Some Results On Analysis And Implementation Of HC-128 Stream Cipher

A thesis presented to Indian Statistical Institute
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Computer Science

by

Shashwat Raizada

under the supervision of

Professor Subhamoy Maitra

Applied Statistics Unit
INDIAN STATISTICAL INSTITUTE
Kolkata, West Bengal, India

Submitted: May 2014, Revised: January 2015

Patience in the present,
faith in the future,
and joy in the doing.

(George Perera)

Abstract

The HC-128 stream cipher is a successful entrant in the eStream candidate list (software profile) and is the lighter variant of HC-256 stream cipher. Apart from the analysis by the designer of the cipher (Hongjun Wu) to conjecture the security of this cipher, there are only a few other observations on this cipher despite being the focus of researchers during the three phases of eStream evaluation and later efforts in the community. Till date none of the security claims in favor of HC-128 by the designer could be broken. One may expect HC-128 stream cipher to be popular in commercial domain in near future, if not already so. This thesis presents a rigorous study in different aspects of this stream cipher covering combinatorial analysis, distinguishers, design modification proposal, side channel analysis on this cipher and finally implementation strategies.

We first show that the knowledge of any one of the two internal state arrays of HC-128 along with the knowledge of 2048 keystream words is sufficient to construct the other state array completely in 2^{42} time complexity. This analysis reveals a structural insight into the cipher's internal state along with theoretically establishing some novel combinatorial properties of HC-128 keystream generation algorithm.

Next, using linear approximation of the addition modulo 2^n of three n-bit integers, we identify linear approximations of g_1, g_2, the feedback functions of HC-128. Here we show that the process of keystream output generation of HC-128 can be well approximated by linear functions wherein the "least significant bit" based distinguisher (presented by the designer of the cipher) of HC-128

can be extended for the other bits of the 32-bit word. Further, using the above linear approximations of g_1, g_2, we also present several other distinguishers in the line of the distinguisher proposed by the designer of the cipher. We also study how HC-128 keystream words leak secret state information of the cipher due to the properties of the functions h_1, h_2 and present improved results.

The third major aspect covered in this thesis is on side channel attacks against HC series of stream ciphers. We extend the existing HC-128 fault attack and the HC-256 cache analysis onto the HC-256 and HC-128 ciphers respectively under similar models. The techniques applied on one variant is not trivially translatable to the other and the issue was left open until this work. Here we propose a technique to recover half the state of HC-128 using cache analysis, which can be cascaded with the differential attack towards a full state recovery and hence key recovery. In a similar line, we analyze the state leakage of HC-256 under differential fault attack model to achieve partial state recovery.

We finally study several implementation issues for HC-128 in a disciplined manner. HC-128 is primarily designed as a software stream cipher aiming for sequential execution on general purpose processors and so we first carry out implementations in this direction on embedded and customizable processors. Next we consider the ASIC implementation for a co-processor design that will house such ciphers. Further we explore several parallelization strategies for faster execution of the cipher. We present a detailed implementation exercise for the HC-128 stream cipher on special purpose hardware.

In summary, though we could not break any security conjecture made by the designer for HC-128, our analysis explores different aspects of the cipher from analysis, design and implementation. Our work has also stimulated further research on this cipher that is evident from the literature.

Acknowledgments

Praise the bridge that carried you over. – George Coleman

Several people have contributed and helped me in myriad ways because of which this thesis has become a reality. I would like to thank them all for their guidance, suggestions and advice throughout the tenure of my Doctoral studies.

With utmost humility, I submit that this thesis would never see the light of day without the unstinted support of several individuals. Foremost amongst them is my supervisor Prof. Subhamoy Maitra. It has been my proud privilege to have worked with him. Not only did he introduce me to the fascinating and challenging world of stream ciphers but also taught me the art of doing fundamental research. A special and warm mention of thanks to Dr. Goutam Paul who besides being a co-author has been a supportive friend, mentor and guide. I gratefully acknowledge the help given by Dr. Sourav Sen Gupta in navigating me through errors and typos across the multiple drafts of my thesis.

The Cryptology Research Group at the Indian Statistical Institute (ISI) has become the foremost breeding ground for aspiring cryptographers in the country. I would thank its inspirational leaders Prof. Bimal Roy, Prof. Palash Sarkar and Prof. Rana Barua, besides Prof. Subhamoy Maitra, for all the motivation besides incubating a culture academic excellence and synergy in the group. On a similiar note I wish to thank all the student members of our cryptology lab for maintaining lively research atmosphere there. I had a great time working in the lab, and have made very good friends. A special word of

thanks to all my faculty and student colleagues in the institute who extended immense help to me during my stay in Kolkata.

I am indebted to my loving parents and brothers for their unstinted enthusiasm and encouragement to take up the doctoral studies. Note of special thanks to my in-laws, my little son Ishaan and my wife Shweta who have been a bolster of support. Their continuous perseverance during the long hours of research has been a source of strength.

Contents

Abstract iii
Contents vii
List of Tables xi
List of Figures xiii

1 Introduction **1**

1.1 Genesis of cryptology 1
1.2 Modern cryptology 2
1.2.1 Attributes of cryptography 3
1.2.2 Functional model of a cryptosystem 4
1.3 Categorization of cryptosystems 6
1.3.1 Symmetric Key cryptosystem 6
1.3.2 Public key cryptosystem 8
1.4 Adversary and the attack models 11
1.5 Overview of stream ciphers 13
1.5.1 Vernam cipher to electro-mechanical rotors 13
1.5.2 Digitization of stream ciphers 15
1.5.3 State-of-the-art in stream ciphers 19
1.6 Stream cipher cryptanalysis 20

1.6.1 Theoretical attack models 21
1.6.2 Side-channel attacks 23
1.7 Motivation for the thesis 25
1.8 Contribution and thesis plan 26
1.9 Prerequisites for the reader 27

2 Background **29**
2.1 The HC series of Stream Ciphers 29
2.2 Description of HC-128 30
2.2.1 Operators and structures 30
2.2.2 Initialization process 31
2.2.3 Keystream generation 32
2.3 Description of HC-256 32
2.4 Chronology of recent works 34
2.4.1 Theoretical Cryptanalysis 34
2.4.2 Side Channel Cryptanalysis 36
2.4.3 Cipher Implementation and Usage 37
2.5 Publications included in this Thesis 39

I Analysis of HC-128 Stream Cipher **41**

3 Internal Structure **43**
3.1 Reconstruction of one array from another 44
3.1.1 Notations and problem formulation 45
3.1.2 State reconstruction strategy 45
3.2 Design modifications 56
3.3 Performance Evaluation 57
3.4 Conclusion 58

4 Study of HC-128 Keystream **59**

4.1 Introduction 59

4.2 Linear approximation of the functions g_1 and g_2 61

4.3 A class of distinguishers by extending the LSB-based distinguisher 65

4.3.1 Brief outline of the LSB distinguisher of HC-128 65

4.3.2 Our extension to other bits 67

4.4 A new distinguisher 71

4.5 State leakage in keystream 75

4.6 Conclusion 78

II Side Channel and Implementation Issues **81**

5 Side Channel Attacks and Impact on HC series of Stream Ciphers **83**

5.1 Motivation 83

5.1.1 Layout of the chapter 84

5.2 Cache and fault attack 85

5.2.1 Cache analysis attack 85

5.2.2 Fault attack 86

5.3 Cache analysis of HC-128 87

5.3.1 Bits obtainable from cache information 87

5.3.2 Constructing bytes 0 and 2 of each array element 88

5.3.3 Finding the remaining sixteen bits for each element . . . 89

5.4 Fault attack on HC-256 89

5.4.1 Faulty Q entering in computation of h_1 90

5.4.2 Faulty Q entering in update of P 92

5.4.3 Differentiating the two cases and obtaining additional elements 94

5.5 Conclusion 94

6 HC-128 Implementation in Hardware **97**

6.1 Motivation and contributions 97

6.2 HC-128 on general purpose CPU's 98

6.3 Experiment with embedded & customizable processors 100

6.3.1 Implementation on embedded processors 100

6.3.2 Implementation on customizable processor 100

6.4 Hardware accelerator implementation of HC-128 102

6.4.1 Implementation of Keystream Generation 103

6.5 Parallelization strategies 104

6.5.1 Parallelizing initialization with keystream generation . . 104

6.5.2 Parallelizing keystream generation 105

6.5.3 Odd-even memory partitioning 108

6.6 Conclusion 109

7 Conclusion and Future Work **111**

7.1 Summary 111

7.2 Future works and open problems 113

7.3 Final words 115

Bibliography **117**

List of Tables

3.1 The evolution of P, Q arrays. 44

3.2 Speed comparisons of our proposals with HC-128 and HC-256. . 58

4.1 Linear Approximaton of Addition for first 8 bits. 61

4.2 Evolution of the Arrays P and Q and Correspondence with the Keystream Words. 65

5.1 Cache block size vs. number of bits learned. 88

5.2 Number of words with first ten bits leaked vs. number of subsequent updates incorporated. 94

6.1 HC-128 Key and IV initialization on general purpose processors 99

6.2 HC-128 Keystream generation on general purpose processors . . 99

6.3 HC-128 Key and IV initialization on embedded & customizable embedded processors. 101

6.4 HC-128 keystream generation on embedded & customizable embedded processors. 102

6.5 Performance of hardware accelerator 106

6.6 Pipeline organization with odd-even memory partitioning 108

List of Figures

1.1 Typical model for a Cryptosystem. 4
1.2 Encryption using a Stream Cipher. 14
1.3 LFSR based Filter Generator 17
1.4 LFSR based Combination Generator. 18
1.5 LFSR based Clock-controlled Generators 18

5.1 Number of bytes of P array elements leaked from h_1 function . . 92
5.2 Probability of finding the first ten bits of P array elements from several updates 93

6.1 Pipeline structure for keystream generation 103
6.2 Pipeline structure for parallel keystream generation 107

Chapter 1

Introduction

"The best things are simple, but finding these simple things is not simple." – Anonymous

1.1 Genesis of cryptology

The desire to discreetly store and disseminate information is as old as humanity itself. Besides it has always been the inquisitive human nature to seek access to knowledge that is not officially available. This was illustrated in the earliest Biblical story of the Garden of Eden where "the tree of the knowledge of good and evil" produced the forbidden fruit that Adam and Eve ate in belief of obtaining the secret knowledge, and were banished forever. The prehistoric man tried to manage information in the cave paintings that are believed to bear some code. As the race evolved there were a number of different ways of discreetly passing on information. These ranged from the coded messages in drum beats to the uncanny smoke signals visible from a distance.

The study of techniques that enabled keeping sensitive information secret, evolved with mankind into a full fledged discipline, today known as *Cryptography*. Concurrently, the field pertaining to techniques for testing the vulnerabilities of cryptography, and thereby obtaining sensitive information, evolved into another discipline known as *Cryptanalysis*. These two subjects with apparently

contradictory aims are jointly referred to as *Cryptology*. The word 'Cryptology' is derived by joining two Greek root words 'kryptós' (hidden) and 'logia' (study). Thus cryptology is the study and practice of hiding information, or the art and science of information security.

Amongst the earliest cryptographic devices was the *Scytale*, popular in the classical Greek civilization (around 400 BC). The device was shaped like a baton with strips of parchment wound around to perform what we now know as a *transposition* cipher. Surprisingly around the same time, the Indian treatise Kamasutra suggests cryptography as one of the important skills that lovers should learn to communicate discreetly. The Romans under Julius Caesar devised the simple alphabetic shift cipher, which has formed the logical basis for the *substitution* operation – a feature present in modern day ciphers.

Subsequently, there have been several epochs in cryptography, each with its own unique underlying technology. These ranged from linguistics to steganography, from manual systems to electro-mechanical systems, and pervaded through the analog era to the modern digital ubiquity.

1.2 Modern cryptology

Contemporary Cryptology can be attributed to be born with the classified work done by Claude Shannon during the World War II, later (in 1949) published in the Bell System Technical Journal as "Communication Theory of Secrecy Systems" [111]. Shannon also co-authored a book with Warren Weaver titled "Mathematical Theory of Communication" [112] that brought out the work done during the war. These in addition to other works in information and communication theory form the initial structure of mathematics based cryptology. These gave birth to novel cryptographic devices that ranged from purely mechanical devices to digital ones. Although these devices were state-of-the-art, they however suffered certain shortcomings that are inherent in symmetric cryptosystems. There existed an inescapable requirement of a secure channel to transfer keys between the sender and receiver. This problem gets compounded when the number of communicating parties increase, and the number of keys required explode.

The next paradigm shift occurred a quarter century later, when Whitfield Diffie and Martin Hellman in 1976 laid down the functional basis for a novel cryptosystem in "New Directions in Cryptography" [35]. This introduced a radically innovative method for handling cryptographic keys. The article also stimulated the almost immediate design and development of a new class of enciphering algorithms, known as Asymmetric or Public Key Cryptography.

1.2.1 Attributes of cryptography

Cryptology has become a key enabler for all the processes of modern digital economy. The requirement of cryptology is turning all-pervasive so as to cover major aspects of everyday life, including Internet banking, Identification cards, secure databases, e-mails, social networking, secure telephony, etc. The attributes that modern-day cryptology aims to enable can be summarized in the following categories.

Confidentiality. The aim achieved by confidentiality is to ensure secure data communication over insecure channels, despite the possible presence of a malicious adversary. This is achieved by sender encrypting the data which is later decrypted by the recipient.

Authentication. The objective of (entity) authentication is to ensure proper identification of the communicating party. This feature is required to ensure that any transaction or communication occurs only between duly authorized parties.

Data Integrity. The objective of integrity is to make sure that the data has not been tampered with. This is to verify the consistency of the data and detect if any illegitimate alterations have been made to the original communication.

Non-Repudiation. The aim of non-repudiation is primarily necessary from a legal angle wherein, an action carried out by an entity can be duly and be verifiably imputed with. In simple words this refers to the undeniable proof of a specific action taken by an entity, to be presented in the court of law or to any third-party arbitrator.

Cryptographic primitives can be broadly classified into different types depending on their usage and functions. These are briefly described in the following subsections. One can also refer to "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography" by S. Singh [115] and "The Codebreakers – The Story of Secret Writing" by D. Kahn [57] for an exciting exposition on the evolution of the subject. For a technical account on modern day Cryptology, one may refer to "Cryptography – Theory and Practice" by D. R. Stinson [121] and "Handbook of Applied Cryptography" by A. J. Menezes, P. C. van Oorschot and S. A. Vanstone [88].

1.2.2 Functional model of a cryptosystem

A functional approach of describing a cryptosystem personifies across three actors, conventionally named Alice, Bob and Oscar. In this model Alice and Bob want to securely communicate over an insecure channel to which Oscar is privy to. Encryption in this case is the enabler that helps Alice and Bob achieve their aim of converting their meaningful communication into apparently meaningless data while traversing the insecure channel. Oscar aims at eavesdropping into the channel to de-scramble the data and derive some sensible information from it, using techniques called cryptanalysis. The deployment model is described in Figure 1.1.

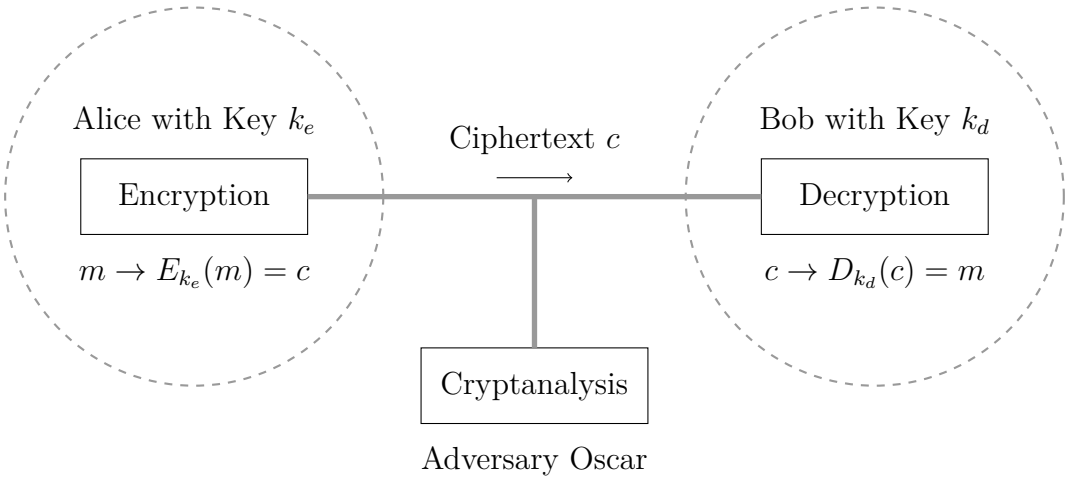

Figure 1.1: Typical model for a Cryptosystem.

The following definitions are essential to describe the components:

Plaintext. The information that Alice or Bob want to share is called the *plaintext*. It can be a natural language, audio, video or a computer file, and is denoted by m.

Message Space. The set of all possible plaintexts is called *message space*, denoted by M. Thus the plaintext message $m \in M$.

Ciphertext. The information sent on the insecure channel in a cryptic format, so that an unauthorized person cannot decipher, is called the *ciphertext*, denoted by c.

Ciphertext Space. The set of all possible ciphertexts is called *ciphertext space*, denoted by C. Thus the ciphertext $c \in C$.

Key. A sequence of bits of predefined length used to control cryptographic transformations like encryption and decryption is termed as key. The key used for encryption is denoted by k_e and that for decryption by k_d.

Key Space. The finite set of all possible keys (of a predefined length) is called *key space*, denoted by K. Thus, $k_d, k_e \in K$.

Encryption. The transformation of the plaintext to ciphertext is called *encryption*. Mathematically, $E : M \times K \to C$ is a function or algorithm that takes as input a plaintext $m \in M$ and an encryption key $k_e \in K$ to produce a ciphertext $c \in C$.

Decryption. The transformation from the ciphertext to plaintext is called *decryption*. Mathematically, $D : C \times K \to M$ is a function or algorithm that takes as input a ciphertext $c \in C$ and a decryption key $k_d \in K$ to produce a plaintext $m \in M$.

Cryptosystem. A cryptosystem (primarily the encryption-decryption model) as a whole may be defined as the composition of the message space M, ciphertext space C and the key space K, such that there exist an encryption algorithm $E : M \times K \to C$ as well as a decryption algorithm $D : C \times K \to M$ satisfying the condition $D(k_d, E(k_e, m)) = m$ for suitable pair of keys $k_e, k_d \in K$ and for all plaintexts $m \in M$.

1.3 Categorization of cryptosystems

Cryptosystems are broadly divided as per the generation and usage of keys. There are two distinct types of cryptosystems:

1. *Private key or Symmetric Key System.* These systems mandate that, the key used for encryption is the same as that for decryption.
2. *Public Key or Asymmetric Key System.* These systems permit that, the key used for encryption to be different from the key used for decryption. Furthermore, the encryption key may be publicly known.

1.3.1 Symmetric Key cryptosystem

Cryptosystems have traditionally been designed using the secret key paradigm. This was because the user set was finite, and the number of shared secret keys was more or less bounded to a practical limit. The keys that are used for encryption are same as that required for decryption. In a typical scenario, sender and receiver require to agree upon a key prior to communication. Thus an elaborate procedure of setting up a secure channel for distribution of cryptographic keys is required. All encryption and decryption are thereafter done using the predetermined shared keys. In other words, here we have a single secret key $k = k_e = k_d$, upon whose secrecy lies the strength of the cryptosystem. Hence the name symmetric or secret key cryptography. Symmetric key systems use two main cryptographic primitives:

Block Cipher. This class of cipher takes in successive chunks of plaintext in finite size denominations known as blocks. Each block is encrypted by applying a key dependent transformation on its bits. This transformation is actually a memory-less (or state-less) permutation, and the decryption typically consists of applying the reverse of key-dependent permutation. Two successful templates around which a number of block cipher designs are modeled are the Substitution-Permutation Network (SPN) [59] and the Feistel structure [78]. Both these approaches involve a complicated mixing of message and key bits which are an integral part of the block ciphers. Modern Block ciphers have a key schedule module

that generates a series of small sub keys from the main keys; thereby providing resistance against Meet-in-the-Middle attacks. In addition they have non-linear building blocks that provide resistance to linear and differential cryptanalysis. The first global standardization of block cipher commenced in 1973 and lead to the promulgation of the *Data Encryption Standard* (DES), which is believed to be derived from an earlier IBM cipher Lucifer. This was done at the behest of the Department of Commerce of United states by the National Institute of Standards and Technology (formerly National Bureau of Standards). DES was in use for over two decades, its major weakness being the small key size (56 bits) that was not sufficient to withstand exhaustive search analysis by modern processors. This weakness was circumvented by using three instances of DES concurrently in a product mode, known as the Triple-DES. Later in 1997, NIST employed a transparent procedure to select a candidate to replace DES. A three year effort led to the promulgation of a new standard known as the *Advanced Encryption Standard* (AES), primarily based on an improved block cipher called Rijndael [32].

Stream Cipher. A stream cipher generates a pseudo-random stream of bits (known as keystream) using a chosen key and an initialization vector of fixed length. The keystream is simply XOR-ed with the plaintext to encrypt, and is XOR-ed with the ciphertext stream to obtain the plaintext back. It can generally be modeled as a finite state machine with internal memory that breaks a plaintext m into successive characters or bits $m_1, m_2, \ldots$, and encrypts each m_r with a different key segment k_r of the keystream $k = k_1, k_2, \ldots$. We have $E_k(m) = E_{k_1}(m_1) \| E_{k_2}(m_2) \| \ldots$ where each encryption is a simple XOR. The decryption function is the same finite state machine that generates the same keystream from an input secret key, and extracts the plaintext from the ciphertext. An overview of stream ciphers is discussed in Section 1.5 of this chapter.

Symmetric Key systems suffer from two major challenges in the context of key management. First, there needs to be a secure channel provisioned between the communicating parties for the initial sharing of the secret key, and second, the system needs to be scalable for multiple users. In a typical

scenario, for n communicating parties, we need to establish $\binom{n}{2}$ or $O(n^2)$ keys across that many secure channels (one for each pair of users).

Key establishment protocols is the study of techniques to address issues of *key distribution* and *key agreement* in symmetric systems [20, 23]. There are myriad of techniques such as Kerberos, trusted key distribution center, key pre-distribution etc., but each has its trade-offs regarding practicality, security and other pertinent parameters.

1.3.2 Public key cryptosystem

Public Key Cryptography was born out of the trials and tribulations of two researchers Whitfield Diffie and Martin Hellman, articulated in the book "Crypto: How the Code Rebels Beat the Government – Saving Privacy in the Digital Age" [75]. It was presented as a concept paper in 1976 [34, 35]. However, according to documents de-classified in 1997, it was revealed that public key algorithms were developed as early as 1973 by James H. Ellis, Clifford Cocks, and Malcolm Williamson under a secret project by UK Government Communications Headquarters (GCHQ). The paper titled "New Directions in Cryptography" [35] was truly a game-changer as it bought cryptography from the elite cartels of political and military organizations to the common masses.

In public key or asymmetric key cryptography, the encryption and the decryption keys are different albeit related by some mathematical relation and together constitute a key pair. In a typical scenario, the sender has the receiver's public key k_e and uses an algorithm to encrypt the message in such a fashion that it can be decrypted using a corresponding private key k_d, securely maintained by the receiver. The part of the key which is used to encrypt is made publicly available, while the decryption key is kept secret. The security of the system vests in the computational infeasibility of tracing out the decryption key with the mere knowledge of encryption key. Public key cryptosystems employ one way functions with trapdoors, typically based on some well known computationally hard mathematical problem. An interesting fact in the public key saga is that the seminal paper [34, 35] was presented just as a concept paper with no concrete examples whatsoever. Subsequently proposals emerged, that would implement concrete public key systems, using hard mathematical

problems. Some of these mathematical problems are listed below.

Knapsack Problem. The knapsack problem, also known as the subset sum problem, aims at the possibility for a set of n positive integers $\{k_i\}$ and a positive integer N to be represented as $N = \sum_{i=1}^{n} a_i \times k_i$ where $a_i \in \{0, 1\}$. Popular proposals based on this approach, like Merkle-Hellman cryptosystem [89], were however found to be weak. A colloquial account on the cryptanalysis of such systems is given in [75], while the technical literature for the same is available in [107, 108].

Integer Factorization Problem. This problem is based on the hardness of finding factors for a large number, and has been the core of the first usable public key cryptosystem RSA [100], developed by Rivest, Shamir and Adleman in 1977. An alternative approach, known as the Rabin cryptosystem [98], also uses the integer factorization problem as its base.

Quadratic Residue Problem. This problem is based on the decision presence of a modular square root for any element $x \in \mathbb{Z}_N$ with the condition that N is a composite number. If the factors of N are known, then QRP in $\mathbb{Z}_N$ is no longer hard. Goldwasser and Micali [42] have proposed cryptosystems based on the Quadratic Residue Problem.

Discrete Logarithm Problem. This problem is based on a large cyclic group $G = \langle g \rangle$, where the issue is to find the exponent a satisfying $y = g^a$ in G. El Gamal cryptosystem [40] is a public key system based on DLP.

Elliptic Curve DLP. Elliptic Curve Cryptography was proposed independently by Koblitz [68] and Miller [90] and is based on DLP in the group of rational points on the elliptic curves. There is no known sub-exponential time algorithm to solve ECDLP, making it suitable for cryptographic applications. In 1989, Neal Koblitz [69] suggested the use of hyper elliptic curves in cryptography with the advantage of having smaller base field over the elliptic curves for the same security level. Elliptic curve based systems have high encryption throughput and are suitable for lightweight cryptographic hardware.

Shortest Vector Problem and Closest Vector Problem SVP. SVP deals with the problem of obtaining the shortest non-zero vector in a high dimen-

sional lattice, and the same idea may be generalized to the Closest Vector Problem. These are hard problems, and a few modern cryptosystems like NTRU [50], Ajtai-Dwork system [5, 6] and [41] are built upon them.

In a public key setting, each communicating entity needs to have a key pair. Thus, for n communicating parties, we need only n key-pairs, i.e., a pair for each person. As brought out in [49], the principle goal of public key cryptography is to allow two entities to exchange confidential information, even if they have never met before. They can communicate only via an insecure channel, monitored by an adversary. The obvious challenge is to devise the mechanism of trusting a public key as the one originating from its reported owner. This authentication of the public keys can be done using undermentioned techniques.

Certifying Agencies. These are trusted agencies running Public Key Infrastructure (PKI), who sign the public keys of all users. The signed keys are known as certificates, and the PKI [2] is a secure system that is used to manage and control these certificates. In such systems there is an implicit trust on the Certifying Authority (CA).

Pretty Good Privacy. Designed by Phil Zimmerman, PGP builds a *web of trust* involving only the users, based on which one of them may take a decision on the status of a public key. PGP does away with the requirement of having dedicated CAs.

Identity Based Encryption. IBE is a logical expansion of public key cryptographic primitive wherein the public key of a subscriber can be chosen to be a publicly known (and trusted) value, such as his/her trusted identity [39, 109], like email id, social security number etc.

The drawback of public key cryptosystems is that they are slower in performance, and require more computation power via-a-vis symmetric key cryptosystems. The situation has slightly improved with Elliptical Curve Cryptography which provides good throughput even on resource constrained platforms. These are however, still not comparable with the throughputs achieved by symmetric key cryptosystems.

The peril that some of the public key cryptosystems have recently been exposed to, is from a new computation paradigm called Quantum Computation [92], initiated by David Deutsch [33]. Hitherto, the classical computers used bits to represent and store data and employed logic gates emulating boolean operations for data processing. In the quantum computation model, the operations would be on quantum bits (qubits) using logic gates based on the principles of quantum mechanics. The first alarm was raised when Peter Shor [113, 114] showed that the integer factorization problem and discrete log problem can be solved in polynomial time complexity employing a quantum variant of the fast Fourier transform. This would have meant the end of several good public key cryptosystems, but for the fact that quantum computer today, remains to be a hypothetical concept with very small prototypes in the horizon. While systems based on integer factorization problem and discrete logarithm problem are susceptible under the quantum computation model, public key systems like NTRU based on the shortest vector problem continue to remain hard in the quantum era. This has resulted in birth of another new field dealing with cryptosystems that would be secure even in the quantum computer based ecosystem called the *Post Quantum Cryptography*. Such systems would be based on cryptographic primitives, such as code-based or lattice-based cryptography, that cannot be broken by quantum computers. On the other hand, another field that relies on quantum physics for design of cryptographic primitives known as *Quantum Cryptography* [22] has evolved. A potential application of quantum cryptography is secret key distribution. Here one may use the Heisenberg's uncertainty principle to thwart the age-old, man in the middle attacks, as an eavesdropper cannot tap into a quantum communication channel even in a promiscuous mode.

1.4 Adversary and the attack models

Generally, there are two modes of attacks, namely *passive* and *active*. A passive attacker merely threatens confidentiality of data by eavesdropping on the communication channel. On the other hand, an active attacker attempts to alter, add or delete the transmissions on the channel, and threatens data integrity and authentication in addition to confidentiality.

While studying the security of cryptosystems, it is assumed that the encryption and decryption algorithms are available to the adversary. This is in consociation with the *Kerckhoff's Principle*, which states that the security of the cipher must vest with the secrecy of the key [61]. This model has been reiterated by Shannon who stated that the adversary 'knows the system'. Based on this underlying assumption, the security of cryptosystems can be assessed in the following models, which also parameterize passive attacks.

1. *Perfect Secrecy.* A cryptosystem is said to be unconditionally or perfectly or information theoretically secure if it cannot be broken even by an adversary with infinite computational resources. This definitely is an ideal security notion for any encryption method.

2. *Computational Security.* A cryptosystem is said to be computationally secure if the best known algorithm for breaking it requires at least N operations, where N is some pre-specified large number.

3. *Provable Security.* A cryptosystem is said to be provably secure if it is as difficult to break as solving some well-known and supposedly difficult problem. Note that a cryptosystem based on a hard problem does not guarantee security. The worst case complexity for solving the problem may be exponential, but the average case complexity or the complexity for some specific instances of the problem may be polynomial.

Apart from the broadly categorized modes mentioned above, there are four basic models [121] of attacks from the point of view of cryptanalysis. In these models, it is by default assumed that the details of the encryption and decryption algorithm are available to the attacker.

1. *Known Ciphertext Attack*: Here the attacker knows the ciphertext of several messages encrypted with the same key and/or several keys and his goal is to recover the plaintext of as many messages as possible, or even better, deduce the key(s).

2. *Known Plaintext Attack*: Here the attacker knows the {plaintext, ciphertext} pairs for several messages, and the goal is to deduce the key to decrypt further messages, or to decrypt a specific ciphertext not belonging to the pairs that (s)he already knows.

3. *Chosen Plaintext Attack*: Here the attacker can choose the plaintext that gets encrypted, thereby potentially garnering more information. This type of situation is possible, for example, when the attacker has access to the encryption device for a limited time.

4. *Chosen Ciphertext Attack*: Here the attacker can choose a series of ciphertexts. It is assumed that a decryption oracle is available and the attacker gets the plaintexts corresponding to these ciphertexts. Based on these information the attacker tries to decrypt ciphertexts not belonging to the pairs that (s)he already knows, or better, deduce the key.

1.5 Overview of stream ciphers

Symmetric key systems are much faster in operation when compared to the public key systems and are therefore a preferred choice for applications requiring high encryption throughputs [72]. Within the private key systems, stream ciphers typically give better throughput and are easier to implement vis-a-vis block ciphers. The past hundred years has seen stream ciphers evolve from the early rudimentary implementations like one-time-pads to the present day sophisticated implementation inside VLSI chips. This metamorphosis has its shares of ups and downs with the Cryptology community questioning the ability of stream ciphers to perform in the evolving information technology landscape.

1.5.1 Vernam cipher to electro-mechanical rotors

In 1917 Gilbert Sandford Vernam conceived of the first stream cipher using additive poly-alphabetic function and successfully obtained a patent two years later. Present day stream ciphers are based on this model and are also known as *Vernam cipher*. In this model if we consider $m_1, m_2, \ldots$ are message bits, $k_1, k_2, \ldots$ are keystream bits and $c_1, c_2, \ldots$ are corresponding ciphertext bits, then for a typical stream cipher, the encryption is performed as $c_r = m_r \oplus k_r$ and the decryption is performed as $m_r = c_r \oplus k_r$. Utilizing unique independent keystream for each message transforms the Vernam cipher into a *one-time pad*.

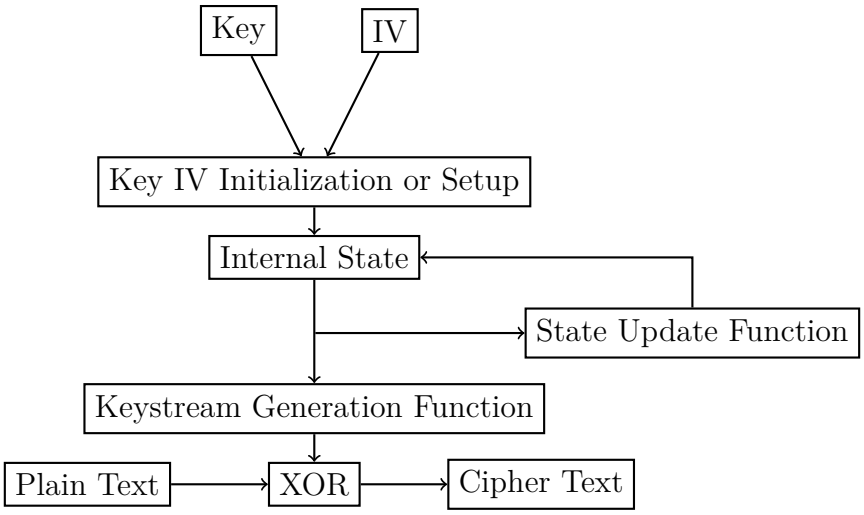

Figure 1.2: Encryption using a Stream Cipher.

Subsequently Vernam and Joseph Mauborgne, an US Army Signals officer, came up with a teleprinter that combined characters of the message with a key on paper tape generated a-priori, to generate the ciphertext. A similar combination function on the ciphertext with same key would yield the message. This product can be considered as the first automated implementation of the one-time pad. Almost three decades later Claude Shannon proved that one-time pads have *perfect secrecy* [111], i.e., the ciphertext leaks absolutely no information about the plaintext.

Ideal one-time pads require an infinite random sequence. Since all finite state machines generate only finite randomness, a True Random Number Generator (TRNG) is required for this purpose. The drawback is that the same random keystream is required at the recipient's end. This problem is compounded by the fact that a keystream can be used only once. In the early days this was achieved using manual means where code books consisting of long random sequences were distributed manually using a courier. The length of the sequence was based on the anticipated traffic size. Such systems being manual were extremely slow and depended on the operators skills.

Attempts to automate the procedure resulted in electro-mechanical devices which dominated the scene for half a century commencing 1920. These machines had a stack of specialized rotors with calibrated positions corresponding to the alphabets which were to be entered through a keyboard. These in turn

were mechanically coupled to other rotors that performed substitution like operations based on the design and the key. These machines served as the workhorse for cryptographic operations for both Allied and Axis forces in the World War II. Some of the popular machines from that era include the Enigma, Lorenz SZ 40/42, Siemens and Halske T52 (Geheimfernschreiber) used by the Germans and Hagelin's family of machines including the C-36, C-52, CD-57 and M-209 used by Americans. Towards the 1970's, microprocessor based computing devices gave sufficient computing power for performing cryptanalysis. The rotor based devices which were constrained by the mechanical movement and thus could not keep pace to incorporate the complexities required to thwart brute-force attacks using microprocessors. The alternative here was to migrate the design onto the microprocessors and thus was born digital cryptography.

1.5.2 Digitization of stream ciphers

In the digital domain implementation, the stream cipher was modeled as a finite state machine having a finite length key (seed) as the input to generate an internal state. The finite state machine derives the next keystream bit (or byte or word) as well as updates the internal state as a function (operation) of the present internal state. The design had provisions of initializing the finite state machine. Due to the deterministic nature of a stream cipher the same key would yield the same keystream which makes a key unsuitable for reuse. To thwart this drawback an *initialization vector* is used along with the key. An initialization vector (IV) is a value, which when changed with each session of the cipher and combined with the secret key, generates a *message (session) key* which is an effective key for the corresponding session of the cipher. Such a mode of usage is called the nonce and with a distinct IV for each message/ session, the output of the stream cipher becomes unique for each message/ session, even with the same key. The keystream bit sequence generated was required to meet the requirements of standard statistical randomness tests [13, 67]. Stream ciphers based on the synchronization mode can be subdivided into one of the following categories.

1. *Synchronous stream cipher.* In *synchronous* stream cipher the keystream is constructed exclusively from the key and initialization vector, indepen-

dent of the plaintext or the ciphertext. There is an inherent requirement for the same state to be maintained between the sender and receiver, or else any offset would result in gibberish data.

2. *Self synchronous stream cipher.* In contrast, *self-synchronous* cipher uses a small number of the previous ciphertext elements to generate the keystream. Thus decryption requires only a finite number of previous ciphertexts. In case of loss of synchronization between sender and receiver, this enables an automatic re-establishment of proper decryption. An example of a modern self synchronizing stream cipher is Moustique [31]. These class of stream ciphers are difficult to design. An alternative and intuitive way of obtaining a self-synchronizing stream cipher is using a block cipher in Cipher Feedback (CFB) mode.

Stream cipher design can be done in several different methods. The final design will be largely influenced on the target implementation platform. Since the implementation could either be directly on the hardware chip or as software running on a processor, there are two broad flavours of stream cipher design paradigms. Although there are diverse and highly creative designs available in literature, one popular example of each design will be described below. A detailed survey on different types of stream ciphers can be found in [101, 103].

Stream ciphers using LFSRs

Linear Feedback Shift Register (LFSR) has been the mainstay for several stream cipher designs. Easy to implement in hardware, an LFSR is essentially a register that contains a series of bits as its internal state. There are tap-points which indicate to specific bits of the register. When the LFSR is clocked the least significant bit (LSB) is delivered as the output and discarded. All other bits shift a position towards the LSB leaving the most significant bit (MSB) vacant. The MSB is replaced with the XORs of values that were contained at the tap-points prior to clocking. LFSRs have been very popular in hardware implementations. Some of the popular stream ciphers based on LFSR are Sober-t-16, Sober-t-32, Snow, A5/1, etc.

The LFSR internal state can be found from l consecutive sequence of bits if the feedback coefficients are known, and from $2l$ sequence of bits otherwise.

The shortest length LFSR called Linear Complexity required to generate the sequence is efficiently calculated using Berlekamp-Massey algorithm, and hence it is essential to mask the linearity of LFSR in stream ciphers. The LFSR based methods of constructing stream ciphers are enumerated as follows. An in-depth description of LFSRs has been done in [44].

1. *Filter Generators.* A filter generator is a combination of different stages of a single LFSR. The function involved is a non linear function and is used to compute the values generated in different stages of a single LFSR. Therefore, the overall complexity depends on the length of the LFSR and algebraic degree of the function used for computation. In order to ensure good statistical properties of the keystream used, it is essential that the filtering function be balanced and the selected LFSR must have a primitive feedback polynomial. Hence, in case of filter generators, while the feedback polynomial, filter function and tapping sequence are known, the unknown factor is the key of the cipher which is the initial state of the LFSR. In this generator a single LFSR is used, having length N (registers $a_1, a_2, a_3, \ldots, a_N$), along with a filter function f of non linear order m, where $m < N$. Given the above inputs, the generated keystream has usually a very high linear complexity.

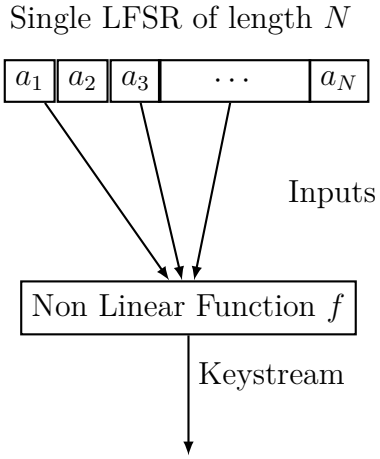

Figure 1.3: LFSR based Filter Generator

2. *Combination Generators.* It is a sequential combination of LFSRs whose output keystream is produced by a non linear boolean function (f) which

is also known as combining function. This function maps the input to generate the output which is binary variable. In order to create a sequence of maximal length, it is necessary that the LFSRs should be carefully chosen. Selecting LFSRs whose length are coprime , LFSRs with feedback polynomials that are primitive with distinct degree are simple methods for obtaining maximal length sequences. The linear complexity of the output sequence is obtained from the linear complexities of the individual output sequences of associated LFSRs and the algebraic normal form (ANF) of the boolean function used. The function used should also have a higher order of correlation immunity.

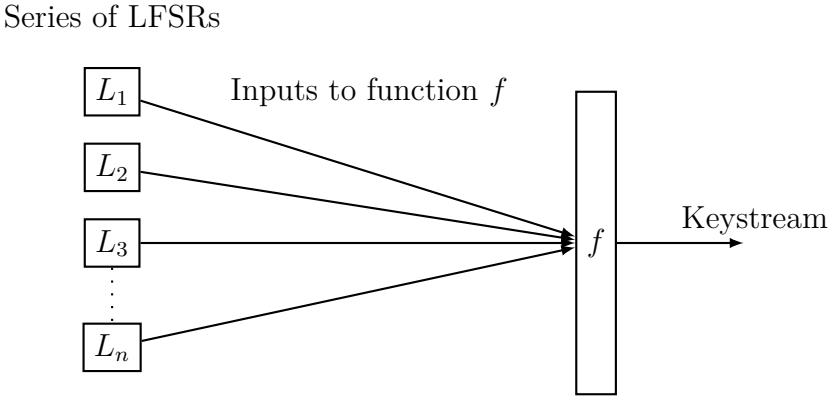

Figure 1.4: LFSR based Combination Generator.

3. *Clock-controlled Generators.* These are based on several registers that together, produce an output sequence. The state of the registers is dependent on some clocking events that could either be an external or an internal one. The register that manages the clocking control is known as control register (CR) and the register which yields keystream in accordance with the output of CR is known as Generator Register (GR). In the non synchronous based Clock controlled generators, the clock of one register is dependent on the output of its previous register.

Figure 1.5: LFSR based Clock-controlled Generators

Stream ciphers in software

Array based stream ciphers have been popular in the software domain. These ciphers have an internal state in the form of a large array of pseudo-random words derived from the key and IV using an initialization process. The output key-stream is a relatively simple function of the state which updates each time keystream is generated. The usage of arrays make these designs extremely efficient and fast in software. The popular stream ciphers that use array are RC4, SEAL, HC-128, HC-256, Py, ISAAC etc. The HC-128 and HC-256 stream ciphers would be described in detail in Chapter 2.

1.5.3 State-of-the-art in stream ciphers

By the year 2000 stream ciphers had faced over three decades of cryptanalysis, and a number of weaknesses were exposed in well established stream ciphers. These weaknesses ranged from loopholes in primitives (cryptanalysis) to flaw in the implementation. Notable amongst these were the flaw in GSM mobile encryption algorithm (A5/1 and A5/2), Bluetooth algorithm E0 and the key and IV usage in Wireless LAN (WEP) implementation of RC4. A European research initiative called New European Schemes for Signatures, Integrity and Encryption (NESSIE) was carried out to identify secure cryptographic primitives from 2000 to 2003. Unfortunately, none of the six stream ciphers submissions to the NESSIE project could make it to the final portfolio. This prompted Adi Shamir to give a talk titled "Stream Ciphers: Dead or Alive?" [110] in Asiacrypt 2004.

Due to the pessimistic state of stream ciphers, a 4-year research initiative was launched by the European Network of Excellence for Cryptology (ECRYPT) within a project called *eStream*. The goal was to study the state-of-the-art in stream cipher design, contrary to other competitions that aim to evolve a standard. Of course the collateral benefit would be the "new stream ciphers that might become suitable for widespread adoption". The eStream competition was carried out in three phases and the designers were given the flexibility to tweak the ciphers so that a good design does not get discarded on account of an avoidable flaw. The submissions to eStream were solicited under either or both of two profiles:

- Profile 1: Stream ciphers for software applications with high throughput requirements (faster than AES-CTR mode), having key size of 128 bit and IV of 64 to 128 bit.
- Profile 2: Stream ciphers for hardware applications with restricted resources such as limited storage, gate count, or power consumption (more compact than AES), having key size of 80 bit and IV of 64 to 128 bit.

There were 34 submissions from across 20 countries. Of these 9 submissions fell into exclusively software category, 12 fell into hardware category, and 13 qualified for both. LFSR appeared to be a popular primitive with 10 ciphers using LFSR, 6 NFSR , 3 T-functions and 2 SPN. In phase one, 22 out of the 34 ciphers were broken. The competition had two more phases and in 2008 eight stream ciphers were selected for the final portfolio of which one was subsequently broken. The present portfolio consists of HC-128, Rabbit, Salsa20/12, SOSEMANUK belonging to Profile 1, and Grain v1, MICKEY v2, Trivium belonging to Profile 2. A comprehensive survey on all of these ciphers is available in [102].

Another recent initiative is the design of a third set of algorithms for 3GPP using ZUC algorithm designed by the Data Assurance and Communication Security Research Center of the Chinese Academy of Sciences (DACAS). Consequent to successful evaluation by the algorithm standardization group ETSI SAGE, and by other closed teams of eminent experts, the algorithms are open for public evaluation. One may visit the ZUC forum for more data [1].

1.6 Stream cipher cryptanalysis

Common techniques using which stream ciphers are typically attacked are described in this section. They range from the traditional attack that aims to exploit mathematical weaknesses in the design structure to side channel cryptanalysis that basically targets the implementation. However in real life cryptanalysis often it is the combination of multiple techniques that need to be used in a complimentary manner.

1.6.1 Theoretical attack models

Exhaustive key search

An exhaustive key search attack aims at trying all possible key combinations in a trial and error mode. Thus an n bit key would require a search across all the possible 2^n key combinations. This undoubtedly is the most basic attack against any symmetric key cryptosystem and is also known as *brute force attack*. For such an attack to succeed there is a requirement of high computation power and often special purpose machines are used for this purpose. This attack can be expedited by using techniques that speed-up the brute force search.

Time Memory Trade-Off (TMTO)

TMTO was originally designed as 'chosen plaintext attack' targeting the DES block cipher. Hellman [47] showed for the first time that by precomputing some values and storing them in memory one can perform a faster exhaustive search. This work was refined further by arranging the precomputing in a format that enabled efficient retrieval. The precomputed values are stored and used in a manner akin to a dictionary attack. The trade-off here is between the exhaustive key search that needs nil memory but time complexity of the order of key-space and the dictionary attack that needs negligible time but memory complexity of the order of key space. The TMTO technique was found suitable in other scenarios like 'known plaintext attack'. Later, TMTO attacks were shown to be useful for stream ciphers as well in [8, 18, 19, 52, 104].

Weak keys

The key and IV form the seed that goes on to generate the pseudo random output sequence of a stream cipher also known as the keystream. The notion of 'weak keys' implies that the cipher would behave in undesirable ways wherein inferences pertaining to secret key information can be deduced from the keystream [66]. A cipher devoid of weak keys is said to have a linear or flat key space. Historically weak keys were present in almost every rotor based cipher machines since 1925. There have been some legendary cryptanalytic

exploits of the same. In the present, such a vulnerability may manifest as related key attacks, key-output correlation attacks, related IV attacks, etc.

Distinguishers

A distinguisher is an algorithm that is able to infer whether the random data bits (either from the keystream of stream cipher, or from a true random number generator) given to it, were generated from the stream cipher or from a random source with a greater probability than the random guess. This is considered relevant because, the output of stream cipher, is mandated to be indistinguishable from a truly random bit-stream. Thus distinguishing attacks, is the study of keystream for statistical weaknesses and patterns, that could attribute the keystream to a specific cipher [83]. In simple words it is discovering the fingerprints of a stream cipher in its keystream. Thus a distinguisher aims to expose within any generic stream cipher design, any structure dependent or initial state dependent patterns. Distinguishers do not generally result in Key recovery but could potentially result in correlation attacks. The strength of the distinguisher is quantified in terms of the keystream sample size required for analysis.

Algebraic attacks

Algebraic attacks are a type of plaintext attack that were initially proposed for public key systems [63] but were found to be equally potent in analyzing practical stream ciphers as well [7, 11, 12, 29, 30, 46]. Algebraic attack being a known plaintext attack, implies that the adversary has some a-priori knowledge of some plaintext bits and the associated ciphertext bits. The modus operandi is extracting the key of the symmetric ciphers by solving a set of polynomial equations. Thus algebraic attack consists of basically two steps – finding a set of equations and then solving them to recover the key. It is generalized that all LFSR related stream ciphers are vulnerable to algebraic attacks provided that the condition or method sustains. This implies that if it is possible to deduce an output bit or bits, low degree multivariate equation, then it can also be used for other states.

Correlation attacks

Correlation attacks are plaintext attacks on stream ciphers where the keystream is produced by the combination of several LFSRs outputs with the help of a Boolean function (non-linear combiner) [25,43,53,54,77,85,86,95]. Correlation attack targets ciphers with poor boolean functions. Thus right and intelligent choice of function used to generate the final keystream is required to prevent such an attack. Such attacks are prevalent where there is significant correlation among the outputs of the associated LFSRs and the final keystream. Correlation attacks can be said to be a type of divide and conquer algorithms.

1.6.2 Side-channel attacks

Side-channel cryptanalysis is an alternative paradigm wherein a crypotographic implementation is attacked on the basis of operating parameters such as power consumption, electromagnetic radiation, memory read patterns, timing for operations, thermal signatures etc. Although side-channels have recently risen to prominence in the cryptographic field, its principles exist from time immemorial and have been exploited for centuries. During the 19th Century, there were issues like cross talk in telephone lines. These issues instead of being sorted out were rather exploited in First World War for espionage. In 1918, H. Yardley et al. discovered that electrical devices could cause leakage of crucial information. Similarly during the Second World War, incidences of tapping information were prevalent again and for the first time devices began to be constructed to prevent leakages. Post war, the saga continued and the theatre of action were signal emanating in and diplomatic embassies. For instance, Chinese are believed to have relied on acoustic waves to read their enemy's intent. Similarly, the Russians are believed to have used microwave signals in metal bars obscured within statues for signal gathering. Electromagnetic radations dominated the scene for several decades and a separate discipline called SIGINT (acronym for SIGnal INTelligence) evolved.

Side-channels today are no longer restricted to electromagnetic radiations and have extended onto physical features of the micro-electronic devices. In order to perform a task, the microprocessors consume time and power and in return radiate electromagnetic radiations, heat and noise. In such cases,

information that appears to be by-products of a computational process, in reality provide sufficient information for the attackers to detect the type of operation and exact task being performed on the device. Side-channel attacks can be classified as invasive or non-invasive based on the method of accessing the device. In an invasive attack there is a direct access to the device and its components. Whereas, a non-invasive attack does not include any dismantling of the device for observations. Side-channel attacks can alternatively be classified as an active or passive attack. An active attack includes direct induction of some errors in the computation to observe the functioning of the device, while a passive attack does not involve any external methods of observing the functionalities, it silently observes the operations carried out by the device.

Owing to the emerging threats of the side-channel attacks, there have been focus on design of side-channel leakage tolerant architectures [125]. Similarly projects like Side Channel Analysis Resistant Design Flow (SCARD) have been undertaken that study on the improvement of the design flow of the micro-chips across all levels of implementation [4]. Side-channel attacks that have relevance to modern ciphers and some of the popular attacks are described below.

Timing attack

In this attack timing variations arising due to various operations being performed by the cipher are observed and exploited. The run time of an operation such as RSA exponentiation etc, depends on the key used to a large extent. This can be exploited in timing attacks [70]. There are several other instances like timing attack on SSH protocol where various keystrokes timing could be inferred using traffic analysis techniques and in return could locate the passwords typed. Typical measures to combat issues of timing attack are Noise Injection and Branch Equalisation. In noise injection method, noise is introduced before performing any cryptographic operation and then removed later. In branch equalisation method, the processing is done in such a way that there exists timing equivalence among the cryptographic operations. These attacks are relevant to stream ciphers also [74, 130, 131].

Power analysis

This attack relies on the power consumption of the devices during conduct of any cryptographic operations and can be classified as either Simple Power Analysis (SPA) or Differential Power Ananlysis (DPA). In SPA the adversary has precise information of the performance details and then employs statistical methods. In DPA the adversary exploits the features with no prior knowledge of the performance details. It just requires correlation of the power consumption along with the secret key to guess a few bits [71]. These attacks have been mounted on stream ciphers [38, 73].

Fault attack

In a fault attack the adversary actively tampers the target device by injecting spurious data during the operation cycle of a cipher thereby disrupting the execution. The effects of the injected fault may be monitored and the resulting variations used to obtain information about the cipher internals [17,21]. Fault attacks are relevant in case of stream ciphers also [9, 10, 16, 48, 51, 64]. Fault attacks will be further clarified in Chapter 5 .

Electromagnetic attacks

This attack is based on the analysis of electromagnetic radiations emanating from the device during certain computational process [3, 99]. Such attacks are preferred in cases where power analysis attack is not feasible. They are an effective means to bypass counter-measures built for power analysis attack. Counter measures against this type of attacks are shielding of the device to avoid leaking radiations.

1.7 Motivation for the thesis

The core motivation for this thesis was to study the diverse aspects of contemporary cryptology. The choice of focusing on HC-128 stream cipher was inspired by the fact that there were no academic results published for this cipher till 2008, despite the cipher being put up to the community throughout

the eStream competition. This was quite paradoxical considering the fact that the cipher bears a structural resemblance with the RC4 stream cipher for which there exists an abundance of academic literature covering almost all aspects of cryptological significance. Another reason that motivated the study of HC-128 stream cipher was the fact that its designer Hongjun Wu had published a number of papers in the cryptanalysis of various stream ciphers [126, 129], prior making this design. The insight gained by the designer would no doubt be reflected in his designs making the task of analysis even more challenging.

1.8 Contribution and thesis plan

A comprehensive study into the myriad aspects of contemporary cryptology should cover the entire spectrum of cryptographic designs, theoretical as well as experimental cryptanalysis, study of emerging side channel attacks as well as implementation on state of art hardware. Accordingly the HC-128 stream cipher has been studied in a disciplined manner. This focus of the thesis is on the papers [127, 128] and presents a rigorous study of the stream cipher covering the broad aspects of modern cryptology, namely, the theoretical study of cryptographic primitives, design modification proposal, experimental and side channel analysis of this cipher and finally implementation strategies.

In Chapter 2, we give the description of the HC-128 and HC-256 stream cipher alongwith an overview of all existing works in the analysis and implementation of this cipher.

In Chapter 3, we first show that the knowledge of any one of the two internal state arrays of HC-128 along with the knowledge of 2048 keystream words is sufficient to construct the other state array completely in 2^{42} time complexity. This analysis reveals a structural insight into the cipher along with theoretically establishing some nice combinatorial properties of HC-128 keystream generation algorithm. This chapter is based on the conference paper [96].

Next in Chapter 4, using linear approximation of the addition modulo 2^n of three n-bit integers we identify linear approximations of g_1, g_2, the feedback functions of HC-128. Here we show that the process of keystream output generation of HC-128 can be well approximated by linear functions wherein

the 'least significant bit' based distinguisher (presented by the designer of the cipher) of HC-128 [128, Section 4] works can be extended for the complete 32-bit word. Further using the above linear approximations of g_1, g_2, we present the first new distinguisher for HC-128 which is slightly weaker than Wu's distinguisher. We also study how HC-128 keystream words leak secret state information of the cipher due to the properties of the functions h_1, h_2 and present improved results. This chapter is based on the journal paper [81] and conference paper [80].

In Chapter 5 we study the Side channel attacks HC series of stream cipher. We extend the established HC-128 fault attack and the HC-256 cache analysis onto the HC-256 and HC-128 ciphers respectively under similar models. The techniques applied on one variant is not trivially translatable to the other and the issue was left open until the current work. Here we propose a technique to recover half the state of HC-128 using cache analysis, which can be cascaded with the differential attack towards a full state recovery and hence key recovery. Similarly, we analyze the state leakage of HC-256 under differential fault attack model to achieve partial state recovery. This chapter is based on the conference paper [97].

In Chapter 6, we study the implementation issues for HC-128 in a disciplined manner. HC-128 is primarily designed as a software stream cipher aiming for sequential execution on general purpose processors and so we first carry out implementations on embedded and customizable processors. Next we consider the ASIC implementation for co-processor design. Further we explore several parallelization strategies for faster execution of the cipher. This is the first detailed implementation exercise as implementation of HC-128 on hardware was never attempted on the eSCARGOT chip. This chapter is based on the conference paper [26].

1.9 Prerequisites for the reader

The thesis is an effort to convey the features of HC-128 stream cipher in a much elaborate way compared to the existing literature. Apart from the summary of the thesis chapters in the aforesaid section, there are certain prerequisites expected on the part of the reader for easier and clearer understanding. It is

required on the reader's part to have a sound knowledge of the mathematical models and field of probability, permutation and combination. Apart from that, it is imperative that a reader has some basic knowledge of classic and modern cryptographic structures and terminologies so as to relate them easily while comprehending the structure of the stream ciphers. Though we have made every effort to provide the minutest details of certain cryptographic terms and also the hardware implementation based terms, yet it would be much easier, quicker and interesting for the reader to understand if (s)he has some elementary knowledge of embedded systems and VLSI.

Chapter 2

Background

Babbage's Rule– "No man's cipher is worth looking at unless the inventor has himself solved a very difficult cipher."

– The Codebreakers by Kahn, 2nd ed, pg 765

2.1 The HC series of Stream Ciphers

The HC series consists of two synchronous stream ciphers – HC-256 and HC-128 – both designed by Hongjun Wu. HC-256 was first presented in FSE 2004 [127] and was subsequently a submission in the eStream initiative to identify new stream ciphers. HC-256 was successful in the Phase I and entered Phase II. HC-128 [128], a lighter version of HC-256, was also a submission to eStream competition, and it made it to the final software portfolio.

In this chapter we first present the description of HC-128 and HC-256 stream ciphers. The eSTREAM [37] Portfolio (revision 1 in September 2008) contains the stream cipher HC-128 [128] in Profile 1 (SW). Initial analysis of the cipher was given by the designer with a view to conjecture the security of this cipher. Subsequent works on the two ciphers have been brought out in the next section. The chapter concludes with a list of papers published as part of this thesis.

2.2 Description of HC-128

2.2.1 Operators and structures

We summarize the key points of the structure and the keystream generation of the cipher below. The following operators are used in HC-128.

$+$: addition modulo 2^{32}.

$\boxminus$: subtraction modulo 512.

$\oplus$: bit-wise exclusive OR.

$\|$: bit-string concatenation.

$\gg$: right shift operator (defined on 32-bit numbers).

$\ll$: left shift operator (defined on 32-bit numbers).

$\ggg$: right rotation operator (defined on 32-bit numbers).

$\lll$: left rotation operator (defined on 32-bit numbers).

HC-128 is a word-oriented stream cipher, with each word of size 32-bits. Two internal state arrays P and Q are used in HC-128, each with 512 many 32-bit words. A 128-bit key array $K[0,\ldots,3]$ and a 128-bit initialization vector $IV[0,\ldots,3]$ are used, each entry being a 32-bit word. The following six functions are used in HC-128.

$$
\begin{aligned}
f_1(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) \\
f_2(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) \\
g_1(x,y,z) &= ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8) \\
g_2(x,y,z) &= ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8) \\
h_1(x) &= Q[x^{(0)}] + Q[256 + x^{(2)}] \\
h_2(x) &= P[x^{(0)}] + P[256 + x^{(2)}]
\end{aligned}
$$

where $x = x^{(3)}\|x^{(2)}\|x^{(1)}\|x^{(0)}$ is a 32-bit word, with $x^{(0)}, x^{(1)}, x^{(2)}$ and $x^{(3)}$ being the four bytes from right to left.

2.2.2 Initialization process

The key and IV setup of HC-128 recursively loads the P and Q array from expanded key and IV, and runs the cipher for 1024 steps with the outputs used to replace the table elements. The initialization is as in Algorithm 1.

Data: Key (K) and IV, each of length 4 words (128 bits)
Result: State arrays P and Q, each consisting of 512 words

`// Expand the key and IV to 8-words each`
for $0 \leq i \leq 3$ **do**
$K[i+4] = K[i]$ and $IV[i+4] = IV[i]$;
end

`// Expand key and IV to 1280-word array W`

$$W[i] = \begin{cases} K[i], & 0 \leq i \leq 7 \\ IV[i-8], & 8 \leq i \leq 15 \\ f_2(W[i-2]) + W[i-7] + \\ \qquad f_1(W[i-15]) + W[i-16] + i, & 16 \leq i \leq 1279 \end{cases}$$

`// Update tables P and Q using the W array`
for $0 \leq i \leq 511$ **do**
$P[i] = W[i+256]$ and $Q[i] = W[i+768]$;
end

`// Update the tables for 1024 rounds, without output`
for $0 \leq i \leq 511$ **do**
$P[i] = (P[i] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])) \oplus h_1(P[i \boxminus 12])$;
end
for $0 \leq i \leq 511$ **do**
$Q[i] = (Q[i] + g_2(Q[i \boxminus 3], Q[i \boxminus 10], Q[i \boxminus 511])) \oplus h_2(Q[i \boxminus 12])$;
end

Algorithm 1: Initialization process of HC-128 stream cipher.

In [76], the ability to recover the key given the internal state post initialization was reported using this algorithm in reverse. Accordingly, a modification was proposed to interleave the KGS. In our analysis, we found that even the modified version was vulnerable in returning the key when the algorithm was used in reverse order.

2.2.3 Keystream generation

The input to the keystream generation algorithm are the state arrays P and Q, each consisting of 512 words. Let s_i denote the keystream word (32 bits) generated at the i-th step. The keystream of HC-128 is generated using Algorithm 2.

Data: State arrays P and Q, each consisting of 512 words
Result: Keystream s_i for $i = 0, 1, 2, \ldots$

$i = 0$;

`// Repeat until enough keystream words are generated`
while *keystream required* **do**
    $j = i \bmod 512$;
    if $(i \bmod 1024) < 512$ **then**
        $P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$;
        $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
    end
    else
        $Q[j] = Q[j] + g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$;
        $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
    end
    $i = i + 1$;
end

Algorithm 2: Keystream generation algorithm of HC-128.

2.3 Description of HC-256

The operators used in HC-256 are similar to HC-128 as described in Section 2.2 (with the exception of ($\boxminus$) which is subtraction modulo 1024). The difference lies in the number of bits used and the processing speed. In Pentium M processor, the speed of HC 256 is 4.4 cycles/byte while that of HC-128 is 3.05 cycles/byte.

Two tables P and Q, each with 1024 many 32-bit elements are used as internal states of HC-256. A 256 bit key array $K[0, \ldots, 7]$ and a 256-bit initialization vector $IV[0, \ldots, 7]$ are used, where each entry of the arrays is a

32-bit element. The following six functions are used in HC-256.

$$
\begin{aligned}
f_1(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) \\
f_2(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) \\
g_1(x,y) &= ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024] \\
g_2(x,y) &= ((x \ggg 10) \oplus (y \ggg 23)) + P[(x \oplus y) \bmod 1024] \\
h_1(x) &= Q[x^{(0)}] + Q[256 + x^{(1)}] + Q[512 + x^{(2)}] + Q[768 + x^{(3)}] \\
h_2(x) &= P[x^{(0)}] + P[256 + x^{(1)}] + P[512 + x^{(2)}] + P[768 + x^{(3)}]
\end{aligned}
$$

where $x = x^{(3)} \| x^{(2)} \| x^{(1)} \| x^{(0)}$ is a 32-bit word, with $x^{(0)}, x^{(1)}, x^{(2)}$ and $x^{(3)}$ being the four bytes from right to left.

The key and IV setup of HC-256 proceeds as in Algorithm 3.

Data: Key (K) and IV, each of length 8 words (256 bits)
Result: State arrays P and Q, each consisting of 1024 words

`// Expand key and IV to 2560-word array W`

$$
W[i] = \begin{cases}
K[i], & 0 \le i \le 7 \\
IV[i-8], & 8 \le i \le 15 \\
f_2(W[i-2]) + W[i-7] + & \\
\qquad f_1(W[i-15]) + W[i-16] + i, & 16 \le i \le 2559
\end{cases}
$$

`// Update tables P and Q using the W array`
for $0 \le i \le 1023$ **do**
    $P[i] = W[i + 512]$ and $Q[i] = W[i + 1536]$;
end
Run keystream generation (Algorithm 4) 4096 steps without output.

Algorithm 3: Initialization process of HC-256 stream cipher.

The input to the keystream generation algorithm are the state arrays P and Q, each consisting of 1024 words. Let s_i denote the keystream word (32 bits) generated at the i-th step. The keystream generation of HC-256 is as shown in Algorithm 4.

The HC-128 and HC-256 stream ciphers are extremely fast on keystream generation. On the flip side, HC-128 and HC-256 stream ciphers have large

Data: State arrays P and Q, each consisting of 1024 words
Result: Keystream s_i for $i = 0, 1, 2, \ldots$

$i = 0$;

`// Repeat until enough keystream words are generated`
while *keystream required* **do**
    $j = i \bmod 1024$;
    if $(i \bmod 2048) < 1024$ **then**
        $P[j] = P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$;
        $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
    end
    else
        $Q[j] = Q[j] + Q[j \boxminus 10] + g_2(Q[j \boxminus 3], Q[j \boxminus 1023])$;
        $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
    end
    $i = i + 1$;
end

Algorithm 4: Keystream generation algorithm of HC-256.

arrays and the time for Key and IV setup is large (around 27,300 and 74,000 clock cycles respectively). This makes these ciphers suitable for point to point link-level encryption that does not require frequent reinitialisation.

2.4 Chronology of recent works

At the time of commencement of work on this thesis in year 2008, there were no research papers on HC-128 or HC-256, despite the fact that the two ciphers had been subject to years of study, as part of the eStream project. The details of research papers that emerged during the course of our research are given in the subsequent paragraphs.

2.4.1 Theoretical Cryptanalysis

The details of research papers pertaining to theoretical cryptanalysis of HC-128 and HC-256 stream ciphers are as follows.

Dunkelman [36]. The first recorded detail on HC-128 stream cipher was an observation made in eStream forum by Dunkelman [36], who showed that

the keystream words of HC-128 leak information regarding the internal state. He observed that $Prob(s_j \oplus s_{j+1} = P[j] \oplus P[j+1]) \approx 2^{-16}$, where s_j, s_{j+1} are two consecutive keystream output words.

Maitra et al. [80]. This is our work and is the first work on the study of the keystream of HC-128, published in WCC 09. This used linear approximation of the addition modulo 2^n of three n-bit integers to identify linear approximations of g_1, g_2, the feedback functions of HC-128. This, in turn, shows that the process of keystream output generation of HC-128 can be well approximated by linear functions. In this direction, the 'least significant bit' based distinguisher (presented by the designer along with the specifications of the cipher) of HC-128 was shown to work for the complete 32-bit word. Further, in the line of Dunkelman's observation above, the leakage pattern of HC-128 secret state information due to keystream words due to the properties of the functions h_1, h_2 was studied and improved results were presented.

Sekar and Preneel [106]. This was the first work on the theoretical cryptanalysis of HC-256 stream cipher. Here a class of distinguishers on HC-256, each of which requires testing the validity of about $2^{276.8}$ linear equations involving binary keystream variables, was presented. These attacks improve the data complexity of the hitherto best-known distinguisher (presented by the designer along with the specifications of the cipher) by a factor of about 12. This improvement is not directly applicable to HC-128 where the elements of the state array are all rotated prior to use in state update function.

Liu and Qin [76]. This work showed that the Key and IV of HC-128 and HC-256 can be recovered from the internal state of the cipher post completion of the Key and IV initialization. Additionally two modifications have been to make the initialization non invertible. The first method was to to interleave the updation of P and Q array terms in the final stages of the KGS. The other method employed generating additional values of W array. In our analysis, we found that the first modification was unsuccessful as it was trivial to obtain the key running the algorithm in reverse order.

Paul et al. [96]. This is our work in IWSEC 2011, which presented a half state exposure of the HC-128 stream cipher wherein knowledge of any one of the two internal state arrays of HC-128 along with the knowledge of 2048 keystream words is sufficient to construct the other state array completely in 2^{42} time complexity. This analysis revealed a structural insight into the cipher and theoretically established certain combinatorial properties of HC-128 keystream generation algorithm. Additionally modifications to HC-128 were proposed, that would take care of the aforesaid issue with little reduction in speed.

Maitra et al. [81]. This is our paper published in Journal of Design, Codes and Cryptography and is an extended version of [80] and gave a new albeit weaker distinguisher for HC-128.

Stankovski et al. [120]. This paper published in Journal of Design, Codes and Cryptography brought out that the keystream requirement of distinguisher published in [128] was underestimated by a factor of 2^8. It accordingly has a keystream requirement of $2^{160.471}$ number of 32-bit keystream blocks. There are two new types of distinguishers presented of which one is based on counting the number of zeros in created blocks of bits. This gave a biased distribution that needs $2^{143.537}$ such constructed block samples, corresponding to $2^{152.537}$ number of 32-bit keystream blocks.

Stankovski et al. [119]. In this paper rotations and XOR operation used for mixing operation was studied to obtain theoretical results on related probability distributions. A distinguisher was presented for a variant of HC-128 where modular addition is replaced by XOR operation. This variant had a keystream complexity of $2^{90.9}$.

2.4.2 Side Channel Cryptanalysis

The details of research papers pertaining to Side Channel cryptanalysis of HC-128 and HC-256 stream ciphers are as follows:-

Zenner [130]. In SAC-2008, Erik Zenner [130] presented a cache-timing analysis attack on HC-256 stream cipher. This attack is applicable for implementations on devices that use cache memory. Here, the measurements

of cache access timings leaks certain bits. The number of bits leaked is inversely proportional to the size of cache page. Based on certain bits the remainder were computed using a guess and verify technique. The attack was able to establish the internal state with 6148 precise cache timing measurements with the knowledge of 2^{16} plaintext bits and also lead to key recovery.

Kircanski and Youssef [64]. In Africacrypt 2010, Aleksandar Kircanski and Amr M. Youssef presented a differential fault attack on HC-128 requiring 7968 faults that recovers the complete internal state of HC-128 by solving a set of 32 systems of linear equations over Z_2 in 1024 variables. The fault model employed here is one in which the attacker will be able to fault a random word of the inner state without any control of its exact location or its new faulted value. The attack is based on the fact that some of the inner state words in HC-128 are utilized several times without being updated.

Paul and Raizada [97]. This is our work presented in SPACE 2012 that explored the effect of a side channel technique on a variant of the cipher algorithm implemented in a similar model. The motivation for such an investigation is to study the feasibility of using a cipher variant as a candidate for replacement to a cipher subjected to a successful side channels attack. The study was centered around the HC series of stream ciphers, viz., HC-128 and HC-256. The HC-128 fault attack and the HC-256 cache analysis was extended onto the HC-256 and HC-128 ciphers respectively under similar models. The techniques applied on one variant is not trivially translatable to the other. A technique was proposed to recover half the state of HC-128 using cache analysis, which can be cascaded with the differential attack towards a full state recovery and hence key recovery. Similarly, the state leakage of HC-256 was studied under differential fault attack model to achieve partial state recovery.

2.4.3 Cipher Implementation and Usage

The details of research papers pertaining to implementation and usage of HC-128 stream ciphers are as follows.

Meiser et al. [87]. In this paper in SIES 2008 implementation results for performance of stream ciphers Dragon, HC-128, LEX, Salsa20, Salsa20/12, and SOSEMANUK on small embedded 8-bit resource constrained microcontrollers was presented.

Kausar and Naureen [60]. The paper deals with the applicability of stream ciphers in the context of Wireless Sensor Network (WSN) devices that are resource scarce by benchmarking WSN-specific NesC based implementations of two stream ciphers namely HC-128 and Rabbit in terms of efficient energy and memory consumption. The two stream ciphers were able to cope to the WSN-specific requirements and perform efficiently under these requirements. This paper concludes that HC-128 software stream cipher is in the area of sensor networks perform as efficiently as Rabbit.

Jolfaei et al. [55]. In this paper published in the International Journal of Electronic Security and Digital Forensics, the performance of HC-128 and HC-256 for image encryption was studied. The two ciphers were found to be efficient, feasible and trustworthy to be adopted for image encryption.

Chattopadhyay et al. [26]. This was our work presented in ISCAS 2012, which studied several implementation issues for HC-128. The cipher was implemented on embedded and customizable processors, and also modeled as a dedicated hardware accelerator. Several parallelization strategies for improving throughput were presented. This novel implementation strategies marked the fastest HC-128 execution reported till date.

Khalid et al. [62]. This paper in the ICISC 2012 is the first reported effort of mapping HC-Series of stream ciphers on GPUs. It presents present various optimization strategies for HC-128 and HC-256 speedup for CUDA device architecture. The peak performance achieved with a single data-stream for HC-128 and HC-256 is 0.95 Gbps and 0.41 Gbps respectively. Similarly for multiple parallel data-stream the implementation has clocked approximately 31 Gbps for HC-128 and 14 Gbps for HC-256 (with 32768 parallel data-streams).

2.5 Publications included in this Thesis

This thesis is built upon one journal paper [81] and four conference papers [26, 80, 96, 97]. The journal paper [81] is a consolidation and considerable extension of the conference paper [80]. A detailed list of publications is given below.

- Subhamoy Maitra, Goutam Paul, Shashwat Raizada, Subhabrata Sen, and Rudradev Sengupta. Some Observations on HC-128. Des. Codes Cryptography, 59(1-3):231–245, 2011 [81].
- Subhamoy Maitra, Goutam Paul, and Shashwat Raizada. Some Observations on HC-128. In International Workshop on Coding and Cryptology, WCC 2009, Ullensvang, Norway, PreProceedings, pages 527–539, 2009 [80].
- Goutam Paul, Subhamoy Maitra, and Shashwat Raizada. A Theoretical Analysis of the Structure of HC-128. In Tetsu Iwata and Masakatsu Nishigaki, editors, IWSEC, volume 7038 of Lecture Notes in Computer Science, pages 161–177. Springer, 2011 [96].
- Anupam Chattopadhyay, Ayesha Khalid, Subhamoy Maitra, and Shashwat Raizada. Designing High-throughput Hardware Accelerator for Stream cipher HC-128. In ISCAS, pages 1448–1451. IEEE, 2012 [26].
- Goutam Paul and Shashwat Raizada. Impact of Extending side channel attack on Cipher Variants: A Case Study with the HC Series of Stream ciphers. In Andrey Bogdanov and Somitra Kumar Sanadhya, editors, SPACE, volume 7644 of Lecture Notes in Computer Science, pages 32–44. Springer, 2012 [97].

THIS PAGE INTENTIONALLY LEFT BLANK

Part I

Analysis of HC-128 Stream Cipher

Chapter 3

On the internal structure of HC-128

In this chapter we study certain issues related to the internal structure of HC-128. We show that the knowledge of any one of the two internal state arrays of HC-128 along with the knowledge of 2048 keystream words is sufficient to construct the other state array completely in 2^{42} time complexity. Though this analysis does not lead to any attack on HC-128, it reveals a structural insight into the cipher. In the process, we theoretically establish some discernible combinatorial properties of HC-128 keystream generation algorithm. We also suggest a modification to HC-128 that prevents the above analysis with minimal reduction in speed.

As described earlier, there are two internal state arrays of HC-128, P and Q, each containing 512 words (each word is of 32 bits). The keystream is generated using two distinct procedures that are alternated in blocks of 512 words. Within a block, one of these arrays gets updated and the keystream word is produced by XOR-ing the updated entry with the sum of two words from the other array. The role of the two arrays is reversed after every block of 512 keystream words generation. In this chapter, we show that the knowledge of one internal state array of HC-128 reveals the other. To be specific, in Section 3.1, we show that if one knows one of P and Q completely, then one can reconstruct the complete other array efficiently. This *Half-State Exposure Analysis* can serve as a general model of analysis when the state of a stream cipher consists of two similar structures of the same size.

Without loss of generality, we consider four consecutive blocks B_1, B_2, B_3

and B_4 of keystream generation such that Q is updated with the completion of blocks B_1 and B_3 and P is updated with the completion of blocks B_2 and B_4. We assume that, the keystream words corresponding to all of these four blocks are known. Here, the symbols P and Q will denote the arrays after the completion of block B_1 and before the start of block B_2. After the completion of block B_2, Q remains unchanged and P is updated to, say, P_N. After the completion of block B_3, Q would again be updated to, say, Q_N.

Block B_1:	**Block B_2:**	**Block B_3:**
P unchanged,	P updated to P_N,	P_N unchanged,
Q updated.	Q unchanged.	Q updated to Q_N.
(Q denotes the updated array)		

Table 3.1: The evolution of P, Q arrays.

Block B_4, is not shown in Table 3.1, as it will only be used for verifying the correctness of our reconstruction.

In Section 3.1.2, we present Algorithm 9 (called *ReconstructState*), that takes the 512 words of the array P as inputs and (assuming that the 2048 keystream words corresponding to the four blocks B_1, B_2, B_3 and B_4 are known) produces 512 words of the array Q_N as output. Since the update of an array is self dependent, it is inferred that that the complete state gets exposed from block B_3 onwards. The proof of correctness of the algorithm is established through Lemma 3.3 and Theorems 3.6, 3.7 and 3.8 and the data and time complexity requirements are analyzed in Theorem 3.9. Further, in Section 3.2, we propose a modification to the existing HC-128 that escapes the above analysis. We also argue the motivation of such design and present performance comparisons with the existing designs of HC-128 and HC-256.

3.1 Reconstruction of one array from another

We introduce a few notations for the ease of analysis. We describe them and then move on to the actual strategy.

3.1.1 Notations and problem formulation

As discussed earlier, we consider four consecutive blocks B_1, B_2, B_3 and B_4. In B_1 and B_3, Q is updated. Let Q denote the updated array after the completion of block B_1 and let Q_N be the new array after Q is updated in block B_3. In B_1, P remains unchanged and in B_2, it is updated to P_N. Let $s_{b,i}$ denote the i-th keystream word produced in block B_b, $1 \leq b \leq 4$, $0 \leq i \leq 511$. The update of P (or Q) depends only on itself, i.e.,

$$P_N[i] = \begin{cases} P[i] + g_1(P[509+i], P[502+i], P[i+1]), & \text{for } 0 \leq i \leq 2; \\ P[i] + g_1(P_N[i-3], P[502+i], P[i+1]), & \text{for } 3 \leq i \leq 9; \\ P[i] + g_1(P_N[i-3], P_N[i-10], P[i+1]), & \text{for } 10 \leq i \leq 510; \\ P[i] + g_1(P_N[i-3], P_N[i-10], P_N[i-511]), & \text{for } i = 511. \end{cases} \tag{3.1}$$

Thus, if one knows the 512 words of P (or Q) corresponding to any one block, then one can easily derive the complete P (or Q) array with respect to any subsequent block.

Consider that the keystream words $s_{b,i}$, $1 \leq b \leq 4$, $0 \leq i \leq 511$, are observable. We formulate a special state reconstruction problem as follows.

Given the partial state information $P[0 \ldots 511]$,
reconstruct the complete state $(P_N[0 \ldots 511], Q_N[0 \ldots 511])$.

Since the update of each of P and Q depends only on P and Q respectively, once we determine P_N and Q_N, we essentially recover the complete state information for all subsequent steps.

3.1.2 State reconstruction strategy

Our state reconstruction proceeds in five phases. The **First Phase** would be to determine P_N from P using Equation (3.1).

The keystream generation of block B_2 follows the equation

$$s_{2,i} = \begin{cases} h_1(P[500+i]) \oplus P_N[i], & \text{for } 0 \le i \le 11; \\ h_1(P_N[i-12]) \oplus P_N[i], & \text{for } 12 \le i \le 511. \end{cases} \tag{3.2}$$

Since $h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}]$, we can rewrite Equation (3.2) as

$$Q[l_i] + Q[u_i] = s_{2,i} \oplus P_N[i] \tag{3.3}$$

where,

$$\begin{aligned} &l_i = (P[500+i])^{(0)} \text{ and } u_i = 256 + (P[500+i])^{(2)}, \quad 0 \le i \le 11, \\ &l_i = (P_N[i-12])^{(0)} \text{ and } u_i = 256 + (P_N[i-12])^{(2)}, \quad 12 \le i \le 511. \end{aligned} \tag{3.4}$$

Here l_i, u_i and the right hand side $s_{2,i} \oplus P_N[i]$ of System 3.3 of equations are known for all $i = 0, 1, \ldots, 511$. Thus, there are 512 equations in 512 unknowns. The above Phase is formalized in Algorithm 5, called *Phase 1 of ReconstructState*.

Input: $P[0 \ldots 511]$.
Output: $P_N[0 \ldots 511], Q_N[0 \ldots 511]$.
First Phase:
for $i \leftarrow 0$ **to** 511 **do**
1 | Determine $P_N[i]$ using Equation (3.1);
end

Algorithm 5: Phase 1 of ReconstructState

Applying Gaussian elimination, one would require a complexity of around $512^3 = 2^{27}$. However, according to Lemma 3.1, a unique solution does not exist for any such system and hence we have to take a different approach to solve the system. Though the proof of Lemma 3.1 is simple, we include here for easy reference.

Lemma 3.1. *Any system of $r + s$ equations in $r + s$ variables $x_1, x_2, \ldots, x_r$ and $y_1, y_2, \ldots, y_s$, where each equation is of the form $x_i + y_j$ for some i in $[1, r]$ and some j in $[1, s]$, does not have a unique solution.*

Proof. Let us consider the $(r + s) \times (r + s)$ coefficient matrix A of the system. Let the columns be denoted by $C_1, \ldots, C_{r+s}$, such that the first r columns $C_1, \ldots, C_r$ correspond to the variables $x_1, \ldots, x_r$ and the last s columns $C_{r+1}, \ldots, C_{r+s}$ correspond to the variables $y_1, \ldots, y_s$.

Every row of A has the entry 1 in exactly two places and the entry 0 elsewhere. The first 1 in each row appears in one of the columns $C_1, \ldots, C_r$ and the second 1 in one of the columns $C_{r+1}, \ldots, C_{r+s}$.

After the elementary column transformations $C_1 \leftarrow C_1 + \ldots + C_r$ and $C_{r+1} \leftarrow C_{r+1} + \ldots + C_{r+s}$, the two columns C_1 and C_{r+1} has 1's in all the rows and hence become identical. This implies that the matrix is not of full rank and hence unique solution does not exist for the system. □

The left hand side of every equation in System 3.3 is of the form $Q[l_i]+Q[u_i]$, where $0 \leq l_i \leq 255$ and $256 \leq u_i \leq 511$. Taking $r = s = 256$, $x_i = Q[i-1]$, $1 \leq i \leq 256$ and $y_j = Q[255+j]$, $1 \leq j \leq 256$, we see that Lemma 3.1 directly applies to this system, establishing the non-existence of a unique solution.

At this stage, one could remove the redundant rows to find a linear space which contains the solution. However, it is not clear how many variables need to be guessed to arrive at the final solution. Below we formulate a graph theoretic approach to derive the entries of the array Q efficiently, by guessing the value of only a single variable.

Definition 3.2. System 3.3 of 512 equations can be represented in the form of a bipartite graph $G = (V_1, V_2, E)$, where $V_1 = \{0, \ldots, 255\}$, $V_2 = \{256, \ldots, 511\}$ and for $l_i \in V_1$ and $u_i \in V_2$, $\exists$ an edge $\{l_i, u_i\} \in E$ if and only if the sum $Q[l_i] + Q[u_i]$ is known. Thus, $|E| = 512$ (counting repeated edges, if any). We call such a graph G with the vertices as the indices of one internal array of HC-128 the *index graph* of the state of HC-128.

Lemma 3.3. *Let M be the size of the largest connected component of the index graph G corresponding to block B_2. Then M out of the 512 words of the array Q are determined up to the value of a single 32-bit word.*

Proof. Consider any one of the 512 equations of System 3.3. Since the sum $Q[l_i]+Q[u_i]$ is known, knowledge of one of $Q[l_i]$, $Q[u_i]$ reveals the other. Thus, if we know one word of Q at any index of a connected component, we can

immediately derive the words of Q at all the indices of the same component. Since this holds for each connected component, we can guess any one 32-bit word in the largest connected component correctly in 2^{32} attempts and thereby the result follows. □

Since the arrays P, Q and the keystream of HC-128 are assumed to be random, our *index graph* G can be considered to be a random bipartite graph.

Theoretical analysis of the size distribution of the connected components of random finite graphs is a vast area of research in applied probability and there have been several works [28,45,58,91,105] in this direction under different graph models. In [105], the model considered is a bipartite graph $G(n_1, n_2, T)$ with n_1 vertices in the first part, n_2 vertices in the second one and the graph is constructed by T independent trials, each of them consists of drawing an edge which joins two vertices chosen independently of each other from distinct parts. This coincides with our index graph model of Definition 3.2 with $n_1 = |V_1|$, $n_2 = |V_2|$ and $T = |E|$.

In general, let $n_1 \geq n_2$, $\alpha = \frac{n_2}{n_1}$, $\beta = (1-\alpha)\ln n_1$, $n = n_1 + n_2$. Let $\xi_{n_1,n_2,T}$ and $\chi_{n_1,n_2,T}$ denote the number of isolated vertices and the number of connected components in $G(n_1, n_2, T)$ respectively. We have the following result from [105].

Proposition 3.4. *If* $n \to \infty$ *and* $(1+\alpha)T = n\ln n + Xn + o(n)$, *where* X *is a fixed number, then* $Prob\,(\chi_{n_1,n_2,T} = \xi_{n_1,n_2,T} + 1) \to 1$ *and for any* $k = 0, 1, 2, \ldots$, $Prob\,(\xi_{n_1,n_2,T} = k) - \frac{\lambda^k e^{-\lambda}}{k!} \to 0$, *where* $\lambda = \frac{e^{-X}(1+e^{-\beta})}{1+\alpha}$.

In other words, if n is sufficiently large and n_1, n_2, T are related by $(1+\alpha)T = n\ln n + Xn + o(n)$, then the graph contains one giant connected component and isolated vertices whose number follows a Poisson distribution with the aforesaid parameter λ.

Corollary 3.5. *If* M *is the size of the largest component of the index graph* G, *then the mean and standard deviation of* M *are respectively given by* $E(M) \approx 442.59$ *and* $sd(M) \approx 8.33$.

Proof. For our index graph, $n_1 = n_2 = 256$, $n = n_1 + n_2 = 512$, $T = 512$, $\alpha = \frac{n_2}{n_1} = 1$, $\beta = (1-\alpha)\ln n_1 = 0$. The relation $(1+\alpha)T = n\ln n + Xn + o(n)$

is equivalent to $\frac{(1+\alpha)}{n}T = \ln n + X + \frac{o(n)}{n}$. As $n \to \infty$, the ratio $\frac{o(n)}{n} \to 0$ and hence $X \to \frac{(1+\alpha)}{n}T - \ln n$. Substituting $\alpha = 1$, $T = 512$ and $n = 512$, we get $X \approx -4.24$.

By Proposition 3.4, the limiting distribution of the random variable $\xi_{n_1,n_2,T}$ is Poisson with mean (as well as variance) $\lambda = \frac{e^{-X}(1+e^{-\beta})}{1+\alpha} \approx e^{4.24} \approx 69.41$. Moreover, in the limit, $\chi_{n_1,n_2,T} = \xi_{n_1,n_2,T} + 1$ and this implies that all the vertices except the isolated ones would be in a single giant component. Thus, $M = n - \xi_{n_1,n_2,T}$ and the expectation $E(M) = n - E(\xi_{n_1,n_2,T}) = n - \lambda \approx 512 - 69.41 = 442.59$. Again, the variance $Var(M) = Var(n - \xi_{n_1,n_2,T}) = Var(\xi_{n_1,n_2,T}) = \lambda$, giving $sd(M) = sd(\xi_{n_1,n_2,T}) = \sqrt{\lambda} \approx 8.33$. □

Simulations with 10 million trials, each time with 1024 consecutive words of keystream generation for the complete arrays P and Q, gives the average of the number $\xi_{n_1,n_2,T}$ of isolated vertices of the index graph of the state of HC-128 as 69.02 with a standard deviation of 6.41.

These values closely match with the theoretical estimates of the mean $\lambda \approx 69.41$ and standard deviation $\sqrt{\lambda} \approx 8.33$ of $\xi_{n_1,n_2,T}$ derived in Corollary 3.5.

Again from Corollary 3.5, theoretical estimates of the mean and standard deviation of the size M of the largest component is 442.59 and 8.33 respectively. However, from the same simulation described above, the average and standard deviation of M are found to be $407.91 \approx 408$ and 9.17 respectively.

The theoretical expectation $E(M)$ overestimates the actual average of M in practice, because the theoretical estimate is asymptotic with $n \to \infty$ and for our index graph we have $n = 512 < \infty$.

In the limit, each vertex is either an isolated one or part of the single giant component. In practice, on the other hand, except the isolated vertices (≈ 69 in number) and the vertices of the giant component (≈ 408 in number), the remaining few ($\approx 512 - 69 - 408 = 35$ in number) vertices form some small components. However, the low (9.17) empirical standard deviation of M implies that the empirical estimate 408 of $E(M)$ is robust.

We would see later that as a consequence of Theorem 3.7, any $M > 200$ is sufficient for our purpose.

If $C = \{y_1, y_2, \ldots, y_M\}$ be the largest component of G, then we can guess

the word corresponding to any fixed index, say y_1. As explained in the proof of Lemma 3.3, each guess of $Q[y_1]$ uniquely determines the values of $Q[y_2], \ldots, Q[y_M]$. According to Corollary 3.5 and the discussion following it, we can guess around 408 words of Q in this method. This is the **Second Phase** of our solution and is formalized in Algorithm 6, called Phase 2 of ReconstructState.

Second Phase:
1 Form a bipartite graph $G = (V_1, V_2, E)$ as follows;
2 $V_1 \leftarrow \{0, \ldots, 255\}$; $V_2 \leftarrow \{256, \ldots, 511\}$; $E \leftarrow \emptyset$;
3 **for** $i \leftarrow 0$ **to** 511 **do**
4 Determine l_i and u_i using Equation (3.4);
$E \leftarrow E \cup \{l_i, u_i\}$;
end
Find all connected components of G;
5 Let $C = \{y_1, y_2, \ldots, y_M\}$ be the largest component with size M;
Guess $Q[y_1]$ and thereby determine $Q[y_2], \ldots, Q[y_M]$ from Equation (3.3); and for each such guess of $Q[y_1]$, repeat the Third, Fourth and Fifth Phases below;

Algorithm 6: Phase 2 of ReconstructState

We use the following result, which we call *Propagation Theorem*, to determine the remaining unknown words.

Theorem 3.6 (Propagation Theorem)**.** *If Q[y] is known for some y in $[0, 499]$, then $m = \lfloor \frac{511-y}{12} \rfloor$ more words of Q, namely, $Q[y{+}12], Q[y{+}24], \ldots, Q[y{+}12m]$, can all be determined from $Q[y]$ in a time complexity that is linear in the size of Q.*

Proof. Consider block B_1. Following our notation in Section 3.1.1, the equation for keystream generation is

$s_{1,i} = h_2(Q[i-12]) \oplus Q[i]$, for $12 \leq i \leq 511$. Written in a different way, it becomes

$$Q[i] = s_{1,i} \oplus \left(P\left[(Q[i-12])^{(0)}\right] + P\left[256 + (Q[i-12])^{(2)}\right]\right).$$

Now, setting $y = i - 12$, we have, for $0 \leq y \leq 499$,

$$Q[y+12] = s_{1,y+12} \oplus \left(P\left[(Q[y])^{(0)}\right] + P\left[256 + (Q[y])^{(2)}\right]\right). \tag{3.5}$$

Equation (3.5) is a recursive equation, in which all s_1 values and the array P are completely known. Clearly, if we know one $Q[y]$, we know all subsequent $Q[y+12k]$, for $k = 1, 2, \ldots$, as long as $y + 12k \leq 511$. This means $k \leq \frac{511-y}{12}$. The number m of words of Q that can be determined is then the maximum allowable value of k, i.e., $m = \lfloor \frac{511-y}{12} \rfloor$. □

By recursively applying Equation (3.5) to the words of the Q array that were determined from the maximum size connected component of the index graph, we derive approximately $104 (= 512 - 408)$ unknown words in the array. This is the **Third Phase** of our solution.

Third Phase:
1 **for** $j \leftarrow 1$ **to** M **do**
2 $y \leftarrow y_j$;
3 **while** $y \leq 499$ **do**
4 **if** $Q[y+12]$ *is still unknown* **then**
5 $Q[y+12] \leftarrow s_{1,y+12} \oplus \left(P\left[(Q[y])^{(0)}\right] + P\left[256 + (Q[y])^{(2)}\right]\right)$;
end
6 $y \leftarrow y + 12$;
end
end

Algorithm 7: Phase 3 of ReconstructState

If we imagine that the words are initially labeled as 'known' or 'unknown', then this step can be visualized as propagation of the 'known' labels in the forward direction. Even after this step, some words remain unknown. However, as Theorem 3.7 would imply, we observe that through this propagation, all the words $Q[500], Q[501], \ldots Q[511]$ become 'known' with probability almost one.

Theorem 3.7. *After the Third Phase, the expected number of unknown words amongst* $Q[500]$, $Q[501]$, …, $Q[511]$ *is approximately* $8 \cdot (1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$, *where* M *is the size of the largest component of the index graph* G.

Proof. After the Second Phase, exactly M words $Q[y_1], Q[y_2], \ldots Q[y_M]$ are known corresponding to the distinct indices $y_1, y_2, \ldots, y_M$ in the largest component C of size M in G. Since G is a random bipartite graph, each of indices $y_1, y_2, \ldots y_M$ can be considered to be drawn from the set $\{0, 1, \ldots, 511\}$ uni-

formly at random (without replacement). We partition this sample space into 12 disjoint residue classes modulo 12, denoted by, $[0], [1], \ldots, [11]$.

Then, each of the indices $y_1, y_2, \ldots, y_M$ can be considered to be drawn from the set $\{[0], [1], \ldots, [11]\}$ (this time with replacement; this is a reasonable approximation because $M \gg 12$) with probabilities proportional to the sizes of the residue classes. Thus, for $1 \leq j \leq M$, $Prob(y_j \in [r]) = \frac{43}{512}$ if $0 \leq r \leq 7$ and $\frac{42}{512}$ if $8 \leq r \leq 11$.

Let $m_r = 1$, if none of $y_1, y_2, \ldots, y_M$ are from $[r]$; otherwise, let $m_r = 0$. Hence, the total number of residue classes from which no index is selected is $Y = \sum_{r=0}^{11} m_r$.

Now, in the Third Phase, we propagate the known labels in the forward direction using Equation (3.5) (see Theorem 3.6, the Propagation Theorem). The indices $\{500, 501, \ldots, 511\}$ are to the extreme right end of the array Q and hence they also form the set of "last" indices where the propagation eventually stops. Further, each index in the set $\{500, 501, \ldots, 511\}$ belongs to exactly one of the sets $[r]$. Hence, the number of unknown words amongst $Q[500]$, $Q[501]$, ..., $Q[511]$ is also given by Y.

We have,

$$E(m_r) = Prob(m_r = 1) = \begin{cases} (1 - \frac{43}{512})^M & \text{for } 0 \leq r \leq 7; \\ (1 - \frac{42}{512})^M & \text{for } 8 \leq r \leq 11. \end{cases}$$

Thus, $E(Y) = \sum_{r=0}^{11} E(m_r) = 8 \cdot (1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$. □

Substituting M by its theoretical mean estimate 443 as well as by its empirical mean estimate 408 yields $E(Y) \approx 0$.

In fact, for any $M > 200$, the expression $(1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$ for $E(Y)$ becomes vanishingly small. Our experimental data also supports that in every instance, none of the words $Q[500], Q[501], \ldots, Q[511]$ remains unknown.

Note that changing bytes 1 or 3 of $Q[y]$ yields no change in Equation (3.5). Combining this with the Second Phase, we could form a new set of equations and attempt to solve them. However, as Theorem 3.7 establishes,

this is not required; propagation of known $Q[y]$ values in steps of 12 covers all the unknowns.

Next, we use the following result to determine the entire Q_N array.

Theorem 3.8. *Suppose the complete array P_N and the 12 words $Q[500]$, $Q[501]$, ..., $Q[511]$ from the array Q are known. Then the entire Q_N array can be reconstructed in a time complexity linear in the size of Q.*

Proof. Following our notation in Section 3.1.1, the equation for the keystream generation of the first 12 steps of block B_3 is $s_{3,i} = h_2(Q[500+i]) \oplus Q_N[i]$, $0 \leq i \leq 11$.

Expanding $h_2(.)$, we get, for $0 \leq i \leq 11$,

$$Q_N[i] = s_{3,i} \oplus \left(P_N\left[(Q[500+i])^{(0)}\right] + P_N\left[256 + (Q[500+i])^{(2)}\right]\right).$$

So, we can determine $Q_N[0], Q_N[1], \ldots, Q_N[11]$ from $Q[500], Q[501], \ldots Q[511]$.

Now, applying Theorem 3.6 on these first 12 words of Q_N, we can determine all the words of Q_N in linear time (in size of Q). □

Applying Theorem 3.8 constitute the **Fourth Phase** of our solution.

Fourth Phase:
1 **for** $i \leftarrow 0$ **to** 11 **do**
2 $\quad Q_N[i] \leftarrow s_{3,i} \oplus \left(P_N\left[(Q[500+i])^{(0)}\right] + P_N\left[256 + (Q[500+i])^{(2)}\right]\right)$;
4 $\quad y \leftarrow i$;
5 $\quad$ **while** $y \leq 499$ **do**
6 $\quad\quad Q_N[y+12] \leftarrow s_{3,y+12} \oplus \left(P_N\left[(Q_N[y])^{(0)}\right] + P_N\left[256 + (Q_N[y])^{(2)}\right]\right)$;
7 $\quad\quad y \leftarrow y + 12$;
$\quad$ **end**
end

Algorithm 8: Phase 4 of ReconstructState

After Q_N is derived, we need to verify its correctness. For this, we update P_N as it would be updated in block B_4 and generate 512 keystream words with this P_N and the derived Q_N. If the generated keystream words entirely match with the observed keystream words $\{s_{4,0}, s_{4,1}, \ldots, s_{4,511}\}$ of block B_4, then our guess is correct. This verification is the **Fifth** (and final) **Phase** of

the algorithm. If we find a mismatch, then we repeat the procedure with the next guess, i.e., with another possible value in $[0, 2^{32} - 1]$ of the word $Q[y_1]$.

Once Q_N is correctly determined, the words of the Q array for all the succeeding blocks can be deterministically computed from the update rule for Q.

The above discussion is formalized in Algorithm 9, called *ReconstructState*.

Theorem 3.9. *The data complexity of Algorithm 9 is* 2^{16} *and its time complexity is* 2^{42}.

Proof. For the First Phase, we do not need any keystream word. For each of the Second, Third, Fourth and Fifth Phases, we need a separate block of 512 keystream words. Thus, the required amount of data is $4 \cdot 512 = 2^{11}$ no. of 32 $(= 2^5)$-bit keystream words.

From Step 0 in the First Phase up to Step 4 of the Second Phase, the total time required is linear in the size of P (or Q), i.e., of complexity 2^9. Step 4 in the Second Phase of Algorithm 9 can be performed through depth-first search which requires $O(|V_1| + |V_2| + |E|)$ time complexity. For $|V_1| = 256$, $|V_2| = 256$ and $|E| = 512$, the value turns out to be 2^{10}. After this, the guess in Step 5 of Algorithm 9 consumes 2^{32} time and for each such guess, the complete Phases 3, 4 and 5 together take time that is linear in the size of the array Q, i.e., of complexity 2^9. Thus, the total time required is $2^9 + 2^{10} + 2^{32} \cdot 2^9 < 2^{42}$. □

Note that for System 3.3 of Equations, one must verify the solution by first generating some keystream words and then matching them with the observed keystream, as is done in the Fifth Phase of Algorithm 9. During Step 5 in the Second Phase, one may exploit the cycles of the largest component to verify correctness of the guess. If the guessed value of a variable in a cycle does not match with the value of the variable derived when the cycle is closed, we can discard that guess. However, in the worst case, all the 2^{32} guesses have to be tried and if there is no conflict in a cycle, the guess has to be verified by keystream matching. Thus, it is not clear if there is any significant advantage by detecting and exploiting the cycles and so we have not considered this in the description of the algorithm.

Input: $P[0 \dots 511]$.
Output: $P_N[0 \dots 511], Q_N[0 \dots 511]$.
First Phase:
for $i \leftarrow 0$ **to** 511 **do**
1 | Determine $P_N[i]$ using Equation (3.1);
end
Second Phase:
2 Form a bipartite graph $G = (V_1, V_2, E)$ as follows;
3 $V_1 \leftarrow \{0, \dots, 255\}$; $V_2 \leftarrow \{256, \dots, 511\}$; $E \leftarrow \emptyset$;
4 **for** $i \leftarrow 0$ **to** 511 **do**
5 | Determine l_i and u_i using Equation (3.4);
| $E \leftarrow E \cup \{l_i, u_i\}$;
end
Find all connected components of G;
6 Let $C = \{y_1, y_2, \dots, y_M\}$ be the largest component with size M;
Guess $Q[y_1]$ and thereby determine $Q[y_2], \dots, Q[y_M]$ from Equation (3.3);
and for each such guess of $Q[y_1]$, repeat the Third, Fourth and Fifth Phases below;
Third Phase:
7 **for** $j \leftarrow 1$ **to** M **do**
8 | $y \leftarrow y_j$;
9 | **while** $y \leq 499$ **do**
10 | | **if** $Q[y+12]$ *is still unknown* **then**
11 | | | $Q[y+12] \leftarrow s_{1,y+12} \oplus \left(P\left[(Q[y])^{(0)}\right] + P\left[256 + (Q[y])^{(2)}\right]\right)$;
| | **end**
12 | | $y \leftarrow y + 12$;
| **end**
end
Fourth Phase:
13 **for** $i \leftarrow 0$ **to** 11 **do**
14 | $Q_N[i] \leftarrow s_{3,i} \oplus \left(P_N\left[(Q[500+i])^{(0)}\right] + P_N\left[256 + (Q[500+i])^{(2)}\right]\right)$;
16 | $y \leftarrow i$;
17 | **while** $y \leq 499$ **do**
18 | | $Q_N[y+12] \leftarrow s_{3,y+12} \oplus \left(P_N\left[(Q_N[y])^{(0)}\right] + P_N\left[256 + (Q_N[y])^{(2)}\right]\right)$;
19 | | $y \leftarrow y + 12$;
| **end**
end
Fifth Phase:
20 With the new Q_N, generate 512 keystream words by updating P_N;
21 Verify correctness of the guess in Step 5 by matching these keystream words with the observed keystream words of block B_4;

Algorithm 9: ReconstructState

3.2 Design modifications

We have two design goals. Firstly, to guard against the available analysis in literature and secondly, to attain the objective at optimal computing or processing speed. Thus, we attempt to keep the same structure as the original HC-128 with minimal changes.

Apart from the present work, we are aware of only three other similiar works on the analysis of HC-128, one by the designer Wu [128], the next as in [80] and the most recent one from [64]. The first two works exploit the fact that $h_1(.)$ as well as $h_2(.)$ make use of only 16 bits from the 32-bit input. Our current work also uses this fact to form Equation (3.3) that eventually leads to reconstruction of the state. Thus, all of these results indicate that the form of $h_1(.), h_2(.)$ need to be modified so as to incorporate all the 32 bits of their inputs. In our new versions of these functions (Equation (3.6)), we suggest XOR-ing the entire input with the existing output (sum of two array entries). However, certain precautions are needed so that other security threats do not come into play.

We replace h_1 and h_2 as follows.

$$\left.\begin{aligned} h_{N1}(x) &= (Q[x^{(0)}] + Q[256 + x^{(2)}]) \oplus x. \\ h_{N2}(x) &= (P[x^{(0)}] + P[256 + x^{(2)}]) \oplus x. \end{aligned}\right\} \tag{3.6}$$

We need to modify the update functions g_1 and g_2 with the twin motivation of preserving the internal state as well as making sure that the randomness of the keystream is ensured. We propose the following.

$$\left.\begin{aligned} g_{N1}(x, y, z) &= \Big((x \ggg 10) \oplus (z \ggg 23)\Big) + Q[(y \gg 7) \wedge 1FF]. \\ g_{N2}(x, y, z) &= \Big((x \lll 10) \oplus (z \lll 23)\Big) + P[(y \gg 7) \wedge 1FF]. \end{aligned}\right\} \tag{3.7}$$

We keep f_1 and f_2 the same as in original HC-128.

We include a randomly chosen word from the Q array in the update of P array elements and a randomly chosen word from the P array while updating

the Q array elements.

This would ensure that each new block of P (or Q) array is dependent on the previous block of Q(or P) array. Thus, our analysis of Section 3.1 would not apply and the internal state would be preserved even if half the internal state elements are known. The fault analysis in [64] explains that the location of fault is uniquely determined by observing subsequent changes in the keystream. This occurs because during the update of terms of one array no term of the other array is used. Further during the update function, the terms used follow a serial sequence. That is if a fault occurs at $Q[f]$ in the block in which P is updated, then $Q[f]$ is not referenced until step $f-1$ of the next block (in which Q would be updated). This assumption does not hold for our design due to the nesting of P and Q in the updates of one another (Equation (3.7)) and hence our design resists the fault analysis. Lemma 1 and 2 of [64] do not hold as the fault propagation will not be straight forward.

Likewise, in the equation of the distinguisher proposed by the designer [128, Section 4], the term $P[i \boxminus 10]$ will get replaced by some random term of Q array. With this replacement, it is not obvious how a similar distinguishing attack can be mounted.

The security of any stream cipher is always a conjecture. We have tried to circumvent the known issues of HC-128. The way we have modified the design, it appears that no new security holes are introduced. However, the new design is open to the community for further analysis.

3.3 Performance Evaluation

We evaluated the performance of our new design using the eSTREAM testing framework [24]. The C-implementation of the testing framework was installed in a machine with Intel(R) Pentium(R) D CPU, 2.8 GHz Processor Clock, 2048 KB Cache Size, 1 GB DDR RAM on Ubuntu 7.04 (Linux 2.6.20-17-generic) OS. A benchmark implementation of HC-128 and HC-256 [127] is available within the test suite. We implemented our modified version of HC-128, maintaining the API compliance of the suite. Test vectors were generated in the NESSIE [93] format. The results presented below correspond to tests

with null IV using the gcc-3.4_prescott_O3-ofp compiler.

	HC-128	Our Proposal	HC-256
Stream Encryption (cycles/byte)	4.13	4.29	4.88

Table 3.2: Speed comparisons of our proposals with HC-128 and HC-256.

The encryption speed of our proposed design is of the same order as that of original HC-128. We also observe that the extra array element access in the new update rules (Equation (3.7)) as compared to the original update rules does not affect the performance much. HC-128 was designed as a lightweight version of HC-256. The idea of cross-referencing each other in the update rules of P and Q has also been used in the design of HC-256 and that is why the half state exposure does not reveal the full state in case of HC-256. However, our modification to HC-128 removes the known weaknesses of HC-128 but keeps the speed faster than HC-256, with only little reduction in speed compared to HC-128.

3.4 Conclusion

The stream cipher HC-128 uses two internal arrays, each containing 512 32-bit words. In this chapter, we show that if one knows only one array completely and has access to 2048 consecutive keystream words then the other array can be completely reconstructed in 2^{42} time complexity. While this does not affect the actual security of the cipher, this is an observation related to the the internal structure. To resist this, we proposed a design modification of HC-128. We also evaluated the performance of our proposal in the eSTREAM testing framework and the performance was close to that of original HC-128 and HC-256.

Chapter 4

Study of HC-128 Keystream

"The world is full of obvious things which nobody by any chance ever observes." – Arthur Conan Doyle

4.1 Introduction

In this chapter we study HC-128 in detail from cryptanalytic point of view. First, we use linear approximation of the addition modulo 2^n of three n-bit integers to identify linear approximations of g_1, g_2, the feedback functions of HC-128. This, in turn, shows that the process of keystream output generation of HC-128 can be well approximated by linear functions. In this direction, we show that the "least significant bit" based distinguisher (presented by the designer of the cipher) of HC-128 works for the complete 32-bit word. Using the above linear approximations of g_1, g_2, we present a new distinguisher for HC-128 which is slightly weaker than Wu's distinguisher. Finally, from Dunkelman's observation, we also study how HC-128 keystream words leak secret state information of the cipher due to the properties of the functions h_1, h_2 and present improved results.

In this work, we identify a few other issues pertaining to HC-128. Though our results do not constitute an attack on HC-128, we believe these will aid further exposure towards analysis of the cipher.

Each keystream word of HC-128 is 32 bit long (the 0th bit is the least

significant bit and the 31st bit is the most significant bit). In [128], bitwise XOR of least significant bits of 10 (possibly) different keystream words (rotated by certain amounts) are considered to propose a distinguisher (that requires 2^{156} keystream words with a success probability of 0.9722) and it has been commented: "But due to the effect of the two '+' operations in the feedback function, the attack exploiting those 31 bits is not as effective as that exploiting the least significant bit". In Section 4.2, we discuss the linear approximation of the feedback functions g_1, g_2. These results are used in Section 4.3 to characterize the distinguisher for all other bits. In Section 4.3.2, we show that for each of the bits 2 to 31, one can have distinguishers of almost the same strength as the distinguisher proposed for the least significant bit in [128]. All these distinguishers can be taken together to mount a word level distinguisher for HC-128.

We present a novel distinguisher in Section 4.4. The distinguisher in [128] uses the update formula for P (or Q) array during the keystream generation. However, in our analysis, we identify a different relation among the words of only either P (or only Q) array,(at a single time) that leads to this new distinguisher. In [128], elements at five distinct indices from two consecutive blocks of only P (or only Q) array have been related to mount the distinguisher. In our distinguisher, elements at five distinct indices from three consecutive blocks of either only P (or only Q) array have been related. Though our distinguisher is marginally weaker than that of [128], ours is the only known distinguisher other than the one identified by the designer himself [128].

In Section 4.5, we study how the keystream output words leak secret state information in HC-128. In [36], it has been observed that XOR of two consecutive keystream words of 32-bit each is equal to the XOR of two consecutive words of the secret array with probability $\approx 2^{-16}$. We study this analysis in more detail and infer a lucid association which gives twice the above probability.

4.2 Linear approximation of the functions g_1 and g_2

HC-128 uses two functions g_1, g_2 of similar kind. The two '+' operations in g_1 or g_2 are believed to be a source of high nonlinearity, but we find good linear approximation in this case by using the result of linear approximation of the addition of three integers.

Let us first present a brief outline to the linear approximation of addition of n-bit integers modulo 2^n. Consider three integers $X = (X_{n-1}, \ldots, X_0), Y = (Y_{n-1}, \ldots, Y_0), Z = (Z_{n-1}, \ldots, Z_0)$ of n-bits each. Let the addition modulo 2^n be $S = (X + Y) \bmod 2^n$, and the GF(2) addition corresponding to each bit be $T = X \oplus Y$, . Similarly consider $S' = (X+Y+Z) \bmod 2^n$, and $T' = X \oplus Y \oplus Z$.

For $n = 8$, the probabilities of $S_i = T_i$ and $S'_i = T'_i$ are presented in the following table. We observe that while $Prob(S_i = T_i)$ tends to $\frac{1}{2}$ as i increases,

i	$Prob(S_i = T_i)$ (For two integers)	$Prob(S'_i = T'_i)$ (For three integers)
0	1.00000000	1.00000000
1	0.75000000	0.50000000
2	0.62500000	0.37500000
3	0.56250000	0.34375000
4	0.53125000	0.33593750
5	0.51562500	0.33398438
6	0.50781250	0.33349609
7	0.50390625	0.33337402

Table 4.1: Linear Approximaton of Addition for first 8 bits.

$Prob(S'_i = T'_i)$ tends to $\frac{1}{3}$. This shows that $Prob(S'_i = 1 \oplus X_i \oplus Y_i \oplus Z_i)$ is approximately $\frac{2}{3}$ for $i \geq 2$ and thus the i-th bit ($i \geq 2$) of addition modulo 2^n of three integers is highly correlated to the complement of the bitwise XOR of the integers

Linear approximations of modulo-2^n addition of k many n-bit integers have been studied in [118]. For $k = 2$, the probability of the equality of XOR and

modulo-2^n sum in the b-th least significant bit tends to $\frac{1}{2}$ as b increases. Below, we briefly discuss the case for $k = 3$, i.e., the XOR-approximation of modulo addition of three integers, that would be subsequently used in approximating g_1, g_2. We do not claim that the probability calculation in Proposition 4.1 below as our contribution, but we present an outline for better understanding.

As we would be using the keystream word number as subscript, we denote the b-th least significant bit of an n-bit word w by $[w]^b$, $0 \leq b \leq n-1$, i.e., $w = ([w]^{n-1}, [w]^{n-2}, \ldots, [w]^1, [w]^0)$. This notation is also extended to $[w]^b$, where $b > n-1$. In that case, $[w]^b$ will mean $[w]^{b \bmod n}$.

Let X, Y, Z be three n-bit integers; $S = (X+Y+Z) \bmod 2^n$, $T = X \oplus Y \oplus Z$, the bitwise XOR. Let $[C]^b$ denote the carry bit produced in the b-th step of the addition of X, Y and Z. Since three bits are involved, $[C]^b$ can take the values $0, 1$ and 2. For the LSB addition, we assume $[C]^{-1} = 0$. Denote $\rho_{b,v} = Prob([C]^b = v)$, $b \geq -1$, $v \in \{0, 1, 2\}$. We know that $Prob([S]^b = [T]^b) = Prob([C]^{b-1} = 0 \text{ or } 2) = \rho_{b-1,0} + \rho_{b-1,2} = 1 - \rho_{b-1,1}$.

The following recurrences are easy to show.

1. $\rho_{b+1,0} = \frac{1}{2}\rho_{b,0} + \frac{1}{8}\rho_{b,1}$.
2. $\rho_{b+1,1} = \frac{1}{2}\rho_{b,0} + \frac{3}{4}\rho_{b,1} + \frac{1}{2}\rho_{b,2}$.
3. $\rho_{b+1,2} = \frac{1}{8}\rho_{b,1} + \frac{1}{2}\rho_{b,2}$.

The solution gives $\rho_{b,1} = \frac{2}{3}(1 - \frac{1}{4^{b+1}})$ and so we have the following result.

Proposition 4.1. *For $0 \leq b \leq n-1$, $Prob([S]^b = [T]^b) = \frac{1}{3}(1 + \frac{1}{2^{2b-1}})$.*

The following corollary gives an immediate approximation to Proposition 4.1.

Corollary 4.2. *For $0 \leq b \leq n-1$, $Prob([S]^b = [T]^b) = p_b$, where*

$$p_b = \begin{cases} 1 & \text{if } b = 0; \\ \frac{1}{2} & \text{if } b = 1; \\ \frac{1}{3} \text{ (approximately)} & \text{if } 2 \leq b \leq n-1. \end{cases}$$

During the keystream generation part of HC-128, the array P is updated as

$$P[i \bmod 512] = P[i \bmod 512] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511]),$$

where

$$g_1(x, y, z) = \Big((x \ggg 10) \oplus (z \ggg 23)\Big) + (y \ggg 8).$$

Thus, the update rule can be restated as

$$\begin{aligned} P_{up}[i \bmod 512] = P[i \bmod 512] + \Big((P[i \boxminus 3] \ggg 10) \\ \oplus (P[i \boxminus 511] \ggg 23)\Big) + (P[i \boxminus 10] \ggg 8). \end{aligned} \tag{4.1}$$

In consistence with the notations in [128, Section 4], we may write the keystream generation step as follows. For $0 \leq i \bmod 1024 < 512$,

$$s_i = \begin{cases} h_1(P[i \boxminus 12]) \oplus P_{up}[i \bmod 512], & \text{for } 0 \leq i \bmod 512 \leq 11; \\ h_1(P_{up}[i \boxminus 12]) \oplus P_{up}[i \bmod 512], & \text{for } 12 \leq i \bmod 512 \leq 511. \end{cases} \tag{4.2}$$

Let P'_{up} denote the updated array P, when we replace the two '+' operators by '$\oplus$' in the right hand side of Equation (4.1).

Then for $0 \leq b \leq n-1$, the b-th bit of the updated words of the array P would be given by

$$\begin{aligned} \Big[P'_{up}[i \bmod 512]\Big]^b = \Big[P[i \bmod 512]\Big]^b \oplus \Big[P[i \boxminus 3]\Big]^{10+b} \\ \oplus \Big[P[i \boxminus 511]\Big]^{23+b} \oplus \Big[P[i \boxminus 10]\Big]^{8+b}. \end{aligned}$$

We define

$$s'_i = \begin{cases} h_1(P[i \boxminus 12]) \oplus P'_{up}[i \bmod 512], & \text{for } 0 \le i \bmod 512 \le 11; \\ h_1(P_{up}[i \boxminus 12]) \oplus P'_{up}[i \bmod 512], & \text{for } 12 \le i \bmod 512 \le 511. \end{cases}$$

Each s'_i can be considered as the "distorted" value of s_i due the XOR-approximation of the sum of three integers in Equation (4.1). From Corollary 4.2 of Proposition 4.1 and the definition of s'_i, the following result is immediate.

Lemma 4.3. *For* $0 \le i \bmod 1024 < 512$ *and* $0 \le b \le n-1$,
$Prob\left(\left[s'_i\right]^b = \left[s_i\right]^b\right) = Prob\left(\left[P'_{up}[i \bmod 512]\right]^b = \left[P_{up}[i \bmod 512]\right]^b\right) = p_b$.

We may construct 32-bit integers ζ_i's as estimates of the keystream words s_i's as follows.

$$[\zeta_i]^b = \begin{cases} \left[s'_i\right]^b & \text{if } b = 0, 1; \\ 1 \oplus \left[s'_i\right]^b & \text{if } 2 \le b < 32. \end{cases}$$

Thus, we have the following result.

Theorem 4.4. *The expected number of bits where the two 32-bit integers* s_i *and* ζ_i *match is approximately* 21.5.

Proof. Let $m_b = 1$, if $[s_i]^b = [\zeta_i]^b$; otherwise, let $m_b = 0$, $0 \le b \le 31$. Hence, the total number of matches is given by $M = \sum_{b=0}^{31} m_b$. By linearity of expectation, $E(M) = \sum_{b=0}^{31} E(m_b) = \sum_{b=0}^{31} Prob(m_b = 1)$.

From Lemma 4.3 and the construction of ζ_i, we have

$$Prob(m_b = 1) = \begin{cases} p_b & \text{if } b = 0, 1; \\ 1 - p_b & \text{if } 2 \le b < 32. \end{cases}$$

This gives $E(M) \approx 1 + \frac{1}{2} + 30 \cdot (1 - \frac{1}{3}) = 21.5$. □

Theorem 4.4 shows the association of the HC-128 keystream word s_i with its linear approximation ζ_i.

...

Old P array:	$P[0]$	$P[1]$	...	$P[511]$
Keystream:	s_0	s_1	...	s_{511}
Intermediate Q array:	$Q[0]$	$Q[1]$	...	$Q[511]$
Keystream:	s_{512}	s_{513}	...	s_{1023}
New P array:	$P[0]$	$P[1]$	...	$P[511]$
Keystream:	s_{1024}	s_{1025}	...	s_{1535}
New Q array:	$Q[0]$	$Q[1]$	...	$Q[511]$
Keystream:	s_{1536}	s_{1537}	...	s_{2047}

...

Table 4.2: Evolution of the Arrays P and Q and Correspondence with the Keystream Words.

4.3 A class of distinguishers by extending the LSB-based distinguisher

In this section, we use the linear approximation of the feedback functions g_1, g_2 described in the previous section to construct 30 more bit level distinguishers.

4.3.1 Brief outline of the LSB distinguisher of HC-128

Before presenting the ideas in this section, let us revisit the keystream word generation of HC-128. The keystream words are generated using both the arrays P and Q, each consisting of 512 many words. However, the updates of P and Q arrays are independent. For 512 iterations, the array P is updated with the older values from P itself and for the next 512 iterations the array Q is updated with the older values of Q. This method continues alternatively. Table 4.2 shows how the keystream words s_i's are related to the array elements $P[i]$'s and $Q[i]$'s.

In general, for $0 \leq (i \bmod 1024) < 512$, the keystream output word of HC-128 is generated as $s_i = h_1(P[i \boxminus 12]) \oplus P[i \bmod 512]$, following an update of $P[i \bmod 512]$ by adding to it $g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])$, as formulated

in Equations (4.1) and (4.2) of Section 4.2. Let $P[i \boxminus 12]$ at the i-th step be denoted by z_i. From Equation (4.2), it is clear that any occurrence of $P[i \bmod 512]$ can be replaced by $s_i \oplus h_1(z_i)$. Performing this replacement in Equation (4.1), we get, for $10 \leq i \bmod 512 < 511$,

$$s_i \oplus h_1(z_i) = \Big(s_{i-1024} \oplus h'_1(z_{i-1024})\Big) + g_1\Big(s_{i-3} \oplus h_1(z_{i-3}), s_{i-10} \oplus h_1(z_{i-10}), s_{i-1023} \oplus h'_1(z_{i-1023})\Big). \quad (4.3)$$

Here $h_1(.)$ and $h'_1(.)$ indicate two different functions since they are related to two P arrays at two different 1024 size blocks that act as two different S-boxes.

Inside g_1, we have three rotations, one XOR and one addition and outside g_1 we have one more addition. Since the LSB of the XOR of any number of words equals the LSB of the sum of those words, we can write Equation (4.3) as

$$[s_i]^0 \oplus [s_{i-1024}]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23}$$
$$= [h_1(z_i)]^0 \oplus [h'_1(z_{i-1024})]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23}.$$

Thus, for $1024\tau + 10 \leq j < i < 1024\tau + 511$,
$[s_i]^0 \oplus [s_{i-1024}]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23}$
$= [s_j]^0 \oplus [s_{j-1024}]^0 \oplus [s_{j-3}]^{10} \oplus [s_{j-10}]^8 \oplus [s_{j-1023}]^{23}$, if and only if $H(\xi_i) = H(\xi_j)$, where,

$$H(\xi_i) = [h_1(z_i)]^0 \oplus [h'_1(z_{i-1024})]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23}.$$

Here $\xi_i = (z_i, z_{i-1024}, z_{i-3}, z_{i-10}, z_{i-1023})$ is an 80-bit input and $H(.)$ can be assumed to be a random 80-bit-to-1-bit S-box.

The following result (with proof for better clarity) gives the collision probability for a general random m-bit-to-n-bit S-box.

Proposition 4.5. *[128, Theorem 1] Let H be an m-bit-to-n-bit S-box and all those n-bit elements are randomly generated, where $m \geq n$. Let x_1 and x_2 be two m-bit random inputs to H. Then $H(x_1) = H(x_2)$ with probability*

$2^{-m} + 2^{-n} - 2^{-m-n}$.

Proof. If $x_1 = x_2$ (this happens with probability 2^{-m}), then $H(x_1) = H(x_2)$ happens with probability 1. If $x_1 \neq x_2$ (this happens with probability $1 - 2^{-m}$), then $H(x_1) = H(x_2)$ happens with probability 2^{-n}. Thus, $Prob\,(H(x_1) = H(x_2)) = 2^{-m} \cdot 1 + (1 - 2^{-m}) \cdot 2^{-n}$. □

Coming back to HC-128, $m = 80$ and $n = 1$ for the S-box whose outputs are $H(\xi_i)$ and $H(\xi_j)$, we have, according to Proposition 4.5,

$$Prob\,(H(\xi_i) = H(\xi_j)) = \frac{1}{2} + 2^{-81}.$$

Hence, for $1024\tau + 10 \leq j < i < 1024\tau + 511$,

$$Prob\Big([s_i]^0 \oplus [s_{i-1024}]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} = [s_j]^0 \oplus [s_{j-1024}]^0 \oplus [s_{j-3}]^{10} \oplus [s_{j-10}]^8 \oplus [s_{j-1023}]^{23}\Big) = \tfrac{1}{2} + 2^{-81}.$$

Thus, a distinguisher can be mounted based on the equality of the least significant bits of the keystream word combinations $s_i \oplus s_{i-1024} \oplus (s_{i-3} \ggg 10) \oplus (s_{i-10} \ggg 8) \oplus (s_{i-1023} \ggg 23)$ and $s_j \oplus s_{j-1024} \oplus (s_{j-3} \ggg 10) \oplus (s_{j-10} \ggg 8) \oplus (s_{j-1023} \ggg 23)$. According to [128, Section 4], this distinguisher requires approximately 2^{164} many equations of the above form or a total of 2^{156} many keystream words for a success probability 0.9772. It has been mentioned in [128] that the distinguishing attack exploiting the other 31 bits will not be effective due to the use of modulo addition. In contrary to the belief of the designer of HC-128, we show in the next section that the distinguisher works for all the bits (except one) in the keystream words.

4.3.2 Our extension to other bits

Our analysis shows that there exist biases in the equality of 31 out of the 32 bits (except the second least significant bit) of the word combinations $s_i \oplus s_{i-1024} \oplus (s_{i-3} \ggg 10) \oplus (s_{i-10} \ggg 8) \oplus (s_{i-1023} \ggg 23)$ and $s_j \oplus s_{j-1024} \oplus (s_{j-3} \ggg 10) \oplus (s_{j-10} \ggg 8) \oplus (s_{j-1023} \ggg 23)$, which leads to a distinguisher for each of those 31 bits separately.

Our analysis generalizes the idea of [128, Section 4] by applying Corollary 4.2. We refer to the visualization of the array P as explained in Table 4.2

of Section 4.3.1 and focus on Equation(4.3):

$$s_i \oplus h_1(z_i) = \Big(s_{i-1024} \oplus h'_1(z_{i-1024})\Big) + g_1\Big(s_{i-3} \oplus h_1(z_{i-3}), s_{i-10} \oplus h_1(z_{i-10}), s_{i-1023} \oplus h'_1(z_{i-1023})\Big).$$

We replace the two '+' operations (one inside and one outside) g_1 by '$\oplus$'. Then as per the discussion following Corollary 4.2, we can write, for $10 \leq i \bmod 1024 < 511$,

$[s_i]^b \oplus [s_{i-1024}]^b \oplus [s_{i-3}]^{10+b} \oplus [s_{i-10}]^{8+b} \oplus [s_{i-1023}]^{23+b}$

$= [h_1(z_i)]^b \oplus [h'_1(z_{i-1024})]^b \oplus [h_1(z_{i-3})^{10+b}] \oplus [h_1(z_{i-10})]^{8+b} \oplus [h'_1(z_{i-1023})]^{23+b}$

holds with probability $p_0 = 1$ for $b = 0$, with probability $p_1 = \frac{1}{2}$ for $b = 1$ and with probability $p_b \approx \frac{1}{3}$ for $2 \leq b \leq 31$.

In short, we can write, for $0 \leq b \leq 31$,

$$Prob\Big([\psi_i]^b = H_b(\xi_i)\Big) = p_b,$$

where the 32-bit integer ψ_i is constructed as:

$$[\psi_i]^b = [s_i]^b \oplus [s_{i-1024}]^b \oplus [s_{i-3}]^{10+b} \oplus [s_{i-10}]^{8+b} \oplus [s_{i-1023}]^{23+b}$$

and

$$H_b(\xi_i) = [h_1(z_i)]^b \oplus [h'_1(z_{i-1024})]^b \oplus [h_1(z_{i-3})]^{10+b} \oplus [h_1(z_{i-10})]^{8+b} \oplus [h'_1(z_{i-1023})]^{23+b}.$$

Here $\xi_i = (z_i, z_{i-1024}, z_{i-3}, z_{i-10}, z_{i-1023})$ is an 80-bit input and each $H_b(.)$, $0 \leq b \leq 31$, is a random 80-bit-to-1-bit S-box. Obviously, for $0 \leq b \leq 31$, $Prob\Big([\psi_i]^b = H_b(\xi_i) \oplus 1\Big) = 1 - p_b$.

Thus, we can state the following technical result.

Lemma 4.6. *For* $1024\tau + 10 \leq j < i < 1024\tau + 511$ *and* $0 \leq b \leq 31$,

$$Prob\Big([\psi_i]^b \oplus [\psi_j]^b = H_b(\xi_i) \oplus H_b(\xi_j)\Big) = q_b$$

where

$$q_b = \begin{cases} 1 & \text{if } b = 0; \\ \frac{1}{2} & \text{if } b = 1; \\ \frac{5}{9} \text{ (approximately)} & \text{if } 2 \leq b \leq 31. \end{cases}$$

Proof. $Prob\left([\psi_i]^b \oplus [\psi_j]^b = H_b(\xi_i) \oplus H_b(\xi_j)\right)$
$= Prob\left([\psi_i]^b = H_b(\xi_i)\right) \cdot Prob\left([\psi_j]^b = H_b(\xi_j)\right)$
$\quad + Prob\left([\psi_i]^b = H_b(\xi_i) \oplus 1\right) \cdot Prob\left([\psi_j]^b = H_b(\xi_j) \oplus 1\right)$
$= p_b \cdot p_b + (1 - p_b) \cdot (1 - p_b)$.
Substituting the values of p_b from Corollary 4.2, we get the result. □

Obviously, for $0 \leq b \leq 31$, $Prob\left([\psi_i]^b \oplus [\psi_j]^b = H_b(\xi_i) \oplus H_b(\xi_j) \oplus 1\right) = 1 - q_b$.

For a given b, all the $H_b(\xi_i)$'s are the outputs of the same random secret 80-bit-to-1-bit S-box $H_b(.)$. So setting $m = 80$ and $n = 1$ in Proposition 4.5, we get the following corollary.

Corollary 4.7. *For* $1024\tau + 10 \leq j < i < 1024\tau + 511$ *and* $0 \leq b \leq 31$,

$$Prob\left(H_b(\xi_i) = H_b(\xi_j)\right) = \frac{1}{2} + 2^{-81}.$$

Obviously, $Prob\left(H_b(\xi_i) = H_b(\xi_j) \oplus 1\right) = \frac{1}{2} - 2^{-81}$.

Combining the above results, we get the following theorem.

Theorem 4.8. *For* $1024\tau + 10 \leq j < i < 1024\tau + 511$, $Prob\left([\psi_i]^b = [\psi_j]^b\right) = r_b$, *where*

$$r_b = \begin{cases} \frac{1}{2} + 2^{-81} & \text{if } b = 0; \\ \frac{1}{2} & \text{if } b = 1; \\ \frac{1}{2} + \frac{2^{-81}}{9} \text{ (approximately)} & \text{if } 2 \leq b \leq 31. \end{cases}$$

Proof. $Prob\left([\psi_i]^b = [\psi_j]^b\right)$
$= Prob\left([\psi_i]^b \oplus [\psi_j]^b = H_b(\xi_i) \oplus H_b(\xi_j)\right) \cdot Prob\left(H_b(\xi_i) = H_b(\xi_j)\right)$
$\quad + Prob\left([\psi_i]^b \oplus [\psi_j]^b = H_b(\xi_i) \oplus H_b(\xi_j) \oplus 1\right) \cdot Prob\left(H_b(\xi_i) = H_b(\xi_j) \oplus 1\right)$.
Substituting values from Lemma 4.6 and Corollary 4.7, we get the result. □

One may consider the probability in the form of $p + \epsilon$, where $p = \frac{1}{2}$ is the expected probability and ϵ is the bias. Then it is possible to directly use the Piling-up lemma as described in [121, page 80]. Note that for the special case of $b = 0$, we have a distinguisher based on the non-uniform probability $\frac{1}{2} + 2^{-81}$ in the equality of the LSB's of ψ_i and ψ_j. This corresponds to a bias of 2^{-81} and is exactly the distinguisher described in [128, Section 4]. Our results show that we can also mount a distinguisher of around the same order for each of the 30 bits corresponding to $b = 2, 3, \ldots, 31$ based on the non-uniform probability $\frac{1}{2} + \frac{2^{-81}}{9}$ with a bias of $\frac{2^{-81}}{9}$.

Two random 32-bit integers are expected to match in 16 bit positions. Below we show that if one performs a bitwise comparison of the 32-bit elements $\psi_i = ([\psi_i]^{31}, [\psi_i]^{30}, \ldots, [\psi_i]^0)$ and $\psi_j = ([\psi_j]^{31}, [\psi_j]^{30}, \ldots, [\psi_j]^0)$ in HC-128, where $1024\tau + 10 \leq j < i < 1024\tau + 511$, then the expected number of matches between the corresponding bits is more than 16, and to be precise, is approximately $16 + \frac{13}{12} \cdot 2^{-79}$.

Theorem 4.9. *For $1024\tau + 10 \leq j < i < 1024\tau + 511$, the expected number of bits where the two 32-bit integers ψ_i and ψ_j match is approximately $16 + \frac{13}{12} \cdot 2^{-79}$.*

Proof. Let $m_b = 1$, if $[\psi_i]^b = [\psi_j]^b$; otherwise, let $m_b = 0$, $0 \leq b \leq 31$. Hence, the total number of matches is given by $M = \sum_{b=0}^{31} m_b$. From Theorem 4.8, we have $Prob(m_b = 1) = r_b$. Hence, $E(m_b) = r_b$ and by linearity of expectation, $E(M) = \sum_{b=0}^{31} E(m_b) = \sum_{b=0}^{31} r_b$. Substituting the values of r_b's from Theorem 4.8, we get $E(M) \approx 16 + \frac{13}{3} \cdot 2^{-81}$. □

Thus our contributions in this section constitute of

- identifying 30 many slightly weaker distinguishers other than the one described in [128] at bit level (Theorem 4.8);
- further, all these distinguishers can be taken together to mount a word level distinguisher for HC-128 (Theorem 4.9) of the same order of complexity as the bit level distinguishers.

These distinguishers have not been identified in [128].

4.4 A new distinguisher

In this section, we present a new distinguisher. Once again, we refer to the visualization of the array P as explained in Table 4.2. For the analysis in this section, we introduce few notations. The keystream generation occurs in blocks of 512 words. If $B_1, B_2, B_3, B_4, \ldots$ denote successive blocks, the array P is updated in blocks $B_1, B_3, B_5, \ldots$ and the array Q is updated in blocks $B_2, B_4, B_6, \ldots$, and so on. Without loss of generality, consider a block B_{2t+1} (for some fixed t) of 512 keystream word generation in which the array P is updated. More specifically, the symbol P without any subscript or superscript denote the *updated* array in the current block B_{2t+1}. Let P_{-1} and P_{-2} denote the *updated* arrays in blocks B_{2t-1} and B_{2t-3} respectively.

From Equation (4.1) (see Section 4.2), using the equality of XOR and sum for the least significant bit, we can write, for $10 \leq i \bmod 512 < 511$,

$$\Big[P[i]\Big]^0 = \Big[P_{-1}[i]\Big]^0 \oplus \Big[P[i \boxminus 3]\Big]^{10} \oplus \Big[P_{-1}[i \boxminus 511]\Big]^{23} \oplus \Big[P[i \boxminus 10]\Big]^8. \quad (4.4)$$

Similarly, we have

$$\Big[P_{-1}[i]\Big]^0 = \Big[P_{-2}[i]\Big]^0 \oplus \Big[P_{-1}[i \boxminus 3]\Big]^{10} \oplus \Big[P_{-2}[i \boxminus 511]\Big]^{23} \oplus \Big[P_{-1}[i \boxminus 10]\Big]^8. \quad (4.5)$$

XOR-ing both sides of Equation (4.4) and Equation (4.5) and rearranging terms, we get, for $10 \leq i < 511$,

$$\begin{aligned} \Big[P[i]\Big]^0 \oplus \Big[P_{-2}[i]\Big]^0 \;=\; & \Big(\Big[P[i \boxminus 3]\Big]^{10} \oplus \Big[P_{-1}[i \boxminus 3]\Big]^{10}\Big) \\ & \oplus \Big(\Big[P[i \boxminus 10]\Big]^8 \oplus \Big[P_{-1}[i \boxminus 10]\Big]^8\Big) \\ & \oplus \Big(\Big[P_{-1}[i \boxminus 511]\Big]^{23} \oplus \Big[P_{-2}[i \boxminus 511]\Big]^{23}\Big). \end{aligned} \quad (4.6)$$

As in Section 4.2, let the primed array name denote the updated array, when the two '+' operators are replaced by '$\oplus$' in the update rule (Equation (4.1)). Thus,

$$\Big[P'[i \boxminus 3]\Big]^{10} = \Big[P_{-1}[i \boxminus 3]\Big]^{10} \oplus \Big[P[i \boxminus 6]\Big]^{20} \oplus \Big[P_{-1}[i \boxminus 514]\Big]^{33} \oplus \Big[P[i \boxminus 13]\Big]^{18}, \quad (4.7)$$

$$\Big[P'[i \boxminus 10]\Big]^{8} = \Big[P_{-1}[i \boxminus 10]\Big]^{8} \oplus \Big[P[i \boxminus 13]\Big]^{18} \oplus \Big[P_{-1}[i \boxminus 521]\Big]^{31} \oplus \Big[P[i \boxminus 20]\Big]^{16}, \quad (4.8)$$

and $\Big[P'_{-1}[i \boxminus 511]\Big]^{23}$

$$= \Big[P_{-2}[i \boxminus 511]\Big]^{23} \oplus \Big[P_{-1}[i \boxminus 514]\Big]^{33} \oplus \Big[P_{-2}[i \boxminus 1022]\Big]^{46} \oplus \Big[P_{-1}[i \boxminus 521]\Big]^{31}. \quad (4.9)$$

Equations (4.7),(4.8) and (4.9) hold for the ranges $13 \leq i < 514$, $20 \leq i < 521$ and $9 \leq i < 510$ respectively.

XOR-ing both sides of Equations (4.6),(4.7), (4.8) and (4.9), and rearranging terms, we get, for $20 \leq i < 510$,

$$\begin{aligned}
&\Big[P[i]\Big]^{0} \oplus \Big[P_{-2}[i]\Big]^{0} \oplus \Big[P[i \boxminus 6]\Big]^{20} \oplus \Big[P[i \boxminus 20]\Big]^{16} \oplus \Big[P_{-2}[i \boxminus 510]\Big]^{14} \\
&= \Big(\Big[P[i \boxminus 3]\Big]^{10} \oplus \Big[P'[i \boxminus 3]\Big]^{10}\Big) \\
&\oplus \Big(\Big[P[i \boxminus 10]\Big]^{8} \oplus \Big[P'[i \boxminus 10]\Big]^{8}\Big) \\
&\oplus \Big(\Big[P_{-1}[i \boxminus 511]\Big]^{23} \oplus \Big[P'_{-1}[i \boxminus 511]\Big]^{23}\Big).
\end{aligned} \quad (4.10)$$

Lemma 4.10. *For* $1024\tau + 20 \leq i < 1024\tau + 510$,

$$Prob\Big(\Big[P[i]\Big]^{0} \oplus \Big[P_{-2}[i]\Big]^{0} \oplus \Big[P[i \boxminus 6]\Big]^{20} \oplus \Big[P[i \boxminus 20]\Big]^{16} \oplus \Big[P_{-2}[i \boxminus 510]\Big]^{14} = 0\Big) \approx \frac{13}{27}.$$

Proof. From Equation (4.10), we can write,

$$Prob\Big(\Big[P[i]\Big]^0 \oplus \Big[P_{-2}[i]\Big]^0 \oplus \Big[P[i \boxminus 6]\Big]^{20} \oplus \Big[P[i \boxminus 20]\Big]^{16} \oplus \Big[P_{-2}[i \boxminus 510]\Big]^{14} = 0\Big) = Prob(\lambda_1 \oplus \lambda_2 \oplus \lambda_3 = 0)$$

where $\lambda_1 = \Big[P[i \boxminus 3]\Big]^{10} \oplus \Big[P'[i \boxminus 3]\Big]^{10}$, $\lambda_2 = \Big[P[i \boxminus 10]\Big]^8 \oplus \Big[P'[i \boxminus 10]\Big]^8$, $\lambda_3 = \Big[P_{-1}[i \boxminus 511]\Big]^{23} \oplus \Big[P'_{-1}[i \boxminus 511]\Big]^{23}$.

Now, from Lemma 4.3 (see Section 4.2), we get $Prob(\lambda_1 = 0) = Prob(\lambda_2 = 0) = Prob(\lambda_3 = 0) \approx \frac{1}{3}$.

Again, XOR of three terms yield 0, if either all three terms are 0 or exactly one is 0 and the other two are 1's.

Hence, $Prob(\lambda_1 \oplus \lambda_2 \oplus \lambda_3 = 0) \approx (\frac{1}{3})^3 + \binom{3}{1}(\frac{1}{3})(\frac{2}{3})^2 = \frac{13}{27}$. □

Now, we present our new distinguisher. As in Section 4.3, we denote $P[i \boxminus 12]$ at the i-th step by z_i and replace $P[i \bmod 512]$ by $s_i \oplus h_1(z_i)$. Thus, for $1024\tau + 20 \leq i < 1024\tau + 510$, the event
$\Big(\Big[P[i]\Big]^0 \oplus \Big[P_{-2}[i]\Big]^0 \oplus \Big[P[i \boxminus 6]\Big]^{20} \oplus \Big[P[i \boxminus 20]\Big]^{16} \oplus \Big[P_{-2}[i \boxminus 510]\Big]^{14} = 0\Big)$
can be alternatively written as $(\chi_i = \mathcal{H}(\mathcal{Z}_i))$, where

$$\chi_i = s_i^0 \oplus s_{i-2048}^0 \oplus s_{i-6}^{20} \oplus s_{i-20}^{16} \oplus s_{i-2046}^{14} \text{ and}$$

$$\mathcal{H}(\mathcal{Z}_i) = h_1(z_i)^0 \oplus h_1''(z_{i-2048})^0 \oplus h_1(z_{i-6})^{20} \oplus h_1(z_{i-20})^{16} \oplus h_1''(z_{i-2046})^{14}.$$

Here $\mathcal{Z}_i = (z_i, z_{i-2048}, z_{i-6}, z_{i-20}, z_{i-2046})$ is an 80-bit input and $\mathcal{H}(.)$ can be assumed to be a random 80-bit-to-1-bit S-box. Here $h_1(.)$ and $h_1''(.)$ indicate two different functions since they are related to two P arrays at two different blocks and hence act as two different S-boxes.

With this formulation, Lemma 4.10 can be restated as

$$Prob((\chi_i = \mathcal{H}(\mathcal{Z}_i)) \approx \frac{13}{27}. \tag{4.11}$$

Further, from Proposition 4.5, for $1024\tau + 20 \leq j < i < 1024\tau + 510$,

$$Prob\left(\mathcal{H}(\mathcal{Z}_i) = \mathcal{H}(\mathcal{Z}_j)\right) = \frac{1}{2} + 2^{-81}. \tag{4.12}$$

Theorem 4.11. *For* $1024\tau + 20 \leq j < i < 1024\tau + 510$,

$$Prob\left(\chi_i = \chi_j\right) \approx \frac{1}{2} + \frac{1}{3^6 \cdot 2^{81}},$$

where $\chi_u = s_u^0 \oplus s_{u-2048}^0 \oplus s_{u-6}^{20} \oplus s_{u-20}^{16} \oplus s_{u-2046}^{14}$.

Proof. We need to combine the probability expressions of Equations (4.11) and (4.12). For $1024\tau + 20 \leq j < i < 1024\tau + 510$,

$$\begin{aligned}
&Prob\left(\chi_i \oplus \chi_j = \mathcal{H}(\mathcal{Z}_i) \oplus \mathcal{H}(\mathcal{Z}_j)\right) \\
&= Prob\left(\chi_i = \mathcal{H}(\mathcal{Z}_i)\right) \cdot Prob\left(\chi_j = \mathcal{H}(\mathcal{Z}_j)\right) + \\
&\qquad Prob\left(\chi_i = \mathcal{H}(\mathcal{Z}_i) \oplus 1\right) \cdot Prob\left(\chi_j = \mathcal{H}(\mathcal{Z}_j) \oplus 1\right) \\
&\approx \left(\frac{13}{27}\right)^2 + \left(1 - \frac{13}{27}\right)^2 = \frac{365}{3^6}.
\end{aligned}$$

$$\begin{aligned}
&Prob\left(\chi_i = \chi_j\right) \\
&= Prob\left(\chi_i \oplus \chi_j = \mathcal{H}(\mathcal{Z}_i) \oplus \mathcal{H}(\mathcal{Z}_j)\right) \cdot Prob\left(\mathcal{H}(\mathcal{Z}_i) = \mathcal{H}(\mathcal{Z}_j)\right) + \\
&\qquad Prob\left(\chi_i \oplus \chi_j = \mathcal{H}(\mathcal{Z}_i) \oplus \mathcal{H}(\mathcal{Z}_j) \oplus 1\right) \cdot Prob\left(\mathcal{H}(\mathcal{Z}_i) = \mathcal{H}(\mathcal{Z}_j) \oplus 1\right) \\
&\approx \frac{365}{3^6} \cdot \left(\frac{1}{2} + 2^{-81}\right) + \left(1 - \frac{365}{3^6}\right) \cdot \left(\frac{1}{2} - 2^{-81}\right) = \frac{1}{2} + \frac{1}{3^6 \cdot 2^{81}}.
\end{aligned}$$

□

The aforesaid analysis shows that there exist biases in the event $(\chi_i = \chi_j)$, which can be used to mount a distinguishing attack. The probability of the above event can be written as $p_\chi(1 + q_\chi)$, where $p_\chi = \frac{1}{2}$ and $q_\chi = \frac{1}{3^6 2^{80}}$. According to [14, Section 4.1], one would require $\frac{4^2}{p_\chi q_\chi^2} = 3^{12}2^{165}$ pairs of keystream word combinations of the form (χ_i, χ_j) for a success probability 0.9772. Since each block of $512 = 2^9$ keystream words provides approximately $\binom{512}{2} \approx 2^{17}$ many pairs, the required number of keystream words is approximately $3^{12}2^{156} \approx 2^{19}2^{156} = 2^{175}$.

Though our distinguisher is slightly weaker than what mentioned in [128], ours is the only known distinguisher other than the one identified in [128]. The distinguisher presented in Theorem 4.11 may be extended to other bits in a similar line that we have studied in Section 4.3.2 for the distinguisher of [128].

However, the bias would be weaker than that presented in Theorem 4.11 and the extended distinguisher would require more keystream words.

Subsequent to our work, Stankovski et al. [120] combined our bit-level distinguishers to mount word-based distinguisher that required $2^{152.537}$ many 32-bit keystream blocks.

4.5 State leakage in keystream

Whereas the previous sections concentrated on the functions g_1, g_2; here, in a different direction, we study the other two functions h_1, h_2. Without loss of generality, we focus on the keystream block corresponding to the P array, i.e., the block of 512 rounds where P is updated in each round and Q remains constant. As j runs from 0 to 511, we denote the corresponding output $h_1(P[j \boxminus 12]) \oplus P[j]$ by s_j. Here, $h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}]$. The results we present in this section are in terms of the function h_1. The same analysis holds for the function h_2 in the other keystream block.

In [36], it has been observed that $Prob(s_j \oplus s_{j+1} = P[j] \oplus P[j+1]) \approx 2^{-16}$, where s_j, s_{j+1} are two consecutive keystream output words. We study that in more detail in this section and in the process we find a sharper association in Theorem 4.15 which gives twice the above probability.

The following technical result establishes that XOR of two words of P is leaked in the keystream words if the corresponding values of $h_1(.)$ collide.

Lemma 4.12. *For $0 \leq u \neq v \leq 511$, $s_u \oplus s_v = P[u] \oplus P[v]$ if and only if $h_1(P[u \boxminus 12]) = h_1(P[v \boxminus 12])$.*

Proof. We have $s_u = h_1(P[u \boxminus 12]) \oplus P[u]$ and $s_v = h_1(P[v \boxminus 12]) \oplus P[v]$. Thus, $s_u \oplus s_v = \Big(h_1(P[u \boxminus 12]) \oplus h_1(P[v \boxminus 12])\Big) \oplus (P[u] \oplus P[v])$. The term $\Big(h_1(P[u \boxminus 12]) \oplus h_1(P[v \boxminus 12])\Big)$ vanishes if and only if $s_u \oplus s_v = P[u] \oplus P[v]$. □

Now we detail the result related to collision in h_1. Note that the array P from which the input to the function h_1 is selected and the array Q from which the output of h_1 is chosen can be assumed to contain uniformly distributed 32-bit elements. In Lemma 4.13, which is in a more general setting than just

HC-128, we use notations h and U; these may be considered to model h_1 and Q respectively.

Lemma 4.13. *Let $h(x) = U[x^{(0)}] + U[x^{(2)} + 2^m]$ be an n-bit to n-bit mapping, where each entry of the array U is an n-bit number, U contains 2^{m+1} many elements and $x^{(0)}$ and $x^{(2)}$ are two disjoint m-bit segments from the n-bit input x. Suppose x and x' are two n-bit random inputs to h. Assume that the entries of U are distributed uniformly at random. Then for any collection of t distinct bit positions $\{b_1, b_2, \ldots, b_t\} \subseteq \{0, 1, \ldots, n-1\}$, $0 \leq t \leq n$,*

$$Prob\left(\bigwedge_{l=0}^{t}\left([h(x)]^{b_l} = [h(x')]^{b_l}\right)\right) = \gamma_{m,t}$$

where $\gamma_{m,t} = 2^{-2m} + (1 - 2^{-2m})2^{-t}$.

Proof. Each value $v \in [0, 2^n - 1]$ can be expressed as the modulo 2^n sum of two integers $a, b \in [0, 2^n - 1]$ in exactly 2^n ways, since for each choice of $a \in [0, 2^n - 1]$, the choice of b is fixed as $b = (v - a) \bmod 2^n$. It is given that each $U[\alpha]$ is uniformly distributed over $[0, 2^n - 1]$. Thus, for uniformly random $\alpha, \beta \in [0, 2^{m+1} - 1]$, the sum $(U[\alpha] + U[\beta]) \bmod 2^n$ is also uniformly distributed and so is any collection of t bits of the sum.

Now, $h(x) = U[x^{(0)}] + U[x^{(2)} + 2^m]$ and $h(x') = U[x'^{(0)}] + U[x'^{(2)} + 2^m]$ can be equal in bit positions $b_1, b_2, \ldots, b_t$ in the following two ways.

1. $x^{(0)} = x'^{(0)}$ and $x^{(2)} = x'^{(2)}$. This happens with probability $2^{-m} \cdot 2^{-m}$.
2. The event "$x^{(0)} = x'^{(0)}$ and $x^{(2)} = x'^{(2)}$" does not happen and still $\left(U[x^{(0)}] + U[x^{(2)} + 2^m]\right)$ and $\left(U[x'^{(0)}] + U[x'^{(2)} + 2^m]\right)$ match in the t bits due to random association. This happens with probability $(1 - 2^{-2m}) \cdot 2^{-t}$.

Adding the two components, we get the result. □

Note that $\gamma_{m,t}$ depends only on m, t and not on n, as is expected from the particular form of $h_1(.)$ that uses only $2m$ bits from its n-bit random input. For HC-128, we have $n = 32$ and $m = 8$. For the equality of the whole output word of $h_1(.)$, we need to set $t = n = 32$. Thus, $Prob\left(h_1(x) = h_1(x')\right) = \gamma_{8,32} = 2^{-16} + 2^{-32} - 2^{-48} \approx 0.0000152590$, which is slightly greater than 2^{-16}. We like

to point out that if one checks the equality of two n-bit random integers, then the probability of that event is 2^{-n} only, which is as low as 2^{-32} for $n = 32$.

Next we formalize the result given in [36].

Theorem 4.14. *In HC-128, consider a block of 512 many keystream words corresponding to the array P. For $0 \leq u \neq v \leq 511$,*

$$Prob\left((s_u \oplus s_v) = (P[u] \oplus P[v])\right) = \gamma_{8,32} > 2^{-16}.$$

Proof. The result follows from Lemma 4.12 and Lemma 4.13. □

Now, we present a sharper result which gives twice the probability of the observation in [36].

Theorem 4.15. *In HC-128, consider a block of 512 many keystream words corresponding to the array P. For any u, v, $0 \leq u \neq v < 500$, if $\left((s_u^{(0)} = s_v^{(0)}) \;\&\; (s_u^{(2)} = s_v^{(2)})\right)$, then*

$$Prob\left((s_{u+12} \oplus s_{v+12}) = (P[u+12] \oplus P[v+12])\right) \approx \frac{1}{2^{15}}.$$

Proof. From Lemma 4.12, $s_u^{(b)} \oplus s_v^{(b)} = P[u]^{(b)} \oplus P[v]^{(b)}$ if and only if

$$h_1(P[u \boxminus 12])^{(b)} = h_1(P[v \boxminus 12])^{(b)},$$

for $b = 0, 1, 2, 3$. Given that $s_u^{(0)} = s_v^{(0)}$ and $s_u^{(2)} = s_v^{(2)}$, we have $P[u]^{(0)} = P[v]^{(0)}$ and $P[u]^{(2)} = P[v]^{(2)}$ if and only if

$$h_1(P[u \boxminus 12])^{(0)} = h_1(P[v \boxminus 12])^{(0)} \text{ and } h_1(P[u \boxminus 12])^{(2)} = h_1(P[v \boxminus 12])^{(2)}.$$

Thus, using Lemma 4.13,

$$\begin{aligned} &Prob\left(P[u]^{(0)} = P[v]^{(0)} \;\&\; P[u]^{(2)} = P[v]^{(2)} \mid s_u^{(0)} = s_v^{(0)} \;\&\; s_u^{(2)} = s_v^{(2)}\right) \\ &= Prob\left(h_1(P[u \boxminus 12])^{(0)} = h_1(P[v \boxminus 12])^{(0)} \;\&\; \right. \\ &\qquad \left. h_1(P[u \boxminus 12])^{(2)} = h_1(P[v \boxminus 12])^{(2)}\right) = \gamma_{8,16} \approx \frac{1}{2^{15}}, \end{aligned}$$

By definition, $h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}]$. So the equalities $P[u]^{(0)} =$

$P[v]^{(0)}$ and $P[u]^{(2)} = P[v]^{(2)}$ give $h_1(P[u]) = h_1(P[v])$ and this in turn gives $s_{u+12} \oplus s_{v+12} = P[u+12] \oplus P[v+12]$ by Lemma 4.12. □

For random keystream, the event $(s_u^{(0)} = s_v^{(0)}$ & $s_u^{(2)} = s_v^{(2)})$ occurs with probability 2^{-16}. If the probability of this event in HC-128 is away from 2^{-16}, then we would immediately have a distinguisher. Our experiments did not reveal any observable bias for this event. However, it would be interesting to investigate if there exists a bias of very small order for this or a similar event in HC-128.

The Glimpse Main Theorem [56, 82] is an important result on the weakness of RC4 stream cipher. It states that at any round, $Prob(S[j] = i - z) = Prob(S[i] = j - z) \approx 2^{-7}$, where S is the internal state of RC4, i and j are the deterministic and the pseudo-random indices respectively and z is the keystream output byte. This result quantifies the leakage of state information into the keystream of RC4. Note that the leakage probability is twice the random association 2^{-8}. Our Theorem 4.15 is a Glimpse-like theorem on HC-128 that shows the leakage of state information into the keystream with a probability $\approx 2^{-15}$ which is much more than 2^{-31} (two times the random association 2^{-32}), and is in fact two times the square-root of the random association. However, in RC4, the Glimpse theorem turns the key-state correlations into key-keystream correlations [65, 79] that lead to practical attacks. On the other hand, in case of HC-128, no key-state correlations have been discovered so far. So the state leakage in HC-128 keystream, as of now, does not pose any threat to the use of the cipher.

4.6 Conclusion

In this chapter, we study the linear approximation of the feedback functions g_1, g_2 of HC-128. Using this result, we extend the least significant bitwise distinguisher proposed by the designer himself to all the bits (except one) of the 32-bit keystream output word. We also present a new distinguisher by properly choosing pairs of five keystream word combinations from three consecutive blocks corresponding to only P (or only Q) array. Further, we have studied the idea of Dunkelman in detail towards the secret state information leakage

in keystream output words. Though our results do not have any immediate threat to the applicability of HC-128, these provide further insight into the cryptanalysis of HC-128.

THIS PAGE INTENTIONALLY LEFT BLANK

Part II

Side Channel and Implementation Issues

Chapter 5

Side Channel Attacks and Impact on HC series of Stream Ciphers

5.1 Motivation

Side channel attacks are extremely implementation specific. An attack is tailor-made for a specific cipher algorithm implemented in a specific model. In a commercially popular device with large user-base, the immediate reaction to such an attack is to change the underlying cryptographic algorithm as quickly as possible. In such a case, a variant of the original algorithm forms a preferential choice, because its structural similarity with the original cipher makes it implementation-friendly on existing hardware. For example, in case of Wi-Fi networks, when the RC4 implementation in WEP protocol suffered several attacks [82,84,122–124], a natural choice in the subsequent WPA protocol was a variant of RC4 implementation (with different key and IV size and different form of key and IV mixing), before the longterm change-over to the block cipher AES in WPA2. A natural question is: *what is the effect of a side channel technique on a variant of the cipher algorithm implemented in a similar model*? The motivation for such an investigation is to study the feasibility of using a cipher variant as a mode of recovering from a successful side channel attack. The following works studied side channel vulnerability of HC series of stream ciphers, viz., HC-128 and HC-256.

- a differential fault attack on HC-128 [64],
- a cache timing analysis analysis on HC-256 [130].

For both the above works, extending the attack on one HC variant to the other is not straight-forward. In [64], there is no mention about the extendibility of the attack to HC-256. On the other hand, the author of [130] mentions the following point in a subsequent work [131] that studied the cache timing analysis of all the eSTREAM finalists.

> HC-128 ... has a slightly smaller inner state ... and surprisingly big changes of the internal workings. Most state update equations are modified, and this has a profound impact on the above cache timing attack. It turns out that the attack can not be transferred to HC-128 in a straightforward way. Thus, further analysis of HC-128 is necessary to determine its resistance against cache timing attacks.

It is important to note that one of the major differences in the structure of HC-128 and HC-256 is that in the former the two state tables are updated independently, whereas in the latter they are inter-dependent. This makes extending the differential fault analysis of HC-128 [64] to HC-256 a challenging task.

In this chapter, the HC-128 fault attack and the HC-256 cache analysis are extended onto the HC-256 and HC-128 ciphers respectively under similar models. The techniques applied on one variant is not trivially translatable to the other and the issue was left open until the work presented in this chapter. A technique has been proposed to recover half the state of HC-128 using cache analysis, which can be cascaded with the differential attack towards a full state recovery and hence key recovery. Similarly, the state leakage of HC-256 is analyzed under differential fault attack model to achieve partial state recovery.

5.1.1 Layout of the chapter

A brief description of cache-analysis and differential fault attack is presented in the first section.

Next, the effect of the attacks on one HC variant to the other is analyzed. Specifically, analysis has been carried out on how much state information can be leaked when the HC-128 variant is chosen against the cache analysis attack on HC-256, and the latter is chosen against the differential fault attack of the former. For the study the same implementations that are considered in [64,130] have been used.

In Section 5.3, the description of the cache analysis model as applied to HC-128 along with the inferences have been presented. This issue had been left as an open problem in [130]. In Section 5.4 we analyze the fault attack on HC-256. We conclude the chapter and discuss possible future works in Section 5.5.

5.2 Cache and fault attack

We first present an overview of Cache analysis and fault attacks that would be used in the chapter.

5.2.1 Cache analysis attack

Cache analysis is a side channel cryptanalysis technique that has been introduced independently by Bernstein [15] and Osvik et al. [94], primarily for the AES block cipher. We give a simple description of the cache analysis adapted from [130]. *Cache* is a temporary storage area that is closer to the CPU compared to the RAM and is used to replicate frequently accessed data for enabling faster access. Data once stored in the cache, can be further used by accessing the cached copy rather than by re-fetching from the RAM. If the CPU finds the data it needs in the cache, a *cache hit* is said to occur. Otherwise a *cache miss* occurs and the cache must be immediately loaded with the requisite data from the RAM. Cache management in modern processors divides the available cache memory into blocks of b bytes. For a given block, all the b bytes must be loaded together. To ensure consistency between the data in the RAM and the cache, a record of cache blocks being loaded into the cache is maintained. This record serves as the initial raw-data for commencing the cache analysis. The adversary first fills the entire cache with his own data. Next, during normal

computation, user's data replaces some data of the adversary. When adversary tries to reload his own data from the cache, if it takes longer time, then he knows the address of the user's data from the cache record.

This method can be considered as a *cache access attack*. However, since the time to load the data plays a crucial role in leaking the address of the data, we follow the same nomenclature as in [130] and call this as *cache timing attack*.

5.2.2 Fault attack

Fault Attack [21] is the kind of SCA where computational faults are induced in the device to extract the information. This involves two steps process. The first step is fault injection and the second step includes exploitation of the induced fault. Faults can be inserted in the system by creating abnormal conditions like high or low voltage, temperature variations, clock cycle tampers and radiations. The fault exploitation involves analysis of generated output of such systems in erroneous environment. The goal may be to corrupt the value of an internal state register or memory location or to make a small change in the execution flow, such as skipping an instruction or changing a memory address etc. The corresponding change in the cipher output obtained are used to extrapolate the internal state.

A differential fault attack (DFA) against a stream cipher resets the cipher with the same key, but injecting different faults. The resulting keystreams have small differences and the attack exploits these differentials. It is most widely used form of Fault Analysis, as the changes and results of the erroneous device can be significantly observed.

Certain measures against fault attack include computing a part or whole of a cryptographic algorithm twice or more, however, it degrades the performance. Apart from that, another measure exists wherein the integrity of a signature is checked after computation.

Though in general, there are certain other countermeasures available, yet owing to the serious amount of threat posed by DFA and the parameters it usually takes into account for analysis, it is required to develop stronger counter measures for ensuring better security.

5.3 Cache analysis of HC-128

In the attack model, we assume that the adversary is running a special process in the CPU in which the cipher is executing. The adversary's process executes calls so as to fill the entire cache with his own data, causing the HC-128 cached data to be evicted. When the HC-128 process gains control of the CPU it must reload data into the cache pertinent to current instruction. The cache block size b in present day processors range from 16 bytes to 128 bytes. The arrays P and Q have 512 many word entries. As four bytes make a word, $b/4$ array elements will be mapped to a single cache block size. If the cache block is aligned with the index of the array, we will have the elements from indices 0 to $(b/4)-1$ in the first cache block and the elements from indices $(b/4)$ to $(b/2)-1$ in the second block and so on, till the elements from indices $511-(b/4)$ to 511 in the final cache block.

5.3.1 Bits obtainable from cache information

The keystream words are generated using both the arrays P and Q, each consisting of 512 many words. However, the updates of P and Q arrays are independent. For 512 many iterations, the array P is updated with the older values from P itself and during this time the array Q is accessed but not updated. For the next 512 many iterations the array Q is updated with the older values of Q and during this time the array P is accessed for keyword generation but not updated. This access and update pattern continues alternatively. At this phase, the key-stream is generated using two functions h_1, h_2 of similar kind. The equations used are as follows.

$$\begin{aligned} s_j &= h_1(P[j \boxminus 12]) \oplus P[j] \quad \text{(keystream when } P \text{ is updated)}, \\ s_j &= h_2(Q[j \boxminus 12]) \oplus Q[j] \quad \text{(keystream when } Q \text{ is updated)}. \end{aligned}$$

Every time the h_1 (or h_2) function is called, it uses bytes 0 and 2 of $P[j \boxminus 12]$ (or $Q[j \boxminus 12]$) to access the elements of Q (or P) array. Since a cache with block size b holds $b/4$ array elements, the cache blocks can be numbered using the first $8 - \log_2(b/4)$ bits. For each call, two elements of the array are accessed. This gives $16 - 2\log_2(b/4)$ bits of each element of the arrays P and Q, corresponding

Cache Block Size b:	16	32	64	128
No. of bits obtained:	12	10	8	6

Table 5.1: Cache block size vs. number of bits learned.

to the cache fill which is, sought at the time h_1 or h_2 is called. Thus, the number of bits obtained depends on b and is given in Table 5.1.

Using the above, we can form an array with the known elements of P and Q. Let P_k and Q_k denote the array corresponding to the k-th iteration of the update. On completion of the KSA, we will get the first array denoted by P_0 and Q_0. On generation of the first 512 keystream words, we get another array denoted by P_1 and for the next 512 words we get the Q_1 array, and so on.

5.3.2 Constructing bytes 0 and 2 of each array element

Since h_1 and h_2 are similar, without any loss of generality we present the case for P_1, and this is valid for Q_1 also. For $0 \leq j \leq 500$, we can rewrite the keystream generation equation $s_j = h_1(P[j \boxminus 12]) \oplus P[j]$ as follows.

$$Q[u] + Q[v] = s(j + 12) \oplus P_1[j + 12], \tag{5.1}$$

where $u = P[j]^{(0)}$ and $v = P[j]^{(2)}$.

The keystream s is known. For a cache block size of 32 bytes, the first 5 most significant bits of each of u and v are known. Thus, $u \gg 3$ and $32 + (v \gg 3)$ denote the cache blocks loaded for computing h_1. Within these blocks (in Q_0), we exhaustively search the elements that would give a sum identical to the right hand side of Equation (5.1).

These additions are across two chunks of 5 bits and so additional carry bits need to be accounted. For each element $P[j]$, there are $(b/4).(b/4) = b^2/16$ calculations. Finding a correct single match gives the unknown bits of bytes 0 and 2 of P_1. Finding more than one match means that we have a set of values out of which one is the correct candidate. Considering the HC-128 keystream is uniformly random, the frequency of such ties would be very low.

The procedure is repeated for all elements of P and Q array.

5.3.3 Finding the remaining sixteen bits for each element

Since bytes 1 and 3 of the array elements are never used in h_1 (or h_2), obtaining any information about these bits is not possible by the above cache analysis. If one considers the g_1 (or g_2) function and propagates the known 10 bits across many updates, one can guess some bits considering the carry propagation. However, we could not get much information from such analysis. In the cache attack of HC-256 [130], the advantage was that all the four bytes of the array elements are used inside the h_1 (or h_2) functions, giving leakage of all the 32 bits eventually.

In order to find the remaining 16 bits for each element in case of HC-128, we propose to use the techniques of differential fault analysis [64] in combination with the cache analysis suggested here. According to [64], they need to solve a set of 32 systems of linear equations over $\mathbb{Z}_2$ in 1024 variables. If one performs the cache analysis proposed in this paper first, immediately the problem reduces to 16 systems of linear equations over $\mathbb{Z}_2$ in 1024 variables.

According to [76], the key schedule of HC-128 is reversible and hence once the full state is recovered, the secret key can be easily found.

5.4 Fault attack on HC-256

The model that we use for inserting faults in HC-256 is similar to that used for the attack on HC-128 [64] with a slightly stronger assumption as regards to the fact that one *needs to know the location of the fault*. Such an assumption is not entirely impractical. In [116,117], the authors discuss how to flip precise bits in SRAM and EEPROM, or change the state of any individual CMOS transistor on a chip. We only require a change in at least one of the 32 bits of a word at a specific location. Like [64], we also do not need to know the value of the fault.

In practice, we know that the updates of P and Q occur alternatively.

The fault is assumed to be inserted into the array that is not being updated. Any variation in the keystream would imply that a faulty element has been accessed. The block of 1024 keystream generation in which P (or Q) is updated is referred as the P (or Q)-block and the keystream is denoted by s_P (or s_Q). Let the primed variables s'_P and s'_Q denote the 'faulty' keystream, i.e., the keystream generated after fault injection. We denote the location (index) of the fault as f.

In P-block, the keystream is generated as

$$\begin{aligned} s_{P,j} &= h_1(P[j \boxminus 12]) \oplus P[j] \\ &= \Big(Q\big[(P[j \boxminus 12])^{(0)}\big] + Q\big[256 + (P[j \boxminus 12])^{(1)}\big] + Q\big[512 + \\ &\quad (P[j \boxminus 12])^{(2)}\big] + Q\big[768 + (P[j \boxminus 12])^{(3)}\big]\Big) \oplus P[j]. \end{aligned}$$

Suppose we inject a fault at $Q[f]$ before $P[0]$ is updated in P-block. We rerun the key generation algorithm 1024 times to generate 1024 *faulty* keystream words corresponding to the current P-block.

We compare $s_{P,j}$ with $s'_{P,j}$ for $j = 0, \ldots, 511$. Whenever $s_{P,j} \neq s'_{P,j}$, we know that the faulty keystream has been accessed.

The noticeable faults observed in the keystream are either due to a faulty value of Q entering the h_1 function or a faulty value of Q entering the update function of P.

Definition 5.1. When a faulty Q (or P) array element enters in h_1 (or h_2), we call this an ***opportune*** event.

Definition 5.2. When a faulty Q (or P) array element is referred inside g_1 (or g_2), we all this a ***traverse*** event.

These cases are analyzed below one by one.

5.4.1 Faulty Q entering in computation of h_1

Suppose an *opportune* event occurs and f is the location of the fault, i.e., $Q[f]$ is the faulty value accessed inside h_1. Here the location of a faulty Q element gives information of a byte of P. The keystream index j where $s_{P,j}$ and $s'_{P,j}$

differs, refers to
byte 0 of $P[j \boxminus 12]$ if $0 \leq f \leq 255$,
byte 1 of $P[j \boxminus 12]$ if $256 \leq f \leq 511$,
byte 2 of $P[j \boxminus 12]$ if $512 \leq f \leq 767$ or
byte 3 of $P[j \boxminus 12]$ if $768 \leq f \leq 1023$.

Since all the four bytes of P are used inside h_1 (as indices to lookup into array Q), one could retrieve all the bytes of P by injecting faults at all the 1024 entries of Q in turn, if the faulty value never entered in the update of P. But in practice, the faulty value enters in the update of P and therefore the keystream indices where $s_{P,j}$ and $s'_{P,j}$ differs do not correspond to distinct bytes of the words of P. Assuming the P and the Q arrays to be uniformly random, we can theoretically estimate how many words of the P array would actually be revealed for the proposed attack model.

Theorem 5.3. *The expected number of bytes of $P[i]$ leaked through h_1 function is given by $4(\frac{1023}{1024})^{i+1}$, $0 \leq i \leq 1023$.*

Proof. Before $P[i]$ is used in the keystream generation, all the $i+1$ elements from $P[0]$ to $P[i]$ are updated using the function g. The faulty $Q[f]$ enters in each of these updates with probability $1/1024$, assuming the array elements to be uniformly random. Thus, the probability that it does not enter in any particular one of these updates is $1 - \frac{1}{1024} = \frac{1023}{1024}$. Assuming the events corresponding to $Q[f]$ entering into the updates of different rounds to be independent, the probability that it does not enter in any of the $i+1$ updates is given by $\alpha_i = (\frac{1023}{1024})^{i+1}$. For $0 \leq b \leq 3$, let $X_{i,b} = 1$, if the byte b of $P[i]$ is revealed successfully; otherwise $X_{i,b} = 0$. The total number of bytes of $P[i]$ revealed is given by $Y_i = \sum_{b=0}^{3} X_{i,b}$. The expectation of $X_{i,b}$ is given by $E[X_{i,b}] = Prob(X_{i,b} = 1) = \alpha_i$, for any b, $0 \leq b \leq 3$. By linearity of expectation, $E[Y_i] = \sum_{b=0}^{3} E[X_{i,b}] = 4\alpha_i$. □

In Fig. 5.1, we compare the theoretical estimate with the empirical values of the number of bytes of $P[i]$ leaked from the h_1 function, $0 \leq i \leq 1023$, when each element of Q is faulted once. We see that the plots are almost identical. The empirical values were obtained by averaging over 1 million iterations, and each time HC-256 was run with a new randomly generated secret key.

An immediate consequence of Theorem 5.3 is Corollary 5.4, that gives the theoretical estimate of the number of words of the array P that are actually leaked.

Corollary 5.4. *The expected number of words of the array P leaked through h_1 function is 647.*

Proof. Refer to Y_i defined in the proof of Theorem 5.3. Total number of bytes revealed is given by $\sum_{i=0}^{1023} Y_i$, whose expected value is given by $4\sum_{i=0}^{1023}(\frac{1023}{1024})^{i+1}$. Thus, the expected number of words revealed is given by $\sum_{i=0}^{1023}(\frac{1023}{1024})^{i+1} = 646.84 \approx 647$. □

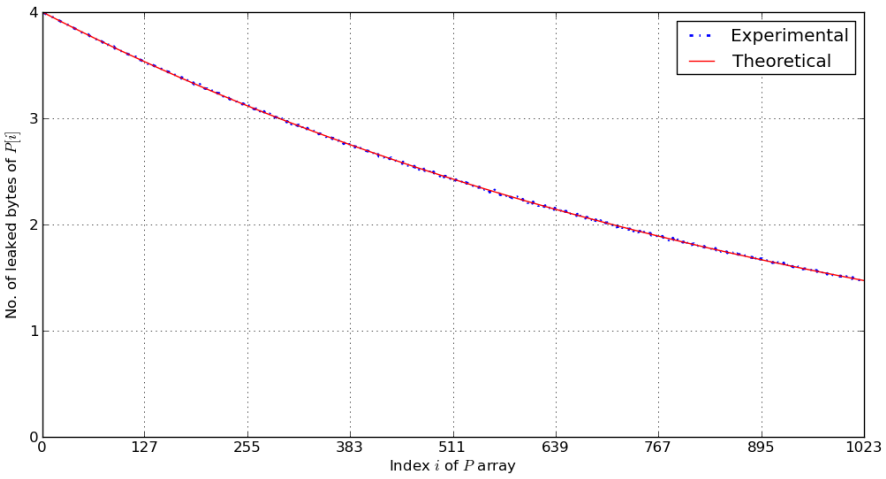

Figure 5.1: Number of bytes of P array elements leaked from h_1 function

5.4.2 Faulty Q entering in update of P

Since the update of P involves Q (unlike HC-128), faulty $Q[f]$ is eventually referred inside g_1. When such a *traverse* event happens, it does not yield any more information within the particular update. However, this case assists in finding elements of the previous updates as follows. Recall the update of P.

$$P[j] \quad = \quad P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023]),$$

where $g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024]$.

So whenever we observe a mismatch between $s_{P,j}$ and $s'_{P,j}$, that may be due to the fact that $P[j \boxminus 3] \oplus P[j \boxminus 1023] \bmod 1024$ refers to the faulty element of Q.

Since the index of the faulty Q element is known, if any one of the elements $P[j \boxminus 3]$ and $P[j \boxminus 1023]$ is known, the other element can be easily computed. Except for $P[0]$, all the elements of the form $P[j \boxminus 1023]$ are from the previously updated P array. Similarly, all the elements of the form $P[j \boxminus 3]$ refer to the updated values of P array, except for $P[0], P[1]$ and $P[2]$.

While keeping track of these indices, the elements found in subsequent updates can be used for finding elements of the previous updates. We find that when we propagate the knowledge from the second update into the first update, the probability increases. Similar trend continues as one increases the number of updates. Our experimental results reveal that after faulting four successive updates of P and Q, the first ten bits of approximately 85% elements of P and Q pertaining to first update can be obtained. Fig. 5.2 shows how the values obtained from the first, the second and the fifth updates help in finding values in the current update. Table 5.2 gives the numerical data for all the five updates. All the values were obtained by averaging over 10000 simulations with randomly generated secret keys.

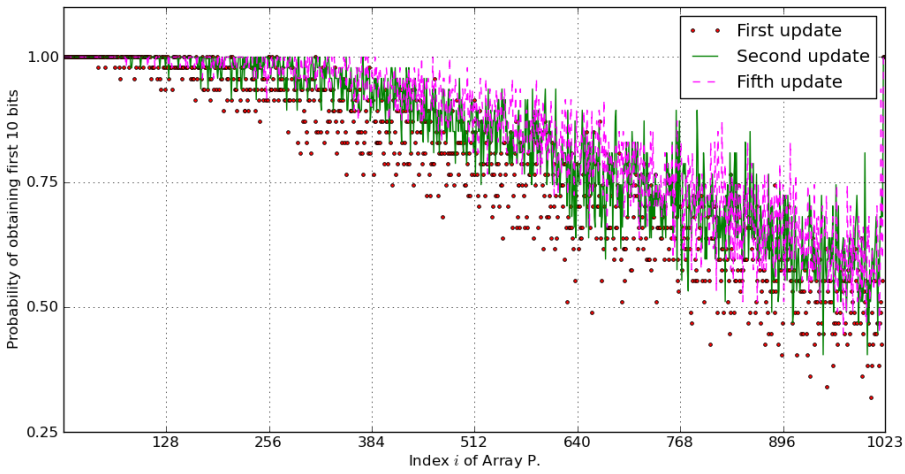

Figure 5.2: Probability of finding the first ten bits of P array elements from several updates

Thus, we are able to get approx. 877 values of the first ten bits, fourth

Type	No. of updates	#words (first 10 bits leaked)
only h fault	1	646.8
both h and g fault	1	805.1
both h and g fault	2	860.0
both h and g fault	3	871.0
both h and g fault	4	875.0
both h and g fault	5	877.6

Table 5.2: Number of words with first ten bits leaked vs. number of subsequent updates incorporated.

array update onwards. Thus, a total $647 \times 22 + 8776$ bits ≈ 719 words for each array are known.

5.4.3 Differentiating the two cases and obtaining additional elements

Experiments show that if the *opportune* event occurs, then the keystream differences ($s_{P,j}$ vs. $s'_{P,j}$) occur at random indices (j), whereas if the traverse event occurs then the keystream differences follow a sequential pattern. This is because a fault in keystream position s_i due to faulty $P[i]$, which in turn is caused via an update involving the faulty $Q[f]$, would result in its being used in the update function of $P[i+3]$, thereby altering the value of s_{i+3}. This in turn creates a fault in position s_{i+6} by virtue of the faulty value being used in the update of $P[i+6]$.

Since the first ten bits of both P and Q function are known, by propagating these values across many updates of P array, one can endeavor to find the missing bits by guess and determine strategy. The analysis is tedious and non-trivial and at this stage we leave it as an open problem. It would be interesting to look at the combinatorial aspects of this guess and determine attack.

5.5 Conclusion

We showed how the cache analysis of HC-256 [130] can be extended to HC-128 and how the differential fault analysis of HC-128 [64] can be extended to

HC-256. The first attack leads to half state recovery of HC-128 and when combined with the differential fault analysis can lead to the full state recovery and key recovery of HC-128. With the second attack, we have been able to perform partial state recovery of HC-256. Two interesting future works could be to study the feasibility of using the cache analysis alone to achieve full state recovery of HC-128 and that of mounting the differential fault attack alone to achieve full state recovery of HC-256.

Our findings show that the side channel vulnerability for a particular implementation of a cipher may percolate to its variants also under similar conditions, albeit in a different degree. This vulnerability is still exploitable through refinement of the attack vectors. So, while selecting a cipher variant to thwart side channel vulnerabilities, extra caution must be exercised.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Implementing HC-128 in Hardware

In the recent years, VLSI technology has grown in a fast pace as predicted by Moore's law. There have been considerable advances in the platforms available for implementation, since the HC-128 cipher was designed eight years ago. In this chapter we present a comprehensive design study of implementing HC-128 on contemporary architectures to achieve high throughput.

6.1 Motivation and contributions

In the present scenario, embedded processors have become an emerging candidates to implement ciphers. After a quick review of the implementation of HC-128 on general purpose CPU, we study the implementation in several embedded processors and on a state-of-the-art customizable processor in Section 6.3. Implementation of HC-128 on modern embedded processors is also an important area of study. Quite naturally, the speed achieved in this situation may not be very enthusiastic compared to general purpose processors. However, once the cipher is widely deployed in commercial domain, one may like to use it on low end hand-held devices. Work in this direction has been reported earlier for embedded micro-controllers [87] and wireless sensor networks. Once HC-128 becomes more popular, one may try to achieve higher speed for the key-stream generation. Given the overheads of general purpose processors, it is clear that one can always obtain further speed-up using application specific hardware accelerator and such a design of HC-128 may efficiently be used in a

cryptographic co-processor.

Next, in Section 6.4, we present detailed ASIC implementation of HC-128 that can be included as a module in a cryptographic co-processor. To boost the performance, in Section 6.5, we propose novel parallelization strategies both in terms of key/IV set-up and keystream generation. These efforts yield more than 3 times throughput improvement compared to the fastest software speed reported in eStream. Our strategies are also useful to completely hide the initialization latency by employing a multi-session HC-128 execution. Our study serves two major purposes towards the deployment of HC-128 stream cipher. First, it reports several novel and generic performance improvement strategies for HC-128. Second, several design points are explored ranging from customizable co-processor to fully dedicated hardware accelerators. Though HC-128 is a software cipher, we believe that instead of software-only benchmarking, a complete design study is crucial to identify advantages/bottlenecks of HC-128.

6.2 HC-128 on general purpose CPU's

HC-128 is primarily designed as a software stream cipher aiming for sequential execution on general purpose processors. However, it contains several coarse-grained and fine-grained parallelism opportunities. Some of these parallelism has been exploited in order to obtain fast software implementation on general purpose processor cores. The software performances reported, for stream encryption, in eSTREAM project web-page [37] are summarized in Table 6.1 and Table 6.2. For the initialization speed, the total duration of key and IV setup are reported.

The parallelism options reported in the available literature [128] mainly relied on loop-unrolling. The optimized implementation, as presented in [128], unrolled 16 consecutive iterations of keystream generation. Though the iterations contain dependencies, it can efficiently execute with deep pipelining and data forwarding as supported by modern general purpose processors. Finally, large bit-width is used for storing the keywords and encryption, thereby exploiting SIMD extensions of modern microprocessors.

Processor	Clock Freq. (GHz)	Initialization	
		cycles/setup	setups/second
Intel Pentium M	1.7	25947.56	65516.76
Intel Pentium 4	2.8	57008.26	49115.69
ADM Athlon 64 X2 4200+	2.2	23346.63	94232.02
Alpha EV6	0.5	42184.67	11852.65
HP 9000/785	0.875	19911.01	43945.54

Table 6.1: HC-128 Key and IV initialization on general purpose processors

Processor	Clock Freq. (GHz)	Key Generation	
		cycles/byte	Gbps
Intel Pentium M	1.7	3.07	4.43
Intel Pentium 4	2.8	3.76	5.95
ADM Athlon 64 X2 4200+	2.2	2.86	6.19
Alpha EV6	0.5	3.9	1.03
HP 9000/785	0.875	2.75	2.54

Table 6.2: HC-128 Keystream generation on general purpose processors

6.3 Experiment with embedded & customizable processors

In this section, we analyze the available performance figures for HC-128 for embedded processors. Furthermore, several experiments have been conducted and reported on customization of an embedded processor for improving the software runtime performance of HC-128.

6.3.1 Implementation on embedded processors

Only one implementation of HC-128 on embedded micro-controller has been reported so far. It is observed that the large RAM size does not fit into many 8-bit micro-controllers [87]. ATmega128l from Atmel's 8-bit AVR micro-controller family, running at 8 MHz clock, is used.

We implemented the HC-128 stream cipher on another dominant embedded processor, namely ARM Cortex-A8. We ran HC-128 on a TI OMAP3530 platform, where ARM Cortex-A8 is run at 550 MHz. For compilation, CodeSourcery ARM compiler is used with *-O3 -march=armv7-a* options. ARM cycle counter is used for estimating the number of cycles. No optimization at C-level or assembly-level is performed. The results of executing HC-128 on ATmega128l and ARM Cortex-A8 are presented in Table 6.3. Note that, both the processors reported here, i.e., ATmega128l and ARM Cortex-A8 can run faster as per their specifications at 16 MHz and 1 GHz respectively. Assuming the cycle count will remain intact at a higher clock, we also report the corresponding speed numbers.

6.3.2 Implementation on customizable processor

Customizing a basic processor templates with instruction-set architecture extensions and micro-architectural modifications are commonly found in the portfolio of several IP vendors. In this experiment, IRISC, a customizable RISC processor template is chosen, which is based on the IPs distributed with a high-level processor design framework. The template processor description supports a basic 32-bit RISC instruction set with predicated instruction execution. The

Processor	Clock Frequency (GHz)	Initialization	
		cycles/setup	setups/second
ATmega128l [87]	8	2084788	3.84
ATmega128l (estimated)	16	2084788	7.68
ARM Cortex-A8 (implemented)	550	85696	6418.04
ARM Cortex-A8 (estimated)	1000	85696	12836.08
IRISC (Core Area 41.35 NAND KGates)	769	231695	3320.02

Table 6.3: HC-128 Key and IV initialization on embedded & customizable embedded processors.

micro-architecture contains a 5-stage pipeline supporting synchronous accesses to program and data memory. The core features 16 32-bit general purpose registers and a range of special purpose registers. On the basis of the initial profiling results of HC-128 on the processor template, several architectural modifications are performed. These are described briefly in the following.

Special Purpose Registers: A special purpose 10-bit counter for HC-128 is defined in the processor template.

Special Purpose Memories: The processor template supported only one data memory. This restricted the possible speed-up of the architecture considerably. Therefore, two additional memories for P and Q array are defined.

Custom Instruction Extensions: Special instructions supporting rotate left and rotate right are defined. In functions f_1, f_2, g_1 and g_2, rotation of two values are done followed by XOR operation. For this, a custom instruction *rotate_xor* is defined. For address generation in h_1 and h_2, a special purpose instruction *shift_xor_add* is defined.
In order to allow fast access of P and Q memories, custom load and store instructions are defined with additional addressing support for *modulo minus* operations based on the counter. While extending for this custom instructions, it was also made sure that the custom instructions do not increase the proces-

Processor	Clock Frequency (GHz)	Key Generation	
		cycles/byte	Gbps
ATmega128l [87]	8	168.81	0.38 Mbps
ATmega128l (estimated)	16	168.81	0.76 Mbps
ARM Cortex-A8 (implemented)	550	11.06	0.39 Gbps
ARM Cortex-A8 (estimated)	1000	11.06	0.79 Gbps
IRISC	769	15	0.41 Gbps

Table 6.4: HC-128 keystream generation on embedded & customizable embedded processors.

sor critical path. The final processor description was processed by Synopsys Processor Designer to generate the HDL description [27]. The HDL description is synthesized with Synopsys Design Compiler, topographical mode for 65 nm technology library. The synthesis results and runtime performance are presented in Table 6.3. From the speed perspective, it could be observed that IRISC significantly outweighs the earlier results on ATmega micro-controller and achieves comparable results as ARM. This study provides us with a background on achievable throughput on embedded processors. As we will observe in the following sections, further speed-up in software-based implementation will require a multi-threaded execution platform.

6.4 Hardware accelerator implementation of HC-128

It should be noted that the HC-128 accelerator implementation needs to be configurable for loading new keys and initialization vectors. To that end, we utilized the same high-level synthesis flow used for processor designing and carefully designed a micro-architecture with least necessary configurability [27]. For example, the micro-architecture contained two sets of instructions for initialization and keystream generation respectively. In order to have an area-

efficient design, the ASIC for initialization and key stream generation are built around the same underlying hardware structure.

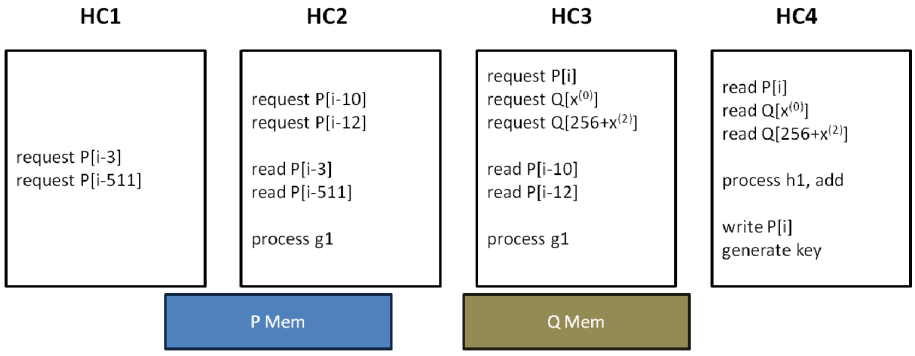

Figure 6.1: Pipeline structure for keystream generation

6.4.1 Implementation of Keystream Generation

The major bottleneck in the ASIC implementation for HC-128 is large area overhead from its storage, i.e., P and Q arrays. For our storage implementation, dual-port read/write synchronous SRAM is used with single-cycle access latency. The accelerator is initially conceived in the 4-stage pipeline structure as shown in Figure 6.1. For the sake of simplicity, only the processing of if-block in the keystream generation algorithm is shown. The else-block utilizes exactly the same structure with an additional multiplexer controlled by the counter k.

The storage accesses of keystream generation algorithm are distributed in the 4-stage pipeline structure $HC1$, $HC2$, $HC3$ and $HC4$ in a manner that the SRAM access restrictions are not violated if only one pipeline stage is active at any given cycle. Efficient distribution of the computation in the pipeline stages is important for achieving a low critical path, i.e., a high clock frequency. To that effect, the computation is triggered from the innermost kernel of functions g_1 or g_2, i.e., the *rotate* operation followed by the *xor* is computed as soon as the data is available in pipeline stage $HC2$. In stage $HC3$, further processing of g_1 is done and the addresses for accessing Q array is computed. In the same stage, the read requests for Q array are placed. In the final stage, the remaining computations for keystream generation and updating the P array are performed. Subsequently, a write request to P array

is made. Note that, because of this write request, a new iteration of keystream generation cannot be started until the next cycle. This means, every iteration of keystream generation occupies the complete pipeline for 4 cycles, thereby providing a keystream generation speed of 1 word per 4 cycles or 1 byte/cycle. The gate-level synthesis results at 65 nm technology node achieved a total area of 3.72 KGates with 4 KBytes of RAM. The fastest clock frequency is determined to be 1.82 GHz indicating 14.56 Gbps throughput.

For improved initialization speed, the initialization module of HC-128 is also included in the design. The initialization part of HC-128 is distributed into 4 phases. The first 3 phases perform a given set of operations using P, Q and auxiliary array W [128]. The last phase essentially runs the keystream generation step 1024 times, where instead of generating the final keystream, the arrays are updated with the final result at each step. For this purpose, we re-used the pipeline structure exactly as depicted earlier in Figure 6.1. The area and timing results after including initialization module is reported in Table 6.5 (base implementation).

6.5 Parallelization strategies

Certain parallelization strategies seem to be interesting towards the implementation aspects of HC-128. First of all, long initialization latency is considered a drawback, especially for stream ciphers. Although our proposed accelerator does the initialization considerably faster than available implementations, it remains important to check if the initialization can be managed without any delay.

6.5.1 Parallelizing initialization with keystream generation

The proposed accelerator for key and IV setup shares several computation with the keystream generation. To invoke the initialization in parallel to keystream generation, it requires duplication of the pipeline structure. A second set of P and Q array will be needed, too. The initialization can work on a set of

P, Q array while the other set is being used for keystream generation. The keystream generation operation can switch to a new key and IV with a latency of only 1 cycle. The support for parallel initialization increases area overhead as reported in Table 6.5.

6.5.2 Parallelizing keystream generation

The keystream generation itself can be further parallelized by efficient duplication of resources. For that, we study the algorithm and partition the keystream generation into two phases namely, key generation and self-update. These two phases can be separately triggered. The key idea is that when, the self update for P array takes place, the key generation will take place from Q array and an old copy of P array.

Key generation

For the key generation, updated value of P or Q array is required at index i. This can be taken directly from the updated $P1$ or $Q1$ array. However, the values stored at index $i \boxminus 12$ gets updated during the self update function, which is not the desired value for key generation function. For example, when the value of i is 1, $i \boxminus 12$ points to array index 501. This P array value at this index gets updated during self-update function later. For key generation, one needs to have the old value though. Therefore, for correct computation of h_1 and h_2 functions, two additional arrays $P3$ and $Q3$ are maintained. During the key generation, $P1$ or $Q1$ arrays reflect the updated versions, whereas $P2$ or $Q2$ contain old values that suffice for current iterations. During key generation from $P1$ or $Q1$, a subsequent data transfer to $P2$ or $Q2$ is done to prepare for the next else or if block respectively.

Self update

The self update always works on the $P1$ or $Q1$ array, with a subsequent loading to $P3$ or $Q3$ array as required. The updating works on a different indexing, j.

	Area			Clock Frequency (GHz)	Keystream Generation		Initialization (setups/second)
	Combinational (NAND Kgates)	Sequential (NAND Kgates)	RAM size (KByte)		(cycles/byte)	(Gbps)	
Base Implementation	6.43	1.76	9	1.67	1	13.36	211499.49
Parallel Initialization	11.73	2.48	13	1.67	1	13.36	211499.49
Parallel Keystream Generation	10.06	3.60	21	1.67	0.75	17.81	192751.62
Odd-Even Memory Partitioning	10.28	2.37	21	1.43	0.5	22.88	243617.23

Table 6.5: Performance of hardware accelerator

Initialization

In order to trigger the parallel keystream generation algorithm, the initialization of key and IV needs to be done with additional care. The 3rd and 4th phase of key and IV setup (as described in [128, Section 2]) needs to be done for all $P1, P2, Q1$ and $Q2$. In this work, it is done by first running the key and IV setups for $P1$, $Q1$ arrays followed by a copying to $P2$ and $Q2$ respectively.

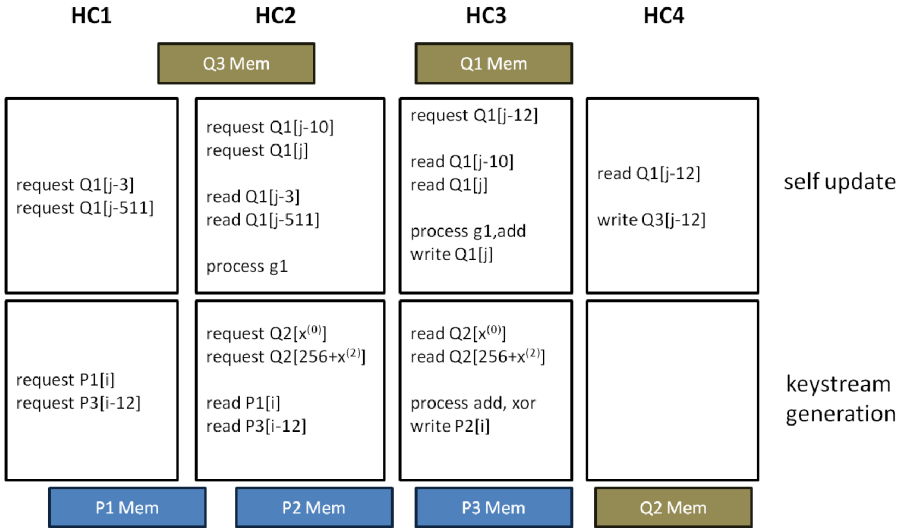

Figure 6.2: Pipeline structure for parallel keystream generation

Pipeline structure

In addition to the parallel initialization structure, two parallel 4-stage ASIC pipelines are conceived to accommodate the parallel keystream generation algorithm. The self update and keystream generation phases are distributed in the 4 pipeline stages as shown in Figure 6.2. Though the key generation phase is much faster now, accomplishing within 3 pipeline stages, the self update process still spans the entire pipeline. However, since the write request in self update process accesses the array $Q3$ (or $P3$), the pipeline can accommodate a new iteration of self update in its first stage. Therefore, one iteration of self-update requires 3 cycles to complete. This ensures that one word of key is generated every 3 cycles.

Pipeline Stage	Operation with Memory Partitioning
HC1	request $Q1_{odd}[j \boxminus 3], Q1_{odd}[j \boxminus 511]$ request $Q1_{even}[j \boxminus 10], Q1_{even}[j]$
HC2	read $Q1_{odd}[j \boxminus 3], Q1_{odd}[j \boxminus 511]$ read $Q1_{even}[j \boxminus 10], Q1_{even}[j]$; request $Q1_{even}[j \boxminus 12]$; process $g1, add$; write $Q1_{even}[j]$
HC3	read $Q1_{even}[j \boxminus 12]$; write $Q3[j \boxminus 12]$

Table 6.6: Pipeline organization with odd-even memory partitioning

6.5.3 Odd-even memory partitioning

A careful study in the update part of the pipeline designed in Figure 6.2 reveals that the accesses to the memory $Q1$ is a bottleneck, spreading the iteration over 3 pipeline stages. Thanks to our intuitive design, the access indices follow a pattern with alternate access to odd and even memory locations. This previous design decision is now leveraged by creating two memories from $Q1$ namely, $Q1\_odd$ and $Q1\_even$. This distribution allowed to squeeze the self update pipeline in 3 stages. The keystream generation still spreads over 3 pipeline stages. However, a new iteration can start in pipeline stage $HC1$ when the current iteration is in pipeline stage $HC3$. The operation distribution per pipeline stage for the update part is shown briefly in the above table.

As we could overlap the write access of the last pipeline stage with a new iteration starting at the first pipeline stage, the self update phase now takes 2 cycles per iteration. This strategy increases the keystream generation throughput to 1 word/2 cycles.

All the above design points are implemented in form of Verilog RTL and synthesized with Synopsys Design Compiler, Topographical mode, using a 65 nm technology library. The detailed results discussed so far are presented in Table 6.5. It is interesting to note that the split up of key generation and self updating, as done for parallel keystream generation also resulted into less combinational circuit. This happened due to the fact that the splitting avoided several combinational logic to be on the critical path, for which sim-

pler implementation could be synthesized. The memory partitioning approach resulted into little increase in the critical path because of the arithmetic operations now being performed in a single stage. For the same design point, a gain in sequential area is observed by avoiding several pipeline registers. The highest throughput for key stream generation in a software implementation is 6.19 Gbps (see Table 6.2). In comparison, the fastest implementation we could achieve is 22.88 Gbps, which is more than 3 *times* the fastest reported software implementation. The initialization task, combining key and IV set up together, is more than 2 *times* faster than the fastest reported software implementation. The results justify a closer look at the multi-core software implementation for HC-128 and/or deploying dedicated accelerator for HC-128 in heterogeneous embedded platforms.

6.6 Conclusion

In this chapter, we investigated different design points of a implementing HC-128 stream cipher on contemporary as well as futuristic platforms. We have achieved significant throughput improvement compared to state-of-the-art systems for the keystream generation as well as for the initialization with novel parallelization strategies. The design points offer a variety of area and throughput options for hardware accelerator-based implementations.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Conclusion and Future Work

In this thesis, a detailed study have been performed on the design, analysis and implementation of HC-128 stream cipher. The study spanned from theoretical cryptanalysis to compact cipher implementation, covering side-channel cryptanalysis also in the journey. As discussed earlier, the primary reason of choosing HC-128 as the thesis topic was, because it was one of the few ciphers amongst 34 submissions of the eStream comptetition with no significant cryptanalytic results. Though the work carried out, does not break the cipher, it has definitely propelled interest in the cryptology community, which is evident by an increase in related publications subsequent to our initial work.

Note that the HC-128 cipher belongs to the genre of array based stream ciphers. Though many ciphers in this class have been analyzed critically and shown to have several vulnerabilities, HC-128 remained a strong exception. Through our work, we have shed light on the important structural features of HC-128, that may lead to realistic attacks in future.

7.1 Summary

After a brief introduction to cryptology and stream ciphers in Chapter 1, we described the HC-128 and HC-256 stream ciphers and enlisted the existing results on the two ciphers in Chapter 2.

In Chapter 3, we showed that if any one of the two internal state arrays of

HC-128 and just 2048 keystream words are known, one can construct the other state array completely in 2^{42} time complexity. In addition, one can recover the Key. This analysis reveals a structural weakness of the cipher and theoretically establishes some novel combinatorial properties of HC-128 keystream generation algorithm. Though this does not affect the actual security of the cipher, it is an interesting observation related to the the internal structure. In this chapter, we also propose a design modification of HC-128 that resists such half-state exposure. We evaluate the performance of our new design in the eStream testing framework and compare the speed with that of actual HC-128.

In Chapter 4, we exploited linear approximation in the feedback functions g_1, g_2 of HC-128. There are three 32-bit additions in each of these feedback functions. We reviewed the analysis of approximating the additions by bit-wise XOR operators, and thereby could extend the "least significant bit" based distinguisher (presented by the designer of the cipher) of HC-128 to other bits of the 32-bit word. We also presented a new distinguisher by properly choosing pairs of five keystream word combinations from three consecutive blocks corresponding to only P (or only Q) array. In this chapter, we also looked into the problem of state leakage in keystream. In [36], it has been observed that $Prob(s_j \oplus s_{j+1} = P[j] \oplus P[j+1]) \approx 2^{-16}$, where s_j, s_{j+1} are two consecutive keystream output words. We studied that in more detail and found a sharper association which gives twice the above probability.

In Chapter 5 we showed how the cache analysis of HC-256 [130] can be extended to HC-128 and how the differential fault analysis of HC-128 [64] can be extended to HC-256. The first attack led to half state recovery of HC-128 and when combined with the differential fault analysis can lead to the full state recovery and key recovery of HC-128. With the second attack, we have been able to perform partial state recovery of HC-256. Our findings show that the side channel vulnerability for a particular implementation of a cipher may percolate to its variants also, albeit in a different degree. This vulnerability is still exploitable through refinement of the attack vectors. So, while selecting a cipher variant to thwart side channel vulnerabilities, one must exercise extra caution.

In Chapter 6, we studied efficient hardware implementation of HC-128 on different platforms. Advancement of VLSI technology (particularly, embed-

ded processors) have removed the boundary between software and hardware. Irrespective of the fact that HC-128 is a software stream cipher, it is has thus become important to analyze the hardware design parameters of HC-128 which happens to be the fastest software stream cipher of eStream. After a review of the implementation of HC-128 on general purpose CPU, we studied the implementation in several embedded processors and on a state-of-the-art customizable processor. Further, we present detailed ASIC implementation of HC-128 that can be included as a module in a cryptographic co-processor. We also proposed novel parallelization strategies both in terms of key/IV set-up and keystream generation. These efforts yielded more than 3 times throughput improvement compared to the fastest software speed reported in eStream. Our strategies can completely hide the initialization latency by employing a multi-session HC-128 execution. In short, the analysis in this chapter served two major purposes towards the deployment of HC-128 stream cipher. First, it reported several novel and generic performance improvement strategies for HC-128. Second, several design points were explored ranging from customizable co-processor to fully dedicated hardware accelerators.

7.2 Future works and open problems

Research is a never-ending journey. A thesis attempts to solve some open problems and in the process creates more open problems than it started with. This thesis is no exception. Here we list the most important ones amongst the possible future extensions of our work and the new problems that our work has opened up.

In the work on internal state in Chapter 3, we showed that half the internal state leaks the other half of the internal state and thus in turn leaks the entire state. A possible generalization could be the case where instead of half if we know only α fraction of the internal state. What fraction β (as a function of α) of the total internal state can be inferred from this. This is very hard and non-trivial. Another caveat of our work is that we assumed the known half-state to be either the entire P array or the entire Q array. What if we know half of the internal state, but partly from P and partly from Q? This is also an interesting open problem. In this chapter, we also argued why our

strategy of half-state to full-state expansion cannot be extended to HC-256. However, there could be other strategies that might expand γ-fraction of the internal state of HC-256 to δ ($> \gamma$)-fraction of its state. Investigating such strategies is also important in the anaylsis of HC series of ciphers.

In Chapter 4, we have discovered few new but weak distinguishers (not of practical complexity). An obvious question is: can we have stronger distinguishers? Moreover, we proved some correlations between state and keystream. So another related open problem is: can this state-leakage or any other undiscovered form of state-leakage be used to mount a (full or partial) state-recovery attack on HC-128? Similar questions can be asked for HC-256 as well.

Two interesting future works that naturally arise from the side channel analysis of Chapter 5 are as follows:

1. Can the cache analysis alone lead to full state recovery of HC-128?
2. Can the differential fault attack alone be used to mount full state recovery of HC-256?

Apart from these, one may also look into alternative fault model. For example, we have assumed that the location of the fault is known for HC-256. A stronger model may work with faults at unknown location.

The speed achieved in the hardware designs attempted in Chapter 6 is not very enthusiastic compared to general purpose processors. However, once the cipher is widely deployed in commercial domain, one may like to use it on low end hand-held devices. Work in this direction has been reported earlier for embedded micro-controllers [87] and wireless sensor networks. Once HC-128 becomes more popular, one may try to achieve higher speed for the key-stream generation. Given the overheads of general purpose processors, it is clear that one can always obtain further speed-up using application specific hardware accelerator and such a design of HC-128 may efficiently be used in a cryptographic co-processor. This gives rise to a wide area of open research - not only for increasing throughput, but also in terms of other performance parameters, like reducing the gate count, power minimization, etc. In addition, it will be interesting to explore the application of these parallelization techniques to other stream ciphers. Alternatively, it is necessary to check the functionality

of the light weight forms of HC-256 and HC-128 and their resiliency in the current scenario of resource constrained environment.

7.3 Final words

The HC series of stream ciphers have been in the horizon for close to a decade. This thesis marks culmination of five years of dedicated research on the cipher. It appears that the designer has taken care of resistance against practical attacks. However, cryptography and cryptanalysis are always at arm's race and we never know what is coming up next.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] 3rd Generation Partnership Project. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. ETSI/SAGE Specification – Document 2: ZUC Specification, v1.6, June 28, 2011.

[2] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[3] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-Channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002.

[4] Manfred Josef Aigner, Stefan Mangard, Francesco Menichelli, Renato Menicocci, Mauro Olivieri, Thomas Popp, Giuseppe Scotti, and Alessandro Trifiletti. Side channel analysis resistant design flow. In *ISCAS*. IEEE, 2006.

[5] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(65), 1996.

[6] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In Frank Thomson Leighton and Peter W. Shor, editors, *STOC*, pages 284–293. ACM, 1997.

[7] Frederik Armknecht and Matthias Krause. Algebraic Attacks on Combiners with Memory. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2003.

[8] S. Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers. European Convention on Security and Detection, IEEE Conference Publication, May 1995.

[9] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2013.

[10] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on the Grain Family of Stream Ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2012.

[11] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer, 2003.

[12] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *J. Cryptology*, 21(3):392–429, 2008.

[13] Lawrence E. Bassham, III, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2010.

[14] Riddhipratim Basu, Shirshendu Ganguly, Subhamoy Maitra, and Goutam Paul. A complete characterization of the evolution of rc4 pseudo random generation algorithm. *J. Mathematical Cryptology*, 2(3):257–289, 2008.

[15] Daniel J. Bernstein. Cache-timing attacks on AES. Technical report, The University of Illinois at Chicago, 2005.

[16] Eli Biham, Louis Granboulan, and Phong Q. Nguyen. Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer, 2005.

[17] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[18] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[19] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[20] C. Blundo and P. D'Arco. The Key Establishment Problem. Lecture Notes in Computer Science, Springer, 2004.

[21] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[22] Dirk Bouwmeester, Artur Ekert, Anton Zeilinger, et al. *The Physics of Quantum Information*, volume 38. Springer Berlin, 2000.

[23] Colin A. Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.

[24] C. D. Cannière. eStream Testing Framework. Available online at `http://www.ecrypt.eu.org/stream/perf` [last accessed on April 30, 2014].

[25] Anne Canteaut and Michaël Trabbia. Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer, 2000.

[26] Anupam Chattopadhyay, Ayesha Khalid, Subhamoy Maitra, and Shashwat Raizada. Designing High-throughput Hardware Accelerator for Stream Cipher HC-128. In *ISCAS*, pages 1448–1451. IEEE, 2012.

[27] Anupam Chattopadhyay, Heinrich Meyr, and Rainer Leupers. LISA: a Uniform ADL for Embedded Processor Modelling, Implementation and Software Toolsuite Generation. *Processor Description Languages*, pages 95–130, 2008.

[28] Colin Cooper and Alan Frieze. The Size of the Largest Strongly Connected Component of a Random Digraph with a Given Degree Sequence. *Comb. Probab. Comput.*, 13(3):319–337, May 2004.

[29] Nicolas Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003.

[30] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[31] Joan Daemen and Paris Kitsos. The Self-synchronizing Stream Cipher Moustique. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2008.

[32] Joan Daemen and Vincent Rijmen. The Block Cipher Rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *CARDIS*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998.

[33] David Deutsch. Quantum theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London*, 400:97–117, 1985.

[34] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, volume 45 of *AFIPS Conference Proceedings*, pages 109–112. AFIPS Press, 1976.

[35] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[36] O. Dunkelman. A Small Observation on HC-128, November 2007. Available online at`http://www.ecrypt.eu.org/stream/phorum/read.php1,1143` [last accessed on April 10, 2011].

[37] ECRYPT Stream Cipher Project eStream. The current eSTREAM portfolio. Available online at `http://www.ecrypt.eu.org/stream/index.html`.

[38] Wieland Fischer, Berndt M. Gammel, O. Kniffler, and J. Velten. Differential Power Analysis of Stream Ciphers. In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007.

[39] Martin Gagné. Identity-Based Encryption. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 594–596. Springer, 2011.

[40] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[41] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-Key Cryptosystems from Lattice Reduction Problems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

[42] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[43] Jovan Dj. Golic and Miodrag J. Mihaljevic. A Generalized Correlation Attack on a Class of Stream Ciphers Based on the Levenshtein Distance. *J. Cryptology*, 3(3):201–212, 1991.

[44] S. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[45] Jennie C. Hansen and Jerzy Jaworski. Large Components of Bipartite Random Mappings. *Random Struct. Algorithms*, 17(3-4):317–342, 2000.

[46] Philip Hawkes and Gregory G. Rose. Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2004.

[47] Martin E. Hellman. A Cryptanalytic time-memory Trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[48] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.

[49] Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[50] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[51] Michal Hojsík and Bohuslav Rudolf. Differential Fault Analysis of Trivium. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[52] Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 353–372. Springer, 2005.

[53] Thomas Johansson and Fredrik Jönsson. Fast Correlation Attacks Based on Turbo Code Techniques. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 1999.

[54] Thomas Johansson and Fredrik Jönsson. Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer, 1999.

[55] Alireza Jolfaei, Ahmadreza Vizandan, and Abdolrasoul Mirghadri. Image Encryption Using HC-128 and HC-256 Stream Ciphers. *Int. J. Electron. Secur. Digit. Forensic*, 4(1):19–42, February 2012.

[56] Robert J. Jenkins Jr. ISAAC and RC4. Published on the Internet at http://burtleburtle.net/bob/rand/isaac.html, 1996.

[57] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, December 1996.

[58] I. B. Kalugin. The Number of Components of a Random Bipartite Graph. *Discrete Math. Appl.*, 1(3):289–299, 1989.

[59] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[60] Firdous Kausar and Ayesha Naureen. A Comparative Analysis of HC-128 and Rabbit Encryption Schemes for Pervasive Computing in WSN Environment. In *Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance*, ISA '09, pages 682–691, Berlin, Heidelberg, 2009. Springer-Verlag.

[61] Auguste Kerckhoffs. La cryptographie Militaire. *Journal des Sciences Militaires*, pages 161–191, 1883.

[62] Ayesha Khalid, Deblin Bagchi, Goutam Paul, and Anupam Chattopadhyay. Optimized GPU Implementation and Performance Analysis of HC Series of Stream Ciphers. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC*, volume 7839 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2012.

[63] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In Michael J. Wiener, editor,

CRYPTO, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[64] Aleksandar Kircanski and Amr M. Youssef. Differential Fault Analysis of HC-128. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2010.

[65] Andreas Klein. Attacks on the RC4 stream cipher. *Des. Codes Cryptography*, 48(3):269–286, 2008.

[66] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of trivium and KATAN. In *Selected Areas in Cryptography*, pages 200–212. Springer, 2012.

[67] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[68] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[69] Neal Koblitz. Hyperelliptic Cryptosystems. *J. Cryptology*, 1(3):139–150, 1989.

[70] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[71] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[72] Sandeep Kumar, Kerstin Lemke, and Christof Paar. Some Thoughts about Implementation Properties of Stream Ciphers. In *SASC - State of the Art of Stream Ciphers Workshop*, 2004.

[73] Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. Power Analysis of Synchronous Stream Ciphers with Resynchronization Mechanism. In *Workshop Record of SASC 2004 – The State of the Art of Stream Ciphers*, pages 327–333, 2004. Available online at http://www.ecrypt.eu.org/stvl/sasc/record.html.

[74] Gregor Leander, Erik Zenner, and Philip Hawkes. Cache Timing Analysis of LFSR-Based Stream Ciphers. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2009.

[75] Steven Levy. *Crypto: How the Code Rebels Beat the Government–Saving Privacy in the Digital Age*. Penguin USA, New York, NY, USA, 2001.

[76] Yunyi Liu and Tuanfa Qin. The Key and IV Setup of the Stream Ciphers HC-256 and HC-128. *Networks Security, Wireless Communications and Trusted Computing, International Conference on*, 2:430–433, 2009.

[77] Yi Lu and Serge Vaudenay. Faster Correlation Attack on Bluetooth Keystream Generator E0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2004.

[78] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[79] Subhamoy Maitra and Goutam Paul. New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2008.

[80] Subhamoy Maitra, Goutam Paul, and Shashwat Raizada. Some Observations on HC-128. In *International Workshop on Coding and Cryptology, WCC 2009, Ullensvang, Norway*, PreProceedings, pages 527–539, 2009.

[81] Subhamoy Maitra, Goutam Paul, Shashwat Raizada, Subhabrata Sen, and Rudradev Sengupta. Some observations on HC-128. *Des. Codes Cryptography*, 59(1-3):231–245, 2011.

[82] Itsik Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2005.

[83] Itsik Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2005.

[84] Itsik Mantin and Adi Shamir. A Practical Attack on Broadcast RC4. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2001.

[85] Willi Meier and Othmar Staffelbach. Fast Correlation Attacks on Certain Stream Ciphers. *J. Cryptology*, 1(3):159–176, 1989.

[86] Willi Meier and Othmar Staffelbach. Correlation Properties of Combiners with Memory in Stream Ciphers. *J. Cryptology*, 5(1):67–86, 1992.

[87] Gordon Meiser, Thomas Eisenbarth, Kerstin Lemke-Rust, and Christof Paar. Efficient Implementation of eStream ciphers on 8-bit AVR microcontrollers. In *SIES*, pages 58–66. IEEE, 2008.

[88] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[89] R. Merkle and M. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Trans. Inf. Theor.*, 24(5):525–530, September 2006.

[90] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[91] Michael Molloy and Bruce A. Reed. The Size of the Giant Component of a Random Graph with a Given Degree Sequence. *Combinatorics, Probability & Computing*, 7(3):295–305, 1998.

[92] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[93] Information Society Technologies (IST) Programme of the European Commission. Performance of Optimized Implementations of the NESSIE Primitives, version 2.0, IST-1999-12324, 2003.

[94] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: The Case of AES. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[95] Sarbani Palit, Bimal K. Roy, and Arindom De. A Fast Correlation Attack for LFSR-Based Stream Ciphers. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *ACNS*, volume 2846 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2003.

[96] Goutam Paul, Subhamoy Maitra, and Shashwat Raizada. A Theoretical Analysis of the Structure of HC-128. In Tetsu Iwata and Masakatsu Nishigaki, editors, *IWSEC*, volume 7038 of *Lecture Notes in Computer Science*, pages 161–177. Springer, 2011.

[97] Goutam Paul and Shashwat Raizada. Impact of Extending Side Channel Attack on Cipher Variants: A Case Study with the HC Series of Stream Ciphers. In Andrey Bogdanov and Somitra Kumar Sanadhya, editors, *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 32–44. Springer, 2012.

[98] M. O. Rabin. Digitalized Signatures and Public-key Functions as Intractable as Factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[99] Josyula R. Rao and Pankaj Rohatgi. EMpowering Side-Channel Attacks. *IACR Cryptology ePrint Archive*, 2001:37, 2001.

[100] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[101] Matthew J. B. Robshaw. Stream ciphers. *Technical Report, RSA Laboratories*, 1995.

[102] 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.

[103] Rainer A. Rueppel. *Analysis and design of stream ciphers*. CCES Communications and Control Engineering Series. Springer-Verlag Berlin Heidelberg New York, Berlin, 1986.

[104] Markku-Juhani Olavi Saarinen. A Time-Memory Tradeoff Attack Against LILI-128. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236. Springer, 2002.

[105] A. I. Saltykov. The Number of Components in a Random Bipartite Graph. *Discrete Mathematics and Applications*, 5(6):86–94, 1995.

[106] Gautham Sekar and Bart Preneel. Improved distinguishing attacks on hc-256. In Tsuyoshi Takagi and Masahiro Mambo, editors, *Advances in Information and Computer Security*, volume 5824 of *Lecture Notes in Computer Science*, pages 38–52. Springer Berlin Heidelberg, 2009.

[107] Adi Shamir. A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, pages 279–288. Plenum Press, New York, 1982.

[108] Adi Shamir. A Polynomial-time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. *IEEE Transactions on Information Theory*, 30(5):699–704, 1984.

[109] Adi Shamir. Identity-Based Cryptosystems and Signature Schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[110] Adi Shamir. Stream Ciphers: Dead or Alive? In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, page 78. Springer, 2004.

[111] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal, Vol 28, pp. 656âĂŞ715*, Oktober 1949.

[112] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana and Chicago, 1949.

[113] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994.

[114] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[115] Simon Singh. *The Code book, the Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor, 1999.

[116] Sergei P. Skorobogatov. Semi-invasive Attacks – A New Approach to Hardware Security Analysis, 2005.

[117] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[118] Othmar Staffelbach and Willi Meier. Cryptographic Significance of the Carry for Ciphers Based on Integer Addition. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 601–614. Springer, 1990.

[119] Paul Stankovski, Martin Hell, and Thomas Johansson. Analysis of Xor-rotation with Application to an HC-128 Variant. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *ACISP*, volume 7372 of *Lecture Notes in Computer Science*, pages 419–425. Springer, 2012.

[120] Paul Stankovski, Sushmita Ruj, Martin Hell, and Thomas Johansson. Improved distinguishers for HC-128. *Des. Codes Cryptography*, 63(2):225–240, 2012.

[121] Douglas R. Stinson. *Cryptography - Theory and Practice*. Discrete mathematics and its applications series. CRC Press, 2006.

[122] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *NDSS*. The Internet Society, 2002.

[123] Erik Tews. Attacks on the WEP protocol. *IACR Cryptology ePrint Archive*, 2007:471, 2007.

[124] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. *IACR Cryptology ePrint Archive*, 2007:120, 2007.

[125] Kris Tiri, Patrick Schaumont, and Ingrid Verbauwhede. Side-Channel Leakage Tolerant Architectures. In *ITNG*, pages 204–209. IEEE Computer Society, 2006.

[126] Hongjun Wu. Cryptanalysis of Stream Cipher Alpha1. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 169–175. Springer, 2002.

[127] Hongjun Wu. A New Stream Cipher HC-256. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2004.

[128] Hongjun Wu. The Stream Cipher HC-128. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 39–47. Springer, 2008.

[129] Hongjun Wu and Feng Bao. Cryptanalysis of Stream Cipher COS(2, 128) Mode I. In Lynn Margaret Batten and Jennifer Seberry, editors, *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 154–158. Springer, 2002.

[130] Erik Zenner. A Cache Timing Analysis of HC-256. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2008.

[131] Erik Zenner. Cache Timing Analysis of eStream Finalists. In Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors,

Symmetric Cryptography, number 09031 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.