $S_{2i+2,j}$ and $S_{i,2i+2}$. Hence, the maximum bifurcation value in this case is $2/3$.

For the subsets $S_{i,J}$ where $|J| > 1$, let us consider the a-tree $\mathcal{A}_a^i$ rooted at node i. The a-tree $\mathcal{A}_a^j$ containing the nodes in J is either this same a-tree (when $i = j$) or it is rooted at a descendant j of i. In any case, the subsets formed by the split are as follows. The subtrees rooted at leaves of $\mathcal{A}_a^i$ form a subset each in the split. From each of these 2^a subtrees, all users under nodes in J are excluded. As a result, some of these 2^a subtrees may be completely excluded. When $i = j$, the maximum bifurcation value is $1/(2^a - |J|)$ which in the worst case would be $1/2$. In case j is in the subtree of i, the nodes in J will be contained in at least one (but not all) of the 2^a subtrees under the a-tree $\mathcal{A}_a^i$. The users in the subtrees of J are excluded from the respective subtrees at the leaves of $\mathcal{A}_a^i$. Since j is in the subtree of i, one of the child subtrees of i would not have any node in J. There will be at least 2^{a-1} subtrees at the leaves of $\mathcal{A}_a^i$ that will not have any node in J. As a result, the bifurcation value in this case will be between $1/2^{a-1}$ and $1/2^a$. This goes to show that the bifurcation property also holds for subsets in the a-ABTSD scheme. Hence, traitor tracing mechanisms can be devised for the scheme introduced in this work in a manner similar to the one described in Chapter 2 [NNL01, NNL02].

The number of queries required by the traitor tracing algorithm depends on the bifurcation value. At every step of the traitor tracing algorithm, a subset S of users that contains a traitor is divided into subsets $S_1, \ldots, S_t$ using the bifurcation property as mentioned above. Each subset S_t is tested for containment of a traitor. The ratio $|S_t|/|S|$ is at most the bifurcation value. The size of the remaining subset from which the traitors have to be traced reduces with the bifurcation value. The bifurcation value of the NNL-SD scheme is $2/3$. The bifurcation value of the a-ABTSD scheme is at most $2/3$ for $a \geq 2$. Hence, traitor tracing in the a-ABTSD scheme will be at least as efficient as the NNL-SD scheme, if not better on an average.

7.5 Reducing User Storage

A user u is provided with the set I_u as secret information. This set is the union of two disjoint sets $I_u^{(1)}$ and $I_u^{(2)}$ where

$$|I_u^{(1)}| = 1 + \ell_0(\ell_0 + 1)/2$$

and

$$|I_u^{(2)}| = (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2.$$

So the user storage is $|I_u| = 1 + \ell_0(\ell_0+1)/2 + (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ where $k = 2^a$ (see (7.10)). For a given ℓ_0, the quantity $|I_u^{(1)}| = 1 + \ell_0(\ell_0+1)/2$ is fixed and does not change with the value of a. As the value of a increases, the component $|I_u^{(2)}| = (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ increases. The main increase is due to the exponential factor 2^{k-1} which is actually doubly exponential in a. Here we describe a technique to somewhat mitigate this increase. For small concrete values of a, the decrease in user storage is quite significant.

Recall that the information provided in $I_u^{(2)}$ is used by u to generate keys for the subsets in $\mathcal{A}$-$\mathcal{S}_u$. For a specified value of a, the new key generation method will provide a user u with a different set, to be denoted $\mathrm{II}_u^{(2)}(a)$, which will enable u to generate keys for the subsets in $\mathcal{A}$-$\mathcal{S}_u$.

It is to be noted that the technique for decreasing user storage described in this section does not change the definition of the collection $\mathcal{S}$ of subsets to which keys are assigned in the a-ABTSD scheme. Hence, the cover generation algorithm remains the same. Only the method of assigning seeds to nodes and keys to SD subsets is altered.

Suppose the number of users is n. Then as discussed earlier, the user storage is not the same for all users. Denote by $\mathsf{us}_a(n)$ the maximum user storage with n users, i.e., $\mathsf{us}_a(n) = \max_u |I_u|$. For $2^{\ell_0-1} < n \leq 2^{\ell_0}$, $\mathsf{us}_a(n) = \mathsf{us}_a(2^{\ell_0})$.

7.5.1 The Basic Idea

Consider a subset $S_{i,J}$ for an allowed pair (i,J). Let j be the a-pivot of J. Then J is a non-simple subset of the set of leaf nodes of $\mathcal{A}_a^j$, i.e., $J \in \mathcal{J}_{ns}(\mathcal{A}_a^j)$. The key $K_{i,J}$ is assigned to $S_{i,J}$ using the hash function H as $K_{i,J} = H[\mathcal{A}_a^j](J,L)$ where j is the a-pivot of J and $L = L_{i,j}$ (7.6). Let u be a user and consider the set $I_u^{(2)}$. The key $K_{i,J}$ is in $I_u^{(2)}$ if the following condition holds: the a-pivot j of J is an ancestor of u and the ancestor v of u at $\mathsf{level}(J)$ is not in J.

Let $\mathcal{T}$ be a full binary tree of height a having $k = 2^a$ leaf nodes. Any subset J of the leaf nodes of $\mathcal{T}$ can be encoded by a k-bit string $\mathsf{str}(J)$ where the $\imath$-th bit from the left of $\mathsf{str}(J)$ is 1 if and only if the $\imath$-th leaf node of $\mathcal{T}$ is in J. By extension of this notation, $\mathsf{str}(\mathcal{J}_{ns}(\mathcal{T}))$ denotes the set of k-bit strings encoding the non-simple subsets of $\mathcal{T}$. Define

$$H : \mathsf{str}(\mathcal{J}_{ns}(\mathcal{T})) \times \{0,1\}^m \to \{0,1\}^m. \qquad (7.11)$$

For $\sigma \in \mathsf{str}(\mathcal{J}_{ns}(\mathcal{T}))$ and $L \in \{0,1\}^m$ define $L_\sigma = H(\sigma, L)$. If w is a leaf node of $\mathcal{T}$, define $\mathsf{keys}[L, \mathcal{T}](w)$ to be the set of all L_σ such that the w-th bit of σ is 0.

Let i be an internal node of $\mathcal{T}^0$ and j be a node of $\mathcal{T}^i$. Let v be a leaf node of the a-tree $\mathcal{A}_a^j$. The seed $L_{i,j}$ is the derived seed from L_i which is assigned to the node j. Let w be a leaf node of $\mathcal{A}_a^j$. The keys in $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](w)$ are to be made available to users in $\mathcal{T}^w$. This is captured by the following definition.

Using the definition of H in (7.11), the key $K_{i,J}$ for the subset $S_{i,J}$ is defined to be

$$K_{i,J} \quad = \quad H(\mathsf{str}(J), L_{i,j}), \tag{7.12}$$

where as before, j is the a-pivot of J. Suppose u is a user. Then the set $I_u^{(2)}$ is the following.

$$I_u^{(2)} \quad = \quad \bigcup_i \bigcup_j \mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v), \tag{7.13}$$

where i is an ancestor of u; j is node on the path from u to i and $\mathsf{level}(j) \geq a$; v is the ancestor of u at level $\mathsf{level}(j) - a$.

Our basic idea of reducing key storage is that instead of directly providing $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v)$ we provide sufficient information for the keys in this set to be computed. This is achieved by defining the function H in a different manner. Note that the function H can itself be defined with respect to a full binary tree $\mathcal{T}$ of height a and without reference to the tree $\mathcal{T}^0$. Once H is defined, the definition of $K_{i,J}$ follows and the set $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v)$ is also obtained from the definition of $\mathsf{keys}[L, \mathcal{T}](w)$.

In the rest of this section, we show how to define suitable H. In the next subsection, we describe this method for the special case of $a = 2$ and in the subsequent subsection we consider the case of general a.

7.5.2 The Case $a = 2$

For $a = 2$, $k = 2^a = 4$. For $a = 2$, the factor $2^{k-1} - 2k + a + 1 = 3$ and so from (7.10) the maximum number of seeds to be stored by a user is

$$1 + \ell_0(\ell_0 + 1)/2 + 3\ell_0(\ell_0 - 1)/2. \tag{7.14}$$

We show how to reduce the factor 3 to 2 by suitably defining the function H.

Let $\mathcal{T}$ be a full binary tree of height a. Then the simple subsets of $\mathcal{T}$ are encoded by the 6 strings $0001, 0010, 0100, 1000, 0011, 1100$ and the non-simple subsets of $\mathcal{T}$ are encoded by the 8 strings

$$0101, 0110, 0111, 1001, 1010, 1011, 0101, 1001, 1101, 0110, 1010, 1110.$$

So, given an m-bit string L and a string σ encoding a non-simple subset of $\mathcal{T}$, we have to define $L_\sigma = H(\sigma, L)$. Let the leaf nodes of $\mathcal{T}$ from the left be $\theta_0, \ldots, \theta_3$. Then

$$\begin{array}{ll} \mathsf{keys}[L,\mathcal{T}](\theta_0) = \{L_{0101}, L_{0110}, L_{0111}\}; & \mathsf{keys}[L,\mathcal{T}](\theta_1) = \{L_{1001}, L_{1010}, L_{1011}\}; \\ \mathsf{keys}[L,\mathcal{T}](\theta_2) = \{L_{0101}, L_{1001}, L_{1101}\}; & \mathsf{keys}[L,\mathcal{T}](\theta_3) = \{L_{0110}, L_{1010}, L_{1110}\}; \end{array}$$

Each of these sets contains 3 m-bit strings which gives the factor 3 in (7.14). Since L and $\mathcal{T}$ will be clear from the context we will drop them from the notation. We show how to define H such that any of the sets $\mathsf{keys}(\theta_0), \ldots, \mathsf{keys}(\theta_3)$ can be obtained from 2 m-bit strings.

We define a new tree T_4. This tree has no relation to the tree $\mathcal{T}^0$. It is solely used to define the function H. The tree T_4 is defined as follows. The root node has four children nodes numbered $0, 1, 2, 3$. The child node numbered i has two children numbered $(i, 0)$ and $(i, 1)$. The structure is shown in Figure 7.6.

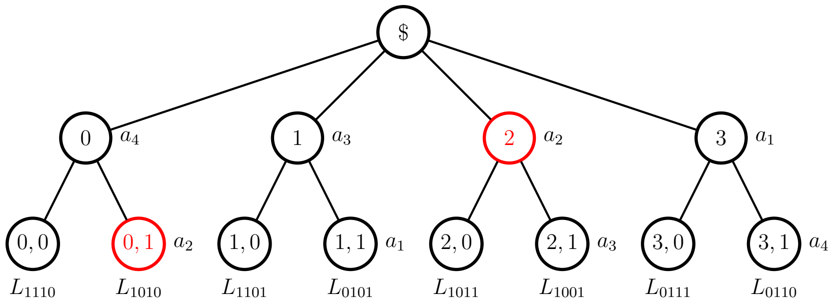


Figure 7.6: The structure of T_4 for $a = 2$.

Define, two hash functions $F_1 : \{0,1,2,3\} \times \{0,1\}^m \rightarrow \{0,1\}^m$ and $F_2 : \{0,1\} \times \{0,1\}^m \rightarrow \{0,1\}^m$. These hash functions are chosen by the broadcast center and made available to the users in the system.

Given an m-bit seed L, define

$$\left.\begin{array}{rcll} \widehat{L}_i & = & F_1(i, L) & \text{for } i = 0, 1, 2, 3; \\ \widehat{L}_{i,b} & = & F_2(b, \widehat{L}_i) = F_2(b, F_1(i, L)) & \text{for } i = 0, 1, 2, 3 \text{ and } b = 0, 1. \end{array}\right\} \quad (7.15)$$

Define

$$\begin{array}{l} L_{1110} = \widehat{L}_{1,0},\ L_{1010} = \widehat{L}_{1,1},\ L_{1101} = \widehat{L}_{2,0},\ L_{0101} = \widehat{L}_{2,1}, \\ L_{1011} = \widehat{L}_{3,0},\ L_{1001} = \widehat{L}_{3,1},\ L_{0111} = \widehat{L}_{4,0},\ L_{0110} = \widehat{L}_{4,1}. \end{array}$$

Then each of the sets $\mathsf{keys}(\theta_0), \ldots, \mathsf{keys}(\theta_3)$ can be obtained from 2 m-bit seeds as indicated below.

$$\begin{array}{l} \mathsf{keys}(\theta_0) : \widehat{L}_3 \text{ and } \widehat{L}_{1,1}; \\ \mathsf{keys}(\theta_1) : \widehat{L}_2 \text{ and } \widehat{L}_{0,1}; \\ \mathsf{keys}(\theta_2) : \widehat{L}_1 \text{ and } \widehat{L}_{2,1}; \\ \mathsf{keys}(\theta_3) : \widehat{L}_0 \text{ and } \widehat{L}_{3,1}. \end{array}$$

It is easy to verify that the above information is sufficient to obtain any set $\mathsf{keys}(\theta_i)$. For example, the users under the node $4j+3$ in $\mathcal{T}^0$ will be able to get the seeds $\{L_{0101}, L_{0110}, L_{0111}\}$.

Fix a user u and an ancestor i of u at level ℓ. For every node j which is an ancestor of u at levels between 2 and ℓ, the set $\mathrm{II}_u^{(2)}(2)$ contains two m-bit seeds. Since ℓ can vary from 2 to ℓ_0, we have

$$|\mathrm{II}_u^{(2)}(2)| \ = \ 2 \times \frac{\ell_0(\ell_0 - 1)}{2} = \ell_0(\ell_0 - 1). \quad (7.16)$$

Based on this we obtain the following improvement to (7.14).

$$\mathsf{us}_2(2^{\ell_0}) \ = \ 1 + \ell_0(\ell_0 + 1)/2 + \ell_0(\ell_0 - 1). \quad (7.17)$$

7.5.3 General Case

The technique for $a = 2$ is somewhat specific since in this case the number of non-simple subsets of an a-tree turns out to be 8 which is a power of 2. More generally, Lemma 34 shows that the number of non-simple subsets of an a-tree is $2^k - 2k$ where $k = 2^a$. The expression

$2^k - 2k$ will not be a power of 2 for $a > 2$. For this case, we directly use the technique from Chapter 6 [BS15] which dealt with the same problem in a different context. We explain this below.

The k-ary Tree Subset Difference Scheme. The underlying structure of the NNL-SD scheme is the binary tree $\mathcal{T}^0$. Chapter 6 [BS15] generalizes the idea to work with k-ary trees for any $k \geq 2$. So, suppose that $\mathcal{T}^0$ is a k-ary tree. Then each internal node has k children. Let i be an internal node of $\mathcal{T}^0$ and J be a non-empty subset of nodes having a common parent j. Let $S_{i,J}$ denote the leaf nodes of the graph formed by taking away from $\mathcal{T}^0$ the subtrees whose root nodes are in J. The collection $\mathcal{S}$ for the k-ary tree scheme consists of all such subsets $S_{i,J}$.

Key assignment in the k-ary tree scheme is done as follows. Each node is assigned a seed L_i and a hash function is iteratively used to define the seed $L_{i,j}$ for any node j in the subtree rooted at i. Given $L_{i,j}$ and the subset J of children nodes of j, a key $L_{i,J}$ is defined. In Chapter 6 [BS15] this is first defined directly and then later it is shown how to define this in a different manner so that the user storage reduces.

Coming back to the a-ABTSD scheme, we note the similarity between the subsets and the key assignment procedure of the two schemes. The relevant difference is that in the k-ary tree scheme the subset J is a non-empty subset of the children nodes of j, whereas in the a-ABTSD scheme, the subset J is a non-simple subset of the leaf nodes of the a-tree rooted at j. For both cases, the key to $S_{i,J}$ is assigned from the seed $L_{i,j}$. So, in both cases the problem is given an m-bit seed L and the subset J, how to define the key based on L and J?

A solution to this problem has been given in Chapter 6 [BS15] which uses the notion of cyclotomic cosets. We do not repeat the solution here and instead refer the reader to Chapter 6 for details. Our main observation is that the solution provided in Chapter 6 also works in the present case. The difference is that the method of Chapter 6 assigns keys to all non-empty subsets of the children nodes of j, whereas in the present case, we only need to assign keys to all non-simple subsets of the leaf nodes of the a-tree rooted at j. This difference, however, is not significant. We simply ignore the keys that are assigned to the simple subsets.

On the other hand, it is also possible to actually modify the key assignment procedure in

Table 7.1: Effect of reduction of user storage. In the second row the entry for $a = 2$ is from (7.16) and the entries for $a = 3$ and $a = 4$ are from (7.18).

storage	$a = 2$	$a = 3$	$a = 4$
$2\|I_u^{(2)}\|/(\ell_0 - a + 2)(\ell_0 - a + 1)$ from (7.9)	3	116	32741
$2\|\mathrm{II}_u^{(2)}(a)\|/(\ell_0 - a + 2)(\ell_0 - a + 1)$	1	36	4116

Chapter 6 so that keys are only assigned to non-simple subsets. We have carried this out for $a = 3$ and $k = 2^a = 8$. The work required us to examine the $2^k - 1 = 256$ non-empty subsets and eliminate the keys assigned to $2k - 1 = 15$ simple subsets. These details are quite tedious and so we do not report them. Directly using the key assignment procedure from Chapter 6 in the present context shows that $\mathrm{II}_u^{(2)}(a)$ for a user u consists of $(\chi_k - 2)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ m-bit keys where χ_k is the number of cyclotomic cosets of k-bit strings, i.e., for $a > 2$,

$$\mathrm{II}_u^{(2)}(a) \quad = \quad \frac{(\chi_{2^a} - 2) \times (\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \tag{7.18}$$

So, for $a > 2$,

$$\mathsf{us}_a(2^{\ell_0}) \quad = \quad 1 + \frac{\ell_0(\ell_0 + 1)}{2} + \frac{(\chi_{2^a} - 2) \times (\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \tag{7.19}$$

For the case of $a = 2$ and $k = 4$, $\chi_4 = 6$. Hence, from (7.19) $\mathsf{us}_2(2^{\ell_0})$ would be $1 + \ell_0(\ell_0 + 1)/2 + 2\ell_0(\ell_0 - 1)$. Previously, however, we have seen that $\mathsf{us}_2(2^{\ell_0}) = 1 + \ell_0(\ell_0 + 1)/2 + \ell_0(\ell_0 - 1)$. So, for the case of $a = 2$, directly using the solution from Chapter 6 is sub-optimal. This is one of the reasons why we considered the case of $a = 2$ as a special case.

For small value of a the reduction that is achieved is shown in Table 7.1. It is clear that the reduction achieved is significant in practical terms.

7.5.4 Full Resilience

A user obtains secret information I_u which allows it to obtain a set of keys. Let us denote this set as $\mathcal{K}_u$. It is to be noted that under certain reasonable cryptographic assumptions on the hash functions G, F_1 and F_2, user u does not obtain any information about keys that are not in $\mathcal{K}_u$. Further, if $\mathcal{K}$ is a set of keys and $\mathcal{U}_{\mathcal{K}}$ is the set of all users such that $\mathcal{K} \cap \mathcal{K}_u = \emptyset$, then $\cup_{u \in \mathcal{U}} \mathcal{K}_u$ does not provide any information about $\mathcal{K}$ (again under reasonable cryptographic assumptions on G, F_1 and F_2). This can be argued formally along the lines of the argument provided in Section 2.1 [NNL01, NNL02]. We skip the details and only remark that this can be intuitively seen by considering the hash functions to be one-way and the outputs of the hash functions to be independent.

7.6 Experimental Studies

The main point of this work is to reduce the header length. As we have already seen, the header length is never more than that of the NNL-SD scheme. This result, however, does not indicate what will happen on average. In this section, we report on this aspect and also compare the average header length and user storage as a varies.

In order to compute the expected header length, one may consider the same random experiment as described in Section 4.4 where r users out of n are randomly revoked without replacement. Then for every non-leaf node i in $\mathcal{T}^0$, one can associate a binary valued random variable X_i which takes the value 1 if a subset of the form $S_{i,j}$ or $S_{i,J}$ is generated and takes the value 0 otherwise. The header length is then $\sum X_i$ and by linearity of expectation, the expected header length is $\sum \Pr[X_i = 1]$.

In order to find $\Pr[X_i = 1]$ one has to consider the situations for which the event $X_i = 1$ can occur. Let us consider two sibling nodes i_1 and i_2 in $\mathcal{T}^0$. A subset S_{i_1,J_1} is generated from i_1 if sibling subtrees in J_1 are the only subtrees within the subtree $\mathcal{T}^{i_1}$ that have at least one revoked node each. Moreover, the level of the nodes in J_1 should be at least a levels below that of i_1. If such a subset is generated from i_1, then there has to be at least one revoked leaf in the subtree $\mathcal{T}^{i_2}$ and a subset S_{i_2,J_2} will be generated. Similarly, if the subset S_{i_2,J_2} generated from i_2 is such that the sibling subtrees in J_2 are the only subtrees in $\mathcal{T}^{i_2}$ with revoked users and the level of nodes in J_2 is at least a levels below i_2, then $\mathcal{T}^{i_1}$ will have at least one revoked leaf and a subset S_{i_1,J_1} will be generated from i_1. This gives rise to a large number of cases in the computation of $\Pr[X^i_{n,r} = 1]$. While in principle it is possible to exhaust all the cases, the resulting algorithm will be quite complicated. It did not seem useful to us to obtain such an algorithm.

Instead, we chose a simulation based approach to get a fair idea of the expected header length. First, we fix the parameter a for the scheme. For given values of n and r, we generate random revocation patterns using Floyd's Algorithm [BF87]. For each such revocation pattern, the cover generation algorithm finds the exact cover and hence we get the header length. The number of iterations is chosen so that the average value of the header length stabilizes. It turns out that 100 iterations are sufficient.

Table 7.2 shows that for different values of r, the expected header length of the 1-ABTSD scheme (the complete tree version of the NNL-SD scheme) is always more than that of the a-ABTSD scheme with $a > 1$. In fact, as a increases, there is a steep fall in the expected header length for fixed n and r. As an example, we see that for $n = 2^{24}$ and $r = 0.4n = 6710886$, the expected header length due to the NNL-SD scheme is 2.29 times that of the a-ABTSD scheme with $a = 3$.

We compare the performance of the a-ABTSD scheme by varying the parameter a. Table 7.2 shows how the mean header length for a given value of a (MHL_a) varies with n and r. We observe the following:

1. For a fixed n, as the parameter a is increased, the user storage increases.
2. For fixed n and a, the ratio MHL_a/r decreases steadily as r increases. This behavior is true for all $a \geq 1$ (including the NNL-SD scheme).
3. For fixed n and r, as a increases, the ratio MHL_a/r decreases steadily. This holds for any value of r.
4. For fixed a and r/n, the value of MHL_a/r is approximately the same for all values of n. Hence, these properties hold good for the full-tree versions (with $n = 2^{\ell_0}$) of the scheme too.

For certain values of r/n, the ratio MHL_a/r is shown in Table 7.3. This behavior is further depicted by plotting the values of Table 7.3 in Figure 7.7.

Practical Impact. Broadcast encryption is used in paid services like cable TV and online broadcasting services (audio, video, gaming and document sharing) for implementing digital rights management [DRMa]. Our scheme with $a > 1$ would reduce the communication overhead of a system that uses the NNL-SD scheme. For any value of r/n, the mean header length for $a > 1$ will be less than $a = 1$ (NNL-SD scheme). From Table 7.2,

Table 7.2: User storage and mean header lengths in the complete a-ABTSD scheme for values of a between 1 and 4. For a fixed n, we report MHL_a/r for three different choices of r namely, $r = (0.1n, 0.2n, 0.4n)$.

n	a	$\mathsf{us}_a(n)$	MHL_a/r	n	a	$\mathsf{us}_a(n)$	MHL_a/r
10^3	1	55	$(1.11, 0.97, 0.71)$	10^4	1	105	$(1.11, 0.97, 0.71)$
	2	145	$(0.96, 0.78, 0.53)$		2	287	$(0.96, 0.78, 0.53)$
	3	1279	$(0.75, 0.53, 0.31)$		3	2757	$(0.75, 0.53, 0.31)$
	4	115247	$(0.52, 0.31, 0.16)$		4	271629	$(0.52, 0.30, 0.16)$
10^5	1	153	$(1.11, 0.97, 0.71)$	10^6	1	210	$(1.11, 0.97, 0.71)$
	2	425	$(0.96, 0.78, 0.53)$		2	590	$(0.96, 0.78, 0.53)$
	3	4233	$(0.75, 0.53, 0.31)$		3	6024	$(0.75, 0.53, 0.31)$
	4	432123	$(0.52, 0.30, 0.16)$		4	629652	$(0.52, 0.30, 0.16)$
10^7	1	300	$(1.11, 0.97, 0.71)$	10^8	1	378	$(1.11, 0.97, 0.71)$
	2	852	$(0.96, 0.78, 0.53)$		2	1080	$(0.96, 0.78, 0.53)$
	3	8902	$(0.75, 0.53, 0.31)$		3	11428	$(0.75, 0.53, 0.31)$
	4	950634	$(0.52, 0.30, 0.16)$		4	1234578	$(0.52, 0.30, 0.16)$

Table 7.3: List of values of the ratio MHL_a/r (for any n) corresponding to the varying ratio r/n for each a. Note that as the value of a increases, the scheme performs better in terms of communication overhead as compared to a smaller value of a.

a \ r/n	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
1	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
2	(1.20,	1.08,	0.96,	0.78,	0.64,	0.53,	0.44,	0.35,	0.27,	0.18,	0.10,	0.00)
3	(1.15,	0.93,	0.75,	0.53,	0.39,	0.31,	0.25,	0.20,	0.17,	0.13,	0.08,	0.00)
4	(1.07,	0.73,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

we see that for a system with $n = 10^6$ user of which $r = 0.4n = 4 \times 10^5$ users are revoked, the expected header length of the 2-ABTSD scheme is $0.96r$ whereas that of the 1-ABTSD scheme is $1.11r$. This means that a system using the NNL-SD scheme will on an average require the header to be smaller by $0.05r$ per session as compared to the 1-ABTSD scheme. In concrete terms, assuming that keys in these systems are 128-bit long, for $r = 4 \times 10^5$, on an average, the header length will be smaller by 312.5KB per session. Thus, each session will save around 0.31MB of additional bandwidth per channel.

Depending upon the length of each session, the savings per channel can be significant. This practical saving of communication bandwidth, however, comes at a cost. Assuming 128-bit key size, the storage for $a = 1$ is 26.25KB, whereas that of the 2-ABTSD scheme is around 73.75KB. Due to steadily falling memory prices, the benefit of savings in communication bandwidth will outweigh the cost of extra memory.

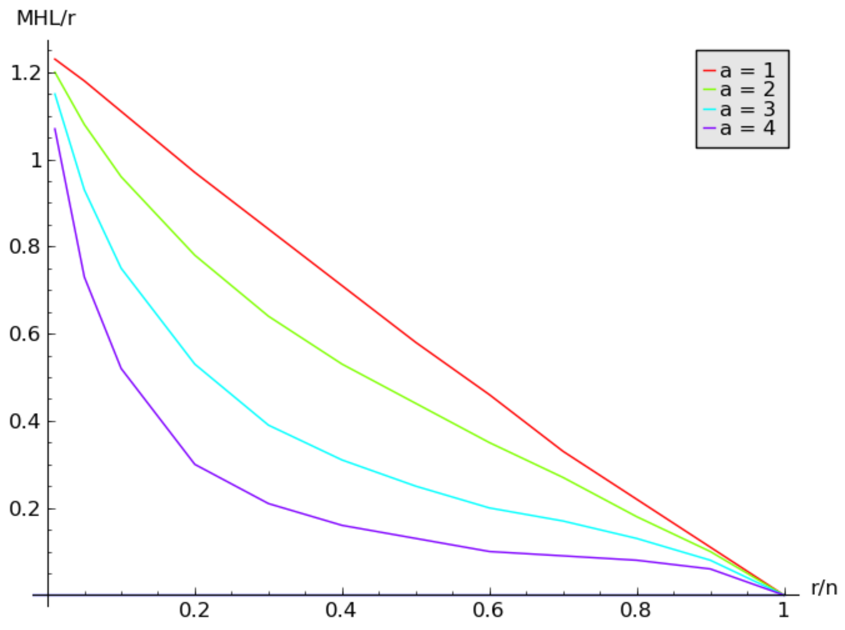


Figure 7.7: Plot showing how MHL_a/r varies with r/n.

In applications like the standard for DRM in optical discs [AAC], the header is stored in a fixed portion of the optical disc. There is an allotted amount of space for the header. This amount of storage allotted for the header may be fixed and hence there would be a limit on the number of revoked users that the system will be able to tolerate. For a given value of r, the average header length due to $a > 1$ will always be less compared to the NNL-SD scheme ($a = 1$). In other words, for an instantiation of the scheme with $a > 1$, a particular value of the expected header length will occur for larger values of r. Given n and r, the maximum header length for $a > 1$ will be at most as much as the NNL-SD scheme and in general less. As a result of the reductions in the average as well has worst-case header lengths, the system with $a > 1$ will be able to tolerate more number of revoked users compared to the NNL-SD scheme.

7.7 Conclusion

Several scenarios implementing BE, require improving the communication efficiency and can tolerate an increase in the user storage. Our goal in this chapter has been to bring down the communication cost. It can be intuitively said that increasing the number of subsets to which keys are assigned, should improve the communication overhead. Based on this intuition, we have proposed the a-augmented binary tree subset difference scheme (a-ABTSD) scheme. This scheme is a generalization of the NNL-SD scheme. It is parameterized by a (height of the augmenting structure), offering varying efficiencies of the user storage and communication overhead. We proved that the header length for any given set of revoked users in this scheme is at most as much as the NNL-SD scheme. The expected header length however, has been experimentally seen to be always less than the NNL-SD scheme for any value of r. Although the storage requirement for both these schemes are asymptotically the same as the NNL-SD scheme, in concrete terms they are more than the NNL-SD scheme. This is the trade-off for the decreased average communication overhead.

Chapter 8

Applications, Implementation Aspects and Future Directions

This chapter is divided into two parts. First, we point out the various real-life applications of BE. This is an elaboration of the brief summary of the applications that was provided in Chapter 1. It includes a very brief account of the content protection systems that have been employed over the years and their short-comings. Following that, we discuss the impact of our results on real-life scenarios that use the NNL-SD scheme and its variants.

8.1 Real-Life Applications of BE

It has been discussed several times in the previous chapters that a BE framework assumes a broadcasting center and a set of users. Broadly, there are two primary security issues that arise out of such scenarios: (1) protection of electronic content from unauthorized access, and (2) privacy of users who access that content. Using a trusted server as the broadcasting center is the most commonly used method for protecting both electronic content and the privacy of users who can access it. Whenever a user wishes to access content, it contacts the server, authenticates itself, and is sent the content over a secure channel. As long as the server behaves correctly, (1) only authorized users will be able to correctly decrypt the content, and (2) no one else other than a user itself (not even other authorized users) will be able to find out which content it is authorized to access. This is the functional framework we have assumed in this thesis.

For scenarios where the broadcasting center may not be trusted, the above framework fails to provide protection of content access or privacy of users. In case the server is compromised both data content and user privacy are subject to attack. An example scenario is the case of content providers who will often not distribute their data directly, but for economic reasons outsource distribution to third parties or use peer-to-peer networks. In this case, the content owners will no longer be directly in control of data distribution. Here, we shall not discuss such systems where the server is not trusted.

We look at different practical scenarios in the following and see how BE can solve the security issues that arise in these scenarios. The BE scenarios described in this section bring out different real-life situations that have some amount of commonality that is captured and has been found to be addressed by BE schemes. It would indeed be nice to be able to report more details of the parameters governing the implementations of BE in these scenarios. However, it is not difficult to guess that the statistical data for these scenarios are sensitive to the businesses or relevant institutions. Unfortunately, we have not been able to gather any significant data that could be reported. Given the unavailability of such data, our intent in this chapter is to provide short descriptions of the scenarios we have come across and the available references that could be quoted to use BE.

All our works in this thesis are within the ambit of the NNL-SD scheme described in Chapter 2 [NNL01, NNL02]. It has been stated several times earlier that the NNL-SD scheme has been suggested for use by the [AAC] standard for DRM [DRMa] in optical discs. Hence, we put additional emphasis on the DRM systems for content protection in optical discs. Given the dominance of Pay-TV systems in the content protection industry, the congregation of our results presented later in the chapter takes examples of parameters from this application.

8.1.1 Content Protection in Optical Discs

First let us see how the problem of content protection in optical discs fits into the BE framework. A BE system is initialized by a *licensing authority* (Licensing Authority (LA)) that generates the keys to be used by the center and the users of the system. The hardware and software players of these optical discs are produced by manufacturers who purchase licenses to embed decryption keys into the players. The broadcasting center here are the various production houses that sell their content (movies, songs, etc.). The copyrighted content is encrypted with keys from the LA on optical discs. The hardware and software players are the users of the BE system that allow run-time decryption of the copyrighted content from these optical discs.

Keys stored in these players may get leaked. These keys may be used for large-scale manufacture of pirate hardware players to be sold in the market. The hardware players will be cheaper since the keys will not be obtained from the licensing authority. The leaked keys may also be used to build software players. The copyrighted content that are decrypted using these software players may be redistributed. Leaked device keys when detected using traitor tracing techniques, they are revoked so that future content cannot be decrypted using those keys.

Here, we look at the various content protection systems and standards that have been adopted for content protection in optical discs and how the shortcoming(s) of one system (often ending in complete collapse of the system) led to another system to be developed.

Content Scrambling System (Content Scrambling System (CSS)). The Content Scrambling System (CSS) was devised by the DVD Copy Control Association (DVD CCA) [DVD] and was introduced in 1996. CSS included both player-host mutual authentication and data encryption. It was used to protect the content of DVDs from piracy and to enforce region-based viewing restrictions.

The idea of region codes worked as follows. Each DVD contained a region code determining the region of the world in which it could be viewed. Each player knew the region in which it was supposed to be sold. If the region code of the player did not match the region code on the DVD, the player would not read the DVD. This was to help the MPAA[1] [MPA] ensure that DVDs don't leak out into parts of the world ahead of their respective scheduled "first screening".

CSS utilized a proprietary 40-bit stream cipher algorithm. The CSS key sets were licensed by the DVD CCA to manufacturers of DVD movie releases as well as hardware drives and software players. The weakness of the CSS system lay in the size of the key. This key length was grossly inadequate in the face of increasing computing power. In addition, structural flaws in CSS resulted in reduction of the effective key length to only around 16 bits. A brute-force attack worked even without the region codes. This allowed region-free DVD player software to work with region-locked drives.

One of the first free computer programs capable of decrypting content on a commercially produced DVD video disc was DeCSS created by Jon Lech Johansen and two people who have

[1] Motion Picture Association of America (MPAA) is a trade association that represents the six major Hollywood studios. It sets guidelines for film content (the Production Code) and administers the MPAA film rating system. More recently, the MPAA has advocated for the motion picture and television industry through lobbying to protect creative content from piracy and for the removal of trade barriers. The MPAA has made consistent attempts to curb copyright infringement, including recent attempts to limit the sharing of copyrighted works via peer-to-peer file-sharing networks.

remained anonymous, by reverse engineering CSS. Before the release of DeCSS, there was no way for computers running a Linux-based operating system to play video DVDs. DeCSS was developed without a license from the DVD CCA. The release of DeCSS resulted in a Norway criminal trial and subsequent acquittal of Jon Lech Johansen. The chief complaint against DeCSS (and similar programs) is that once the unencrypted source video is available in digital form, it could be copied without degradation. So DeCSS could be used for copyright infringement.

Content Protection for Recordable Media and Pre-Recorded Media (CPRM/CPPM). Content Protection for Recordable Media and Pre-Recorded Media (CPRM/CPPM) was a DRM system developed by the 4C Entity, LLC[2] (comprising of IBM, Intel, MEI, and Toshiba) for content protection in secure digital (SD) cards[3] and DVD-audio discs. It was agreed upon in mid-2000.

CPPM used the Cryptomeria Cipher (C2) as the successor to the CSS algorithm for content encryption. C2 was a proprietary block cipher defined and licensed by the 4C Entity. The C2 symmetric key algorithm had a 10-round Feistel structure that had a key size of 56 bits and a block size of 64 bits. Implementations of C2 required the secret values of the substitution box (S-box), which were only available under a license from the 4C Entity. The 4C Entity licensed a different set of S-boxes for each application (such as DVD-Audio, DVD-Video and CPRM). It proved to be stronger than CSS.

Full-round C2 was broken in [BKLM09] in three different scenarios. This work presented (1) an attack with time complexity 2^{24} to recover the S-box in a chosen-key scenario, (2) a 2^{48} time complexity boomerang attack to recover the key with a known S-box using 2^{44} adaptively chosen plaintext-ciphertext pairs, and (3) a $2^{53.5}$ time complexity attack when both the key and S-box are unknown.

Advanced Access Content System (AACS). The Advanced Access Content System (AACS) is a standard for digital rights management and content protection of the post-DVD generation of optical discs. The theoretical foundations of the AACS standard was laid by

[2] Limited Liability Company (LLC) is a flexible form of enterprise in the US that blends elements of partnership and corporate structures that may not be organized for profit [Wik].

[3] The Secure Digital (SD) format included four card families available in three different form factors. The four families are the original Standard-Capacity (SDSC), the High-Capacity (SDHC), the eXtended-Capacity (SDXC), and the SDIO which combines data storage with I/O functions.

the work of Naor et. al. [NNL01, NNL02] (described in Chapter 2) in 2001 that introduced the SD scheme which formed the basis of the AACS standard. Since its public release in 2005, the specification standard has been adopted for content protection in HD DVD and Blu-ray Disc (BD). It was developed by AACS Licensing Administrator, LLC (AACS LA), a consortium that included Disney, Intel, Microsoft, Panasonic, Warner Bros., IBM, Toshiba and Sony.

The main difference between AACS and CSS lay in how the device decryption keys and codes were organized. In CSS, all players of a given model group carried a single shared decryption key. Content was encrypted under the title-specific key (the session key), which was further encrypted under each model's key. Thus each disc contained a collection of encrypted session keys, one for each licensed player model. Consequently, a licensor could revoke a given player model by omitting to encrypt future title keys with the player model's key. Revoking all players of a particular model was costly since it caused many users to lose playback capability. Furthermore, since the same decryption key was shared by all players of a model, key compromise was significantly more likely.

The NNL-SD scheme that was suggested for use by AACS, provides each individual player with a unique set of decryption keys. AES is used with 128-bit keys for encrypting the content. Hence, a licensor could revoke (the decryption keys of) individual players. Thus, compromised and published keys were revoked by the AACS LA in future content, making the keys/player useless for decrypting new titles.

In addition to the traitor tracing for pirate decryption boxes that was provided by Naor et. al. [NNL01, NNL02] (described in Chapter 2), AACS also incorporated traitor tracing techniques to trace re-broadcasted content. The standard allowed for short sections of a movie to be encrypted with different keys to make unique versions. A given player would only be able to decrypt exactly one version of each section. The manufacturer would embed unique digital watermarks (involving *sequence keys*) in these sections for each copy of the distributed content. Upon subsequent analysis of the pirated release, the compromised keys could be identified and revoked.

Since appearing in devices in 2006, several AACS decryption keys have been extracted from software players and published on the Internet, allowing decryption by unlicensed software. One of the techniques used by hackers for key-extraction is by using debuggers to inspect the memory of software player programs. In fact, this issue is common and inherent to all existing DRM systems that allow decryption in software. The keys used to finally decrypt the content has to be available somewhere in the memory and hence is susceptible to attacks. A hacker named "muslix64" [Mus] used the specifications on the AACS website [AAC] to create software that can decrypt any HD-DVD movie given the title key (which is actually the session key). Measures like providing software patches and updating device keys with new uncompromised ones have also not succeeded because the attackers would have used only a few keys which could be traced and revoked, and just after the systems were updated (which was costly for hardware updates), they started using the keys that were held back. This made the update process futile.

Another possible reason for the failure of the AACS standards to stop piracy through revocation for DVD players is that DVD players became cheaper compared to the latest movie releases. Using a new DVD player for each new movies release would not hurt the profit margins of the DVD video pirates.

Self-Protecting Digital Content (SPDC). Self-Protecting Digital Content (SPDC) [SPD] designed by Cryptography Research, Inc. [Res] was designed to provide an additional layer of security for a content protection system, in addition to the key management systems such as AACS. A primary goal of the SPDC framework was to provide renewability of the encryption system in the event that an entire class of devices becomes vulnerable to compromise. The SPDC systems are hence dynamic that allow compromised keys to be replaced by new ones. SPDC executes code from the encrypted content on the device, so that the content providers can change DRM systems. If some weakness is found in the method of playback used in previously released content, code embedded into content released in the future will change the method. Thus, a fresh attack has to be launched on the new method. If a certain model of players are compromised, code specific to the model can be activated to verify that the particular player has not been compromised. The player can be fingerprinted if found to be compromised and the information can be used to detect the traitor. Code inserted into content can add fingerprints to the output that specifically identifies the player. Hence, in case the content is re-distributed on a large scale, it can be used to trace the player.

BD+ is a component of the Blu-ray Disc DRM system that was developed by Cryptography Research, Inc. and the Blu-ray Disc Association leaders Twentieth Century Fox, Sony, and Panasonic. The BD+ virtual machine embedded in authorized players, use the SPDC framework for content protection. The content providers can include executable programs on Blu-ray Discs to test for threats, patch the existing system and if required, circumvent the vulnerability that may have been introduced in the device. Hence, the SPDC framework plays a significant role in safeguarding the capabilities of the AACS system that in turn takes care of the content protection.

8.1.2 Pay-TV

Television subscription has been handled using two different techniques. The first technique uses cables to deliver the programs to the subscriber homes. The second technique scrambles channels to be broadcasted through wireless broadcasting frequencies such that only subscribers will be able to view the programs. Both these techniques (especially the second) uses BE systems to ensure that only a subscriber is able to view a channel or program.

Several works [MQ95, Woo98, Woo00, MV01, NRK03] have addressed the issues associated with Pay-TV systems. These systems use BE for the subscription management. The service provider is the broadcasting center. The user equipment called set-top-boxes (STBs) are provided by the service provider. Hence, each service provider employs its own BE system. The channels are encrypted in a manner such that the STBs of only the subscribed users will be able to view them. The cryptographic algorithms and keys are typically stored in the smart cards. Smart cards have embedded integrated circuits and can be re-programmed if necessary. Televisions now-a-days come with the capabilities of STBs inbuilt.

According to the white-paper [Ros], PayTV often leads the way in content protection technologies and hence is the most important industry driving these technologies. To quote from the document:

> "*... among digital content delivery modalities, Pay-TV is unique for two reasons: one technical and one economical. The technical reason is that many digital Pay-TV systems provide some kind of communication channel from the client device, such as a set-top box, back to the server at the head end. That means that the device is able to "phone home" for various purposes, such as to register itself on the network or respond to various types of security messages. Such two-way communication capability can facilitate some of the advanced security techniques discussed here that would otherwise be impossible to implement. Contrast this with delivery modalities such as physical media (DVDs, Blu-ray discs), where no*

server connectivity can be assumed at all, or even Internet delivery modalities such as PC downloads, where consumers expect to be able to use content offline.

The economic reason for Pay-TV's uniqueness is that the incentives of Pay-TV operators (such as cable and satellite providers) and content owners (such as movie studios) regarding content protection are aligned. The content owner doesn't want its copyrights infringed, and the operator doesn't want its signal stolen. Again, contrast this with other delivery modalities: makers of consumer electronics (such as Smart-Phones) generally don't build content protection technologies into their devices, and content retailers would rather not have to pay the cost of implementing DRM and similar technologies. As we'll see in this white paper, some of the more notable failures in content protection technology have been due to consumer device makers trading security off in favor of lower unit cost. ”

8.1.3 File Sharing in Encrypted File Systems

Encrypted File Systems implement read access control by encrypting the contents of files such that only users with read permission will be able to perform decryption. Typical encrypted file systems, such as Windows EFS, encrypt each file under its own session key, and then encrypt the session key separately under the keys of the users authorized to access the file.

To quote from [BGW05],

“*Abstractly, access control in an encrypted file system can be viewed as a broadcast encryption problem. The file system is the broadcast channel and the key K_F is broadcast (via the file header) to the subset of users that can access file F. Many encrypted file systems implement the straightforward broadcast system where the number of ciphertexts in the file header grows linearly in the number of users that can access the file. As a result, there is often a hard limit on the number of users that can access a file. For example, the following quote is from Microsoft's knowledge base: “EFS has a limit of 256 Kbytes in the file header for the EFS metadata. This limits the number of individual entries for file sharing that may be added. On average, a maximum of 800 individual users may be added to an encrypted file.”* ”

Here, the key K_F is the session key and the file F is the message. This clearly brings out the importance of reducing the header length in BE schemes that are used in Encrypted File Systems. Although Windows EFS uses public-key BE, there may be other scenarios where the computation power may be assumed to be limited or the speed of decrypting the session key for each session may be crucial. A very common example where the decryption speed is crucial is a shared file server which has to handle millions of online sessions at a time. A symmetric key BE system requiring small header lengths per session would be very suitable for such scenarios.

8.1.4 Sending Encrypted Email to Mailing Lists

OpenPGP is arguably the most widely used email encryption standard. The OpenPGP standard was originally derived from PGP (Pretty Good Privacy), first created by Phil Zimmermann in 1991. The standard was defined by the OpenPGP Working Group of the Internet Engineering Task Force (IETF) proposed standard RFC 4880 [CDF$^+$07]. In [BGW05, BBW06], it was pointed out that when encrypting a message to multiple recipients, OpenPGP functions as a broadcast encryption system. It encrypts each message under a session key and then encrypts the session key for the intended users using their keys. Such systems may have varied efficiency requirements. Reducing communication overhead and user storage may both be important.

8.1.5 Online Content Sharing and Distribution

In commercial online content distribution (like websites that allow viewing of videos against a fee), a company may wish for its digital media to be available only to paying users. It was mentioned in [BBW06] that such applications use BE schemes to implement content protection. As an example of smaller scale, suppose a department's faculty need to access the academic transcripts of graduate applicants. If electronic copies of the transcripts were stored on the department's file-server, they should only be accessible by the faculty and the respective students and not by anyone else.

Movie rental companies like Netflix use BE systems for copyright protection [JL07]. In these scenarios, the respective centers would assign users with their long-lived secret information. The data (movie files, academic transcript image files, etc.) would be encrypted with a random session key and the session key would be attached in encrypted form as header with the encrypted data such that only privileged users will be able to decrypt the data.

It was also pointed out in [BBW06] that it is often equally important to protect the identities of the users who are able to access protected content. Commercial sites will often not want to disclose identities of customers because competitors might use this information for targeted advertising. A website that provide subscription-based adult material would wish to keep the identities of their customers private. We however do not address this issue of anonymity in this thesis.

Apple Inc.'s Fairplay and Windows Media DRM (WMDRM) are two very popular DRM systems used in online content distribution systems. The iTunes Store is a software-based online digital media (songs, apps, TV episodes, films, books) store. It uses FairPlay – a DRM technology – to guard the digital media (other than songs) available on iTunes Store against unauthorized access. WMDRM for the Windows Media platform was designed to provide delivery of audio and/or video content over an IP network to a PC or other playback device in such a way that the distributor can control how that content is used.

8.1.6 Online Gaming

Fifth generation and later video game consoles (Microsoft's XBox, Sony's PlayStation, Nintendo's Wii) have revolutionized living room computing entertainment [Wik]. The console vendors usually have an online multiplayer gaming and digital media delivery service. The gaming console usually receives regular updates during its lifetime. These online services are available in free and subscription-based varieties. These services include: playing games online; downloading games and their demos; purchasing and streaming music, television programs and films through video portals; and access third-party content services through media streaming applications. In addition to online multimedia features, these consoles typically allow users to stream media from local PCs. Several peripherals and additional services have been released which helped this industry grow from gaming-only to encompassing all multimedia.

All subscription-based online services need to be protected using DRM technologies and as with every multi-user system, BE may be used to implement DRM in these systems. To implement a BE system, the service provider would be the center while the consoles will be the user equipment.

8.1.7 Web-Based Electronic Commerce

The financial institution that facilitates the buying and selling of financial securities between a buyer and a seller is called a brokerage firm. In addition to carrying out a stock or bond trade, the brokerage firm is entrusted with the responsibility of researching the markets to provide appropriate recommendations, up-to-date stock prices and quotes.

An online broker helps its clients perform trades via automated, computerized trading systems. The firm will want to ensure that the broadcasts of online stock quotes and proprietary market analysis (often carrying trade secrets) are available only to the clients and are not leaked to an outsider. BE can be used to ensure confidentiality of these broadcasts. The firm will have a central server for broadcasting and handling the client requests. The investor clients will be provided with a trading platform that acts as the hub for transactions for the user.

Reducing communication overhead per session for these broadcasts is important to ensure increased speed of data being fed. At the user end, the Internet connectivity may not be of very high speed. Reducing the overhead will be of practical importance in such scenarios.

8.1.8 Peer-to-Peer DRM

The term “peer-to-peer” (P2P) refers generally to software that enables a device to locate a content file on another networked device and copy it to its own local storage [ER05]. P2P technology often attracts people who use it to reproduce or distribute copyrighted music and movies without authorization of rights owners. Early P2P systems did not use encryption and had no DRM implementation. By establishing an access control mechanism (DRM) for these shared digital content, unauthorized reproduction may be rendered useless. The system would work in a manner similar to DRM in optical discs. The production house while publishing the digital content would encrypt it such that only legitimate (software) players will have the decryption capabilities. Napster was a very popular P2P network used to share multimedia content followed by Gnutella, Freenet and others.

8.1.9 Military Broadcasts

Broadcasting information from the headquarters or military base (center) to the outposts or handheld receivers (user devices) with soldiers have to be cryptographically secured so that even if the enemy intercepts the signals, it will not be able to extract any significant information. An example of such a system is the Global Broadcast Service (GBS) [Wik, FAS] which is a combined United States space and Command, Control, Communications, and Intelligence (C3I) system. It provides a one-way high-throughput of information to forces garrisoned, deployed, or on the move. GBS uses the popular commercial direct broadcast satellite technology. The European counterpart of GBS is Joint Broadcast System (JBS). The GBS is supported with multi-level security which should include the use of BE for confidentiality. To quote from [FAS],

> "*The GBS system consists of broadcast management, space, and terminal segments. The broadcast management segment, integrates, encrypts and packages multi-media information and provides a bit stream to the Primary Injection Points (PIP) for Radio Frequency (RF) transmission to the satellite. The user receive terminal, consisting of a small satellite antenna, low noise block and receiver, will receive and convert the RF down-link signal into a bit stream for receive broadcast management decryption and distribution to end users.*"

8.1.10 Home Networks

In today's homes, multiple digital devices are connected to a peer-based cluster and seamlessly work together. It was mentioned in [JL09] that such networks need a content protection system that would allow a recording device inside the home network to bring the streaming content into the home network in a secure way that devices and only devices in the same home network can playback the recording. The technology will enable the secure sharing of premium quality HD content across a consumer's all audio-video devices at its home network. The recorded content should however be such that in case of a piracy attack, it can be used to obtain forensic information, to identify the source devices that participated in the attack. The identified traitor devices will have to be revoked for future content access.

The High-Definition Audio-Video Network Alliance (HANA) [Wik] is a cross-industry collaboration that was set up to address the end-to-end needs of connected, HD, home entertainment products and services. To quote from [Wik], HANA is

> "*. . . based on broadcast encryption, the same basic technology used in 4C and AACS content protection. Similar to AACS, a compromised device or class of devices is repairable by revocation of device the keys which can occur any time new content is imported into the domain, or a connection is made to a content service.* "

IBM, a HANA member, developed a content protection technology called Advanced Secure Content Cluster Technology (ASCCT). ASCCT was designed specifically for home networks. The BE setup in a HANA home network is called an "authorized domain". Devices (users of the BE system) while joining a HANA network receive BE keys and hence become part of the authorized domain. When content is received into the authorized domain, it is encrypted by the BE system. A non-authorized device will not be able to decrypt, rendering its copy useless. To share content, a device must join a network at which time its keys from any previous domain are destroyed and the previous content rendered unreadable.

The xCP (eXtensible Content Protection system) technology, based on IBM's Cluster Protocol (backed by Intel, Matsushita and Toshiba) works with peer-to-peer BE systems. It connects devices like MP3 players, DVD players, cell phones, PDAs, televisions and entertainment systems in vehicles. Any device in the network with the hard-disk is set up as the BE center to which the other devices become users.

A very popular home network solution standard is Digital Living Network Alliance (DLNA) [DLNa]. The DLNA trade group was founded by Sony in 2003 to define the interoperability guidelines for devices on home networks. Although the DLNA guidelines are not publicly accessible by individuals, according to [DLNb],

> "*With more than 4 billion DLNA-certified products in the market - including TVs, Blu-ray players, storage devices, media boxes, smartphones, tablets, game consoles and software chances are good you already have more than one compliant device or application in your home. Depending on the manufacturer, the product may use a branded version of DLNA such as SmartShare (LG), SimplyShare (Philips), or AllShare (Samsung), but rest assured its all the same technology and it will all interoperate.* "

A key component of the DLNA standard is DRM and content protection and our guess is they use BE like the others mentioned before.

8.1.11 Mobile Broadcast

The mobile phone industry standards body, Open Mobile Alliance (OMA), developed a leading DRM technical specification called OMA DRM Version 2.0 in late 2004 [4]. BE is expected to constitute the core of the OMA DRM specification. Many mobile service providers use OMA DRM for their content services. For example, most of the ring-tones pre-installed on mobile phones have implemented DRM. Another example is Mobile TV broadcast for which the OMA BCAST Smart-card profile has been recommended by all the industries to be the unified standard [Wik]. The mobile service providers have a broadcasting center of the BE setup while the mobile devices are the user equipment.

Content Management License Administrator (CMLA) [CML] was developed at the same time as OMA DRM with the overall objective of enabling a wide and trusted ecosystem for the distribution of premium digital content. Another objective of CMLA was to develop and operate the trust system to enable commercialization of the OMA DRM specification. CMLA example deployments include a full spectrum of mobile services like mobile broadcast streaming for major sporting events and download services of music and movies. Within CMLA Mobile Broadcast protected services, any digital content such as music, images, video or even applications may be distributed. CMLA Mobile Broadcast service offerings may be independently defined by service providers or broadcasters including free-to-air, pre or post-paid subscription based, and pay-per-view services.

[4] According to [DRMb], this standard is being used till date.

8.2 Practical Impact of our Contributions

The importance of the subset difference technique and especially the NNL-SD scheme in practical scenarios have been emphasized several times. The theoretical impact of this thesis was explained in Chapter 1. Our work began with developing tools to better analyze and understand the subset difference technique. Through those exercises we understood that the behaviour of any BE scheme is governed by the choice of the collection $\mathcal{S}$ of subsets to which keys are assigned. The analysis and the results conformed with and strengthened our intuition that as the size of the collection $\mathcal{S}$ is increased, generally the user storage increases while the expected header length decreases and vice versa. This has been the crux of our understanding in coming up with new techniques for achieving practically useful trade-offs. We obtained generalizations of the NNL-SD and HS-LSD schemes that have opened up interesting avenues of optimization and trade-offs of the two most important parameters of any BE scenario - the header length and the user storage. Here we see how our contributions would impact practical use of BE. It was mentioned in Section 8.1.2 that Pay-TV is one of the leading industry applications using BE for content protection. We see the effect of using our results in the context of Pay-TV systems (they apply equally well for any similar practical scenario). The estimates of the number of users in the system have been done based on the data available on the internet [Cab, FCC06].

The techniques used for these generalizations evolved from the analysis of the subset difference technique. In Chapter 4 we have described combinatorial and probabilistic analysis of the SD technique and devised an algorithm to compute the expected header length of the NNL-SD scheme. The main idea behind this method is to compute the probability of contribution of each node in the underlying tree that add up to give the expected header length. This technique works for all known subset difference based schemes including all the extensions and generalizations of the NNL-SD scheme that have been proposed in this thesis. Using this algorithm we showed that for $n = 2^{23} + 1$ and $r = 10^6$ (in Table 4.4), using the CTSD scheme that we proposed, around 1300 Kbytes can be saved per session. To understand how important this bandwidth saving may be, we see that for a Pay-TV connection with download speed 10 Mbytes per second (through ADSL channels), one can save around 13% bandwidth for sessions that are one second long [5]. For a Pay-TV connection

[5] The justification for assuming the length of a Pay-TV session to be no longer than in seconds is as follows. A stateless system can not store any dynamic information. Hence, the session key has to be extracted from the broadcast after every reboot. A typical set-top-box of a Pay-TV system typically takes a few seconds to

with download speed 100 Mbytes per second (through optical fiber channels), one can save around 1.3% bandwidth for sessions that are one second long.

The storage requirement of the SD scheme was successfully reduced in Chapter 5. The minimum storage that could be achieved using two-way splitting of SD subsets was proposed. With an idea of the minimum number of revoked users that will exist in the system, the average communication bandwidth requirement could also be reduced. This can be used in miniscule devices for which storage might be costly. In-ear receivers used in military broadcasts are a good example. With the advent of TV viewing services on mobile devices with access to high-speed internet and sufficient bandwidth, our results on reduction of storage can be very useful. For $n = 2^{28}$ users in a Pay-TV scenario, using the SML_1 layering strategy, the user storage is reduced from 406 in the NNL-SD scheme to 119 which is 70.69% reduction in the user storage. More importantly, this is the minimum possible storage that can be obtained by the two-way splitting of the SD subsets. Compared to the e-HS-LSD scheme, the savings is 18.49%. However, the use of the SML_0 or SML_1 strategies result in the header length performance to be roughly as bad as that of the e-HS-LSD layering strategy. Hence, the most interesting result in practical terms is the CML strategy where for $n = 2^{28}$, the user storage reduces by 46.06% while for most values of r while the expected header length remains the same.

In most applications of BE that have been listed in Section 8.1, the communication bandwidth is the costliest parameter and hence is often the most important driving force of the respective industry. Examples are Pay-TV, online content sharing, mobile broadcasts, etc. We have proposed two schemes with different trade-offs, both of which reduce the communication overhead of the NNL-SD scheme, at the cost of increased user storage. The k-ary tree SD scheme proposed in Chapter 6 reduces the communication overhead for most practical number of revoked users (greater than a threshold) while the a-ABTSD scheme of Chapter 7 reduces the average communication overhead for any number of revoked users. For $n = 2^{28}$ users in a Pay-TV scenario, taking $k = 4$, the user storage increases by 37.25% while for $r = 0.4n$ [6] the header length decreases by 16.9%. Taking $a = 2$ (closest to $k = 4$),

start including the booting time of the operating system in the set-top-box. Hence, the length of a session can only be a few seconds at the most.

[6] We could not find any public data on the distribution of subscribers to channels. However, if we consider Pay-TV channels, they are either part of the basic subscription or they require additional subscription fees. For channels that are part of the basic subscription, the revoked users will only be those who have unsubscribed from the Pay-TV service altogether. The number of such revoked users will keep growing with time. For any non-basic channel that requires additional subscription fees, it is our guess that at least 40%

the user storage increases almost by 186% while for $r = 0.4n$ the header length decreases by 25.35%. In both these scenarios, a one-time increase in the storage cost will decrease the cumulative communication cost significantly. While the k-ary tree SD scheme works well only after the ratio r/n crosses a threshold, the a-ABTSD scheme always performs better than the NNL-SD scheme in terms of the header length for larger values of their respective parameters k and a. These reductions of the communication overhead can be used to attain significant savings of the bandwidth and hence the cost.

8.3 Implementations

The works in this thesis have been supported by several implementations. They include implementations of the subset cover algorithms, the combinatorial tools like recurrences used for analysis of the subset cover algorithms, probability and expected header length computations, dynamic programming algorithms, and several others. All these programs have been written in the C programming language. A collection of these implementations and some of the respective output files have been uploaded on the web that can be accessible through the link [BS]. These are very basic implementations done as and when required and they do not comply with any coding standard. Here we provide some details.

Each program is kept in a separate directory. A directory name indicates the chapter number (example: "Ch4" denotes Chapter 4) for which the program has been written. Each directory contains a "makefile" that can be used in a Unix-based system to compile the respective program and link it to the appropriate libraries used.

As mentioned before, the analysis done in Chapter 4 played an important role in the understanding of the subset difference technique. The generalization of the NNL-SD scheme was initially done for *incomplete*[7] binary trees for [BS11]. Then, incomplete trees were replaced by complete trees to achieve better results in [BS13]. The implementations of all the subset cover finding algorithms used and proposed in this thesis, have used arrays to store only those nodes in the tree that lie on the paths joining the revoked leaf nodes to the root of the tree. The header length analysis for these schemes were verified using programs written for the worst case and expected values (computed by running the subset cover finding algorithm on random revocation patterns and the combinatorial tools that

of all users of the system will not be subscribing to it.

[7] Trees that may have leaf nodes at any non-root level.

were developed). For large values of n, this analysis was done using a MySQL database to store the results. The behavior of the NNL-SD scheme using dummy users (that may be privileged or revoked) were also analyzed using programs. The dummy users were assumed to be clustered at the right-most end of the tree. Analysis has also been done assuming the distribution of the revoked users at the leaves to be random throughout. The algorithm to compute the expected header length in the CTSD scheme, that was proposed in Chapter 4 helped in generating a lot of relevant data very efficiently.

Chapter 5 deals with the layering of the underlying trees. The primary intent behind layering of the trees, has been to reduce the user storage. Using the implementation of the general layering strategy that we introduced, the individual and collective effect of various layering strategies could be analyzed. The significance of making the root of the tree non-special was verified using this implementation. The dynamic programming algorithm to compute the storage minimal layering has provided us all concrete instances of the SML strategies. The probabilities that the different levels of the underlying tree would generate subsets, was computed. This provided important insights that could be used in determining strategies that may be adopted for layering in a more profitable way. It was understood that it is sufficient to make a portion of the underlying tree behave as in the NNL-SD scheme, in order to reduce storage without significant increase in the expected header length. The constrained minimization of the storage successfully achieved this.

The generalization of the subset difference based schemes using underlying trees of general arity k, was proposed in Chapter 6. We implemented the subset cover finding algorithm for the k-ary tree SD scheme using complete trees. This algorithm has been useful in the header length analysis for random revocation patterns. A program to compute cyclotomic cosets modulo $2^k - 1$ helped us in identifying the additional tree structure that was used in reducing the user storage for the scheme. Comparisons of the k-ary tree SD scheme behavior for $k = 2$ and $k = 3$ were done by finding the user storage and the subset cover for selected revocation patters. The implementation of the a-ABTSD scheme of Chapter 7 was run for random revocation patterns to analyze the expected header length behavior.

8.4 Possible Future Directions

As mentioned in Chapter 1, the problem of securely broadcasting information to users of a system may have requirements specific to applications. The criteria set at the beginning of this thesis may not all be necessary for an application. Each such application may have its own specific needs leading to scopes for optimization and improvement of existing schemes. However, since this thesis has been set with a clear goal, before concluding it let us look at some possible directions of future research that stems out immediately from that goal.

- **More hierarchies of optimization**: Like any BE scheme, the NNL-SD scheme is characterized primarily by the collection of subsets to which keys are assigned. In Figure 1.5 we see that any BE scheme (represented by its collection of subsets) is only a step in the hierarchy of optimization between the singleton set scheme and the power set scheme.
    - In this thesis, we have provided two different techniques for choosing the subsets in the collection to obtain more such steps along the hierarchy between the NNL-SD and the power set scheme. One of them is the a-ABTSD scheme that clearly improves the NNL-SD scheme in terms of the header length. It will be interesting to have other techniques of choosing the collection of subsets that result in smaller header lengths as compare to the NNL-SD scheme with at least as good or possibly better trade-offs among the parameters of BE compared to the ones proposed in this thesis. A very important goal in this context would be the simultaneous improvement of the header length as well as the user storage while the other parameters stay practically usable.
    - The NNL-CS and the HS-LSD schemes are two other steps in the hierarchy between the singleton set scheme and the NNL-SD scheme. Techniques to obtain schemes that would provide more steps in the hierarchy between the singleton set scheme, the NNL-CS scheme, the HS-LSD scheme and the NNL-SD scheme will be interesting for applications where user storage space is constrained.
    - We have explored layering based techniques for obtaining hierarchies between the NNL-SD and HS-LSD. It will also be interesting to know if there are other such techniques to reduce user storage.
    - Obtaining hierarchies of optimization using techniques other than subset difference (some have been enlisted in Chapter 3) will be very interesting.
- **Header Length**: In applications like Pay-TV, header length is the costliest parameter. The following are the possible directions in improving the header length and its analysis.

  - All header length analyses of the NNL-SD based schemes done in this thesis, assume that the distribution of the revoked users are uniform. However, for certain applications, this may not be true. For example, for DRM in optical discs, the users may have been assigned keys such that subtrees in the underlying tree represent regions of sale of the disc players. In such a case, user revocation may be required for the keys of an entire region. The header length analysis of the NNL-SD and other related schemes with more appropriate distributions will be a significant contribution to the area.
  - In symmetric key BE, the header length is one of the most important parameters that determines the cost of usage. Although the Cheon-Jho-Kim-Yoo [JHC$^+$05, CJKY08] work successfully reduces the worst case header length below r, the user storage becomes impractical. Reducing the worst case header length below r with a practical amount of user storage will be an important contribution to this area. As mentioned before, the choice of the collection of subsets $\mathcal{S}$ as well as the technique for assignment of keys will play important roles in this regard.

- The public-key variants of the hierarchies obtained in this thesis (and others) may follow directly from the results by Dodis-Fazio [DF02]. It will be interesting to check if parameters of the public-key versions of specific schemes can be improved.

- In stateless BE schemes, once a user has received secret keys from the center, the user secrets do not change over time. The revocation of users ensure that future broadcasts are secure. However, once a user's secrets are leaked, all previous broadcasts intended for that user (that may have been recorded by an adversary) can be decrypted. Hence, stateless BE schemes (including the NNL-SD scheme and its variants) do not ensure forward security. However, if users can update their own states with time, it may be possible to achieve forward security in BE schemes that receive secret keys only once from the center. Making the NNL-SD and other stateless symmetric key BE schemes forward secure will be important contributions.

We hope that the new techniques and other findings of this thesis will be useful both in theory and practice in taking forward the area of research in symmetric key broadcast encryption and possibly others too.

Bibliography

[AAC] AACS. Advanced Access Content System, http://www.aacsla.com. Accessed on 31st July, 2014.

[ABK98] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A Proposal for the Advanced Encryption Standard, 1998.

[AG05] André Adelsbach and Ulrich Greveler. A Broadcast Encryption Scheme with Free-Riders but Unconditional Security. In Reihaneh Safavi-Naini and Moti Yung, editors, *DRMTICS*, volume 3919 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2005.

[AK08] Per Austrin and Gunnar Kreitz. Lower Bounds for Subset Cover Based Broadcast Encryption. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2008.

[AKI03] Nuttapong Attrapadung, Kazukuni Kobara, and Hideki Imai. Sequential Key Derivation Patterns for Broadcast Encryption and Key Predistribution Schemes. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 2003.

[Asa02] Tomoyuki Asano. A Revocation Scheme with Minimal Storage at Receivers. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2002.

[BBC$^+$08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Sosemanuk, a Fast Software-Oriented Stream Cipher. In Robshaw and Billet [RB08], pages 98–118.

[BBW06] Adam Barth, Dan Boneh, and Brent Waters. Privacy in Encrypted Content Distribution Using Private Broadcast Encryption. In Giovanni Di Crescenzo and Aviel D. Rubin, editors, *Financial Cryptography*, volume 4107 of *Lecture Notes in Computer Science*, pages 52–64. Springer, 2006.

[BCD$^+$99] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr, Luke O'Connor, Mohammad Peyravian, Jr. Luke, O'connor Mohammad Peyravian, David Stafford, and Nevenko Zunic. MARS - a candidate cipher for AES. *NIST AES Proposal*, 1999.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. In Robshaw and Billet [RB08], pages 191–209.

[Bel00] Mihir Bellare, editor. *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*. Springer, 2000.

[Ber91] Shimshon Berkovits. How to Broadcast a Secret. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 535–541. Springer, 1991.

[Ber08] Daniel J. Bernstein. The Salsa20 Family of Stream Ciphers. In Robshaw and Billet [RB08], pages 84–97.

[BF87] Jon Bentley and Bob Floyd. Programming Pearls: A Sample of Brilliance. *Commun. ACM*, 30(9):754–757, September 1987.

[BF99] Dan Boneh and Matthew K. Franklin. An Efficient Public Key Traitor Tracing Scheme. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[Bir07] Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

[BKL$^+$07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *THE PROCEEDINGS OF CHES 2007*. Springer, 2007.

[BKLM09] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Krystian Matusiewicz. Cryptanalysis of C2. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 2009.

[BMS96] Carlo Blundo, Luiz A. Frota Mattos, and Douglas R. Stinson. Trade-offs Between Communication and Storage in Unconditionally Secure Schemes for Broadcast Encryption and Interactive Key Distribution. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 387–400. Springer, 1996.

[BMS98] Carlo Blundo, Luiz A. Frota Mattos, and Douglas R. Stinson. Generalized Beimel-Chor Schemes for Broadcast Encryption and Interactive Key Distribution. *Theor. Comput. Sci.*, 200(1-2):313–334, 1998.

[Bri89] Ernest F. Brickell. Some Ideal Secret Sharing Schemes. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT*, volume 434 of *Lecture Notes in Computer Science*, pages 468–475. Springer, 1989.

[BS] Sanjay Bhattacherjee and Palash Sarkar. Implementations Related to This Thesis, `https://drive.google.com/folderview?id=0B7azs7qqqdS0UnB5aHp3WmJwcDQ&usp=sharing_eil`. Uploaded on 13th August, 2014.

[BS11] Sanjay Bhattacherjee and Palash Sarkar. An Analysis of the Naor-Naor-Lotspiech Subset Difference Algorithm (For Possibly Incomplete Binary Trees). In Daniel Augot and Anne Canteaut, editors, *Workshop on Coding and Cryptography, April 11-15, 2011*, Workshop on Coding and Cryptography, pages 483–492. INRIA, 2011.

[BS13] Sanjay Bhattacherjee and Palash Sarkar. Complete Tree Subset Difference Broadcast Encryption Scheme and its Analysis. *Des. Codes Cryptography*, 66(1-3):335–362, 2013.

[BS14a] Sanjay Bhattacherjee and Palash Sarkar. Concrete Analysis and Trade-Offs for the (Complete Tree) Layered Subset Difference Broadcast Encryption Scheme. *IEEE Trans. Computers*, 63(7):1709–1722, 2014.

[BS14b] Sanjay Bhattacherjee and Palash Sarkar. Reducing Communication Overhead of the Subset Difference Scheme. *IACR Cryptology ePrint Archive*, 2014:577, 2014.

[BS15] Sanjay Bhattacherjee and Palash Sarkar. Tree Based Symmetric Key Broadcast Encryption. *J. Discrete Algorithms*, 34:78–107, 2015.

[BVZ08] Martin Boesgaard, Mette Vesterager, and Erik Zenner. The Rabbit Stream Cipher. In Robshaw and Billet [RB08], pages 69–83.

[Cab] CableLabs. CableLabs, `http://www.cablelabs.com/downloadable-security-and-the-future-of-cablecards/`. Accessed on 31st July, 2015.

[CDF$^+$07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and F. Thayer. RFC 4880 - OpenPGP Message Format. Technical report, Internet Engineering Task Force, November 2007.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing Traitors. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.

[CFNP00] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing Traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CGI$^+$99] Ran Canetti, Juan A. Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *INFOCOM*, pages 708–716, 1999.

[CGZ$^+$04] Weifeng Chen, Zihui Ge, Chun Zhang, James F. Kurose, and Donald F. Towsley. On Dynamic Subset Difference Revocation Scheme. In Nikolas Mitrou, Kimon P.

Kontovasilis, George N. Rouskas, Ilias Iliadis, and Lazaros F. Merakos, editors, *NETWORKING*, volume 3042 of *Lecture Notes in Computer Science*, pages 743–758. Springer, 2004.

[CJKY08] Jung Hee Cheon, Nam-Su Jho, Myung-Hwan Kim, and Eun Sun Yoo. Skipping, cascade, and combined chain schemes for broadcast encryption. *IEEE Transactions on Information Theory*, 54(11):5155–5171, 2008.

[CML] CMLA. Content Management License Administrator, https://www.cm-la.com/. Accessed on 31st July, 2014.

[CMM13] Mickaël Cazorla, Kevin Marquet, and Marine Minier. Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks. In Pierangela Samarati, editor, *SECRYPT*, pages 543–548. SciTePress, 2013.

[CMN99] Ran Canetti, Tal Malkin, and Kobbi Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 459–474. Springer, 1999.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Robshaw and Billet [RB08], pages 244–266.

[CT89] Gerald C. Chick and Stafford E. Tavares. Flexible Access Control with Master Keys. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 316–322. Springer, 1989.

[DF02] Yevgeniy Dodis and Nelly Fazio. Public Key Broadcast Encryption for Stateless Receivers. In Joan Feigenbaum, editor, *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 61–80. Springer, 2002.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public Key Trace and Revoke Scheme Secure Against Adaptive Chosen Ciphertext Attack. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2003.

[DFKY05] Yevgeniy Dodis, Nelly Fazio, Aggelos Kiayias, and Moti Yung. Scalable Public-Key Tracing and Revoking. *Distributed Computing*, 17(4):323–347, 2005.

[DLNa] DLNA. Digital Living Network Alliance, http://www.dlna.org/. Accessed on 31st July, 2015.

[DLNb] DLNA. Everything You Need to Know about DLNA: The De-facto Home Entertainment Network Standard, http://www.techhive.com/article/2020825/how-to-get-started-with-dlna.html. Accessed on 31st July, 2015.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[DRMa] DRM. Digital Rights Management, http://en.wikipedia.org/wiki/Digital_rights_management. Accessed on 31st July, 2014.

[DRMb] DRMtoday. DRMtoday, http://www.drmtoday.com. Accessed on 31st July, 2015.

[DVD] DVDCCA. DVD Copy Control Association, http://www.dvdcca.org/. Accessed on 31st July, 2014.

[EOPR08] Christopher Eagle, Mohamed Omar, Daniel Panario, and Bruce Richmond. Distribution of the Number of Encryptions in Revocation Schemes for Stateless Receivers. In Uwe Roesler, Jan Spitzmann, and Marie-Christine Ceulemans, editors, *Fifth Colloquium on Mathematics and Computer Science*, volume AI of *DMTCS Proceedings*, pages 195–206. Discrete Mathematics and Theoretical Computer Science, 2008.

[ER05] Michael A. Einhorn and Bill Rosenblatt. Peer-to-Peer Networking and Digital Rights Management: How Market Tools Can Solve Copyright Problems, http://www.cato.org/publications/policy-analysis/peerpeer-networking-digital-rights-management-how-market-tools-\can-solve-copyright-problems, 2005. Accessed on 31st July, 2014.

[FAS] FAS. Federation of American Scientists, http://fas.org/spp/military/program/com/gbs.htm. Accessed on 31st July, 2014.

[FCC06] FCC. FCC Annual Report 2006, https://apps.fcc.gov/edocs_public/attachmatch/FCC-06-11A1.pdf, 2006. Accessed on 31st July, 2015.

[FKTS08] K. Fukushima, S. Kiyomoto, T. Tanaka, and K. Sakurai. Ternary Subset Difference Method and Its Quantitative Analysis. In *The 9th International Workshop* *on Information Security Applications (WISA2008)*, volume 5379, pages 225–239. LNCS, 2008.

[FN93] Amos Fiat and Moni Naor. Broadcast Encryption. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[FT01] Amos Fiat and Tamir Tassa. Dynamic Traitor Tracing. *J. Cryptology*, 14(3):211–223, 2001.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A New Family of Lightweight Block Ciphers. In Ari Juels and Christof Paar, editors, *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GPPR12] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. *IACR Cryptology ePrint Archive*, 2012:600, 2012.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient Tree-Based Revocation in Groups of Low-State Devices. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004.

[GSW00] Juan A. Garay, Jessica Staddon, and Avishai Wool. Long-Lived Broadcast Encryption. In Bellare [Bel00], pages 333–352.

[GSY99] Eli Gafni, Jessica Staddon, and Yiqun Lisa Yin. Efficient Methods for Integrating Traceability and Broadcast Encryption. In Wiener [Wie99], pages 372–387.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain Family of Stream Ciphers. In Robshaw and Billet [RB08], pages 179–190.

[HS02] Dani Halevy and Adi Shamir. The LSD Broadcast Encryption Scheme. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002.

[HSH$^+$06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[ISSK09] Maryam Izadi, Babak Sadeghiyan, Seyed Saeed Sadeghian, and Hossein Arabnezhad Khanooki. MIBS: A New Lightweight Block Cipher. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS*, volume 5888 of *Lecture Notes in Computer Science*, pages 334–348. Springer, 2009.

[JG04] Shaoquan Jiang and Guang Gong. Multi-service Oriented Broadcast Encryption. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP*, volume 3108 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2004.

[JHC$^+$05] Nam-Su Jho, Jung Yeon Hwang, Jung Hee Cheon, Myung-Hwan Kim, Dong Hoon Lee, and Eun Sun Yoo. One-Way Chain Based Broadcast Encryption Schemes. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2005.

[JKL06] Mattias Johansson, Gunnar Kreitz, and Fredrik Lindholm. Stateful Subset Cover. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 178–193, 2006.

[JL07] Hongxia Jin and Jeffery Lotspiech. Renewable Traitor Tracing: A Trace-Revoke-Trace System For Anonymous Attack. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 563–577. Springer, 2007.

[JL09] Hongxia Jin and Jeffrey B. Lotspiech. Unifying Broadcast Encryption and Traitor Tracing for Content Protection. In *ACSAC*, pages 139–148. IEEE Computer Society, 2009.

[KM15] Neal Koblitz and Alfred Menezes. The random oracle model: A twenty-year retrospective. *IACR Cryptology ePrint Archive*, 2015:140, 2015.

[KRS99] Ravi Kumar, Sridhar Rajagopalan, and Amit Sahai. Coding Constructions for Blacklisting Problems without Computational Assumptions. In Wiener [Wie99], pages 609–623.

[KY01] Aggelos Kiayias and Moti Yung. Self Protecting Pirates and Black-Box Traitor Tracing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2001.

[LK05] Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In JooSeok Song, Taekyoung Kwon, and Moti Yung, editors, *WISA*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2005.

[LM90] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In Ivan Damgård, editor, *EUROCRYPT*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.

[LPPS07] Gregor Leander, Christof Paar, Axel Poschmann, and Kai Schramm. New Lightweight DES Variants. In Biryukov [Bir07], pages 196–210.

[LS98] Michael Luby and Jessica Staddon. Combinatorial Bounds for Broadcast Encryption. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 1998.

[Mih03] Miodrag J. Mihaljevic. Key Management Schemes for Stateless Receivers Based on Time Varying Heterogeneous Logical Key Hierarchy. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 2003.

[MMW09] Thomas Martin, Keith M. Martin, and Peter R. Wild. Establishing the Broadcast Efficiency of the Subset Difference Revocation Scheme. *Des. Codes Cryptography*, 51(3):315–334, 2009.

[MP88] Chris J. Mitchell and Fred Piper. Key Storage in Secure Networks. *Discrete Applied Mathematics*, 21(3):215–228, 1988.

[MPA] MPAA. Motion Picture Association of America, `http://www.mpaa.org/`. Accessed on 31st July, 2014.

[MQ95] Benoit Macq and Jean-Jacques Quisquater. Cryptology for digital TV broadcasting. In *Proceedings of the IEEE*, volume 83, pages 944–957, 6 1995.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-holland Publishing Company, 2nd edition, 1978.

[Mus] Muslix64. HD-DVD content protection already hacked?, `http://www.techamok.com/?pid=1849`. Accessed on 31st July, 2014.

[MV01] Yi Mu and Vijay Varadharajan. Robust and Secure Broadcasting. In C. Pandu Rangan and Cunsheng Ding, editors, *INDOCRYPT*, volume 2247 of *Lecture Notes in Computer Science*, pages 223–231. Springer, 2001.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

[NNL02] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. *Electronic Colloquium on Computational Complexity (ECCC)*, (043), 2002.

[NP98] Moni Naor and Benny Pinkas. Threshold Traitor Tracing. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 1998.

[NP00] Moni Naor and Benny Pinkas. Efficient Trace and Revoke Schemes. In Yair Frankel, editor, *Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2000.

[NP10] Moni Naor and Benny Pinkas. Efficient Trace and Revoke Schemes. *Int. J. Inf. Sec.*, 9(6):411–424, 2010.

[NRK03] Arvind Narayanan, C. Pandu Rangan, and Kwangjo Kim. Practical Pay-TV Schemes. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2003.

[PB06] E. C. Park and Ian F. Blake. On the Mean Number of Encryptions for Tree-Based Broadcast Encryption Schemes. *J. Discrete Algorithms*, 4(2):215–238, 2006.

[Pfi96] Birgit Pfitzmann. Trials of Traced Traitors. In Ross J. Anderson, editor, *Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 1996.

[PGMM02] Carles Padró, Ignacio Gracia, Sebastià Martín Molleví, and Paz Morillo. Linear Key Predistribution Schemes. *Des. Codes Cryptography*, 25(3):281–298, 2002.

[PGMM03] Carles Padró, Ignacio Gracia, Sebastià Martín Molleví, and Paz Morillo. Linear Broadcast Encryption Schemes. *Discrete Applied Mathematics*, 128(1):223–238, 2003.

[Pin04] Benny Pinkas. Efficient State Updates for Key Management. *Proceedings of the IEEE*, 92(6):910–917, 2004.

[PW97] Birgit Pfitzmann and Michael Waidner. Asymmetric Fingerprinting for Larger Collusions. In Richard Graveman, Philippe A. Janson, Clifford Neumann, and Li Gong, editors, *ACM Conference on Computer and Communications Security*, pages 151–160. ACM, 1997.

[RB08] Matthew J. B. Robshaw and Olivier Billet, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[Res] Cryptography Research. Cryptography Research Inc., http://www.cryptography.com/. Accessed on 31st July, 2014.

[Ros] Bill Rosenblatt. The New Technologies for Pay TV Content Security, http://www.irdeto.com. Accessed on 31st July, 2014.

[RRYS98] Ronald L. Rivest, M. J. B. Robshaw, Y.L. Yin, and R. Sidney. The RC6 Block Cipher, 1998.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[SIH$^+$11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2011.

[ski98] SKIPJACK and KEA algorithm specifications, http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf. Technical report, May 1998.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-Bit Block Cipher. In *in First Advanced Encryption Standard (AES) Conference*, 1998.

[SKW+99] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. *The Twofish Encryption Algorithm: A 128-bit Block Cipher*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[SM03] Alan T. Sherman and David A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Trans. Software Eng.*, 29(5):444–458, 2003.

[SMMK12] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. ${{twine}}$: A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[SNW00] Reihaneh Safavi-Naini and Yejing Wang. Sequential Traitor Tracing. In Bellare [Bel00], pages 316–332.

[SPD] SPDC. Self-Protecting Digital Content, http://www.cryptography.com/public/pdf/SelfProtectingContent.pdf. Accessed on 31st July, 2014.

[SPGQ06] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In Josep Domingo-Ferrer, Joachim Posegga, and Daniel Schreckling, editors, *CARDIS*, volume 3928 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2006.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Biryukov [Bir07], pages 181–195.

[SSW01a] Alice Silverberg, Jessica Staddon, and Judy L. Walker. Efficient Traitor Tracing Algorithms Using List Decoding. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 175–192. Springer, 2001.

[SSW01b] Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial Properties of Frameproof and Traceability Codes. *IEEE Transactions on Information Theory*, 47(3):1042–1049, 2001.

[SSW03] Alice Silverberg, Jessica Staddon, and Judy L. Walker. Applications of List Decoding to Tracing Traitors. *IEEE Transactions on Information Theory*, 49(5):1312–1318, 2003.

[Sti97] Douglas R. Stinson. On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption. *Des. Codes Cryptography*, 12(3):215–243, 1997.

[SvT98] Douglas R. Stinson and Tran van Trung. Some New Results on Key Distribution Patterns and Broadcast Encryption. *Des. Codes Cryptography*, 14(3):261–279, 1998.

[SW98a] Douglas R. Stinson and Ruizhong Wei. Combinatorial Properties and Constructions of Traceability Schemes and Frameproof Codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

[SW98b] Douglas R. Stinson and Ruizhong Wei. Key Preassigned Traceability Schemes for Broadcast Encryption. In Stafford E. Tavares and Henk Meijer, editors, *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 144–156. Springer, 1998.

[TT01] Wen-Guey Tzeng and Zhi-Jia Tzeng. A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2001.

[TT05] Wen-Guey Tzeng and Zhi-Jia Tzeng. A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares. *Des. Codes Cryptography*, 35(1):47–61, 2005.

[WGL00] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.

[WHA99] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures. RFC 2627 (Informational), June 1999.

[Wie99] Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

[Wik] Wiki. Wikipedia, `http://en.wikipedia.org`. Accessed on 31st July, 2014.

[WN94] David J. Wheeler and Roger M. Needham. TEA, a Tiny Encryption Algorithm. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.

[Woo98] Avishai Wool. Key Management for Encrypted Broadcast. In Li Gong and Michael K. Reiter, editors, *ACM Conference on Computer and Communications Security*, pages 7–16. ACM, 1998.

[Woo00] Avishai Wool. Key Management for Encrypted Broadcast. *ACM Trans. Inf. Syst. Secur.*, 3(2):107–134, 2000.

[Wu08] Hongjun Wu. The Stream Cipher HC-128. In Robshaw and Billet [RB08], pages 39–47.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In Javier Lopez and Gene Tsudik, editors, *ACNS*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344, 2011.