Cryptanalysis of Selected SPN and NLFSR-based Symmetric-Key Ciphers

Submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the Degree of
Doctor of Philosophy in Computer Science

Author: **Amit Jana**
Senior Research Fellow

Supervisor: **Goutam Paul**
Associate Professor

Cryptology and Security Research Unit
Indian Statistical Institute
Kolkata - 700108, India

June 2023

Dedicated to

my parents, wife, son, and my family members.

DECLARATION OF AUTHORSHIP

I, Amit Jana, a student of Cryptology and Security Research Unit, of the Ph.D. program of Indian Statistical Institute, Kolkata, hereby declare that the research work presented in this thesis titled "Cryptanalysis of Selected SPN and NLFSR-based Private-Key Ciphers" is based on my works. To the best of my knowledge, the materials presented in this thesis have not previously been published or written by any other person, nor it has been submitted as a whole or as a part of any degree/diploma or any other academic award anywhere before.

Amit Jana

Cryptology and Security Research Unit

Indian Statistical Institute, Kolkata

203, Barrackpore Trunk Road

Kolkata 700108, INDIA.

LIST OF PUBLICATIONS/MANUSCRIPTS

1. **Amit Jana**, Mostafizar Rahman and Dhiman Saha: "Depend on DEEPAND: Cryptanalysis of NLFSR-Based Lightweight Ciphers TinyJAMBU and KATAN", Cryptology ePrint Archive, Paper 2022/1123, url: https://eprint.iacr.org/2022/1123, Communicated.

2. **Amit Jana** and Goutam Paul: "Differential Fault Attack on SPN-based Sponge and SIV-like AE Schemes", Journal of Cryptographic Engineering, 2024, 363–381, url: https://doi.org/10.1007/s13389-024-00354-4.

3. **Amit Jana** and Goutam Paul: "Differential Fault Attack on PHOTON-Beetle", Proceedings of the 6th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2022, Los Angeles, CA, USA, November 11, 25–34, ACM 2022, url: https://doi.org/10.1145/3560834.3563824.

4. **Amit Jana**: "Differential Fault Attack on Feistel-Based Sponge AE Schemes", Journal of Hardware and Systems Security, 2022, 1–16, url: https://doi.org/10.1007/s41635-022-00124-w.

5. **Amit Jana**, Anirban Nath, Goutam Paul and Dhiman Saha: "Differential fault analysis of NORX using variants of coupon collector problem", Journal of Cryptographic Engineering, 2022, 433–459, url: https://doi.org/10.1007/s13389-022-00285-y.

6. **Amit Jana**, Dhiman Saha and Goutam Paul: "Differential Fault Analysis of NORX", Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2020, Virtual Event, USA, November 13, 67–79, ACM 2020, url: https://doi.org/10.1145/3411504.3421213.

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere appreciation and heartfelt gratitude to all those who have supported me throughout my journey of writing this thesis. I am deeply grateful to my esteemed Ph.D. supervisor, Dr. Goutam Paul, who is an Associate Professor of Cryptology and Security Research Unit at the Indian Statistical Institute, Kolkata. His profound knowledge, insightful comments, unwavering support, and constructive criticisms have been invaluable to me throughout the research process. Without his guidance, this achievement would not have been possible, and I will always be indebted to him for his encouragement and motivation.

I also wish to extend my special thanks to Dr. Dhiman Saha, from the Department of Electrical Engineering and Computer Science at IIT Bhilai, for his constant guidance, support, and motivation. His technical and non-technical discussions have provided me with invaluable insights and have played a crucial role in shaping my research work. I am truly grateful for his contributions to this thesis, and I will always remember his insightful comments and suggestions.

I would like to express my heartfelt appreciation and gratitude to Dr. Yu Sasaki, from NTT Social Informatics Laboratories, for his invaluable guidance during my 2-month internship at NTT Secure Platform Laboratories, Japan in 2019. His vast knowledge and expertise in the field of cryptography and security have been instrumental in shaping my research work, and his technical and non-technical discussions have provided me with invaluable insights. I am deeply grateful for his mentorship and support during my internship.

Additionally, I would also like to extend my special thanks to Dr. Avik Chakraborty, with whom I had the privilege of working during my internship. His guidance, support, and motivation have been crucial to my learning and growth during my time at NTT Secure Platform Laboratories, and I am grateful for the opportunity to have worked with him.

Once again, I express my gratitude to Dr. Yu Sasaki and Dr. Avik Chakraborty for their invaluable contributions to my research work, and for their unwavering support and encouragement throughout my internship.

I thank Dr. Mostafizar Rahman for his support and collaboration during my thesis work, and Dr. Ritam Bhaumik for his help in simplifying complex concepts.

I would like to express my heartfelt gratitude to the individuals who have been an integral part of my academic and personal journey. Firstly, I would like to extend a special thanks to my dear friend, Dr. Ashwin Jha, who has been a constant source of support and encouragement throughout our Masters and PhD years. Our frequent discussions on research, football, cricket, cinema, and music during tea breaks have been one of the most cherished memories of my academic life.

I would also like to thank Nishant Nikam for his invaluable contributions during our late-night lab sessions with Ashwin. I am also deeply grateful to my other close friends, Dr. Diptendu Chatterjee, Aniruddha, Subhadip, and Jyotirmoy, who have been by my side during various stages of my PhD journey.

I extend my sincere appreciation to my beloved seniors, Avik da, Butu da, Sanjay da, Srimanta da, Nilanajan da, Kaushik da, Binanda da, Pandey da, Indranil da, Karati da, Ritam da, and Avijit, who have generously shared their wisdom and insights with me during our tea adda sessions.

I would also like to acknowledge my lab mates, Diptendu, Nishant, Mostafizar, Nayana, Pritam, Prabal, Laltu, Samir, and Avishek, for their constant support and collaboration during my research work. I am grateful for the support and encouragement of my junior lab mates, Anup, Gourab, Anirban, Prabhat, Manish, Animesh, Sushanta, and Suman. Lastly, I would like to express my appreciation to Arindam, Tanmoy, Sankhadeep, Archan, Kushal, Harmendar, Ravindra, Sebati, Arghya, Ritam da, Bishwajit, Anik, and others, who have been my friends and companions, and with whom I have shared my interests in literature, cinema, music, football, cricket, and more. Your company has enriched my life and contributed to my personal growth.

To all of you, I extend my heartfelt gratitude for your contributions, support, and encouragement throughout my academic and personal journey.

I would like to express my heartfelt gratitude to the esteemed faculty members of our department for their invaluable contribution to my academic and personal growth. I am deeply indebted to Prof. Arijit Bishnu, Dr. Mridul Nandi, Dr. Debrup

Chakraborty, Prof. Mandar Mitra, Prof. Sandip Das, Prof. Krishendu Mukhopadhay, Prof. Sasthi Charan Ghosh, Prof. Anshuman Banerjee, Prof. Subhas Nandi, Prof. Kishan Chand Gupta, Dr. Goutam Paul, Prof. Bhargov Bhattacharya, Dr. Pinakpani Pal, Prof. Utpal Garain, Prof. Diganta Mukherjee, and Prof. Guruprasad Kar for their unparalleled knowledge, guidance, and support throughout my academic journey. Their dedication to teaching and commitment to excellence have inspired me to delve deeper into the fascinating world of computer science. Their teachings have not only helped me excel academically but have also prepared me to face the challenges of the real world. Once again, I extend my heartfelt gratitude to all the esteemed faculty members of our department.

I am deeply grateful to my school life teachers Ranjit da, Manu da, and Mr. Dipak Das for laying a strong foundation for my academic journey. I would also like to express my gratitude to my graduate life teachers Gopal da and Mr. Joydev Panda, who have been a constant source of inspiration and guidance.

During my graduation, I would like to thank all the members of "Snehaneer Mess" for their camaraderie and support, which made those three years memorable. In my masters, I am indebted to all the members of "Balaka Mess" for making my stay comfortable and enjoyable.

I am also thankful for the unwavering support of my closest friends, including Tapas, Manabendra, Kamal, Pintu, Amalesh, Rudra, Debasis, Sushovan, Samiran da, Milan, Dagar, and Banka, who have been by my side throughout my academic journey. Furthermore, I would like to mention my beloved seniors Malay da, Rintu da, Tapas da, Chanchal da, and Kalu da from "Balaka Mess" for their affection and guidance.

I would like to express my gratitude to my Baba, Maa, and all my family members for their unwavering support, love, and encouragement throughout my academic journey. Their constant support and motivation have been invaluable to me.

Finally, I would like to thank all the participants who helped me in my research work and contributed to my academic growth. Without their participation, my research work would not have been possible.

Amit Jana.

ABSTRACT

The thesis focuses on the cryptanalysis of private-key ciphers, which are widely used encryption methods due to their fast encryption/decryption computing ability and low memory requirements. The thesis covers two different aspects of cryptanalysis: traditional attack techniques and physical attacks. For physical attacks, the thesis presents a differential fault attack on the CAESAR scheme NORX with parallelism levels of 2 and 4. By introducing faults in NORX in parallel mode, the state collides with the internal branches to produce an all-zero state, which can be replayed despite different nonces and messages. The secret key of NORX is recovered using secondary faults and faulty tags, utilizing both internal and classical differentials. The attack strategy is demonstrated using different fault models to showcase its versatility. Additionally, the thesis identifies and solves a new variant of the coupon collector problem called the *Non-circular Consecutive Coupon Collector Problem*, which estimates the expected faults for the consecutive bit-fault model. The problem is extended to the circular variant and validated using hypothesis testing. The outcomes of this study may hold significance and relevance to the research community as a standalone contribution. Furthermore, the thesis investigates the faulty forgery attack on the decryption query to recover the state, leading to key recovery, for sponge-based authentication schemes with internal permutations following the SPN-based GFN structure. The attack is then extended to retrieve the secret key of any SPN-based sponge/SIV-like schemes. For traditional cryptanalysis, the thesis analyzes differential cryptanalysis of single or multiple AND-based NLFSR-like ciphers. Recent trends in automated cryptanalysis involve modeling classical cryptanalysis tools as optimization problems to leverage state-of-the-art solvers and improving existing models to make them more efficient and accurate. The thesis contributes to this trend by devising a general MILP model referred to as "DEEPAND" that captures the correlations among multiple AND gates in NLFSR-based lightweight block ciphers. The DEEPAND model builds upon and generalizes the idea of joint propagation of differences through AND gates, captured using refined MILP modeling of TinyJAMBU by Saha *et al.* in FSE 2020. The proposed model has been applied to TinyJAMBU and KATAN and can detect correlations that were missed by earlier models. This leads to more accurate differential bounds for both ciphers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACROYNMS AND ABBREVIATIONS

Expansion	Acronyms/ Abbreviations
Authenticated Encryption	AE
Authenticated Encryption with Associated Data	AEAD
Advanced Encryption Standard	AES
Lightweight Cryptography	LWC
National Institute of Standards and Technology	NIST
Exclusive-OR	XOR
That is	i.e.,
Message Authentication Codes	MAC
Substitution Permutation Networks	SPN
Generalized Feistel Networks	GFN
Balanced Feistel Networks	BFN
Unbalanced Feistel Networks	UFN
Logical-Operations Rotation XOR	LRX
ADD-Rotation-XOR	ARX
Tweakable Block Cipher	TBC
Block Cipher	BC
Difference Distribution Table	DDT
AddConstants	AC
SubCells	SC
MixColumnSerial	MCS
ShiftRows	SR
Ciphertext-only Attack	COA
Known-plaintext Attack	KPA
Adaptive Chosen-ciphertext Attack	CCA2
Chosen-ciphertext Attack	CCA
Chosen-plaintext Attack	CPA
Adaptive Chosen Plaintext and Ciphertext Attack	ACPC
Indistinguishability	IND
Non-malleability	NM
Key-privacy	KP
Key-indistinguishability	KI

LIST OF ACROYNMS AND ABBREVIATIONS

Expansion	Acronyms/ Abbreviations
Fault Detection	FD
Coupon Collector Problem	CCP
Consecutive Coupon Collector Problem	CCCP
Side-channel Analysis	SCA
Collision Fault Analysis	CFA
Differential Fault Analysis	DFA
Ineffective Fault Analysis	IFA
Safe-Error Analysis	SEA
Fault Sensitivity Analysis	FSA
Linear Fault Analysis	LFA
Statistical Fault Attack	SFA
Statistical Ineffective Fault Attack	SIFA
Statistical Effective Fault Attack	SEFA
Linear Feedback Shift Register	LFSR
Non-linear Feedback Shift Register	NLFSR
Mixed Integer Linear Programming	MILP
Constrained Programming	CP
Boolean Satisfiability Problem	SAT
Chained AND Bit Pattern	BAND
Multiple AND Bit Pattern	MAND
Without Loss of Generality	WLOG

1 INTRODUCTION

The words cryptography or cryptology comes from Ancient-Greek words, kryptós meaning "hidden secret", graphein meaning "to write" and logia means "to study". According to the definition in Oxford Dictionary, cryptography is the art of writing codes, where "code" refers to a system of symbols–such as a letter, word, text, sound, image, etc.– to convert information secrcetly into another form. Historically, cryptography was an art that focuses solely on the problem of secret communication. In the ancient times, constructing good codes, or breaking the codes mostly relied on creativity and personal skills. But as a science, the rigorous study of cryptography with time has been radically changed based on some emerging rich theories. The field of cryptography now encompasses much more than secret communication, including message authentication, digital signatures, protocols for exchanging secret keys, authentication protocols, electronic auctions, elections, and digital cash. Formally, modern cryptography is the science or study of the techniques of secret writing, especially code and cipher systems to secure digital information, transactions, and distributed computations. In short, Cryptography is the study of hidden writing, or the science of encrypting and decrypting message and ciphertexts.

Since the beginning of civilization, in military, diplomatic or otherwise, people have been felt a need for secrecy in communication. Indeed, the oldest known encrypted text occurred some 4000 years ago in the Egyptian town of Menet Khufu where the hieroglyphic inscriptions on the tomb of the nobleman `KHNUMHOTEP` II were written with a number of unusual symbols to obscure the meaning of the inscriptions. Also, in the 5-th century BC the Spartans developed a cryptographic device `Scytale` based on a transposition cipher to send and receive secret messages. A development of this cryptography field starts after

the two world wars followed by the subsequent cold war. Then the Internet revolution with the growth of IoT technologies and products, have acted as catalysts in the rapid development of this field. Cryptography has evolved significantly over the centuries with several key developments. In the 18^{th} century, the Vienna Black Chamber was important for intercepting and decrypting communications for the Habsburg Empire. In the 19^{th} century, Auguste Kerckhoffs established principles for cryptographic security, stating that the security should depend only on the secrecy of the key. In the 20^{th} century, Gilbert Vernam introduced *The One-Time Pad*, a method that provides theoretically unbreakable encryption, transforming secure communication. In [1], Kahn comprehensively described the history and development of cryptography. The era of modern cryptography begins with Claude Shannon, with his pioneering works during world war II on communications security. The first paper [2] in 1948, titled "A mathematical theory of communication", while the second paper, titled "Communication Theory of Secrecy Systems" [3] in 1949. These, in addition to his other works on information and communication theory established a solid theoretical basis for cryptography and also for much of cryptanalysis.

Until the 1970s, secure cryptography was largely the preserve of governments and concerned with just confidentiality or privacy in communication. Prior to that time, all useful encryption algorithms had been private-key ciphers in which the same cryptographic key is used with the underlying cipher, and both the sender and the recipient must have to kept it secret. The mid-1970s saw two major public advances. First was the publication of the draft Data Encryption Standard (DES) in the U.S. Federal Register on 17 March 1975. The proposed DES cipher was submitted by a research group at IBM, at the invitation of the National Bureau of Standards (now NIST), in an effort to develop secure electronic communication facilities for businesses such as banks and other large financial organizations. The second development, in 1976, the paper "New Directions in Cryptography" [4] by Whitfield Diffie and Martin Hellman introduced a radically new method of distributing cryptographic keys (known as the Diffie-Hellman key exchange), which went far toward solving one of the fundamental problems of cryptography. The article also stimulated the almost immediate public development of a new class of enciphering algorithms, the asymmetric-key (or the public-key) algorithms.

Modern cryptographic schemes are classified into two major classes: private-key (or symmetric-key/secret-key) and public-key (or asymmetric-key). For secret communication,

the setting in which the communicating parties share some secret information in advance is known as the private-key setting. In the private-key setting, two parties share some secret information called a key, and use this key when they wish to communicate secretly with each other. This shared key serves to distinguish the communicating parties from any other parties who may be eavesdropping on their communication (which is assumed to take place over a public channel). This explains why this setting is known as the private-key setting, where the symmetry lies in the fact that both parties hold the same key which is used for both encryption and decryption. Whereas in contrast to the setting of public encryption, different keys are used for encryption and decryption. For private setting, communicating parties must have some way of initially sharing a key in a secret manner either by a physical meeting or by sharing the key through a private channel. In fact, most of the modern communication protocols employ a hybrid of public and private-key schemes, where the public-key scheme is used to fulfill the initial prerequisites like key exchange, and then, the actual communication takes place using private-key schemes. While both can serve same cryptographic goals but the private-key ciphers are always have the advantage over the public-key ciphers for ease of implementation and high-performance.

Cryptology includes the study and practice of both cryptography and cryptanalysis, where the later consists of techniques used to analyze a cipher so as to gain some useful information which may help in breaking it. In short, cryptanalysis is the science of breaking those encrypted messages to recover their meaning. Here, we are concerned about both making and breaking a cipher which indirectly implies the importance of the design and their subsequent analysis. The analysis generally proceeds in two directions where one concentrated on theoretical aspects of the design, called as the classical attacks under the black-box model assumptions and the other relying on information leaked due to implementation, called as the physical attacks under the gray-box model assumptions. In the black-box model, the assumption is made that an adversary only has access to the inputs and outputs of a cryptographic cipher. In the gray-box model, the attacker has access to a cipher's implementation. This makes gray-box models more realistic than black-box models for applications such as smart cards, embedded systems, and virtualized systems, to which attackers often have physical access and can thus tamper with the algorithms' internals. The implementation attacks were first introduced in the seminal work of Boneh, DeMillo, and Lipton [5] and Kocher [6]. This started the field of physical attacks on cryptographic

devices. The introduction of physical attacks moved the field of cryptography from the black-box model to a gray-box model. These physical attacks can generally be split up into two classes. The first one is passive where an attacker only observes the operation of the device. The other ones are active attacks where the attacker actively manipulating the device operation to extract secret information from it. Active attacks, also called fault attacks, aim to disturb the device's operation outside its normal operating conditions so that the device can output erroneous results. Based on various classical cryptanalytic techniques as well as fault-based side-channel analysis, this work primarily focuses on scrutinizing some of the modern-day lightweight and authenticated ciphers.

1.1 Private-key Cryptography

Historically, classical ciphers work on letters rather than on bits using computers and are much simpler than modern private-key ciphers. Some of the famous classical ciphers are Caesar cipher, Vigenère cipher, and Vernam cipher (*The One-Time Pad*). The Caesar cipher, named after Julius Caesar, is a substitution cipher that shifts each letter of the plaintext by a fixed number of positions down the alphabet. The Vigenère cipher improves on this by using a repeating keyword to determine the shift for each letter, making it more secure against frequency analysis. The Vernam cipher, or *The One-Time Pad*, where a plaintext is paired with a random secret key and offers the highest level of security by using a random key as long as the message itself when the key is used only once and kept secret.

Abstractly, these ciphers have mainly two main components: permutation and mode of operation. A permutation is a function that has a unique inverse and a mode of operation is an algorithm that internally uses a permutation to process data of arbitrary size. In today's world, most of these classical ciphers can be easily broken using simple computer programs that analyze the statistical distribution of the alphabet. Although *The One-Time Pad* is perfectly secure encrypting large data becomes impractical because it needs a very large key as the given data size. A perfectly secure ciphers are such ciphers that are provably secure even against an adversary who has unbounded computational power. Thus, to need a cipher to be both secure and usable, we need some security notion in a combination of some security goals with some attack model.

A private key encryption algorithm is a type of encryption that uses a single secret key for both encryption and decryption. These algorithms can be broadly categorized into two

classes: stream ciphers and block ciphers. Stream ciphers are designed to encrypt individual characters or bits of a plaintext message one at a time. The synchronous stream cipher generates a stream of pseudorandom bits independently of the plaintext and ciphertext messages from the key and Initialization vector (IV). It then uses the stream to encrypt the plaintext by making XOR/modular addition with the pseudorandom bits. Additionally, the self-synchronous stream cipher is another approach that uses several of the previous ciphertext bits to compute the keystream bits, such as ciphertext-feedback mode (CFB). The encryption process is typically performed in real time as the plaintext is being transmitted. One of the key features of stream ciphers is that they can encrypt plaintext of any size, and the encryption process is simple and efficient. Block ciphers, on the other hand, encrypt a fixed-size group of characters (usually 64 or 128 bits) at a time. These ciphers mix the plaintext block with key bits to produce a ciphertext block of the same size using a fixed encryption function. The encryption process is typically performed on an entire message or block of data at once, which makes it more secure than stream ciphers. Block ciphers are typically slower than stream ciphers when implemented in software due to their more complex encryption process. However, the assertion that stream ciphers are faster than block ciphers on hardware platforms may be debatable. For instance, to optimize AES encryption and decryption, modern CPUs include AES-NI (Advanced Encryption Standard New Instructions), which often outperforms certain efficient stream cipher designs in terms of speed and efficiency.

In the absence of dedicated hardware instruction sets for block ciphers, stream ciphers are typically faster and have simpler hardware circuits. This advantage arises because stream ciphers encrypt data bit by bit, whereas block ciphers encrypt data in predefined block sizes. This means that stream ciphers can encrypt data in real time, while block ciphers typically need to wait until they have enough data to form a complete block. However, when block ciphers are used in Counter (CTR) mode, they can encrypt data similarly to stream ciphers. CTR mode generates a keystream that can be XORed with plaintext bits or bytes as they come, allowing for real-time encryption without waiting for a complete block. In summary, both stream ciphers and block ciphers have their own advantages and disadvantages. Stream ciphers are simple, efficient, and can encrypt data in real time, while block ciphers are more secure but slower and more complex. The choice of which one to use depends on the specific requirements of the application.

1.1.1 Attack Models

In cryptography, an attacker attempts to break a cipher by analyzing the ciphertext and trying to obtain the key. This is done by exploiting any vulnerabilities in the encryption and decryption algorithms, or by accessing the system in various ways. According to Kerckhoffs's principle [7], it is generally assumed that the encryption and decryption algorithms themselves are publicly known and available to all. To evaluate the security of modern cryptographic algorithms, cryptographers use different attack models to estimate the capabilities of an attacker. An attack model is a set of assumptions about how the attackers might interact with a cipher so that modern cryptographic algorithms and their implementations are analyzed in order to estimate their security. Cryptographers currently use three common attack models to estimate an attacker's capabilities.

The traditional security and attacker model is the black-box model, where cryptographic implementations are considered as a black-box and the attacker can only observe the input and output behavior. In this model, the attacker's power is limited to making queries to the cipher, which is the operation that sends input to some function and gets the output in return, without exposing the details of that function. There are several different black-box attack models, ranging from the weakest to the strongest, based on the amount of access the attacker has to the system.

Ciphertext-only Attack (COA): In this model, the attacker can only see the ciphertext and cannot perform any encryption or decryption operations.

Known-plaintext Attack (KPA): In this attack model, the attacker can see both the ciphertext and the corresponding plaintext, which are assumed to be randomly selected. The attacker is given a list of plaintext-ciphertext pairs.

Chosen-plaintext Attack (CPA): In this attack model, the attacker can select any plaintext of their choice and perform encryption on it to observe the resulting ciphertext. This model is considered stronger than the previous ones, as the attacker can choose which plaintexts they want to encrypt and observe the corresponding ciphertexts.

Chosen-ciphertext Attack (CCA): This attack model is similar to the chosen-plaintext attack, where the attacker has the ability to make the system decrypt any ciphertext of their choice. Mathematically, the ability to choose plaintexts is equivalent to the

ability to choose ciphertexts. However, in practice, the latter model is often considered stronger because it allows the attacker to specifically target the decryption of certain ciphertexts. CCA2 is an enhanced version of the CCA attack, where the attacker can not only choose the ciphertext but also adaptively choose the ciphertexts after seeing the results of the previous decryption queries. CCA2 is considered to be a stronger attack model than CCA because the attacker has more information and control over the decryption process. In summary, in the security model of CCA, the adversary is permitted to submit decryption queries only before the challenge ciphertext query. In contrast, in the security model of CCA2, the adversary can submit decryption queries both before and after the challenge ciphertext query.

The black-box model, which only allows the attacker to observe the input and output of the encryption and decryption algorithms, is being replaced by the gray-box model. This is due to the increasing use of embedded systems for security purposes on a large scale. The gray-box model includes the limitations of the black-box model but also assumes that the attacker has limited access to the internal workings of the implementation. Gray-box attacks, such as side-channel and fault attacks, focus on the implementation rather than the algorithm, making them more realistic for applications such as smart cards, embedded systems, and virtualized systems, where attackers often have physical access and can tamper with the internals of the algorithm. Based on this partial knowledge, there are several types of attacks that can be launched against a system:

Side-channel attacks (SCA): These attacks exploit information about the system that can be obtained through observing its behavior during operation, such as power consumption or timing information.

Implementation attacks: These attacks exploit specific details about the way the system is implemented, such as the specific algorithm used for encryption or the specific hardware used to implement the system.

Key-recovery attacks: These attacks aim to recover the secret key used for encryption.

It is important to note that in a gray-box attack model, the attacker may have different levels of access and knowledge about the system and the attack can be tailored accordingly.

The white-box model is a relatively new addition to attack models in cryptography, focusing on software implementations of cryptographic algorithms. In 2002, Chow *et al.* [8, 9] introduced the white-box model with initial implementations of white-box AES and DES. In this model, the attacker is assumed to have complete control over the implementation and the execution environment, giving them virtually unrestricted capabilities. The security of implementation under the white-box model is assessed by its ability to maintain security levels similar to those in the black-box model, meaning that even a white-box attacker should not gain any additional advantage over black-box attackers. This model is particularly relevant for software implementations of cryptographic algorithms, where the attacker may have full access to the implementation and the execution environment, and the challenge is to make the algorithm secure under such conditions.

The ideal white-box implementation for cryptographic algorithms would involve a single look-up table mapping plaintexts to ciphertexts, including the hidden secret key. However, such a solution is impractical for modern ciphers with large block and key sizes, like 128 bits or more, due to security and efficiency challenges. Instead, a more practical approach is to use multiple smaller look-up tables and apply secret and invertible encodings to each table individually. This method aims to protect and conceal the secret key material by transforming the cryptographic primitive into a functionally equivalent implementation through these smaller tables and encodings. Despite this approach's practicality for modern ciphers, many table-based white-box implementations have been shown to fall short of required security levels, as discussed in the literature [10, 11].

1.1.2 Security Goals

Indistinguishability (IND) is an important security goal in cryptography as it helps to protect the confidentiality of the plaintext. When a plaintext is encrypted, the resulting ciphertext should be indistinguishable from random noise, making it impossible for an attacker to determine the content of the plaintext by analyzing the ciphertext. This is known as pseudo-random permutation, and it is achieved by using a key to encrypt the plaintext in such a way that the resulting ciphertext is random and unique for every plaintext. This property of indistinguishability is often tested through the IND-CPA experiment, where the attacker selects two plaintexts and is then provided with the ciphertext of one of them, chosen at random. The goal of the attacker is to determine which plaintext was encrypted.

If the attacker is unable to distinguish the encryption of chosen plaintexts, the encryption scheme is said to be indistinguishable. In summary, indistinguishability is a security goal that ensures that the ciphertexts produced by a cipher look like random strings and an attacker should not be able to determine the content of the plaintext by analyzing the ciphertext, which is an important property for maintaining the confidentiality of the plaintext.

Non-malleability (NM) is a security goal that ensures that it is impossible for an attacker to create another ciphertext that corresponds to a plaintext that is related to the original plaintext in a meaningful way. This means that an attacker should not be able to alter the ciphertext in a way that would change the original message in a meaningful way. This is important for maintaining the integrity of the message, as it ensures that the message received is the same as the one sent and that it has not been modified by any unauthorized parties. One example of a malleable encryption scheme is *The One-Time Pad*, where an attacker can XOR the ciphertext with any value and still obtain a meaningful plaintext. This is why *The One-Time Pad* is not considered a secure encryption scheme. In the NM-CCA experiment, an encryption scheme is tested for its non-malleability by evaluating whether an adversary can manipulate ciphertexts to infer information about the encrypted plaintext. The game involves the adversary submitting decryption queries for various ciphertexts, including one chosen at random, and then attempting to forge a new ciphertext. The challenge is to determine if the adversary can produce a related ciphertext that reveals or manipulates the plaintext of the original ciphertext without being able to distinguish it from a random guess. The encryption scheme is considered non-malleable if the adversary's success probability in distinguishing the encrypted plaintext remains negligible. In summary, Non-malleability is a security goal that ensures that it is impossible for an attacker to create another ciphertext that corresponds to a plaintext that is related to the original plaintext in a meaningful way and is important for maintaining the integrity of the message, as it ensures that the message received is the same as the one sent.

Both Indistinguishability and Non-malleability are important security goals in cryptography as they protect the confidentiality and integrity of the message respectively. Cryptographic algorithms are designed to achieve both these properties to provide a secure and reliable means of communication.

1.1.3 Security Notion

To measure the security of a cipher, security goals are combined with an attack model to define a security notion as GOAL-MODEL. Under the black-box attack model, the attacker is assumed to have no knowledge about the system except for the ability to interact with it through a public interface, such as an encryption or decryption oracle. The security notions that are typically considered under this model include:

Indistinguishability under chosen-plaintext attack (IND-CPA): This notion requires that an attacker should not be able to distinguish the encryptions of two different plaintexts, even if they are allowed to choose the plaintexts and observe the corresponding ciphertexts.

Indistinguishability under chosen-ciphertext attack (IND-CCA): This notion requires that an attacker should not be able to distinguish the encryptions of two different plaintexts, even if they are allowed to choose the ciphertexts and request for their corresponding plaintexts in the decryption queries.

Non-malleability under chosen-ciphertext attack (NM-CCA): This notion requires that an attacker should not be able to modify a ciphertext in a way that the resulting plaintext can be related to the original plaintext.

Also, the security notions that are typically considered under a gray-box attack model include:

Indistinguishability under chosen-plaintext attack (IND-CPA) (or chosen-ciphertext attack (IND-CCA)): For IND-CPA, the attacker cannot distinguish the encryptions of two different plaintexts, even if they are allowed to choose the plaintexts and observe the corresponding ciphertexts. Similarly, for IND-CCA, the attacker cannot distinguish the encryptions of two different plaintexts even if they can choose the ciphertexts and obtain the decryption of other ciphertexts.

Non-malleability under chosen-ciphertext attack (NM-CCA): The attacker should not be able to modify a ciphertext in a way that the resulting plaintext can be related to the original plaintext. This ensures that the ciphertext cannot be altered without detection.

Key-privacy (KP-CPA) or Key-privacy under chosen-ciphertext attack (KP-CCA): This notion requires that an attacker should not be able to learn the secret key used for encryption

Key-indistinguishability (KI-CPA) or Key-indistinguishability under chosen-ciphertext attack (KI-CCA): This notion requires that an attacker should not be able to distinguish the secret key used for encryption from a random key.

The security notions typically considered under a white-box attack model include:

Code Obfuscation: Making the code difficult to understand and reverse-engineer, even when the attacker has full access to the source code or binary.

Key Extraction Resistance: Ensuring that the attacker cannot extract the secret key from the implementation, despite having full access to the code and runtime environment.

Code Lifting Resistance: Ensuring that the cryptographic implementation cannot be easily extracted and reused in another environment by an attacker.

Dynamic Analysis Resistance: Protecting against attacks that involve monitoring and manipulating the running program to understand its inner workings and extract secrets.

Tamper Resistance: Protecting the implementation from being altered or tampered with by an attacker.

These security notions define the ability of the attacker to manipulate the plaintext or ciphertext in a way that it could be distinguished or tampered with. In general, when evaluating the security of a cryptographic system, it is important to consider both the security goals and the attack models that are relevant to the system's intended use. More details can be found in the following books [12, 13, 14]. However, since this thesis does not focus on white-box designs or related cryptanalysis, we will omit the security notions specific to white-box models.

1.1.4 Other Private-key Primitives

Besides stream or block ciphers, there are many other private-key cryptographic primitives. These schemes are employed in defense communications, banking operations, and, probably the most important of them all, network protocols such as SSH [15], SRTP [16], TLS [17], WPA2 [18] etc.. Since a block cipher can only handle a fixed-size plaintext, in order to encrypt messages of arbitrary length, a mode of operation specifies how the message is processed. In general, it involves partitioning a message into blocks of suitable length for the underlying block cipher, padding the message block to the required block length when necessary, and iteratively processing the message blocks using block cipher encryption. The private-key ciphers protect primarily the confidentiality of the transmitted information. Besides confidentiality, there are two other properties that are important in practice, the integrity and authenticity of the information. Any private-key scheme is also expected to guarantee either confidentiality or authenticity (and integrity), or both. Authenticated encryption (AE) aims at combining the goals of privacy and authenticity under a single function. Conventionally, an encryption scheme offers confidentiality while a message authentication code (MAC) provides authentication and message integrity. An authenticated cipher tries to integrate both these primitives together. Initially, there have been various attempts to address the challenge of designing efficient authenticated ciphers providing a reasonable security margin. However, the exposure of serious vulnerabilities [19, 20, 21] in OpenSSL and TLS highlighted the lack of proper understanding of the problem. Thus the need for well-studied innovative designs lead to the initiation of a public competition CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness [22]. The CAESAR competition began in 2014 with 57 proposals submitted from all around the world. The competition has announced its 7 finalists in March 2018. Also, the ever-increasing collection of IoT devices interconnected with each other through the Internet, which includes countless small and constrained devices such as Radio-Frequency IDentification (RFID) tags and wireless sensor nodes. This gives rise to a new subfield of cryptography– `lightweight cryptography`. On a day-to-day life, there is a shift from general-purpose computers to dedicated resource-constrained devices. Unfortunately, conventional cryptographic

primitives may not be suitable for such devices as they can only dedicate a small portion of the already constrained hardware for security purposes. Even when it can be implemented, the performance may not be acceptable. In 2019, NIST [23] initiated a process to solicit, evaluate, and standardize lightweight cryptographic algorithms suitable for use in constrained environments. Initially, it has received 57 submissions out of which 56 were accepted for Round 1. Then, after the evaluation, 32 candidates were selected for Round-2 from which 10 finalists have recently been announced. Finally, NIST-LwC announced Ascon as the winner of this competition.

1.1.5 Cryptanalysis

Cryptanalysis is the process of breaking a cryptographic system in order to gain access to the plaintext or secret key without the knowledge of the encryption key. In the case of symmetric key ciphers, cryptanalysis is the process of attempting to determine the secret key used for encryption. There are several types of cryptanalysis techniques that can be used against symmetric key ciphers. Some of these techniques are as follows.

- Brute-force attack: This type of attack involves trying every possible key until the correct one is found. This technique is computationally infeasible for large key sizes.
- Differential cryptanalysis: This type of attack involves studying the difference between the encryption of two plaintexts in order to determine information about the secret key.
- Linear cryptanalysis: This type of attack is based on linear approximations of the encryption function, which allows the attacker to determine information about the secret key.
- Algebraic Cryptanalysis: This attack involves solving algebraic equations derived from the encryption algorithm. It is used to find the key by exploiting algebraic structures in the cipher.
- Meet-in-the-Middle Attack: This technique is used primarily against encryption schemes that use multiple stages or keys, such as double encryption. The attacker encrypts from the plaintext side and decrypts from the ciphertext side

simultaneously, storing intermediate results for comparison. This significantly reduces the time complexity compared to a brute force attack, making it a powerful method for breaking schemes with multiple layers of encryption.

It's important to note that the effectiveness of these techniques depends on the specifics of the symmetric key cipher in question and the amount of information available to the attacker. In general, the security of symmetric key ciphers can be enhanced by increasing the key size and the complexity of the encryption algorithm, but these methods also make the encryption process more computationally expensive.

1.1.5.1 Brute-force Attack

A brute-force attack is a type of *generic attacks* in cryptanalysis in which an attacker systematically tries all possible combinations of characters or words in an attempt to guess a password or key. These attacks can be automated and run through a computer program, making them efficient and fast. They are often used to crack encryption or other types of security mechanisms and can be especially effective against weak or easily guessable passwords. The attack exhausts all possible combinations, rather than applying any strategy to reduce the search space, also called an exhaustive search.

1.1.5.2 Differential Cryptanalysis

Differential cryptanalysis is a powerful method for analyzing and attacking cryptographic systems, particularly symmetric-key ciphers. It works by studying the difference between pairs of plaintext and ciphertext pairs, known as "differences". By analyzing the patterns of these differences, differential cryptanalysis aims to either distinguish the ciphertext from random texts or to determine the secret key used in the encryption process. Note that most key recovery attacks depend on the distinguisher. This technique was first introduced by Biham and Shamir in [24].

The basic idea behind differential cryptanalysis is to find a set of plaintext and ciphertext pairs that have a specific relationship between their differences. This relationship is known as a "differential" and is specific to the encryption algorithm being used. *Differential Characteristics* describe the propagation of differences through multiple

rounds, whereas *Differential Trail* provides detailed instances of these characteristics, including specific intermediate differences. By analyzing the differential characteristic, the cryptanalyst can determine the internal state of the encryption algorithm and, with enough data, the secret key used. This leads to a distinguishing and even a key-recovery attack.

1.1.5.3 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack that takes advantage of a potential linear bias in a cipher. Developed by Matsui [25], it involves expressing certain plaintext and ciphertext bits as a linear Boolean expression, which holds true (equals 0) with probability p. The stronger the bias (p is further away from the probability $\frac{1}{2}$), the more successful the attack is. The attack uses a linear mask to extract specific bits from the plaintext or ciphertext to form the linear expression. The method involves collecting many plaintext-ciphertext pairs and detecting biases, which can then be used to conduct a distinguishing or key-recovery attack.

1.1.5.4 Side-channel Attack

Side-channel analysis is a type of attack in which an attacker observes physical information leaked from a target device through a side channel. This information, known as side-channel information, contains sensitive information about the intermediate values of a cryptographic operation. By analyzing this information and understanding the relationship between intermediate values and the side-channel information, the attacker can potentially disclose the intermediate value, which can often lead to the recovery of the secret key used in the cipher.

There are several types of sources of leakage for side-channel attacks, including power analysis, electromagnetic analysis, Timing Analysis, and Cache Attacks. These leakages can be used to extract encryption keys, passwords, and other sensitive information from a wide range of systems, including smart cards, cryptographic devices, and embedded systems.

To protect against side-channel attacks, it is important to use cryptographic techniques and protocols that are resistant to side-channel attacks, such as masking and blinding, as well as to use secure hardware and software designs that can reduce the

amount of information leaked through side channels. Additionally, regular monitoring and testing of the systems to detect any signs of side-channel attacks.

1.1.5.5 Fault Attacks

Fault attacks are one of the types of implementation attacks that fall under the category of active attacks. Fault attacks disturb the calculations of the cryptographic algorithms intentionally, i.e., the attackers force some known property of the intermediate values to occur in the target device. The assumption of the active attack implies the alternation of the original calculation of the cryptographic algorithms by performing the so-called fault injections. More precisely, for the active attacks, the attackers need to interact with the target device to perform the key recovery. Targeting a cryptographic device is a two-step process. The first one is the fault injection step where an attacker needs to inject faults into the device and collect faulted outputs. The second one is the fault analysis step where the collected outputs need to be processed in order to retrieve the secret key.

The first academic publication on fault-based cryptanalysis, by Boneh, DeMillo, and Lipton [5], pointed out that faults can be used to break cryptographic algorithms. Immediately after this work, Biham and Shamir first demonstrated the vulnerability of private-key ciphers by presenting several attacks on block ciphers, including differential fault analysis (DFA). Differential fault analysis is a combination of round-reduced differential cryptanalysis with pairs of correct and faulty ciphertexts. Simply speaking, DFA requires an attacker to to run an encryption of the exact same input twice, where the first one is fault-free and the second one is faulted. Once a sufficient amount of faulty and correct ciphertext pairs have been collected apply the differential cryptanalysis technique to retrieve the secret key. Further, the number of required faults to recover the key will depend on the structure of the cipher and the round the attacker chooses to target. Also, some other state-of-the-art fault attack techniques have been applied to break the private-key ciphers. These techniques are as follows: Collision Fault Analysis (CFA) [26], Ineffective Fault Analysis (IFA) [27], Safe-Error Analysis (SEA) [28], Fault Sensitivity Analysis (FSA) [29], Linear Fault Analysis (LFA) [30], Statistical Fault Attack (SFA) [31], Statistical Ineffective Fault Attack (SIFA) [32, 33], Statistical Effective Fault Attack (SEFA) [34].

1.1.5.5.1 Differential Fault Attack Differential Fault Attack (DFA) has been introduced by Biham and Shamir [35] in 1997. Since then DFA has grown in both depth and breadth and is considered one of the most effective forms of physical attacks. The basic idea to perform DFA is to collect non-faulty and faulty outputs and then apply differential cryptanalysis to recover its key. An important requirement of DFA is the ability to replay the execution of the cipher in order to exploit the difference between faulty and non-faulty outputs. This is referred to as the *Replaying criterion.*

1.2 Motivation and Objectives

Authenticated Encryption (AE) is a cryptographic primitive that provides both confidentiality and integrity to data. It is designed to ensure that the data has not been tampered with and comes from a trusted source. Since its inception, there have been several attempts to improve its design and analysis. One of the most significant events in the history of AE is the announcement of the CAESAR competition [22] in 2013 and the NIST LwC [36] competition in 2019. These competitions invited researchers and practitioners to submit new and innovative designs for AE, with the aim of improving its security and efficiency. As a result, there has been a surge in research in this domain, leading to the development of several new schemes. One of the challenges in analyzing AE schemes is that they must defend against a variety of attacks, including traditional cryptographic attacks, fault attacks, and side-channel attacks, and they must also provide misuse resistance. Traditional attacks such as ciphertext-only, known-plaintext, and chosen-plaintext attacks are well-known, and many cryptographic schemes are designed to withstand them. Some AE schemes are also designed to remain secure even if nonces are reused. This helps prevent major failures if nonces (numbers used once) are accidentally repeated. Therefore, AE schemes should ensure security even in the case of nonce reuse. However, fault/side-channel attacks are a relatively new type of attack that can be used to exploit weaknesses in the implementation of the scheme. Fault attacks involve deliberately inducing errors or faults in the implementation of the cryptographic scheme to reveal its secrets. These attacks can be challenging to detect and can result in the disclosure of sensitive

information. Therefore, it is essential to analyze AE schemes for their vulnerability to fault attacks.

Differential fault attacks (DFA) involve introducing errors into the computation of the encryption or decryption function, potentially revealing the secret key or other sensitive information. For nonce-based AE schemes, the use of a unique nonce in the encryption process prevents attackers from replaying the algorithm and introducing errors in an encryption query, thus making the scheme resistant to differential fault attacks. This work mainly focuses on performing DFA on several nonce-based AE schemes to reveal vulnerabilities under gray-box attack models and retrieve the key. The other research work of this thesis is based on classical cryptanalysis of the nonlinear feedback shift register (NLFSR)-based block ciphers.

Non-linear Feedback Shift Registers (NLFSRs) are frequently used as components in the design of cryptographic primitives, including lightweight primitives like stream ciphers, block ciphers, and authenticated encryption (AE) schemes. Initially, NLFSRs were introduced in modern stream ciphers for applications such as RFID and smart-cards. They are particularly well-suited for lightweight cipher designs because they offer a compact, low-power, and fast solution ideal for real-time protocols. Recently, cryptographers have proposed several NLFSR-based block ciphers and AE designs for these reasons. These ciphers have been the focus of much research in recent years, as they are widely used in resource-constrained environments such as Internet of Things (IoT) devices. One of the methods used to analyze the security of cryptographic primitives is the use of mathematical modeling, such as MILP-based cryptanalysis. MILP-based cryptanalysis involves the use of linear and integer programming techniques to solve a system of equations that model the cryptographic primitive. The aim is to find a differential or linear characteristic that has a high probability of occurring in the cipher and can be exploited to recover the secret key. In our research, we used MILP-based differential cryptanalysis to analyze two lightweight ciphers, TinyJAMBU and KATAN , both of which are NLFSR-based. TinyJAMBU is a finalist in the NIST lightweight cryptography competition, while KATAN is a widely used block cipher in IoT devices. The MILP-based differential cryptanalysis approach allowed us to model the ciphers as a set of linear equations and then solve them to find differentials with

a high probability of occurrence. Overall, our research highlights the importance of using mathematical modeling techniques such as MILP-based cryptanalysis to analyze the security of lightweight ciphers such as TinyJAMBU and KATAN . This approach can help identify potential weaknesses in the ciphers and improve their overall security.

1.3 Thesis Outline and Our Contributions

In the thesis, the focus is on the cryptanalysis of private key ciphers using both traditional and physical attack techniques. The contributions are presented in Chapter 3 onwards and can be broadly grouped into two parts: Difference-based Fault Analysis on AE Schemes and Cryptanalysis of NLFSR-based ciphers. Chapters 3 to 6 deal with the first part of the contribution which is focused on the Difference-based Fault Analysis on AE Schemes. In particular, the study investigates the vulnerability of AE schemes to differential fault attacks. The research identifies a simple property of AE schemes that seemed to protect these ciphers implicitly from being exposed to differential fault attacks. The study shows that most of the ciphers submitted to the CAESAR and NIST competitions are resilient to such attacks, even in the presence of faults. Chapter 7 discusses the second part of the contribution which is the cryptanalysis of NLFSR-based ciphers. Finally, Chapter 8 concludes the thesis by discussing potential issues that may arise from the research presented.

Overall, the thesis contributes to the research on the cryptanalysis of private key ciphers using traditional and physical attack techniques. The research highlights the importance of using mathematical modeling techniques such as MILP-based cryptanalysis to analyze the security of lightweight ciphers and identify potential weaknesses that can be exploited. The study also identifies a property of AE schemes that protect them implicitly from being exposed to differential fault attacks, which is an important finding in the field of authenticated encryption. The following subsections give a brief account of the results obtained during this research.

1.3.1 Difference-based Fault Analysis on AE Schemes

The difference-based analysis is somewhat similar and can be considered as an extension of DFA. Whereas, the DFA can be regarded as an extension of the classical differential attack and it requires an attacker to be able to inject faults in a cryptosystem while replaying a previous fault-free run of the algorithm. Generally speaking, DFA or its variants can be successfully applied to any unprotected deterministic private-key block or stream ciphers. Also, DFA is successfully applied to other kinds of primitives like hash functions, message authenticated encryptions (MAC), and authenticated encryptions (AE) as well. Although, researchers have proposed several state-of-the-art countermeasures to inhibit such kind of difference-based fault attacks. Some of the well-studied countermeasures in symmetric key cryptography include *Detection*, *Infection*, and *Prevention*. These strategies often use redundant computations to enhance security. Detailed explanations of these methods can be found in the survey work [37].

In this part of the work, we address an interesting problem where a simple property of cryptographic design proves to be a great inhibitor for one of the most effective forms of physical attacks. Here the former refers to the paradigm of Nonce-based design while the latter implies the area of fault-based cryptanalysis. The problem posed by a nonce derives from his definition itself. Nonce-based encryption was formalized by Rogaway in [38] where it is assumed that security is provided as long as the nonce does not repeat. One of the interesting consequences of a single nonce in an encryption algorithm is to ensure that old communications cannot be reused in replay attacks. Also, the use of a nonce in a cipher leads to a direct contradiction to the ability to replay a cipher and thereby resulting in automatic protection from DFA in the encryption query. Moreover, in public key cryptography, nonces have been used to counteract fault attacks. Though it has been shown in some limited settings that these nonces can be tackled but the techniques used to rely on theoretical constructs that may not directly work with their private-key counterparts. Thus, nonce-based encryption seems to have an in-built protection against DFA.

In the classic security concepts of Authenticated Encryption (AE), the delivery of the decrypted plaintext is subject to successful verification. In their pioneering ar-

ticle [39] in Asiacrypt 2014, Andreeva et al. questioned this model by introducing and formalizing the idea of releasing unverified plaintexts (RUP). The basic idea is to separate the plaintext computation and verification during AE decryption so that the plaintexts are always released irrespective of the status of the verification process. In such a scenario, the nonce can be easily bypassed by choosing the decryption query which opens up new avenues for attacking the decryption module [40]. Further differential fault attacks [41, 42] on nonce-based AE schemes are applied either by misusing the nonce or by avoiding the nonce using the internal DFA to parallel counter-mode encryptions.

In this part of the thesis, we explore other ways to overcome the nonce effect in DFA. We focus primarily on the emerging area of authenticated encryption schemes, although most of the results obtained meet a greater subset of cryptographic primitives. We begin by identifying certain subdomains of authenticated encryption systems based on design principles. Then attempt to exploit certain inherent properties to re-establish the *Replaying Criterion* that makes the DFA viable in the presence of a nonce. Finally, to illustrate the ideas to perform the DFA, we are interested in some specific types of SPN-based AE design modes. The case studies conclude with the practical set-up of differential fault attacks on some of the selected designs. The investigation has been done in three ways.

1.3.1.1 Analysis of NORX using Variants of Coupon Collector Problem

In chapter 4, we first show the differential fault attack on CAESAR scheme NORX with a level of parallelism $p \in \{2, 4\}$(applicable to all the versions v1, v2.0, v3.0) in the presence of a nonce. This demonstrates a scenario when faults introduced in NORX in parallel mode can be used to collide the internal branches to produce an all-zero state. Later, this fault is used to replay on NORX despite being instantiated by different nonces and messages. Once replayed, the secret key of NORX can be recovered using secondary faults and the faulty tags. The attack presents a case where for the first time both internal and classical differentials are used to mount a DFA on a nonce-based authenticated cipher. Different fault models are used to showcase the versatility of the attack strategy. A detailed theoretical analysis of the

expected number of faults is furnished under various models. Under the random bit-flip model, around 1384 faults need to be induced to reduce the key space from 2^{128} to 2^{32}, while the random byte-flip model requires 332 faults to uniquely identify the key. Moreover, Finally, we furnish a discussion to assess the DFA vulnerability of FORK-256-256 based on a strategy similar to the one used for NORX.

Moreover, we have identified and provided a solution to a new theoretical problem for the consecutive bit-flip fault model that is a special variant of the *generalized coupon collector problem*. The problem essentially states the following: *If an attacker repeatedly injects consecutive bit faults in an n-bit register and in each trial, he randomly chooses a bit from the register and then flips k bits* ($k \leq n$) *to the right of the bit, then what is the expected number of trials so that t* ($1 \leq t \leq n$) *number of bits get flipped*. We refer to the new problem as the *non-circular consecutive coupon collector problem*. We present a mathematical solution to this problem for the first time in the literature. Additionally, we corroborate that our theoretical values are matched very closely to the simulated values. Further, we show the validation of our calculations of the problem using hypothesis testing. We additionally extend the problem to the Circular variant and refer to this problem as *circular consecutive coupon collector problem*. This problem may be of independent interest to the community. For this circular variant, we provide the mathematical proof of this problem and also validate it using hypothesis testing. We will discuss these variants of coupon collector problem and their proofs completely in chapter 3.

Publication History: Chapters 3–4 are based on our works [43], published at ASHES@CCS and its extended work [44], accepted in Journal of Cryptographic Engineering.

1.3.1.2 Analysis on Feistel-based Sponge AE Schemes

In Chapter 4, we present a problem with a certain type of encryption called authenticated encryption that uses a technique called sponge mode of operation. We explain that carrying out a differential fault attack on this encryption is challenging. This difficulty arises because of the use of a unique value, known as a nonce, which changes with each encryption and is used only once. However, we have found that if

the nonce is repeated through multiple decryption queries, it becomes possible to perform an attack. The sponge duplex mode, which is the mode of operation in question, is commonly used in constructing authenticated encryption schemes. Many of the submissions to the current NIST lightweight cryptography standardization process, which is a process for evaluating and selecting new encryption standards, are based on this mode. A majority of NIST-LwC submissions use Substitution Permutation Networks or Feistel-like structures in their underlying permutation. This means that a large number of encryption schemes that are being considered for standardization are potentially vulnerable to this attack.

In this work, we are looking into a specific type of encryption called Generalized Feistel Networks (GFN) that is used in a technique called sponge AE construction. It is also assumed that the internal round function used in GFN follows an SPN-like structure. We observed that by intentionally introducing small errors or faults during the decryption process, it may be possible to recover the key used in the encryption. We then present an attack on a specific type of encryption called CiliPadi family of authenticated encryption scheme, under a random fault model. This attack is used to retrieve the state of the encryption and ultimately recover the secret key. Further, we demonstrate that our method is effective and efficient, showing that it is possible to recover the secret key with a small number of faulty queries and low time and memory complexities. Thus, we propose a generalized method for any GFN-based sponge AE where SPN is internally used inside the GFN. To recover the internal state of this encryption, we present two different fault attacks. Both attacks involve performing faulty forgery at the final permutation call under two different fault models. We also provide a theoretical analysis of how to perform these faulty forgeries and propose general countermeasures against these types of fault attacks. To the best of our knowledge, this is the first report of a fault attack on GFN-based sponge AE.

Publication History: Chapters 5 is based on our work [45], published at Journal of Hardware and Systems Security.

1.3.1.3 Analysis on Sponge and SIV-like AE Schemes

This work presents the first successful differential fault attack (DFA) on the nonce-based AE scheme PHOTON-BEETLE, which is one of the finalists in the ongoing NIST LwC competition. Furthermore, this work reports the first DFA on other sponge and SIV-based NIST LwC schemes, namely ORANGE, SIV-TEM-PHOTON, and ESTATE. Although performing DFA on any nonce-based sponge/SIV-based AE is challenging due to the unique nonce in the encryption query, the decryption procedure (with a fixed nonce) remains vulnerable to DFA. To accomplish the attack, we propose various fault attack models and estimate the number of faulty queries required to obtain multiple forgeries. Our simulated values corroborate closely the theoretical estimates. Finally, we devise an algorithm to recover the state using the collected forgeries.

Under the random fault attack model, this work reports that approximately $2^{37.15}$ faulty queries are needed to retrieve the secret key. The offline time and memory complexities of this attack are respectively 2^{16} and 2^{10} nibbles. In contrast, under the random bit fault attack model, this work reports that around $2^{11.5}$ faulty queries are necessary to retrieve the key for PHOTON-based schemes, while $2^{13.1}$ faulty queries are needed for AES-based scheme ESTATE.

Also, in the known fault attack model, this work reports that approximately $2^{11.05}$ faulty queries are required to retrieve the secret key for PHOTON-based schemes, while $2^{13.01}$ faulty queries are necessary for AES-based scheme ESTATE. The time and memory complexities of the state recovery attack (for PHOTON-based schemes) are respectively 2^{11} and 2^{9} nibbles. Moreover, this work successfully reduces the number of faulty queries required to $2^{9.32}$ under the precise bit-flip fault model.

PUBLICATION HISTORY: Chapter 6 is based on our work [46], which was published at ASHES@CCS, and its extended version [47] is published in Journal of Cryptographic Engineering.

1.3.2 Traditional Cryptanalysis of NLFSR-based Lightweight Ciphers: TinyJAMBU and KATAN

Since Mouha *et al.*'s pioneering work [48] showcased the power of Mixed Integer Linear Programming (MILP) in automated cryptanalysis, it has taken center stage in this arena. Research in this area has primarily moved in two directions: modeling classical cryptanalysis tools as optimization problems to leverage state-of-the-art solvers, and improving existing models to make them more efficient and/or accurate. In this work, we aim to contribute to the latter by devising a general model referred to as "DEEPAND" that captures the correlations among multiple AND gates in NLFSR-based lightweight block ciphers.

DEEPAND builds upon and generalizes the idea of joint propagation of differences through AND gates, captured using refined MILP modeling of TinyJAMBU by Saha *et al.* [49] in FSE 2020. The proposed model has been applied to TinyJAMBU and KATAN and can detect correlations that were missed by earlier models. This leads to more accurate differential bounds for both ciphers.

This new model has found a 384-round (full-round as per earlier specification) Type-IV trail for TinyJAMBU with 14 active AND gates, while the refined model reported this figure to be 19. This reaffirms the decision of the designers to increase the number of rounds from 384 to 640. Moreover, the model succeeds in searching a full-round Type-IV differential characteristic of TinyJAMBU keyed permutation $\mathcal{P}_{1024}$ with probability 2^{-108} (much greater than 2^{-128}). This reveals the non-random properties of $\mathcal{P}_{1024}$, thereby showing it to be non-ideal. Hence it cannot be expected to provide the same security levels as robust block ciphers. Further, the provable security of TinyJAMBU AEAD scheme should be carefully revisited.

Similarly, for KATAN32 , DEEPAND modeling improves the 42-round differential with 2^{-11} probability to 2^{-7}. Also, for KATAN48 and KATAN64 , this model, respectively improves the designer's claimed 43-round and 37-round differential probabilities. Moreover, in the related-key setting, the DEEPAND model is able to make a better 140-round boomerang distinguisher (for both the data and time complexity) in comparison to the previous boomerang attack by Isobe *et al.* in ACISP 2013. In summary, DEEPAND seems to capture the underlying correlation better when multiple

AND gates are at play and can be adapted to other classes of ciphers as well.

Publication History: Chapter 7 is based on our work [50], which is currently under submission.

2 BACKGROUND

This chapter begins by revisiting fundamental design paradigms and exploring cryptographic tools that are relevant to the research conducted in this thesis.

2.1 Generalized Feistel Networks (GFN)

A classical Feistel network is a general method of constructing any function into a permutation. It was invented by Horst Feistel in his design of Lucifer [51], and has been used in many block cipher designs (with some variations) like DES [52], FEAL [53], GOST [54], Khufu and Khafre [55], LOKI [56], CAST [57], Blowfish [58], and RC5 [59]. The fundamental building block of a Feistel network is a key-dependent nonlinear mapping, called an f-function. In design, f can be made arbitrarily complicated, since it does not need to be designed to be invertible.

Traditionally, most of the existing Feistel designs divide the input message block into two parts, whereas GFN divides an input message block into $l \geq 2$ sub-blocks (called it as branches). GFNs can be considered a comprehensive collection of various Feistel-based design variants. Despite their extensive use, the parameterized definition of GFNs is not well formulated. Here, we provide a parameterized definition of GFNs.

2.1.1 Classification of GFN

GFN is a generalized structure for Feistel with an arbitrary number of rounds (we denote it by r), several branches (we denote it by l), and branch lengths (we denote them by $n_0, n_1, \ldots, n_{l-1}$) as well as branch permutations (permutation π to define

which branch is going where). Below, we start with the definition of the basic GFN structure with $l = 2$ and only one internal round function $f \in \mathcal{F}$, called the classical Feistel. The state is divided into left and right branches with lengths n_0 and n_1 respectively. This is the most popular choice for Feistel designs. The structure is parameterized by n_0, n_1 and f. We denote this GFN with $l = 2$ by $\mathsf{GFN}^f[r, n, 2]$. It is defined as follows.

Definition 1 (Two Branch Feistel)**.** Let $l = 2$ and $f : \{0,1\}^{n_0} \to \{0,1\}^{n_1}$ be a function with $n_0, n_1 \in \mathbb{N}$. Consider a permutation $\phi : \{0,1\}^{n_0+n_1} \to \{0,1\}^{n_0+n_1}$ such that $\phi(B_1, B_2) = (B_2, B_1 \oplus f(B_2))$ with $|B_1| = n_0$ and $|B_2| = n_1$. Then a Feistel with two branch ($l = 2$) $\mathsf{GFN}^f[r, n, 2]$ with the input $B_1 \| B_2$ is defined as

$$\mathsf{GFN}^f[r, n, 2](B_1 \| B_2) = \phi^r(B_1 \| B_2).$$

Typically ϕ is called round function. $n_0 + n_1$ is called the block size, denoted by $n (= n_0 + n_1)$, where n_0, n_1 denote branch lengths. If the two branches are of equal size, i.e., $n_0 = n_1 = n/2$, it is called a classical Feistel or a balanced Feistel with $l = 2$, given in Figure 2-1a. Otherwise it is called *unbalanced*, given in Figure 2-1c.

Further generalization of Feistel design for any arbitrary branch number $l \in \mathbb{N}$ with equal-sized sub-blocks $n_0, n_1, \ldots, n_{l-1}$, was proposed in [60]. They had proposed three different types of balanced Feistel designs, called Type-1, Type-2, Type-3 respectively.

Definition 2 (Type-1 Feistel)**.** Let $l \geq 2$ and $f : \{0,1\}^{n_0} \to \{0,1\}^{n_0}$ be a function with $n_0 \in \mathbb{N}$. Consider a permutation $\phi_1 : \{0,1\}^{l \cdot n_0} \to \{0,1\}^{l \cdot n_0}$ with $n = l \cdot n_0$ such that $\phi_1(B_1, B_2, \cdots, B_{l-1}, B_l) = (B_2 \oplus f(B_1), B_3, \cdots, B_l, B_1)$ with $|B_1| = \cdots = |B_l| = n_0$. Then a Type-1 Feistel $\mathsf{GFN}_1^f[r, n, l]$ with the input $B_1 \| \cdots \| B_l$ is defined as

$$\mathsf{GFN}_1^f[r, n, l](B_1 \| \cdots \| B_l) = \phi_1^r(B_1 \| \cdots \| B_l).$$

Definition 3 (Type-2 Feistel)**.** Let $l \geq 2$ is an even number and $f_i : \{0,1\}^{n_0} \to \{0,1\}^{n_0}, i \leq \frac{l}{2}$ be a function with $n_0 \in \mathbb{N}$. Consider a permutation $\phi_2 : \{0,1\}^{l \cdot n_0} \to \{0,1\}^{l \cdot n_0}$ with $n = l \cdot n_0$ such that $\phi_2(B_1, B_2, \cdots, B_{l-1}, B_l) = (B_2 \oplus f_1(B_1), B_3, B_4 \oplus f_2(B_3), B_5, \cdots, B_l \oplus f_{l/2}(B_{l-1}), B_1)$ with $|B_1| = \cdots = |B_l| = n_0$. Then a Type-2

Feistel $\mathsf{GFN}_2^f[r,n,l]$ with the input $B_1\|\cdots\|B_l$ is defined as

$$\mathsf{GFN}_2^f[r,n,l](B_1\|\cdots\|B_l) = \phi_2^r(B_1\|\cdots\|B_l).$$

Definition 4 (Type-3 Feistel)**.** Let $l \geq 2$ and $f_i : \{0,1\}^{n_0} \to \{0,1\}^{n_0}, i \leq l-1$ be a function with $n_0 \in \mathbb{N}$. Consider a permutation $\phi_3 : \{0,1\}^{l\cdot n_0} \to \{0,1\}^{l\cdot n_0}$ with $n = l \cdot n_0$ such that $\phi_3(B_1, B_2, \cdots, B_{l-1}, B_l) = (B_2 \oplus f_1(B_1), B_3 \oplus f_2(B_2), \cdots, B_l \oplus f_{l-1}(B_{l-1}), B_1)$ with $|B_1| = \cdots = |B_l| = n_0$. Then a Type-3 Feistel $\mathsf{GFN}_3^f[r,n,l]$ with the input $B_1\|\cdots\|B_l$ is defined as

$$\mathsf{GFN}_3^f[r,n,l](B_1\|\cdots\|B_l) = \phi_3^r(B_1\|\cdots\|B_l).$$

Another new Feistel design was proposed in [61], called alternating Feistel in Figure 2-1e, where the round functions alternate between contracting and expanding. It is defined as follows.

Definition 5 (Alternating Feistel)**.** For $l = 2$ and $n_0 \leq n_1$, let $f : \{0,1\}^{n_0} \to \{0,1\}^{n_1}$, $f' : \{0,1\}^{n_1} \to \{0,1\}^{n_0}$ are respectively an expanding and contracting functions with $n_0, n_1 \in \mathbb{N}$. Consider a permutation $\phi_{alt} : \{0,1\}^{n_0+n_1} \to \{0,1\}^{n_0+n_1}$ such that $\phi_{alt}(B_1, B_2) = (B_1 \oplus f'(B_2 \oplus f(B_1)), B_2 \oplus f(B_1))$ with $|B_1| = n_0$ and $|B_2| = n_1$. Then an alternating Feistel with two branch ($l = 2$) $\mathsf{GFN}_{alt}^f[r,n,2]$ with the input $B_1\|B_2$ is defined as

$$\mathsf{GFN}_{alt}^f[r,n,2](B_1\|B_2) = \phi_{alt}^r(B_1\|B_2).$$

2.1.2 Parameterized Definition of GFN

We first start by setting up the parameters for generalized GFN. Next we define the round function of GFN and then define all the existing classes of GFN case by case. Let $\mathsf{GFN}^{\mathbf{f}}[r,l,n_0,n_1,\ldots,n_{l-1},q,d]$: $\{0,1\}^n \to \{0,1\}^n$ be the GFN, where the parameters are as follows,

- $r \in \mathbb{N}$ denotes the number of rounds,
- $l \in \mathbb{N}$ denotes the number of branches

– $n_0, \ldots, n_{l-1} \in \mathbb{N}$ denotes the length of the branches in bits such that $n = \sum_{i=0}^{l-1} n_i$ is the block size,

– $q \in \mathbb{B}$, denotes whether all branches are of equal length $\frac{n}{l}$ or not. Precisely, when $q = 0$ then $n_0 = n_1 = \cdots = n_{l-1}$ and $q = 1$ denotes $\exists i, j$ s.t, $n_i \neq n_j$,

– $\mathbf{f}$ is a tuple of l functions $(f_0, \ldots, f_{l-1})$ and $d = d_0 \ldots d_{l-1} \in \{0, 1, 2\}^l$ such that $f_i : \{0,1\}^{n_{i+d_i}} \to \{0,1\}^{n_{i+1-d_i}}, f_i \in \mathcal{F}$ will be assigned to the branches based on the values of $d_i, \forall i$. For $d_i = 0/1$, we assign an internal branch function to the i^{th} branch (direction: branch i to $i+1$ for $d = 0$ and branch $i+1$ to i for $d = 1$). When $d_i = 2$, we assign $f_i = \mathbf{0}$, i.e., no non-identity function will be assigned for the i^{th} branch.

$\mathsf{GFN}^{\mathbf{f}}[r, l, n_0, \ldots, n_{l-1}, q, d]$ takes $x \in \{0,1\}^n$ as the input and is defined as

$$\mathsf{GFN}^{\mathbf{f}}[r, l, n_0, \ldots, n_{l-1}, q, d](x) = (\phi_d)^r(x^0 || \cdots || x^{l-1}),$$

where, ϕ_d is the round function iteartes r times. Each of the iterations uses a branch permutation π_l and an internal round function $\tau_d^{\mathbf{f}}$. Thus by fixing all the parameter values, one round of the GFN is defined as

$$\begin{aligned}
\phi_d(x) &= \phi_d(x^0 || \cdots || x^{l-1}) \\
&= \pi_l \circ \tau_d^{\mathbf{f}}(x^0 || \cdots || x^{l-1}) \\
&= \pi_l(y^0 || \cdots || y^{l-1}) \\
&= (z^0 || \cdots || z^{l-1}).
\end{aligned}$$

Different types of Feistels described above are pictorially represented below.

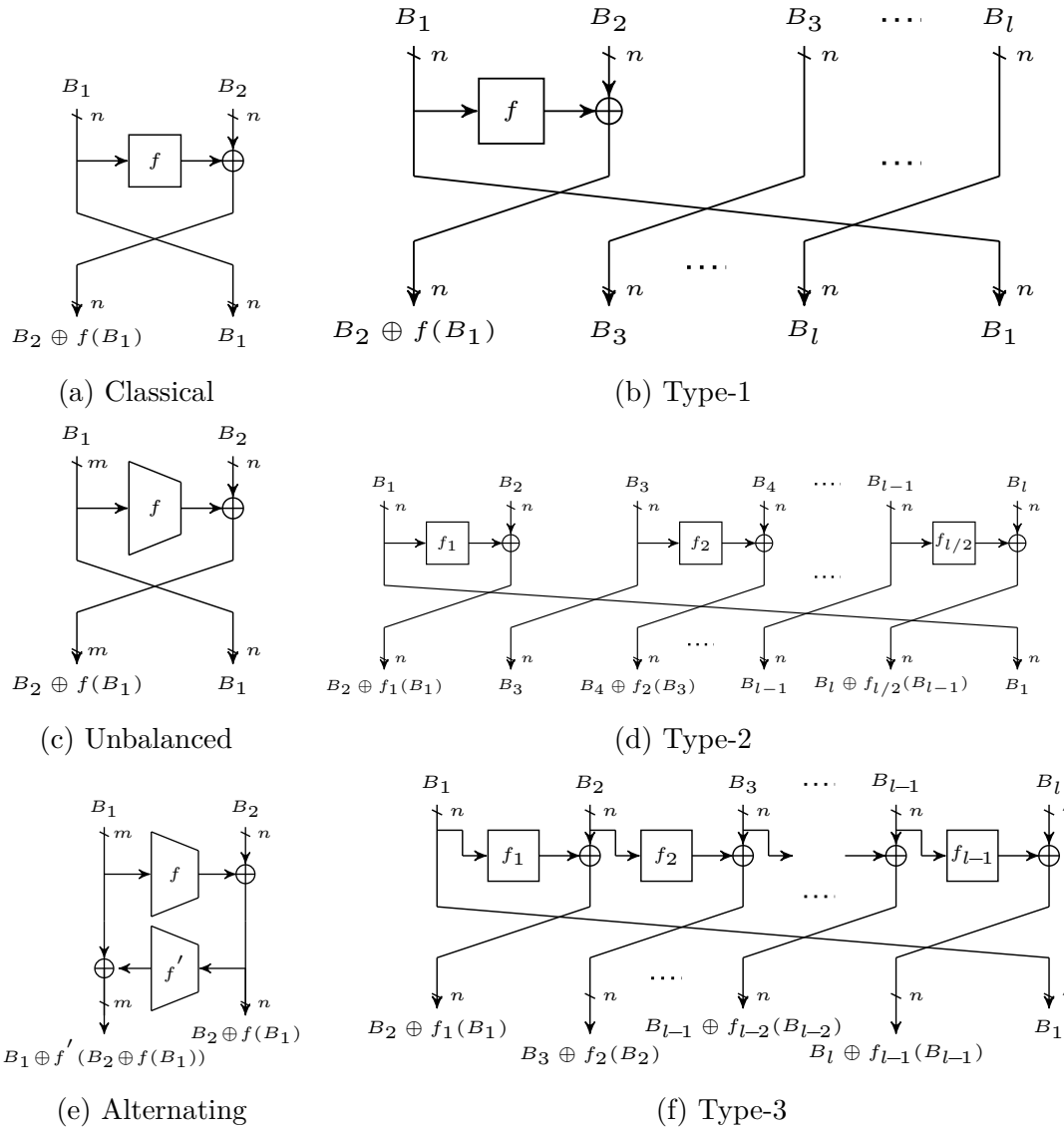
One Round Balanced/Unbalanced GFN: One round of balanced/unbalanced GFN with x_i/x_{i+1} as the input/output is defined as:

$$x_{i+1} = \pi_l \circ \tau_d^{\mathbf{f}}(x_i^0 || \cdots || x_i^{l-1}) = \pi_l(y_i^0 || \cdots || y_i^{l-1}),$$

where $\forall j \in [0, l-1]$, $|x_i^j| = n_j$ and

$$y_i^j = \begin{cases} x_i^j \text{ op } f_j(x_i^{j-1+2d_j}) & \text{if } d_j = 0, 1 \\ x_i^j & \text{if } d_j = 2. \end{cases}$$

It is noted that, if $\forall i, j \in [0, l-1]$, $n_i = n_j$, then it is called balanced GFN. Otherwise, we call it unbalanced GFN. Here if we fix some parameters as $\boxed{\text{l} = 2,\ \text{q} = 0,\ \text{d} = 02,\ \pi_2 = (2, 1)}$, then one round GFN represents a classical balanced Feistel network, given in Figure 2-1a. Similarly, for any values of $\boxed{\text{l} (> 2),\ \text{q} = 0,\ \pi_l = (2, 3, \cdots, l, 1)}$, one round GFN can be represent as Type-1 (Fig-



(a) Classical (b) Type-1

(c) Unbalanced (d) Type-2

(e) Alternating (f) Type-3

Figure 2-1: Generalized Feistel Network

ure 2-1b), Type-2 (Figure 2-1d), Type-3 (Figure 2-1f) for

$\boxed{\text{d} = 02\cdots 22,\ d = 02\cdots 02,\ d = 00\cdots 02}$ respectively.

Also, if we fix inputs as $\boxed{\text{l} = 2,\ \text{q} = 1,\ \text{p} = 10,\ \pi_2 = \{2,1\}}$, then one round GFN represents a classical unbalanced Feistel network, given in Figure 2-1c.

One Round Alternating GFN: For the alternating GFN, we need another tuple of functions $\mathbf{f'} = (f_0^{'}, \ldots, f_{l-1}^{'})$ together with $\mathbf{f}$, where $f_i^{'} : \{0,1\}^{n_i+d_i} \to \{0,1\}^{n_i+1-d_i}$. Let $\mathsf{GFN}_{\mathbf{f}'}^{\mathbf{f}}[r, l, n_0, \ldots, n_{l-1}, q, d]$ be the modified GFN. We also denote $\bar{d} \in \{0,1,2\}^l$, where $\bar{d_i}$ will be defined as follows:

$$\bar{d_i} = \begin{cases} d_i + 1 \mod 2 & \text{if } d_i = 0, 1 \\ 2 & \text{if } d_i = 2. \end{cases}$$

Then $\mathsf{GFN}_{\mathbf{f}'}^{\mathbf{f}}[r, l, n_0, n_1, \ldots, n_{l-1}, q, d]$ with input $x \in \{0,1\}^n$ is defined as

$$\mathsf{GFN}_{\mathbf{f}'}^{\mathbf{f}}[r, l, n_0, \ldots, n_{l-1}, q, d](x) = (\phi_d)^r(x^0||\cdots||x^{l-1}) = (\pi_l \circ \tau_{\bar{d}}^{\mathbf{f}'} \circ \tau_d^{\mathbf{f}})^r(x^0||\cdots||x^{l-1}).$$

Where, ϕ_d is the round function, which can be viewed as the composition of π_l, $\tau_{\bar{d}}^{\mathbf{f}'}$, $\tau_d^{\mathbf{f}}$. By fixing all the parameter values, one round alternating GFN is defined as

$$\begin{aligned} \phi_d(x) = \phi_d(x^0||\cdots||x^{l-1}) &= \pi_l \circ \tau_{\bar{d}}^{\mathbf{f}'} \circ \tau_d^{\mathbf{f}}(x^0||\cdots||x^{l-1}) \\ &= \pi_l \circ \tau_{\bar{d}}^{\mathbf{f}'}(u^0||\cdots||u^{l-1}) \\ &= \pi_l(y^0||\cdots||y^{l-1}) \\ &= (z^0||\cdots||z^{l-1}). \end{aligned}$$

Thus, one round alternate Feistel network with x_i/x_{i+1} as the input/output can be defined as

$$\begin{aligned} x_{i+1} = \phi_d(x_i^0||\cdots||x_i^{l-1}) &= \pi_l \circ \tau_{\bar{d}}^{\mathbf{f}'} \circ \tau_d^{\mathbf{f}}(x_i^0||\cdots||x_i^{l-1}) \\ &= \pi_l \circ \tau_{\bar{d}}^{\mathbf{f}'}(u_i^0||\cdots||u_i^{l-1}) \\ &= \pi_l(y_i^0||\cdots||y_i^{l-1}) \\ &= (z_i^0||\cdots||z_i^{l-1}). \end{aligned}$$

where $\forall j \in [0, l-1]$, $|x_i^j| = n_j$,

$$u_i^j = \begin{cases} x_i^j \text{ op } f_j(x_i^{j-1+2d_j}) & \text{if } d_j = 0, 1 \\ x_i^j & \text{if } d_j = 2, \end{cases}$$

and

$$y_i^j = \begin{cases} u_i^j \text{ op } f_j'(u_i^{j-1+2\bar{d}_j}) & \text{if } \bar{d}_j = 0, 1 \\ u_i^j & \text{if } \bar{d}_j = 2. \end{cases}$$

It is noted that, if $\forall i, j \in [0, l-1]$, $n_i = n_j$, then it is called balanced alternating GFN. Otherwise, we call it as unbalanced alternating GFN. E.g., by fixing inputs as $l = 2, q = 1, d = 02, \pi_2 = (1, 2), n_0 > n_1$, it represents a alternate Feistel network, given in Figure 2-1e.

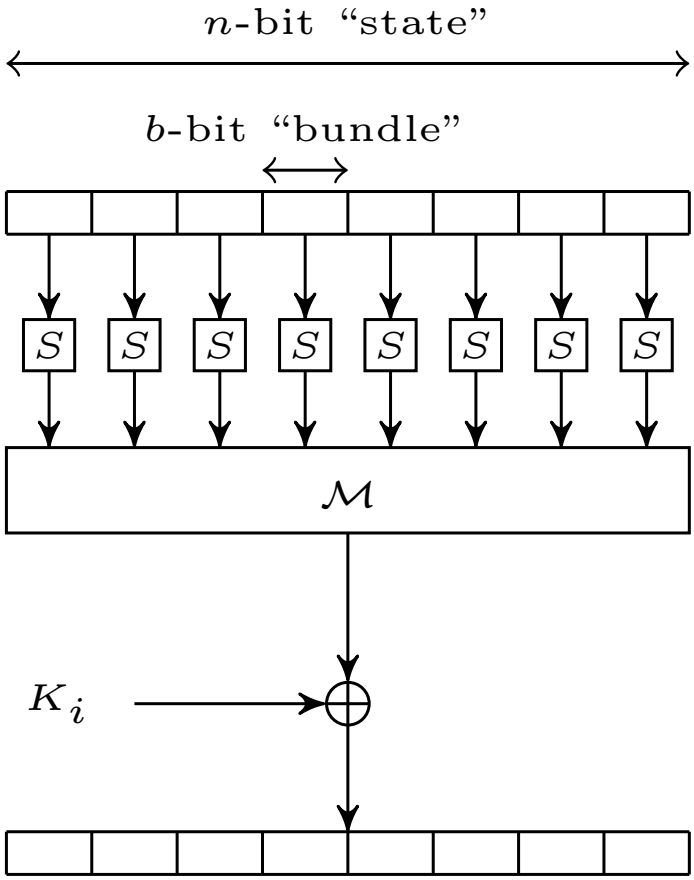


Figure 2-2: One Round of an SPN

2.2 Substitution Permutation Networks (SPN)

In an n-bit SPN cipher, its internal state can typically be divided into $\frac{n}{b}$ many b-bit words, sometimes we also use nibble (resp. byte) to describe a word if $b = 4$ (resp. $b = 8$). In a round function, the S-box layer usually consists of b-bit S-boxes updating all the words in parallel. Then, the $\mathcal{M}$-layer applies linear transformations on every b-bit words. Confusion and diffusion are two important design principles for a secure cipher. The property of confusion hides the relationship between the ciphertext and the key so that it makes difficult to find the key from the ciphertext. This property ensures that a small alteration in the key results in a significant and broad impact. The S-box is a crucial component of an SPN cipher for introducing non-linearity to the system. Diffusion means that if we change a single bit of the plaintext, then about half of the bits in the ciphertext should change. For the diffusion property, it is often achieved by the $\mathcal{M}$-layer and to some extent through the S-box layer as well.

A substitution permutation network (SPN) is made up of multiple rounds and operates on a state of n bits. Each round (as shown in Figure 2-2) involves dividing the input into b-bit words, applying a substitution function (S-box) to each word individually, and then diffusing the words using a linear transformation ($\mathcal{M}$) that exhibits certain "branching" properties. Hence, the state of an SPN can be regarded as a sequence of m words, where $m = \frac{n}{b}$.

An SPN $\mathsf{SPN}_k : \{0,1\}^n \to \{0,1\}^n$ is defined mathematically by three parameters and two functions, indexed by a key $k = k_0 \,||\, k_1 \,||\, \cdots \,||\, k_r \in (\{0,1\}^n)^{r+1}$.

- $r \in \mathbb{N}$, denotes the number of rounds.
- $b \in \mathbb{N}$, denotes the S-box input size, i.e., the word size.
- $m \in \mathbb{N}$, denotes the total number of words in SPN state.
- $S : GF(2^b) \to GF(2^b)$, denotes the S-box
- $\mathcal{M} : (GF(2^b))^m \to (GF(2^b))^m$, denotes the linear transformation.

The input and output size of SPN_k are determined by the product of m and b, i.e., $n = m \cdot b$. For instance, AES uses parameters $b = 8$, $n = 128$, and $r \in 10, 12, 14$, and the computation of SPN_k involves r rounds. The SPN state is initially XORed with

the subkey k_0, after which each of the r rounds ($\forall i \in 1, \cdots, r$) is carried out in three steps:

1. The S-boxes are applied to m parallel words.
2. The entire state undergoes the linear transformation $\mathcal{M}$.
3. The entire state is XORed with the round key k_i.

It is worth noting that each round in the SPN has the same structure, except for step (3)[1]. Specifically, given an input x, $\mathsf{SPN}_k(x)$ uses $x \oplus k_0$ as input for the first round. The output of round i serves as the input to round $i+1$ (for $1 \leq i < r$), and ultimately, the output of $\mathsf{SPN}_k(x)$ is the ciphertext.

Generally speaking, one round SPN structure can be viewed as the composition of Sbox Layer and Linear Layer. Further, a SPN structure can be categorized into two classes based on the **Linear Layer**. The first one [62, 63] is based on Mixcolumn (MDS or almost-MDS) matrix (strong diffusion), where the second one [64, 65] is based on bit permutation (lighter diffusion compared to the first one).

2.3 Sponge Construction

The primary property associated with a hash function's output is that it is fixed. However, it's possible to create a more general form of such functions that allows for variable length output in a straightforward way compared to regular hash functions. Moreover, the sponge is introduced to create a cryptographic primitive that can serve multiple purposes, such as hashing, encryption, and pseudo-random number generation, all within a single framework.. The sponge construction [66] provides a concrete way to realize such functions. As defined by Bertoni *et al.*, it is a simple iterated construction that builds a function (f) with variable-length input and arbitrary output length based on a fixed-length transformation or permutation (f) operating on a fixed number (b) of bits. This b is called the width. The sponge construction operates on a state of $b = r + c$ bits, where r is called the *bitrate* and c is called the *capacity*. At the

[1] SPN can be defined more generally by allowing variations in the S-box across rounds or a more complex interaction with k than XOR

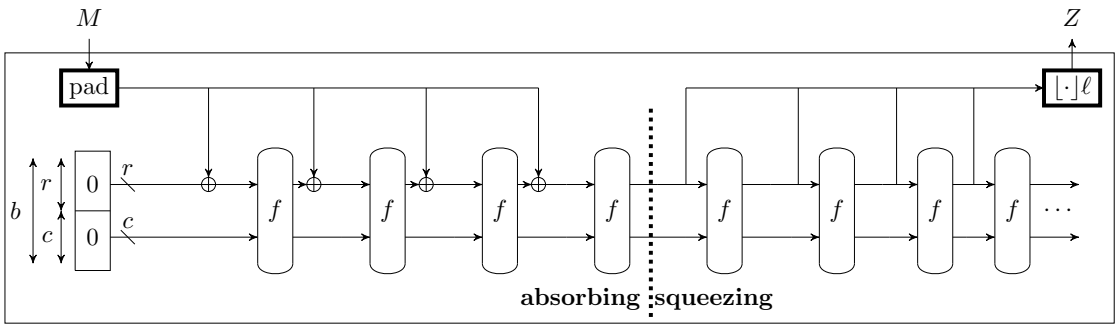


Figure 2-3: The Sponge Construction [66]

initialization phase, the internal state is set to all zeros. Then, the sponge construction process begins with two phases of iteration. The first phase is the absorption phase, where input data is absorbed into the internal state, and the second phase is the squeezing phase, where the output is generated from the internal state. This process is illustrated in Figure 2-3. The sponge construction is a method for creating cryptographic hash functions with variable length output, and it is used as a reference for security claims. The designers have demonstrated that *random sponges* can be as strong as a *random oracle* under certain conditions, specifically when the internal state size is sufficiently large. A random sponge refers to a sponge function where the internal permutations or transformations behave like random permutations and a random oracle is a theoretical black box that provides truly random and independent responses to every unique query. The designers suggest using permutations instead of more complicated structures like block ciphers or special compression functions as the foundation for their cryptographic method.

Hermetic Sponge Strategy

The hermetic sponge strategy is a design strategy based on the concept that any attack against a sponge function implies that the permutation used can be distinguished from a randomly-chosen permutation. This strategy plays a fundamental role in the analysis of permutations for structural distinguishers. Additionally, the security level of the strategy is determined by the capacity c, which can be traded for speed using bitrate r. The sponge construction can also be used to design other primitives such as MACs and key-stream generators. With the help of an equivalent construction called

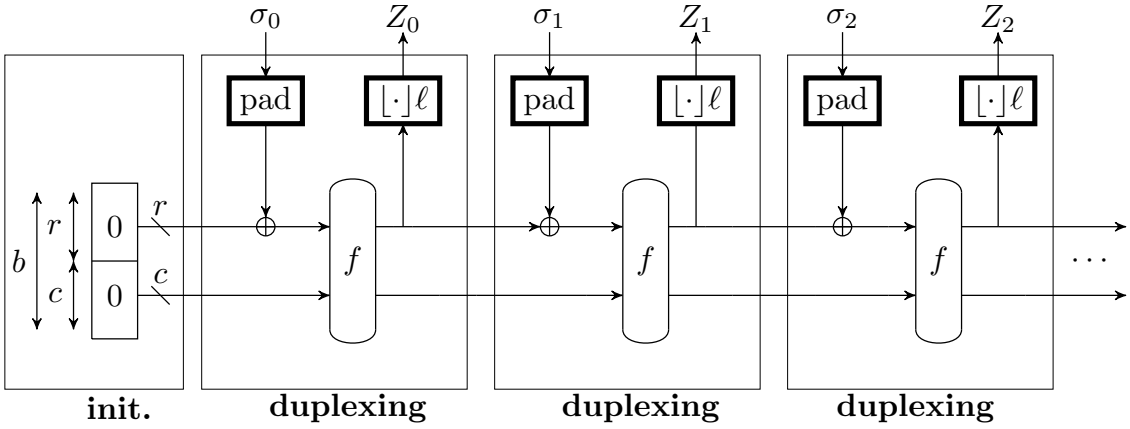


Figure 2-4: The Duplex Construction [66]

Duplex, efficient reseedable pseudo-random bit sequence generation and authenticated encryption schemes can also be implemented.

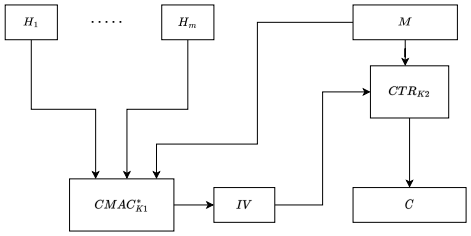
The duplex construction (refer to Figure 2-4), introduced by Bertoni, Daemen, Peeters, and Van Assche [67], is a variant of the sponge construction that allows for the generation of efficient reseedable pseudo-random bit sequences and the implementation of authenticated encryption schemes. It is based on fixed-width permutations and allows for the absorption of one r-bit input block while producing one r-bit output block separated by one permutation call f. The output block depends on all previously absorbed input blocks. The duplex construction enables the creation of single-pass authenticated encryption schemes. The duplex construction is a useful tool for various cryptographic needs and has been utilized in various authenticated encryption schemes.

2.4 Synthetic IV (SIV) Construction

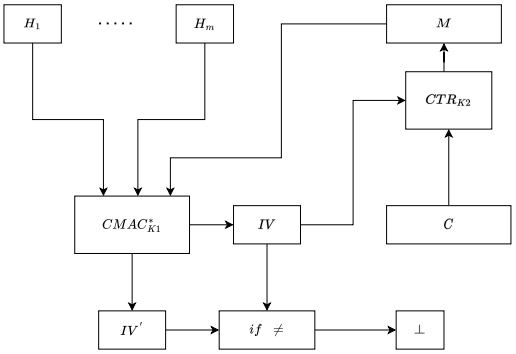
Synthetic Initialization Vector or Synthetic IV, or SIV, is a mode of operation for blockciphers that uses a key, plaintext, and header (a sequence of zero or more strings) as inputs to produce a deterministic associated ciphertext. This ciphertext maintains the privacy of the plaintext and authenticates both the ciphertext and the header. It can be used to resolve two issues: the key-wrap problem (deterministic authenticated encryption), is a type of authenticated encryption scheme where the encryption pro-

cess does not involve any randomness and conventional (two-pass, nonce-based) authenticated encryption, depending on its usage. In [68], Phillip Rogaway and Thomas Shrimpton specified a new block cipher mode, SIV, that provides both nonce-based authenticated encryption and deterministic, nonce-less key wrapping.

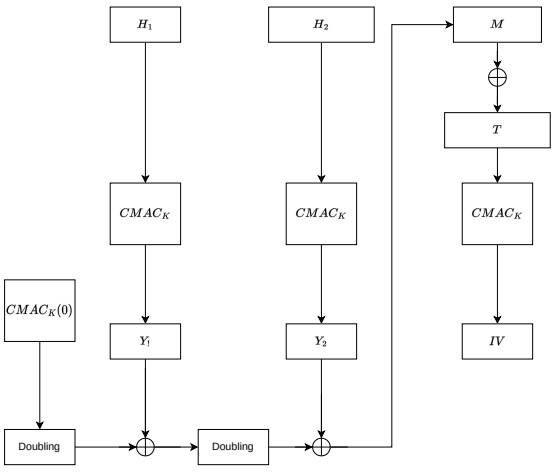
A unique initialization vector (IV) or nonce is essential for the security of authenticated encryption modes like Galois Counter Mode (GCM) [69]. Without it, the encryption could become vulnerable to various attacks. Reusing an IV can lead to a loss of confidentiality and/or authenticity. The SIV mode of operation, when used with a unique nonce as part of the associated data, provides protection against nonce



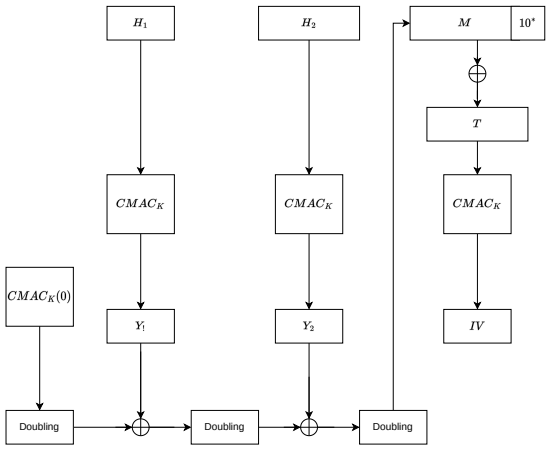
(a) SIV Encryption



(b) SIV Decryption



(c) CMAC without Padding



(d) CMAC with Padding

Figure 2-5: The SIV Construction [68]

reuse. It allows for no loss of authenticity and minimal loss of confidentiality in case of nonce reuse. It is specified as a generic composition of an IV-based encryption scheme and a pseudorandom function (PRF). The construction (see Figure 2-5), contains a Pseudo-Random Function (PRF) construction called S2V and an encryption/decryption construction, called CTR. S2V uses AES in Cipher-based Message Authentication Code mode [70] and CTR uses AES in counter mode [70]. The most commonly used instantiation of the SIV mode is AES-SIV [71], which is built entirely from the AES block cipher, using AES-CMAC [72] as the PRF and AES in CTR mode for confidentiality. It is attractive because it requires only an AES encryption operation to implement all aspects of the mode and it also has the convenient property that AES-CMAC produces a 128-bit tag, and AES-CTR requires a 128-bit IV, which allows the tag to be used directly as the (synthetic) IV.

2.5 Authenticated Encryption

Authenticated encryption is a method of encryption that combines the goals of confidentiality and data integrity into one mechanism. It is used to ensure that sensitive information is kept private while also verifying that the data has not been tampered with and that it was sent by the correct sender. This is accomplished by combining symmetric encryption with a message authentication code. Real-world scenarios such as sending medical information about a patient require both confidentiality and authenticity, which is why authenticated encryption is of practical significance. The concept of authenticated encryption was first introduced by Bellare and Namprempre in their pioneering work [73, 19, 74]. An Authenticated Encryption (AE) scheme is a type of symmetric encryption scheme where the decryption process returns an error if the ciphertext is not authentic. AE has become a widely used method in many standards, such as SSH [75], TLS [76], IPSec [77]. This widespread use has increased the motivation for further research in this area of cryptography.

The security implications of the three forms of generic composition: MAC-then-Encrypt (MtE), Encrypt-and-MAC (E&M), Encrypt-then-MAC (EtM) have been studied in depth in various literature, such as [73, 19, 74]. These studies have examined the security properties of each composition method and how they compare to one another.

Both Krawczyk [19] and Bellare and Namprempre [73] have independently analyzed the security properties of the three forms of generic composition: MtE, E&M, and EtM. Both studies seem to favor the EtM method over the other two. Even though the EtM composition is generally considered to be more secure, other compositions may be appropriate in certain circumstances, as demonstrated by Namprempre *et al.* in [78]. They have shown that nonce-based authenticated encryption with associated data can be realized using various building blocks like encryption schemes that require an initial value IV to be random, or encryption schemes that just require a nonce N to be unique. They also show that EtM constructions can fail if, for example, the nonce is not authenticated.

Authenticated Encryption has evolved over time to address performance and security issues, and can be classified based on the number of passes on the message or underlying construction. Dedicated AE schemes, which do not rely on composition, are initial schemes mainly based on block ciphers. Single-pass schemes, such as IAPM [79, 80], XCBC/XECB [81], and OCB [82, 83], have minimal message expansion, while two-pass schemes, such as CCM [84], EAX [85], CWC [86], and GCM [69], emulate generic composition but use a single key throughout. GCM is the most popular and widely standardized. While two-pass modes are widely adopted, they are not as efficient as single-pass schemes. However, the use of single-pass schemes is limited.

Authenticated ciphers have also evolved by moving away from the use of block-ciphers and using other symmetric-cryptographic primitives to achieve the goal of authenticated encryption. Examples include schemes that use stream ciphers [87, 88, 89], hash/compression functions [90], or more recently, key-less permutations [91].

Authenticated encryption with associated data (AEAD) is a technique that allows for encrypting and authenticating a message, as well as sending additional unencrypted data. An example of this could be a network packet where the payload is encrypted and authenticated, but the header is not encrypted to allow for routing information to be in plaintext. The concept of AEAD was first introduced by Rogaway [92], who added a third input to the standard form of authenticated encryption. Many different schemes now support AEAD, including the nonce-based AE and deterministic AE (DAE) schemes. The SIV [68] and HBS [93] schemes are examples of DAE schemes.

However, some DAE schemes were found to not be able to be computed in real-time, which is desirable in many practical applications. To address this, the concept of online AEAD was introduced in the form of the McOE scheme [94]. The field of AEAD is constantly evolving and new developments and advancements are being made all the time.

Formal Definition of Nonce-based AEAD

Nonce-based authenticated encryption with associated data is a method of encrypting data that uses a unique nonce (number used once) or initialization vector (IV) in combination with a secret key to encrypt the data. A formal definition of nonce-based authenticated encryption can be stated as follows:

Definition 6. A quadruple of probabilistic polynomial-time algorithms $(\mathcal{K}, \mathcal{N}, \mathcal{E}, \mathcal{D})$ satisfying the following properties for any plaintext $m \in \mathcal{M}$, associted data $A \in \mathcal{A}$, any key K chosen from the key space $\mathcal{K}$, any nonce N chosen from the nonce space $\mathcal{N}$:

- Key Generation Algorithm: $\mathcal{K}(\lambda) \rightarrow K$, where λ is the security parameter.
- Nonce Generation Algorithm: $\mathcal{N}(\lambda) \rightarrow N$
- Encryption Algorithm: $\mathcal{E}(K, N, A, M) \rightarrow (C, T) = C^{'}(say)$, where $C \in \mathcal{C}$ is the ciphertext and $T \in \mathcal{T}$ is the authentication tag.
- Decryption Algorithm: $\mathcal{D}(K, N, A, C, T^{'}) \rightarrow M^{'}$, where $M^{'}$ is the plaintext M if $T^{'} = T$, otherwise $M^{'} = \perp$, an authentication error symbol.

The nonce-based AEAD scheme is said to be secure if, for any polynomial-time adversary $\mathbb{A}$, the following two conditions hold:

- Confidentiality: For any two messages M_1 and M_2, and any nonce N, the ciphertexts $\mathcal{E}(K, N, A, M_1)$ and $\mathcal{E}(K, N, A, M_2)$ should be indistinguishable by an adversary without the key K.
- Integrity: For any ciphertext $C^{'}$, nonce N, and associated data A, it should be computationally infeasible for an adversary to produce a valid $(N, C^{'}, A)$ without knowing the key K such that $\mathcal{D}(K, N, A, C, T^{'}) \rightarrow \perp$.

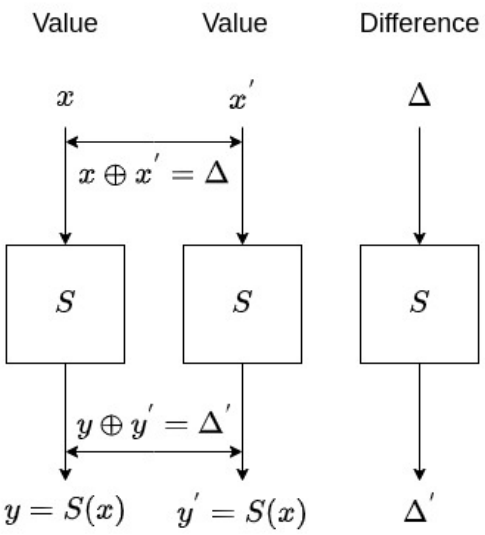


Figure 2-6: Illustration of Difference Propagation

– Authenticity: The scheme should ensure that a message M decrypted with $\mathcal{D}$ under key K is indeed the message M that was encrypted with $\mathcal{E}$ under the key K, and that it has not been tampered with.

This definition highlights the importance of the additional component nonce generation and how it is used in combination with key and encryption and decryption algorithms to ensure confidentiality, authenticity and integrity. Nonce-based AEAD is widely used in protocols such as GCM(Galois/Counter Mode) and CCM(Counter with CBC-MAC), which provides confidentiality, integrity and authenticity of the data.

2.6 Differential Cryptanalysis

Differential cryptanalysis is one of the two most widely used cryptanalysis on private-key ciphers, was first introduced by Biham and Shamir [24]. It is a chosen-plaintext attack (CPA) that exploits the high probability of certain plaintext difference propagating through multiple round functions to a specific ciphertext difference, we call it the differential. The differential cryptanalysis focuses on the difference of two computations in a cipher.

Suppose $S : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is a non-linear function and Δ' be the output difference cor-

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S-box	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

Table 2.1: The PRESENT S-box

reponds to a pair of input (x, x') with the input difference Δ. An illustration of the difference propagation for the S function is given in Figure 2-6.

Definition 7. Let $\Delta, \Delta^{'} (\in \mathbb{F}_2^b)$ be the input and output differences of S. We denote the differential probability (p) from Δ to $\Delta^{'}$ under S as $\Delta \xrightarrow{p} \Delta^{'}$ or $\Pr[\Delta \rightarrow \Delta^{'}]$ and define it as follows:

$$\Pr[\Delta \rightarrow \Delta^{'}] = \frac{\#\{x \in \mathbb{F}_2^b : \ S(x) \oplus S(x \oplus \Delta) = \Delta^{'}\}}{2^b}.$$

The Substitution layer is a crucial component of an SPN/Feistel cipher for introducing non-linearity to the system. The differential transition of an S-box is often represented in a $2^b \times 2^b$ *difference distribution table* (DDT) where each $(\Delta, \Delta^{'})$-entry shows the number of occurrences for which the differential transition holds, i.e., $\#\{x \in \mathbb{F}_2^b : S(x) \oplus S(x \oplus \Delta) = \Delta^{'}\}$.

Δ \ $\Delta^{'}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
a	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
b	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
c	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
d	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
e	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

Table 2.2: DDT of PRESENT S-box

Example 1. Consider the 4-bit S-box in PRESENT cipher [64] and its truth table is given in Table 2.1. The DDT of the PRESENT S-box is presented in Table 2.2.

According to the Table 2.2, we can see that $3 \xrightarrow{2^2} 6$, i.e., $\Pr[3 \to 6] = \frac{4}{16} = 2^{-2}$.

The differential cryptanalysis technique is used to find some high probability differential trails. For an r round n-bit cipher, let $\Delta_0 = M \oplus M^{'}$ be the difference between the two plaintexts $M, M^{'}$. The objective is to analyse how the difference propagates through r rounds in a cipher. Let us consider that at round i, the difference Δ_{i-1} propagates to Δ_i with probability p_i. Note that, at any round, an S-box with $\Delta \neq 0, \Delta^{'} \neq 0$ is called as an active S-box. At each round i, the probability p_i is calculated by multiplying the probabilities of each individual active S-boxes in that round. At the end, we can find a differential trail through the entire cipher as

$$\Delta_0 \xrightarrow{p_0} \Delta_1 \xrightarrow{p_1} \Delta_2 \xrightarrow{p_2} \Delta_3 \cdots \xrightarrow{p_{r-2}} \Delta_{r-1} \xrightarrow{p_{r-1}} \Delta_r. \tag{2.1}$$

We call this as the differential characteristic. Assuming that the independence between rounds, which is not the case in reality as the subkeys are XORed in each round. Thus, the overall probability of a differential characteristic is only an estimation, i.e.,

$$\Pr[\text{A differential characteristic in 2.1}] = \prod_{i=0}^{r-1} p_i.$$

However, in practice, this is proven to be reasonably accurate for many ciphers. However, it is worth noting that there may be multiple differential characteristics $(\Delta_0 \xrightarrow{r} \Delta_r)$ with the same initial (Δ_0) and final (Δ_1) difference that contribute to the overall probability of the differential. In that case the probability of the differential characteristic is calculated by summing of the probability of each differential characteristics with same input and output difference. Thus, the differential probability of $\Delta_0 \xrightarrow{r} \Delta_r$ will be calculated as follows.

$$\Pr[\Delta_0 \xrightarrow{r} \Delta_r] = \sum_{\text{all intermediate differences}} \Pr[\Delta_0 \to \Delta_1 \to \Delta_2 \to \cdots \Delta_{r-1} \to \Delta_r].$$

Differential cryptanalysis involves identifying differential characteristics, which are patterns of differences in input pairs that lead to specific differences in output pairs with high probability. Once such characteristics are found, they can be used to launch distinguishing attacks and potentially key-recovery attacks. In a distinguishing

attack, the encryption of ζ pairs of plaintext with the same input difference Δ_0 is queried. The number of ciphertext differences that match the target difference Δ_1 is then compared to the differential probability, and if they are close, it is likely that the encryption oracle is the target cipher. If not, it is likely a random permutation. In a key-recovery attack, a $(r-1)$-round differential is used. After obtaining the ciphertext pairs, the last round subkey is guessed, and the internal states of the previous rounds are then worked backwards to check for the target difference. The correct subkey candidate should result in a significantly higher number of matches than the incorrect candidates. Hence the amount of plaintext pairs needed to launch a differential attack is estimated to be $\zeta = \frac{c}{\Pr[\Delta_0 \xrightarrow{r} \Delta_r]}$, where c is a small constant.

Searching for Characteristics

Manual searching for characteristics in cryptography can be time-consuming and prone to errors, therefore, several methods and tools have been developed to aid in the search. These methods can be broadly categorized into building tools from scratch or building on existing MILP, CP or SAT solvers. The aim of these tools can vary from finding or bounding the probability of the best existing characteristic to searching for good characteristics that can be useful in attacks. The ability to give a tight bound on probability and finding the characteristic highly depend on the analyzed primitive. In [95], Matsui introduced a method for finding the best differential and linear characteristics for DES. Later, Biryukov and Nikolić developed a tool [96] for searching related-key differential characteristics that works well for byte-aligned ciphers. Biryukov, Velichkov, and Le Corre also developed a tool [97] for best characteristics for ARX-based primitives. Other tools [98, 99] have been developed for specific primitives such as Keccak, exploiting its structural properties to provide bounds on probability.

Recently, it has become popular to use mixed integer linear programming (MILP) and SAT solvers to search for differential and linear characteristics. Mouha, Wang, Gu, and Preneel [48] were among the first to use MILP to prove bounds on the minimum number of active S-boxes in a differential or linear characteristic. This method has been extended to the related-key setting for SPN-structures. More recently, MILP

and SAT solvers [100, 101, 102] have been used to search for differential and linear characteristics for ARX-based primitives.

2.7 Boomerang Attack

The boomerang attack is a method for the cryptanalysis of block ciphers based on differential cryptanalysis. In most of the ciphers, the searching of high probability differential characteristics might not be always possible. In such cases, Wagner in [103] have proposed a variant of differential attack, known as the boomerang attack, which may become very useful to exploit the differntial properties of a cipher by combining two high probability differentials in an elegent way for different segments of the cipher. Consider an n-bit block cipher E with k-bit key, which can be expressed as a cascade cipher $E = E_1 \circ E_0$, where E_0 has a differential $\Delta_0 \xrightarrow{p} \Delta_1$, and E_1 has a differential $\nabla_0 \xrightarrow{q} \nabla_1$. In this attack, the cipher E is considered as a composition of two sub-ciphers E_0 and E_1, i.e., $E = E_1 \circ E_0$, where we suppose that the input difference Δ_0 is propagated to the difference Δ_1 by E_0 with probability p and the difference ∇_0 is propagated to ∇_1 by E_1 with probability q. This is described in Figure 2-7. The boomerang attack leverages two short differentials with high probability, rather than relying on one long differential with low probability, to enhance the chances of success. The method for mounting the distinguisher is as follows.

1. Ask for the ciphertexts $C_0 = E(P_0)$ and $C_1 = E(P_1)$, where $P_1 = P_0 \oplus \Delta_0$.
2. Ask for the plaintexts $P_2 = E^{-1}(C_2)$ and $P_3 = E^{-1}(C_3)$, where $C_2 = C_0 \oplus \nabla_1$ and $C_3 = C_1 \oplus \nabla_1$.
3. Check whether $P_2 \oplus P_3 = \Delta_0$.

The expected probability of this attack, assuming the independence of the characteristics, exploits the following differential:

$$\Pr(E^{-1}(E(x) \oplus \nabla_1) \oplus E^{-1}(E(x \oplus \Delta_0) \oplus \nabla_1) = \Delta_0] = p^2 \cdot q^2. \tag{2.2}$$

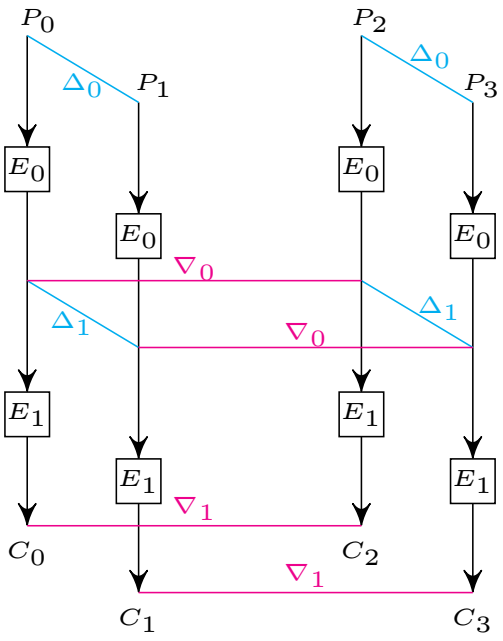


Figure 2-7: Boomerang

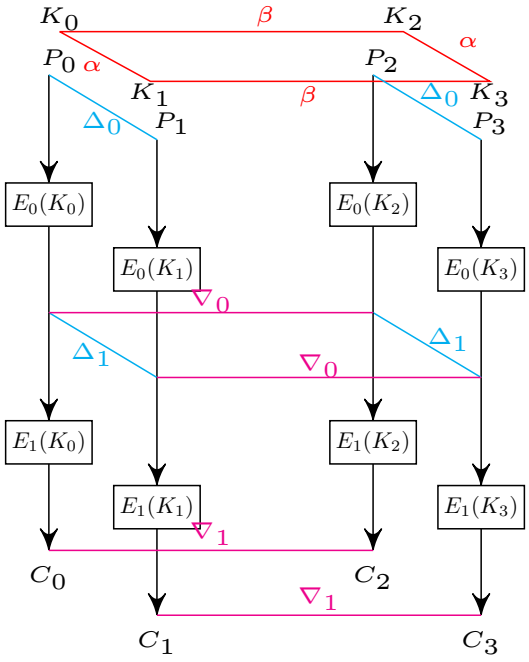


Figure 2-8: Related-key Boomerang

Equation 2.2 shows that by performing $\frac{1}{p^2 \cdot q^2}$ number of adaptively chosen plaintext and ciphertext queries with the Δ_0 difference on the encryption queries and the ∇_1 difference on the decryption queries, the attacker can distinguish E from ideal cipher. Later, the amplified boomerang attack [104] (also called the 'rectangle attack [105]') was proposed which works in a chosen-plaintext attack (CPA) scenario. In this attack, the expected probability to get a right quartet will be $p^2 \cdot q^2 \cdot 2^{-n}$. Furthermore, in [106, 107], they have pointed out that any value of Δ_1 and ∇_0 can be considered as long as $\Delta_1 \neq \nabla_0$. As a result, the probability of the right quartet is increased to $2^{-n} \cdot \hat{p}^2 \cdot \hat{q}^2$, where $\hat{p} = \sqrt{\sum_i \Pr^2(\Delta_0 \to \Delta_1^i)}$ and $\hat{q} = \sqrt{\sum_j \Pr^2(\nabla_0^j \to \nabla_1)}$. Later, Dunkelman *et al.* [108] formalized the sandwich attack, decomposing the target cipher into three parts, $E = E_1 \circ E_m \circ E_0$, with the middle part E_m having relatively short transformations. Sandwich attacks extend boomerang attacks by adding an encryption layer to exploit specific differential properties. To systematically calculate the probability r for E_m and find effective switches, Cid *et al.* [109] proposed the Boomerang Connectivity Table (BCT), which refines boomerang attacks by analyzing connectivity between input and output differences.

2.8 Related-key Boomerang Attack

The related-key boomerang attack, as illustrated in Figure 2-8, uses key differences in addition to plaintext differences. It assumes that the upper sub-cipher E_0 has a differential characteristic $\Delta_0 \xrightarrow{p} \Delta_1$ under a key difference $\alpha = K_0 \oplus K_1 = K_2 \oplus K_3$ and the lower sub-cipher E_1 has a differential characteristic $\nabla_0 \xrightarrow{p} \nabla_1$ under a key difference $\beta = K_0 \oplus K_2 = K_1 \oplus K_3$. A related-key distinguisher is constructed by using four different unknown keys: $K_0, K_1 = K_1 \oplus \Delta_0, K_2 = K_1 \oplus \beta$, and $K_3 = K_0 \oplus \beta$. Under adaptively chosen plaintext and ciphertext queries, the procedure for executing the related-key boomerang distinguisher is as follows.

1. Ask η ciphertext pairs (C_0, C_1), where $C_0 = E_{K_0}(P_0)$, $C_1 = E_{K_1}(P_1)$ and $P_0 \oplus P_1 = \Delta_0$. We denote the set of these pairs as $\mathcal{S}$.
2. Ask η ciphertext pairs (C_2, C_3), where $C_2 = E_{K_2}(P_2)$, $C_3 = E_{K_3}(P_3)$ and $P_2 \oplus P_3 = \Delta_0$. We denote the set of these pairs as $\mathcal{S}'$.
3. Find right quartets satisfying the following conditions from $\mathcal{S}$ and $\mathcal{S}'$: $P_0 \oplus P_1 = P_2 \oplus P_3 = \Delta_0$ and $C_0 \oplus C_2 = C_1 \oplus C_3 = \nabla_1$.

2.9 MILP-based Cryptanalysis

Linear programming (LP) involves optimizing a linear objective function subject to linear inequalities with decision variables. When certain decision variables are required to be integers, the problem becomes mixed-integer linear programming (MILP). MILP has practical applications in fields such as economy and business, but its use in cryptography has been limited. If all decision variables must be integers, the problem is referred to as pure integer linear programming (ILP).

In cryptography, MILP and ILP solvers have been used to solve problems related to stream ciphers, hash functions, and Feistel networks. Borghoff *et al.* [110] used CPLEX to solve the MILP problem for recovering the internal state of Bivium, while Bouillaguet *et al.* [111] used an ILP solver to find a differential characteristic for SIMD. Bogdanov [112] used an MILP program to calculate the minimum number of linearly and differentially active S-boxes of unbalanced Feistel networks. These approaches

are relatively time-consuming and involve a substantial amount of manual labor, as well as the construction of multiple linear programming (LP) programs.

In [113, 48], MILP was used to assess the security of block ciphers against differential and linear cryptanalysis. Mouha *et al.* [48] introduced a model framework to systematically calculate lower bounds on the minimum number of active S-boxes for word-oriented ciphers. A short description to prove the security of word-oriented ciphers against differential cryptanalysis of this model technique is given below.

In the context of truncated differences, a binary variable x is utilized to represent a word-level difference, where x equals 1 if and only if the input word is non-zero. For a cipher that comprises *XOR, Linear Transformation, and S-box operations*, the constraints below are employed to depict how word-level differences propagate through the cipher:

Constraints for XOR Operation. Let the word-level input differences of the XOR operation are u, v and w be its corresponding output difference. For XOR, the *differential branch number* is 2, where the differential branch number is the minimum number of input and output bytes that contain differences, excluding the case where there are no differences in inputs nor outputs. The following constraints are used to describe the XOR operation.

$$\begin{cases} u + v + w \geq 2d_{\oplus}, \\ d_{\oplus} \geq u, d_{\oplus} \geq v, d_{\oplus} \geq w, \end{cases}$$

where $d_{\oplus} \in \{0, 1\}$ is a new binary dummy variable.

Constraints for Linear Transformation. Let $(x_0, \cdots, x_{m-1})$ and $(y_0, \cdots, y_{m-1})$ denote word-level input and output differences of the linear transformation L respectively. Given the differential branch number $B_{\mathcal{D}}$, the linear transformation L can be constrained by the following linear equations:

$$\begin{cases} \sum_{i=0}^{m-1} x_i + \sum_{i=0}^{m-1} y_i \geq B_{\mathcal{D}} D_L, \\ D_L \geq x_i, D_L \geq y_i, i \in \{0, 1, \cdots, m-1\}, \end{cases}$$

where $d_L \in \{0, 1\}$ is a dummy binary variable.

Additional Constraints. To prevent the trivial solution where no S-box is active, an additional linear constraint is included in the ILP problem. If all d-variables and x-variables corresponding to plaintext are restricted to binary values, the resulting program becomes a pure ILP. However, if only d-variables are binary, and plaintext variables are binary, the equations guarantee that the optimal solution for other x-variables will also be binary. This approach leads to an MILP problem that can be solved more efficiently.

Objective Fucnction The objective function is to minimize the number of active S-boxes,i.e, the sum of all variables representing word-level input differences of S-boxes of eachround.

An MILP model was constructed using the above framework to determine the minimum number of active S-boxes for a word-oriented block cipher. However, this model does not take into account bitwise operations, making it unsuitable for bit-oriented block ciphers. Sun *et al.* [114] extended this framework to SPN ciphers with bitwise permutation diffusion layers. At ASIACRYPT 2014, Sun *et al.* [115] improved the MILP-based method for automatically evaluating block cipher security against related-key differential cryptanalysis. They proposed a heuristic algorithm to find actual related-key differential characteristics by introducing two systematic methods for generating inequalities to remove impossible differential characteristics from the model's feasible region. In [116], Sun *et al.* encoded differential probabilities and linear approximations of S-boxes into the MILP model and argued that the feasible region of the model built using the convex hull computation method for S-boxes corresponds to the set of all possible related-key differential and linear characteristics. Moreover, several automatic search algorithms based on MILP [100, 117, 118, 49] have been developed to identify the most favorable linear and differential characteristics.

2.10 Differential Fault Analysis

Differential Fault Analysis (DFA) is a method that was first introduced by Biham and Shamir in [119]. It is similar to differential cryptanalysis, and it typically requires

knowledge of both correct and faulty data. The goal of DFA is to induce a fault in the last rounds of a cipher and inject a difference in the intermediate state, which allows the attacker to apply differential cryptanalysis to just a few rounds. In the case of DES, it was shown that an attacker can flip a single bit in a register and if the fault occurs in the last round, it can be used to reduce the possible keys to four 6-bit keys on average, by knowing the input and output differences of one S-box.

DFA can also be applied to SPN structures like AES, where the last round does not have a MixColumn operation. In [120], Piret and Quisquater, demonstrated how DFA can be used to attack AES by injecting a difference in one byte just before the last MixColumns operation of AES. This results in differences in 4 bytes of the ciphertext. The attacker can then guess the 4 last round key bytes at the position of these differences, and use that information to invert the last round and an inverse MixColumns operation. By discarding any key guesses that lead to a state with more than one byte difference, the attacker can determine the 4 key bytes uniquely with just two faulty ciphertexts. This method can also be applied to the second to last application of MixColumns, allowing to recover 16 bytes of the last round key in parallel. The fault analysis for attacking SPN ciphers, particularly those that use a diffusion matrix as the linear layer instead of bit permutation, is similar and can be generalized from a theoretical perspective to some extent. The security properties and potential vulnerabilities of high-profile ciphers like AES [121, 122, 123], CLEFIA [124], present [125], PRINCE [126], and MIDORI [127] have been recently studied using DFA, which is a commonly employed method for analyzing the security of such ciphers.

2.11 Statistical Fault Analysis

Statistical fault analysis (SFA) differs from DFA in that it does not rely on injecting differences in a pair of intermediate states. Instead, SFA attacks exploit the way faults can affect the distribution of intermediate values across many encryptions. This means that instead of encrypting the same input twice, many different unknown inputs are processed. A simple version of this attack on AES is when a fault occurs before the last round key application, which immediately reveals the key. Fuhr *et al.* [31] first studied different fault models on the byte level, such as a stuck-at-zero

fault, a stuck-at-one fault, and a stuck-at-random fault are used to show attacks on AES.

2.12 Counting Elements Having at Least m Properties

Counting elements with at least m properties is a fundamental problem in combinatorics and applied mathematics. Let's consider a finite collection of sets $A_1, A_2, \ldots, A_n$. We define B_m as the set of elements x that belong to at least m of the sets $A_1, A_2, \ldots, A_n$, i.e., $B_m = x \in A_1 \cup \ldots \cup A_n : x$ belongs to at least m of $A_1, A_2, \ldots, A_n$. The objective is to determine the number of elements in B_m. We state this problem in Proposition 2. Another variant of the problem is to count elements that have exactly m properties. Both of these problems can be proved using similar techniques. The formal proof of the latter variant can be found in [128, Theorem 7.4], where basic counting methods are employed. However, it is also possible to prove this variant using the mathematical induction technique. In this exposition, we will present the proof of proposition using mathematical induction, without loss of generality. Furthermore, we utilize the principle of inclusion-exclusion (stated in Proposition 1) as a crucial tool in our proof. A formal proof of the inclusion-exclusion property can be found in [128, Theorem 7.1]. For the sake of completeness, we will provide a brief presentation of the proof of this inclusion-exclusion property.

Proposition 1 (The Principle of Inclusion and Exclusion)**.** Suppose, we have n number of finite sets as $A_1, \ldots, A_n$. Then, the cardinality of their unions can be calculated using the following formula:

$$\Big|\bigcup_{1\le i\le n} A_i\Big| = \sum_{1\le i_1\le n} |A_{i_1}| - \sum_{1\le i_1\le i_2\le n} |A_{i_1}\cap A_{i_2}| + \sum_{1\le i_1\le i_2\le i_3\le n} |A_{i_1}\cap A_{i_2}\cap A_{i_3}| - \ldots + (-1)^{n+1}\big|\cap_{i=1}^{n} A_i\big|.$$

Proof. We prove this Inclusion-Exclusion principle by counting the number of occurances of an element from both sides in the relation. Let $x \in A_1 \cup \ldots \cup A_n$. Then, in the left hand side, the count will be 1. Now, we have to count the number of

occurances of x in the right hand side.

Let us assume that, x belongs to ℓ number of sets out of $A_1, \ldots, A_n$. Then, the counts of x in the right hand side can be calculated in the following way.

$$\sum_{1\le i_1\le n} |A_{i_1}| = \binom{\ell}{1},$$
$$\sum_{1\le i_1\le i_2\le n} |A_{i_1}\cap A_{i_2}| = \binom{\ell}{2},$$
$$\sum_{1\le i_1\le i_2\le i_3\le n} |A_{i_1}\cap A_{i_2}\cap A_{i_3}| = \binom{\ell}{3},$$
and, so on.

Thus, the right hand side can be written as:

$$\begin{aligned}
&\binom{\ell}{1} - \binom{\ell}{2} + \binom{\ell}{3} - \ldots + (-1)^{\ell+1}\binom{\ell}{\ell}\\
&= 1 - [1 - \binom{\ell}{1} + \binom{\ell}{2} - \binom{\ell}{3} + \ldots - (-1)^{\ell+1}\binom{\ell}{\ell}]\\
&= 1 - [1 + (-1)]^{\ell} \text{ [Using Binomial Theorem]}\\
&= 1.
\end{aligned}$$

This completes the proof. □

Proposition 2 (Counting elements having at least m properties)**.** Suppose we have a finite n number of sets $A_1, \cdots, A_n$. Let B_m be the set of elements belonging to at least m number of sets of $A_1, \cdots, A_n$. Then,

$$|B_m| = \sum_{h=0}^{n-m} (-1)^h \binom{m+h-1}{m-1} S_{m+h},$$

where

$$S_j = \sum_{i_1\le\cdots\cdots\le i_j} \left|A_{i_1}\cap\cdots\cap A_{i_j}\right|.$$

Proof. Let us prove Proposition 2 by using principle of mathematical induction. For $m = 1$, the equality in Proposition 2 holds by using principle of inclusion and exclusion

(Proposition 1). Let us assume that Proposition 2 is true for m. Now, we have to prove whether it is true for $m+1$ or not. Therefore, we have to show that

$$|B_{m+1}| = \sum_{h=0}^{n-m-1} (-1)^h \binom{m+h}{m} S_{m+h+1}.$$

Let $C(k,m)$ denote the number of times an element x belonging to exactly k sets out of $A_1, A_2, \ldots, A_n$ which is actually counted on the right hand side of the expression $| B_m |$. So, by induction hypothesis, we can write

$$C(k,m) = \begin{cases} 1, & \text{if } k \geq m; \\ 0, & otherwise. \end{cases}$$

Let us assume that an element x belonging to exactly k sets out of the given n sets $A_1, A_2, \ldots, A_n$. Then we have to show that,

$$C(k,m+1) = \begin{cases} 1, & \text{if } k \geq m+1; \\ 0, & otherwise. \end{cases}$$

Now, for $k \geq m+1$, $C(k,m+1)$ denote the number of times x belonging to exactly k sets of n sets. This number is actually counted on the right hand side of the expression $| B_{m+1} |$. Therefore, we can write,

$$C(k,m+1) = \sum_{i=0}^{n-m-1} (-1)^i \binom{m+i}{m} \binom{k}{m+i+1}.$$

$$(As\ S_{m+i+1} = \sum | A_{j_1} \cap A_{j_2} \cap \ldots \cap A_{j_{m+i+1}} |)$$

Now, by putting $h = i+1$ in the above expresssion, we get

$$C(k,m+1) = \sum_{h=1}^{n-m} (-1)^{h-1} \binom{m+h-1}{m} \binom{k}{m+h}.$$

Further, we have

$$C(k,m) = \sum_{h=0}^{n-m} (-1)^h \binom{m+h-1}{m-1} \binom{k}{m+h} = 1.$$

Therefore, we have

$$
\begin{aligned}
&C(k,m) - C(k,m+1) \\
&= \sum_{h=0}^{n-m} (-1)^h \binom{m+h-1}{m-1} \binom{k}{m+h} - \sum_{h=1}^{n-m} (-1)^{h-1} \binom{m+h-1}{m} \binom{k}{m+h} \\
&= \binom{k}{m} + \sum_{h=1}^{n-m} (-1)^h [\binom{m+h-1}{m-1} + \binom{m+h-1}{m}] \binom{k}{m+h} \\
&= \binom{k}{m} + \sum_{h=1}^{n-m} (-1)^h \binom{m+h}{m} \binom{k}{m+h} \\
&= \binom{k}{m} + \binom{k}{m} \sum_{h=1}^{n-m} (-1)^h \binom{k-m}{h} \\
&= \binom{k}{m} \sum_{h=0}^{n-m} (-1)^h \binom{k-m}{h} \\
&= \binom{k}{m} \sum_{h=0}^{k-m} (-1)^h \binom{k-m}{h} = \binom{k}{m} (1-1)^{k-m} = 0.
\end{aligned}
$$

Hence, $C(k, m+1) = C(k, m) = 1$. Again, for $k \leq m$, we have $C(k, m+1) = 0$ as

$$|B_{m+1}| = \sum_{h=0}^{n-m-1} (-1)^h \binom{m+h}{m} S_{m+h+1}.$$

This completes the proof. □

3 NEW VARIANTS OF COUPON COLLECTOR PROBLEM

3.1 Introduction

The Coupon Collector Problem (CCP) is a widely studied mathematical problem that has various applications in different fields. It involves collecting different types of coupons with the goal of getting one of each type. The expected number of coupons needed to complete the collection can vary based on the probabilities of getting each type of coupon and can be affected by various factors such as the arrival of coupons in groups of constant size, or the intentions of a group of friends to complete multiple collections. The problem has been applied in fields such as combinatorial mathematics, computer science, cryptography, machine learning, operations research, marketing, queueing theory, and simulation. It can also be used to analyze the performance of systems that use multiple servers or to generate random samples for simulating complex systems.

The CCP and its variants are of long-standing and recurrent interest in the field of mathematics and computer science, as evidenced by their numerous mentions in multiple studies [129, 130, 131, 132, 133]. The problem is well known for its rich theoretical structures and implications, which have led to a wide range of practical applications. Some of these applications include dynamic resource allocation, hashing, and fault detection in combinatorial circuits.

The Fault Detection (FD) problem in digital circuit testing is particularly relevant to

the coupon collector problem. The goal of the FD problem is to identify the minimal list of input vectors that can detect all the faults in a digital circuit. The standard fault model assumes that faults occur infrequently and independently, with the only possible faults being lines that are either stuck at 0 or 1. To solve the FD problem, one randomly selects input vectors, determines the faults detected by each input, and continues this process until all faults are detected. The probability of detecting a fault is proportional to the number of input vectors that produce incorrect output when that fault is present. This makes the FD problem similar to the coupon collector problem, as both involve collecting items with a certain probability of success.

It has been found that in certain permutation-based ciphers under DFA, an attacker who can cause l consecutive bit-flips in the final rounds of the state can obtain the l consecutive state bits. Although this consecutive bit-flip model in the context of fault injection into a device may seem unrealistic, research has shown that consecutive bit-flips can occur quite frequently. For instance, it has been shown [134, 135] that a single-spot laser can induce these consecutive bit-flips and that the size of the laser spot plays a major role. Additionally, a multi-spot laser setup [136] has been demonstrated to be capable of injecting faults on more than two adjacent bits.

We present a new variant of the CCP that models the expected number of attempts needed to complete the collection in this scenario. We call it the "Consecutive Coupon Collector Problem (CCCP)". The problem is as follows: if an attacker repeatedly injects consecutive bit-faults into an n-bit register and in each trial, randomly selects a bit and then flips k bits ($k \leq n$) to the right of the selected bit, what is the expected number of trials to flip t ($1 \leq t \leq n$) bits? To the best of our knowledge, this problem has not been treated previously in the literature.

We conduct a theoretical analysis of the problem, provide a mathematical proof, and show a close match between our theoretical predictions and simulation results. We also perform hypothesis testing to validate our calculations. Additionally, we extend the problem to the "Circular" variant and treat it similarly. This problem may be of independent interest to the research community.

3.2 Summary of Known Variants of the Coupon Collector Problem

The CCP is a popular problem with many variants in Discrete Probability Theory. Interestingly, some of the most common variants can be combined together in a generalized problem statement as follows.

3.2.1 Generalized Problem Statement

Given a set of n coupons, the collector draws randomly l $(1 \leq l \leq n)$ many coupons at each trial with replacement. What is the expected number of trials necessary to collect m copies of each coupon of the n coupons?

3.2.2 Solution for Different Cases

Let $X^l_{m,n}$ be a random variable to denote the number of trials necessary to collect m copies of each coupon. We need to calculate $E(X^l_{m,n})$.

1. For $l = 1$ and $m = 1$, this problem is the Classic CCP [137]. In this case,

$$E(X^1_{1,n}) = nH_n, \tag{3.1}$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} Harmonic number, which for large n, equals $\log(n) + O(1)$.

$$E(X^1_{1,n}) = n\log(n) + nO(1).$$

2. For $l = 1$ and $m \geq 1$ it can be shown that [133], we have

$$E(X^1_{m,n}) = \int_0^1 (1 - (1 - S_m(t)e^{-t})^n)\, dt,$$

where $S_m(t) = \sum_{i=0}^{m-1} \frac{t^i}{i!}$.
For fixed m and large n,

$$E(X^1_{m,n}) = n\log(n) + n(m-1)\log(\log(n)) + nO(1).$$

3. For $l \geq 1$ and $m = 1$, one can show the following [138].

$$E(X_{1,n}^{l}) = \sum_{i=0}^{n-1} \frac{1}{\left(1 - \frac{\binom{i}{d}}{\binom{n}{d}}\right)},$$

where $\binom{i}{d} = 0$ if $i < d$.

For fixed m and large n,

$$E(X_{m,n}^{1}) \approx \frac{1}{l} n \log(n).$$

4. For $l \geq 1$ and $m \geq 1$, one can derive the following bound [139].

$$\frac{\binom{n}{l}}{n^l} \frac{E(X_{m,n}^{1})}{l} \leq E(X_{m,n}^{l}) \leq \frac{E(X_{m,n}^{1})}{l} + mn\left(1 - \frac{1}{l}\right).$$

For $l \geq 1$ (e.g., Cases 3 and 4 above), the problem is sometimes referred to as the Generalized Coupon Collector Problem. Several other variants of the CCP exist in the literature [130, 140, 141, 129, 142, 143, 144, 145, 146] with different assumptions and models.

3.3 Consecutive Coupon Collector Problem

In this chapter, we introduce a new variant called the Consecutive Coupon Collector Problem (CCCP) and derive the relevant expected number of trials for both non-circular and circular cases in the context of the consecutive bit-flip model. The result in Proposition 2 is directly used to calculate the probability of an event for the CCCP.

3.3.1 Non-Circular Consecutive Coupon Collector Problem

The Non-Circular variant of the CCCP is a scenario in cryptography and information security that considers an attacker repeatedly selecting a random bit from an n-bit register and flipping l bits to the right of the selected bit. The objective is to either select all n bits in the register or some t (less than or equal to n) number of bits

through multiple trials. This problem is an extension of the CCP and is of interest to researchers and practitioners in the field of cryptography and information security. To solve the problem, the first step is to find a solution for the boundary case, i.e. finding the expected number of trials required to flip all n bits in the register. Then, the solution for all cases, i.e. finding the expected number of trials to flip t bits (where t is between 1 and n) in the register, is provided.

3.3.1.1 Solution to a Boundary Case of Non-Circular Consecutive Coupon Collector Problem

Problem Statement.

Suppose an attacker targets to inject consecutive bits fault in a register containing n bits. In each trial, the attacker randomly chooses a bit from the register and then flips l bits (for some pre-defined number $l \leq n$) from the flipped one to the right and marks them as flipped. Then, what is the expected number of trials so that all the n bits get flipped?

Exact Theoretical Solution.

For a fixed n, l, let us define the random variable X_i as

$X_i :=$ Number of trials required for the bit position i to be flipped.

n/l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1.0																							
2	3.0	2.0																						
3	5.5	3.5	3.0																					
4	8.3	5.0	4.3	4.0																				
5	11.4	6.7	5.6	5.2	5.0																			
6	14.7	8.4	6.9	6.4	6.2	6.0																		
7	18.1	10.2	8.3	7.6	7.3	7.2	7.0																	
8	21.7	12.1	9.7	8.8	8.4	8.2	8.1	8.0																
9	25.5	14.1	11.1	10.0	9.5	9.3	9.2	9.1	9.0															
10	29.3	16.0	12.6	11.3	10.7	10.4	10.2	10.2	10.1	10.0														
11	33.2	18.0	14.1	12.6	11.9	11.5	11.3	11.2	11.1	11.1	11.0													
12	37.2	20.2	15.6	13.9	13.0	12.6	12.4	12.3	12.1	12.1	12.0	12.0												
13	41.3	22.2	17.2	15.2	14.3	13.8	13.5	13.3	13.2	13.1	13.1	13.1	13.0											
14	45.5	24.4	18.7	16.5	15.4	14.9	14.6	14.4	14.3	14.2	14.1	14.1	14.1	14.0										
15	49.8	26.6	20.3	17.8	16.7	16.1	15.7	15.4	15.3	15.2	15.1	15.1	15.1	15.1	15.0									
16	54.1	28.8	21.9	19.2	17.9	17.2	16.8	16.5	16.4	16.3	16.2	16.1	16.1	16.1	16.1	16.0								
17	58.5	31.1	23.5	20.5	19.1	18.3	17.9	17.6	17.4	17.3	17.2	17.2	17.1	17.1	17.1	17.1	17.0							
18	62.9	33.3	25.2	21.9	20.3	19.5	18.9	18.7	18.5	18.3	18.3	18.2	18.2	18.1	18.1	18.1	18.1	18.0						
19	67.4	35.6	26.8	23.3	21.6	20.7	20.1	19.8	19.5	19.4	19.3	19.2	19.2	19.1	19.1	19.1	19.1	19.1	19.0					
20	71.9	37.9	28.5	24.7	22.8	21.8	21.2	20.9	20.6	20.5	20.3	20.3	20.2	20.2	20.1	20.1	20.1	20.1	20.1	20.0				
21	76.5	40.3	30.1	26.1	24.1	22.9	22.3	21.9	21.7	21.5	21.4	21.3	21.2	21.2	21.1	21.1	21.1	21.1	21.1	21.1	21.0			
22	81.2	42.6	31.8	27.5	25.3	24.2	23.5	23.1	22.7	22.5	22.4	22.3	22.2	22.2	22.1	22.1	22.1	22.1	22.1	22.1	22.1	22.0		
23	85.9	45.1	33.5	28.9	26.6	25.3	24.6	24.1	23.8	23.6	23.5	23.4	23.3	23.2	23.2	23.1	23.1	23.1	23.1	23.1	23.1	23.0	23.0	
24	90.6	47.5	35.3	30.3	27.9	26.5	25.7	25.2	24.9	24.7	24.5	24.4	24.3	24.2	24.2	24.2	24.1	24.1	24.1	24.1	24.1	24.1	24.0	24.0

Table 3.1: Theoretical values for the boundary case of non-circular CCCP

Now, observe that, if we want bit position i to be flipped, then there exist some fixed bit positions which need to be flipped in the trials that makes our concerned position checked. We want the expected number of trials so that all the positions get checked. So, we need to find the expectation of the maximum number of trials required for n number of bits to get checked. Let us define a new random variable X as

$$X := \max_i X_i,$$

and we want to calculate $\mathbb{E}(X)$. Before proceeding into mathematical details, let us first fix some notations.

1. Distance between node i and node j is $d_{ij} :=\mid i - j \mid$.

2. WLOG, let $i < j$. The number of joint favorable hitting positions contributed by j given i is $v_{ij} := \min(d_{ij}, l)$.

Then, WLOG given $A = \{i_1, i_2, \ldots, i_p\} \subset \{1, 2, \ldots, n\}$, where $i_1 < i_2 < \ldots < i_p$ the total number of favorable hitting positions such that at least one of them is flipped is given by

$$h\left(i_1, i_2, \ldots, i_p\right) := v_{0i_1} + v_{i_1i_2} + v_{i_2i_3} + \cdots + v_{i_{p-1}i_p}.$$

Where, following our definition above, $v_{0i_1} = \min(d_{0i_1}, l) = \min(|0-i_1|, l) = \min(i_1, l)$. Thus, we can write down the joint probability as

$$\Pr(X_{i_1} > r, X_{i_2} > r, \ldots, X_{i_p} > r) = \left[1 - \frac{h\left(i_1, i_2, \ldots i_p\right)}{n}\right]^r.$$

Due to the independence of the trials, we will get by using the Principle of inclusion and exclusion,

$$\begin{aligned}
&\Pr(X > r) = \Pr(\max_i X_i > r) \\
&= \sum_i \Pr(X_i > r) - \sum_{i_1 < i_2} \Pr(X_{i_1} > r, X_{i_2} > r) + \cdots + (-1)^{n+1} \Pr(X_1 > r, X_2 > r, \ldots, X_n > r).
\end{aligned}$$

As the last term will be 0 for all $r \geq 1$. Then we have,

$$\Pr(X > r)$$
$$= \sum_i \left[1 - \frac{l}{n}\right]^r - \sum_{i_1<i_2} \left[1 - \frac{h(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \left[1 - \frac{h(i_1, i_2, \ldots i_{n-1})}{n}\right]^r .$$

Finally, we can conclude that,

$$\mathbb{E}(X) = \sum_{r\geq 0} \Pr(X > r)$$
$$= \sum_{r\geq 0}\sum_i \left[1 - \frac{l}{n}\right]^r - \sum_{r\geq 0}\sum_{i_1<i_2} \left[1 - \frac{h(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{r\geq 0}\sum_{i_1<\cdots<i_{n-1}} \left[1 - \frac{h(i_1, i_2, \ldots i_{n-1})}{n}\right]^r$$
$$= \sum_i\sum_{r\geq 0} \left[1 - \frac{l}{n}\right]^r - \sum_{i_1<i_2}\sum_{r\geq 0} \left[1 - \frac{h(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}}\sum_{r\geq 0} \left[1 - \frac{h(i_1, i_2, \ldots i_{n-1})}{n}\right]^r$$
$$= \sum_i \frac{n}{l} - \sum_{i_1<i_2} \frac{n}{h(i_1, i_2)} + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \frac{n}{h(i_1, i_2, \ldots i_{n-1})}$$
$$= n\left[\sum_i \frac{1}{l} - \sum_{i_1<i_2} \frac{1}{h(i_1, i_2)} + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \frac{1}{h(i_1, i_2, \ldots i_{n-1})}\right].$$

Empirical Validation.

We simulate the procedure 1,00,000 times for each value of n and l and take averages over those values to compare it with the theoretical values. Both the theoretical and the simulated values are given in Table 3.1 and Table 3.2 respectively. It is clear that our theoretical result is very close to the simulated one.

Counting observations beyond 3σ limits.

The corrplot in Figure 3-1 of the difference matrix obtained from the theoretical and the simulated values shows an idea of how close the values are.

Also, we plot the 3σ limits in Figure 3-2 around the mean differences to see how many of the observations fall beyond 3σ limit.

Notice that, out of the 300 observations, only 3 of them are not significantly close which is $\frac{3}{300} = 1\%$ of the total number of observations. Hence there is clear evidence that our calculations are correct.

Now, to make things more rigorous, we can do some hypothesis testing, which results in an interesting observation.

Hypothesis Testing.

For each fixed n and l, let, $Y^1_{n,l}, Y^2_{n,l}, \dots Y^m_{n,l}$ be m observations coming from some distribution with mean μ_{nl}. Then, we can frame our hypothesis as:

$$\mathcal{H}_0 : \mu_{nl} = t(n,l), \qquad \mathcal{H}_A : \mu_{nl} \neq t(n,l),$$

where, $t(n,l)$ is our calculated theoretical value. For our purpose, $m = 1,00,000$. So, we can do a large sample approximation here. We want to test at 5% level of significance. Then, the test statistic will be,

$$T = \frac{\sqrt{m}\,(\overline{Y_{n,l}} - \mu_{nl})}{s_{nl}},$$

where,

$$\overline{Y_{n,l}} = \frac{1}{m}\sum_{i=1}^{m} Y^i_{n,l}, \quad \text{and} \quad s_{nl} = \frac{1}{m-1}\sum_{i=1}^{m}(Y^i_{n,l} - \overline{Y_{n,l}})^2.$$

Since we are doing a both-sided test, our decision rule will be,

n/l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1.0																							
2	3.0	1.9																						
3	5.5	3.5	2.9																					
4	8.3	5.0	4.3	4.0																				
5	11.4	6.7	5.6	5.2	4.9																			
6	14.7	8.4	6.9	6.4	6.2	6.0																		
7	18.2	10.2	8.3	7.6	7.3	7.2	6.9																	
8	21.7	12.1	9.7	8.8	8.4	8.2	8.1	8.0																
9	25.4	14.0	11.1	10.0	9.5	9.3	9.2	9.1	9.0															
10	29.3	16.1	12.6	11.3	10.7	10.4	10.2	10.2	10.0	9.9														
11	33.2	18.1	14.0	12.5	11.9	11.4	11.3	11.2	11.1	11.1	10.9													
12	37.2	20.1	15.6	13.8	13.1	12.6	12.4	12.2	12.2	12.1	12.1	12.0												
13	41.4	22.3	17.2	15.1	14.2	13.8	13.5	13.4	13.2	13.2	13.2	13.1	13.0											
14	45.5	24.4	18.7	16.6	15.5	14.9	14.6	14.3	14.3	14.2	14.2	14.1	14.0	13.9										
15	49.8	26.6	20.3	17.8	16.7	16.0	15.6	15.4	15.4	15.2	15.1	15.1	15.1	15.1	14.9									
16	54.1	28.9	21.9	19.2	17.9	17.2	16.7	16.5	16.3	16.2	16.1	16.1	16.1	16.0	16.0	15.9								
17	58.5	31.0	23.5	20.4	19.1	18.4	17.9	17.6	17.4	17.3	17.2	17.1	17.1	17.1	17.1	17.0	16.9							
18	62.9	33.3	25.1	21.9	20.3	19.5	18.9	18.6	18.4	18.4	18.3	18.2	18.2	18.2	18.1	18.0	18.0	17.9						
19	67.4	35.6	26.8	23.3	21.6	20.7	20.1	19.9	19.4	19.4	19.3	19.3	19.2	19.2	19.2	19.0	19.0	19.1	19.0					
20	72.0	37.8	28.5	24.7	22.8	21.8	21.2	20.8	20.6	20.4	20.3	20.2	20.1	20.2	20.1	20.2	20.1	20.1	20.0	19.9				
21	76.4	40.3	30.1	26.0	24.0	23.1	22.3	21.9	21.6	21.5	21.4	21.2	21.1	21.2	21.2	21.1	21.1	21.1	21.1	20.9	20.9			
22	81.1	42.6	31.9	27.5	25.4	24.1	23.4	22.9	22.7	22.7	22.4	22.3	22.3	22.3	22.1	22.1	22.1	22.1	21.9	22.1	21.9	22.1		
23	85.9	45.1	33.5	28.7	26.7	25.5	24.6	24.2	23.8	23.6	23.5	23.4	23.2	23.2	23.2	23.1	23.1	23.3	23.0	23.1	23.1	23.2	22.9	
24	90.6	47.6	35.3	30.3	27.8	26.6	25.7	25.2	24.8	24.6	24.4	24.3	24.3	24.2	24.1	24.2	23.9	24.1	24.1	24.0	24.1	23.9	24.0	23.9

Table 3.2: Simulated values for the boundary case of non-circular CCCP

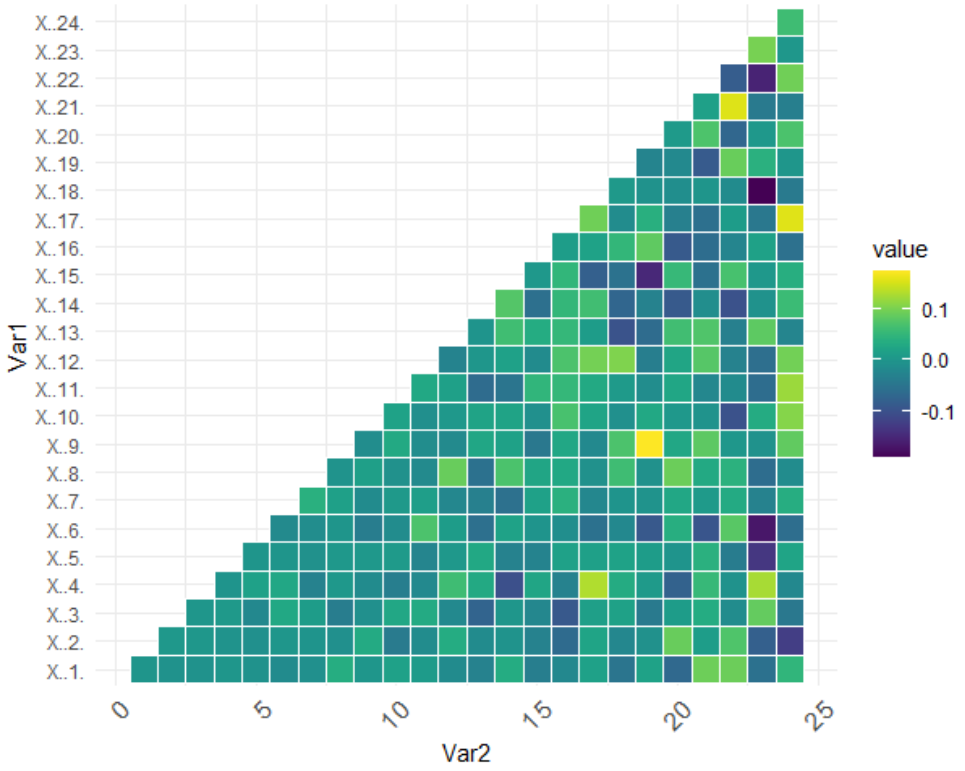


Figure 3-1: Corrplot of the difference matrix between theoretical and simulated values for the boundary case of non-circular CCCP

$$\boxed{\text{Reject } \mathcal{H}_0 \text{ if } |T_{obs}| > 1.96}.$$

Now, we are doing $\frac{25\times24}{2} = 300$ many testings. Out of them 17 was rejected which is $\frac{17}{300} \times 100\% = 5.67\%$ of the total values.

3.3.1.2 Solution to All Cases of Non-Circular Consecutive Coupon Collector Problem

Problem Statement.

Suppose an attacker targets to inject consecutive bits fault in a register containing n number of bits. In each trial, the attacker randomly chooses a bit from the register and then flips l bits (for some pre-defined number $l \leq n$) from the flipped one to the right and marks them as flipped. Then, what is the expected number of trials so that t $(1 \leq t \leq n)$ number of bits get flipped?

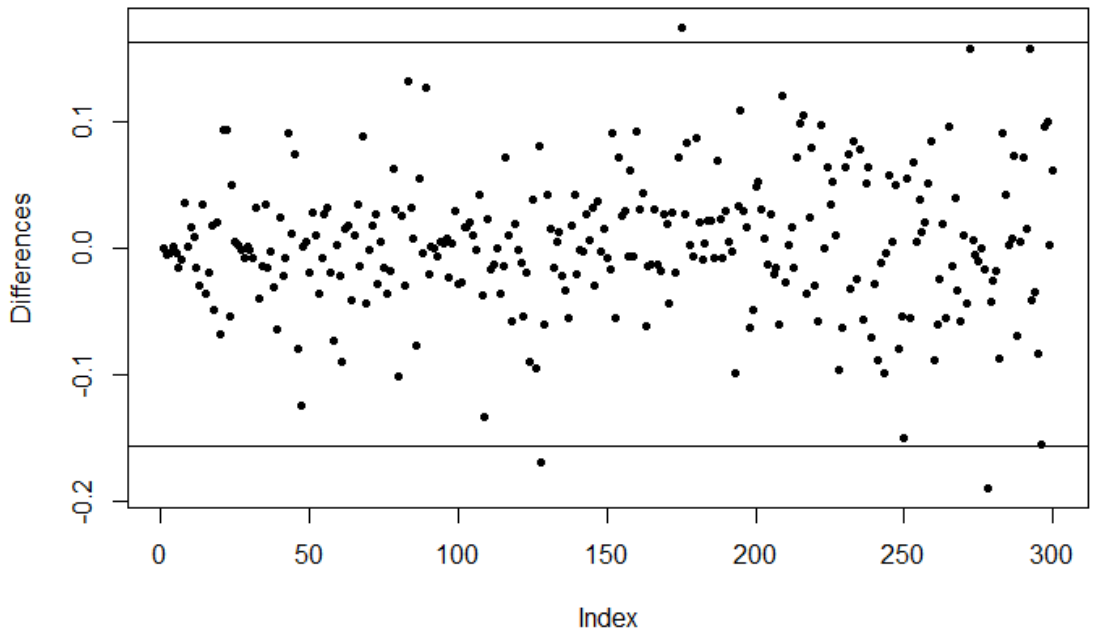


Figure 3-2: Mean differences between theoretical and simulated values for the boundary case of non-circular CCCP with 3σ limit

Exact Theoretical Solution.

For a fixed n, l, and t, let us define the random variable X_i as

$$X_i := \text{ Number of trials required for the bit position } i \text{ to be flipped.}$$

Now, observe that, if we want bit position i to be flipped, then there exist some fixed bit positions which need to be flipped in the trials that make our concerned position checked. We want the expected number of trials so that all the positions get checked. So, we need to find the expectation of the maximum number of trials required so that $1 \leq t \leq n$ number of bits get checked in the register. Let us define a new random variable X as

$$X := \min_{\{i_1, i_2, \cdots, i_t\} \subset \{1, 2, \cdots, n\}} \quad \max_{i \in \{i_1, i_2, \cdots, i_t\}} X_i,$$

and we want to calculate $\mathbb{E}(X)$. Before proceeding to the mathematical details, let us first fix some notations.

1. Distance between node i and node j is $d_{ij} :=\mid i-j \mid$.

2. WLOG, let $i < j$. The number of joint favorable hitting positions contributed by j given i is $v_{ij} := \min(d_{ij}, l)$.

Then, WLOG given $A = \{i_1, i_2, \ldots, i_p\} \subset \{1, 2, \ldots, n\}$, where $i_1 < i_2 < \ldots < i_p$ the total number of favorable hitting positions such that at least one of them is flipped is given by

$$h\left(i_1, i_2, \ldots, i_p\right) := v_{0i_1} + v_{i_1 i_2} + v_{i_2 i_3} + \cdots + v_{i_{p-1} i_p}.$$

Where, following our definition above, $v_{0i_1} = \min(d_{0i_1}, l) = \min(|0-i_1|, l) = \min(i_1, l)$. Thus, we can write down the joint probability as

$$\Pr(X_{i_1} > r, X_{i_2} > r, \ldots, X_{i_p} > r) = \left[1 - \frac{h\left(i_1, i_2, \ldots i_p\right)}{n}\right]^r.$$

Due to the independence of the trials, we will get by using Proposition 2,

$$\begin{aligned}
&\Pr(X > r) = \Pr(X_i > r \text{ for at least } n-t+1 \text{ values of } i) \\
&= \sum_{i_1<\cdots<i_{n-t+1}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+1}} > r) - \binom{n-t+1}{1} \times \sum_{i_1<\cdots<i_{n-t+2}} \Pr(X_{i_1} > r, \cdots \\
&, X_{i_{n-t+2}} > r) + \binom{n-t+2}{2} \times \sum_{i_1<\cdots<i_{n-t+3}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+3}} > r) + \ldots \\
&+ (-1)^{t-1} \cdot \binom{n-1}{t-1} \times \sum_{i_1<\cdots<i_n} \Pr(X_{i_1} > r, \cdots, X_{i_n} > r) \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+k}} > r)\right] \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \left[1 - \frac{h\left(i_1, i_2, \ldots, i_{n-t+k}\right)}{n}\right]^r\right].
\end{aligned}$$

Finally, we can conclude that,

$$
\begin{aligned}
&\mathbb{E}(X) = \sum_{r\geq 0} \Pr(X > r) \\
&= \sum_{r\geq 0}\sum_{k=1}^{t}\left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \left[1 - \frac{h\left(i_1, i_2, \ldots, i_{n-t+k}\right)}{n}\right]^r\right] \\
&= \sum_{k=1}^{t}\left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \sum_{r\geq 0} \left[1 - \frac{h\left(i_1, i_2, \ldots, i_{n-t+k}\right)}{n}\right]^r\right] \\
&= \sum_{k=1}^{t}\left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \frac{n}{h\left(i_1, i_2, \ldots, i_{n-t+k}\right)}\right] \\
&= n \cdot \sum_{k=1}^{t}\left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \frac{1}{h\left(i_1, i_2, \ldots, i_{n-t+k}\right)}\right].
\end{aligned}
$$

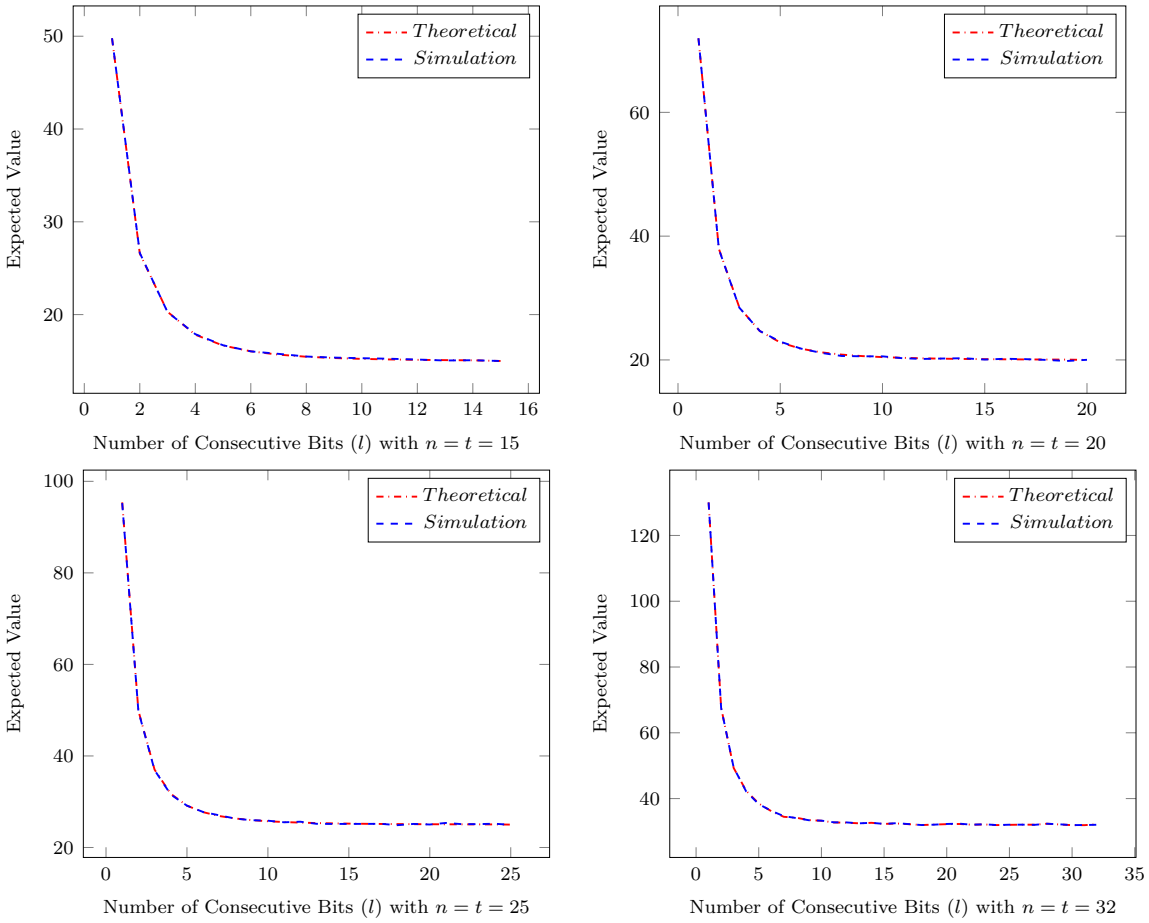


Figure 3-3: Graphical comparison of expected values for $n = t = 15, 20, 25, 32$

Empirical Validation of the Above Solution.

Now we want to verify whether our calculations empirically matched with the simulation. We simulate the procedure 1,00,000 times for each value of n, l, and t. Upon comparison, we find that the theoretical values closely match the corresponding experimental ones. Figure 3-3 shows a graphical representation of the expected value (vertical axis) with the l consecutive number of bits (horizontal axis) for some specific values of $n = t = 10, 15, 25, 32$.

Additionally, we perform hypothesis testing to validate our calculations of the problem as follows.

Hypothesis Testing.

For each fixed n, t, and k, let, $Y^1_{n,t,k}, Y^2_{n,t,k}$,
$\dots Y^m_{n,t,k}$ be m observations coming from some distribution with mean μ_{ntk}. Then, we can frame our hypothesis as:

$$\mathcal{H}_0 : \mu_{ntk} = nc(n,t,k), \qquad \mathcal{H}_A : \mu_{ntk} \neq nc(n,t,k),$$

where, $nc(n,t,k)$ is our calculated theoretical value. For our purpose, $m = 50,000$. So, we can do large sample approximation here. We want to test at 5% level of significance. Then, the test statistic will be,

$$T = \frac{\sqrt{m}\ (\overline{Y_{n,t,k}} - nc(n,t,k))}{s_{n,t,k}},$$

where,

$$\overline{Y_{n,t,k}} = \frac{1}{m}\sum_{i=1}^{m} Y^i_{n,t,k}, \quad \text{and} \quad s_{n,t,k} = \frac{1}{m-1}\sum_{i=1}^{m}(Y^i_{n,t,k} - \overline{Y_{n,t,k}})^2.$$

Under $\mathcal{H}_0$, $T \sim \mathcal{N}(0,1)$. Since we are doing a both-sided test, our decision rule will be,

$$\boxed{\text{Reject } \mathcal{H}_0 \text{ if } |T_{obs}| > 1.96}.$$

Now, observe that some of the observations must be accurate to the theoretical values. So, those observations totally coincides with their theoretical values and do not qualify for testing. To be very specific,

- For some fixed n and t, it is enough to consider k's only from 1 to t because if $k \geq t+1$, then, the events are identical to the one when $k = t$.
- For any n, when $t = 1$, that means we want to see how many trials are required to mark at least one of the entries checked and that value is always 1.
- Since the calculation of theoretical values is computationally expensive, we are testing up to $n = 23$.

3.3.1.3 Total Number of Hypothesis Testing

For each fixed n, we are taking t from 2 to n, that is $n-1$ values and for each fixed t value, we are considering k values from 1 to t. So, for $n = 1, 2, \ldots, 23$, the total number of test performed is given by

$$\sum_{n=2}^{23}\sum_{t=2}^{n} t = \sum_{n=2}^{23}\left[\frac{(n+1)n}{2} - 1\right] = \frac{1}{2}\sum_{n=2}^{23}[n^2+n-2] = \frac{1}{2}\left[\sum_{n=2}^{23} n^2 + \sum_{n=2}^{23} n - 22\times 2\right]$$

$$= \frac{1}{2}\left[\sum_{n=1}^{23} n^2 + \sum_{n=1}^{23} n - 23\times 2\right] = \frac{1}{2}\left[\frac{23\times 24\times 47}{6} + \frac{23\times 24}{2} - 46\right] = 2277.$$

Out of these 2277 many testing procedures, only 123 many have been rejected which is $\frac{123}{2277}\times 100 = 5.4\%$ of the total testing problems, which is significantly small, conforming to our theoretical estimates.

3.3.2 Circular Consecutive Coupon Collector Problem

The above problem can extend to a new problem where an attacker circularly flips consecutive bits. In this case, an attacker first selects the bit position i and then flips l consecutive bits areas $i, i+1, \cdots, (i+k-1) \pmod n$. This problem looks very similar to the Non-circular Consecutive problem. We will give the solution to this problem from our theoretical interests. To the best of our knowledge, this might

also be the first time where we will provide a theoretical solution to this problem and show that these values closely behave with the simulated values.

3.3.2.1 Solution to a Boundary Case of Circular Consecutive Coupon Collector Problem

Problem Statement.

Suppose an attacker targets to inject consecutive bits fault in a register containing n number of bits. In each trial, the attacker randomly chooses a bit i from the register and then flips l bits circularly (for some pre-defined number $l \leq n$), i.e., bits $i, i+1, \cdots, i+l-1 \pmod{n}$ and marks them as flipped. Then, what is the expected number of trials so that all the n bits get flipped?

Exact Theoretical Solution.

For a fixed n and l, let us define the random variable X_i as

$$X_i := \text{ Number of trials required for the bit position } i \text{ to get selected.}$$

n/l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1																							
2	3	1																						
3	5.5	2.5	1																					
4	8.3	3.7	2.3	1																				
5	11.4	5.2	3.1	2.2	1																			
6	14.7	6.7	4	2.8	2.2	1																		
7	18.1	8.3	5.1	3.4	2.6	2.2	1																	
8	21.7	10.1	6.1	4.2	3.1	2.5	2.1	1																
9	25.5	11.9	7.3	5.1	3.7	2.9	2.4	2.1	1															
10	29.3	13.7	8.4	5.9	4.3	3.4	2.8	2.4	2.1	1														
11	33.2	15.6	9.6	6.7	5	3.9	3.2	2.7	2.3	2.1	1													
12	37.2	17.5	10.9	7.6	5.7	4.4	3.6	3	2.6	2.3	2.1	1												
13	41.3	19.5	12.1	8.5	6.4	5.0	4.0	3.3	2.9	2.5	2.3	2.1	1											
14	45.5	21.6	13.4	9.4	7.1	5.6	4.5	3.7	3.2	2.8	2.5	2.2	2.1	1										
15	49.8	23.6	14.8	10.4	7.8	6.1	5.0	4.1	3.5	3.0	2.7	2.4	2.2	2.1	1									
16	54.1	25.7	16.1	11.3	8.6	6.7	5.5	4.5	3.8	3.3	2.9	2.6	2.4	2.2	2.1	1								
17	58.5	27.9	17.5	12.3	9.3	7.3	5.9	5.0	4.2	3.6	3.2	2.8	2.6	2.4	2.2	2.1	1							
18	62.9	30	18.9	13.3	10.1	7.9	6.5	5.4	4.6	3.9	3.4	3.1	2.8	2.5	2.3	2.2	2.1	1						
19	67.4	32.2	20.3	14.4	10.9	8.6	7.0	5.9	5.0	4.3	3.7	3.3	2.9	2.7	2.5	2.3	2.2	2.0	1					
20	71.9	34.4	21.7	15.4	11.7	9.2	7.5	6.3	5.4	4.6	4.0	3.6	3.2	2.9	2.7	2.5	2.3	2.2	2.0	1				
21	76.5	36.7	23.2	16.4	12.5	9.9	8.1	6.8	5.7	5.0	4.3	3.8	3.3	3.1	2.8	2.6	2.4	2.3	2.2	2.1	1			
22	81.2	38.9	24.6	17.5	13.3	10.5	8.6	7.2	6.2	5.3	4.7	4.1	3.6	3.3	3.0	2.8	2.6	2.4	2.3	2.1	2.0	1		
23	85.9	41.3	26.1	18.6	14.1	11.2	9.2	7.7	6.6	5.7	5.0	4.4	3.9	3.5	3.2	2.9	2.7	2.5	2.4	2.2	2.1	2.0	1	
24	90.6	43.6	27.6	19.7	14.9	11.9	9.7	8.2	6.9	6.0	5.3	4.7	4.2	3.7	3.4	3.1	2.9	2.7	2.5	2.4	2.2	2.1	2.0	1

Table 3.3: Theoretical values for the boundary case of circular CCCP

Now, observe that, if we want the bit position i to be flipped, then there exist some fixed bit positions which need to be flipped in the trials that make our concerned position checked. We want the expected number of trials so that all the bit positions get checked. So, we need to find the expectation of the maximum number of trials required so that all bits in the register get selected. Let us define a new random variable X as

$$X := \max_i X_i,$$

and we want to calculate $\mathbb{E}(X)$. Before proceeding to the mathematical details, let us first fix some notations.

1. Anti clock-wise distance between node i and node j is

$$d_{ij} := \begin{cases} |i-j| & |i-j| < \frac{n}{2} \\ n - |i-j| & |i-j| \geq \frac{n}{2}. \end{cases}$$

2. WLOG, let $i < j$ and $|i-j| < \frac{n}{2}$. Then the number of joint favorable hitting positions contributed by j given i is $v_{ij} := \min(d_{ij}, l)$.

Then, WLOG given $A = \{i_1, i_2, \ldots i_p\} \subset \{1, 2, \ldots, n\}$, where $i_1 < i_2 < \ldots < i_p$ the total number of favorable hitting positions such that at least one of them is selected is given by

$$h\left(i_1, i_2, \ldots, i_p\right) := v_{i_1 i_2} + v_{i_2 i_3} + \ldots v_{i_{p-1} i_p} + v_{i_p i_1}$$

Thus, we can write down the joint probability as

$$\Pr(X_{i_1} > r, X_{i_2} > r, \ldots, X_{i_p} > r) = \left[1 - \frac{h\left(i_1, i_2, \ldots, i_p\right)}{n}\right]^r.$$

since the trials are independent, we will get by using the Principle of inclusion and exclusion

$$\begin{aligned}
&\Pr(X > r) = \Pr(\max_i X_i > r) \\
&= \sum_i \Pr(X_i > r) - \sum_{i_1<i_2} \Pr(X_{i_1} > r, X_{i_2} > r) + \cdots + (-1)^{n+1} \Pr(X_1 > r, X_2 > r, \ldots, X_n > r) \\
&= \sum_i \left[1 - \frac{l}{n}\right]^r - \sum_{i_1<i_2} \left[1 - \frac{h\,(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \left[1 - \frac{h\,(i_1, i_2, \ldots i_{n-1})}{n}\right]^r .
\end{aligned}$$

Since the last term will be 0 for all $r \geq 1$. And finally, we can conclude that,

$$\begin{aligned}
&\mathbb{E}(X) = \sum_{r\geq 0} \Pr(X > r) \\
&= \sum_{r\geq 0}\sum_i \left[1 - \frac{l}{n}\right]^r - \sum_{r\geq 0}\sum_{i_1<i_2} \left[1 - \frac{h\,(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{r\geq 0}\sum_{i_1<\cdots<i_{n-1}} \left[1 - \frac{h\,(i_1, i_2, \ldots i_{n-1})}{n}\right]^r \\
&= \sum_i\sum_{r\geq 0} \left[1 - \frac{l}{n}\right]^r - \sum_{i_1<i_2}\sum_{r\geq 0} \left[1 - \frac{h\,(i_1, i_2)}{n}\right]^r + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}}\sum_{r\geq 0} \left[1 - \frac{h\,(i_1, i_2, \ldots i_{n-1})}{n}\right]^r \\
&= \sum_i \frac{n}{l} - \sum_{i_1<i_2} \frac{n}{h\,(i_1, i_2)} + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \frac{n}{h\,(i_1, i_2, \ldots i_{n-1})} \\
&= n \left[\sum_i \frac{1}{l} - \sum_{i_1<i_2} \frac{1}{h\,(i_1, i_2)} + \cdots + (-1)^n \sum_{i_1<\cdots<i_{n-1}} \frac{1}{h\,(i_1, i_2, \ldots i_{n-1})}\right].
\end{aligned}$$

Empirical Validation.

Now we want to verify whether our calculations are true. To check this, we have simulated the procedure 1,00,000 times for each value of n and l and taken averages over those values to compare them with the theoretical values. Both the theoretical and the simulated results are given in Table 3.3 and Table 3.4 respectively and showed that our theoretical result is very close to the simulated one.

Counting observations beyond 3σ limits.

The corrplot is given in Figure 3-4 of the difference matrix obtained from the theoretical and the simulated values to have an idea about how close the values are.

Next, we plot the 3σ limits in Figure 3-5 around the mean differences to see how many of the observations fall beyond 3σ limit.

Notice that, out of the 300 observations, only 7 of them are not significantly close

n/l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1																							
2	3.0	1																						
3	5.5	2.5	1																					
4	8.3	3.7	2.3	1																				
5	11.4	5.2	3.15	2.2	1																			
6	14.7	6.7	4.0	2.8	2.2	1																		
7	18.1	8.3	5.1	3.4	2.6	2.2	1																	
8	21.7	10.1	6.2	4.2	3.1	2.5	2.1	1																
9	25.4	11.8	7.3	5.0	3.7	2.9	2.4	2.1	1															
10	29.3	13.7	8.4	5.9	4.3	3.4	2.8	2.3	2.1	1														
11	33.3	15.6	9.6	6.7	5.0	3.9	3.1	2.7	2.3	2.1	1													
12	37.2	17.5	10.9	7.6	5.7	4.4	3.6	3.0	2.6	2.3	2.1	1												
13	41.4	19.5	12.1	8.5	6.4	5.0	4.0	3.3	2.9	2.5	2.3	2.1	1											
14	45.5	21.6	13.4	9.4	7.1	5.6	4.5	3.7	3.2	2.8	2.5	2.2	2.0	1										
15	49.8	23.6	14.8	10.4	7.8	6.1	5.0	4.1	3.5	3.0	2.7	2.4	2.2	2.1	1									
16	54.1	25.7	16.1	11.4	8.5	6.8	5.5	4.5	3.9	3.3	2.9	2.6	2.4	2.2	2.1	1								
17	58.4	27.8	17.5	12.3	9.3	7.3	5.9	5.0	4.2	3.6	3.2	2.8	2.6	2.4	2.2	2.1	1							
18	62.9	30.1	18.9	13.3	10.1	7.9	6.5	5.4	4.6	3.9	3.4	3.1	2.8	2.5	2.3	2.2	2.1	1						
19	67.3	32.2	20.3	14.4	10.9	8.6	7.0	5.9	5.0	4.3	3.7	3.3	2.9	2.7	2.5	2.3	2.2	2.1	1					
20	71.9	34.4	21.8	15.4	11.7	9.2	7.6	6.3	5.4	4.6	4.0	3.6	3.2	2.9	2.7	2.5	2.3	2.2	2.0	1				
21	76.5	36.7	23.2	16.4	12.5	9.9	8.1	6.8	5.8	5.0	4.3	3.8	3.4	3.1	2.8	2.6	2.4	2.3	2.1	2.0	1			
22	81.2	38.9	24.7	17.5	13.3	10.5	8.6	7.2	6.2	5.3	4.7	4.1	3.6	3.3	3.0	2.8	2.6	2.4	2.3	2.1	2.0	1		
23	85.9	41.3	26.1	18.6	14.1	11.2	9.2	7.7	6.6	5.7	4.9	4.4	3.9	3.5	3.2	2.9	2.7	2.5	2.4	2.2	2.1	2.0	1	
24	90.6	43.6	27.6	19.6	14.9	11.9	9.7	8.1	6.9	6.0	5.3	4.7	4.2	3.7	3.4	3.1	2.9	2.7	2.5	2.4	2.2	2.1	2.0	1

Table 3.4: Simulated values for the boundary case of circular CCCP

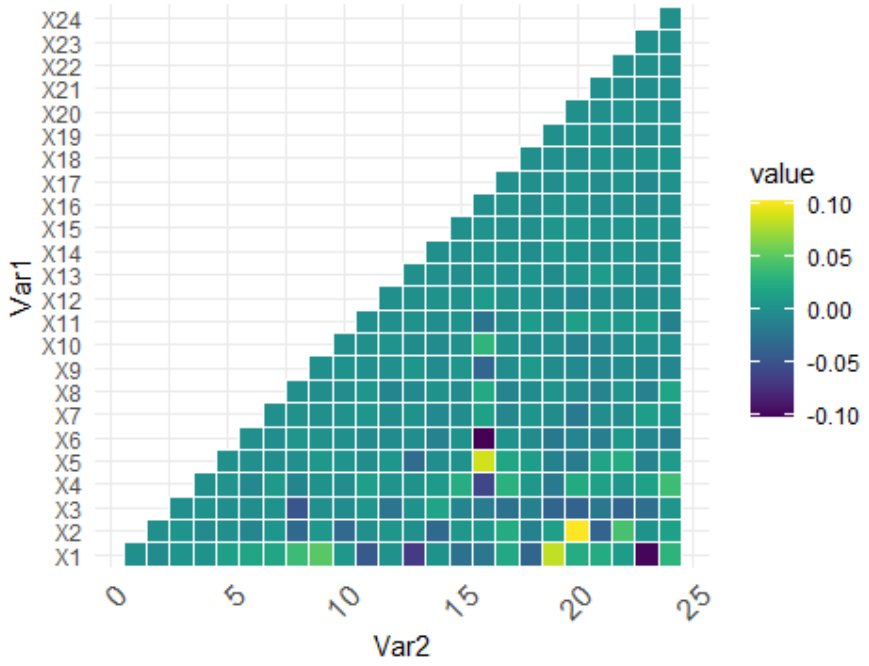


Figure 3-4: Corrplot of the difference matrix between theoretical and simulated values for the boundary case of circular CCCP

which is $\frac{7}{300} = 2.33\%$ of the total number of observations. Hence there is clear evidence that our calculations are correct.

Now, to make things more rigorous, we can do some hypothesis testing, which results in an interesting observation.

Hypothesis Testing.

For each fixed n and l, let, $Y_{n,l}^1, Y_{n,l}^2, \ldots Y_{n,l}^m$ be m observations coming from some distribution with mean μ_{nl}. Then, we can frame our hypothesis as:

$$\mathcal{H}_0 : \mu_{nl} = t(n,l), \qquad \mathcal{H}_A : \mu_{nl} \neq t(n,l),$$

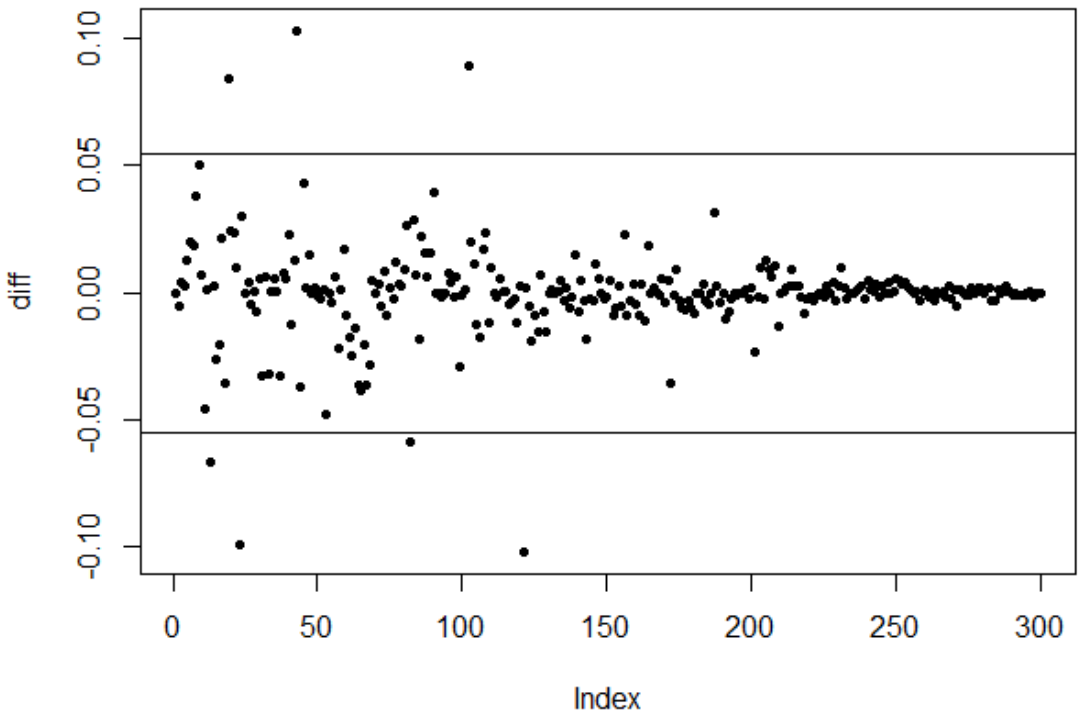


Figure 3-5: Mean differences between theoretical and simulated values for the boundary case of circular CCCP with 3σ limit

where, $t(n,l)$ is our calculated value. For our purpose, $m = 1,00,000$. So, we can do large sample approximation here. We want to test at 5% level of significance. Then, the test statistic will be,

$$T = \frac{\sqrt{m}\ (\overline{Y_{n,l}} - \mu_{nl})}{s_{nl}},$$

where, $\overline{Y_{n,l}} = \frac{1}{m}\sum_{i=1}^{m} Y_{n,l}^{i}$, and $s_{nl} = \frac{1}{m-1}\sum_{i=1}^{m}(Y_{n,l}^{i} - \overline{Y_{n,l}})^2$. Since we are doing a both-sided test, our test procedure will be,

$$\boxed{\text{Reject } \mathcal{H}_0 \text{ if } |T| > 1.96}.$$

Now, when $n = l$, then all simulated values match exactly with the theoretical values. So, we are just testing for the cases when $l < n$ i.e. we are doing $\frac{23 \times 24}{2} = 276$ many testings. Out of them 22 was rejected which is $\frac{22}{276} \times 100\% = 7.9\%$ of the total values.

3.3.2.2 Solution to All Cases of Circular Consecutive Coupon Collector Problem

Problem Statement. Suppose an attacker targets to inject consecutive bits fault in a register containing n number of bits. In each trial, the attacker randomly chooses a bit i from the register and then flips l bits circularly (for some pre-defined number $l \leq n$), i.e., bits $i, i+1, \cdots, i+l-1 \pmod{n}$ and marks them as flipped. Then, what is the expected number of trials so that t $(1 \leq t \leq n)$ number of bits get flipped?

Exact Theoretical Solution.

For a fixed n, l, and t, let us define the random variable X_i as

$$X_i := \text{ Number of trials required for the bit position } i \text{ to be flipped.}$$

Now, observe that, if we want the bit position i to be flipped, then there exist some fixed bit positions which need to be flipped in the trials that make our concerned position checked. We want the expected number of trials so that *tleqn* bit positions get checked. In summary, we need to find the expectation of the maximum number

of trials required so that $1 \leq t \leq n$ many bits get selected in the register. Let us define a new random variable X as

$$X := \min_{\{i_1,i_2,\cdots,i_t\} \subset \{1,2,\cdots,n\}} \max_{i \in \{i_1,i_2,\cdots,i_t\}} X_i,$$

and we want to calculate $\mathbb{E}(X)$. Before proceeding into mathematical details, let us first fix some notations.

1. Anti clock-wise distance between node i and node j is

$$d_{ij} := \begin{cases} |i-j| & |i-j| < \frac{n}{2} \\ n - |i-j| & |i-j| \geq \frac{n}{2}. \end{cases}$$

2. WLOG, let $i < j$ and $|i-j| < \frac{n}{2}$. Then the number of joint favorable hitting positions contributed by j given i is $v_{ij} := \min(d_{ij}, l)$.

Then, WLOG given $A = \{i_1, i_2, \ldots, i_p\} \subset \{1, 2, \ldots, n\}$, where $i_1 < i_2 < \ldots < i_p$ the total number of favorable hitting positions such that at least one of them is selected is given by

$$h\left(i_1, i_2, \ldots, i_p\right) := v_{i_1 i_2} + v_{i_2 i_3} + \cdots + v_{i_{p-1} i_p} + v_{i_p i_1}$$

Thus, we can write down the joint probability as

$$\Pr(X_{i_1} > r, X_{i_2} > r, \ldots, X_{i_p} > r) = \left[1 - \frac{h\left(i_1, i_2, \ldots i_p\right)}{n}\right]^r$$

Since the trials are independent, we will get by using Proposition 2,

$$\begin{aligned}
&\Pr(X > r) = \Pr(X_i > r \text{ for at least } n-t+1 \text{ values of } i) \\
&= \sum_{i_1<\cdots<i_{n-t+1}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+1}} > r) - \binom{n-t+1}{1} \times \sum_{i_1<\cdots<i_{n-t+2}} \Pr(X_{i_1} > r, \cdots \\
&, X_{i_{n-t+2}} > r) + \binom{n-t+2}{2} \times \sum_{i_1<\cdots<i_{n-t+3}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+3}} > r) + \ldots \\
&+ (-1)^{t-1} \cdot \binom{n-1}{t-1} \times \sum_{i_1<\cdots<i_n} \Pr(X_{i_1} > r, \cdots, X_{i_n} > r) \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \Pr(X_{i_1} > r, \cdots, X_{i_{n-t+k}} > r) \right] \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \left[1 - \frac{h(i_1, i_2, \ldots, i_{n-t+k})}{n} \right]^r \right].
\end{aligned}$$

Finally, we can conclude that,

$$\begin{aligned}
&\mathbb{E}(X) = \sum_{r\geq 0} \Pr(X > r) \\
&= \sum_{r\geq 0} \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \left[1 - \frac{h(i_1, i_2, \ldots, i_{n-t+k})}{n} \right]^r \right] \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \sum_{r\geq 0} \left[1 - \frac{h(i_1, i_2, \ldots, i_{n-t+k})}{n} \right]^r \right] \\
&= \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \frac{n}{h(i_1, i_2, \ldots, i_{n-t+k})} \right] \\
&= n \cdot \sum_{k=1}^{t} \left[(-1)^{k-1} \times \binom{n-t+k-1}{k-1} \times \sum_{i_1<\cdots<i_{n-t+k}} \frac{1}{h(i_1, i_2, \ldots, i_{n-t+k})} \right].
\end{aligned}$$

Now we want to verify whether our calculations are true. To check this, we have simulated the procedure 1,00,000 times for each value of n, l, and t. Figure 3-6 shows a graphical comparison of the theoretical vs. simulation results for the specific values of $n = t = 10, 15, 25, 32$. From the above graphical comparison, our theoretical calculations perfectly matched the simulated values.

Hypothesis Testing.

For each fixed n, t and k, let, $Y^1_{n,t,k}, Y^2_{n,t,k}, \ldots Y^m_{n,t,k}$ be m observations coming from some distribution with mean μ_{ntk}. Then, we can frame our hypothesis as:

$$\mathcal{H}_0 : \mu_{ntk} = c(n,t,k), \qquad \mathcal{H}_A : \mu_{ntk} \neq c(n,t,k),$$

where, $c(n,t,k)$ is our calculated theoretical value. For our purpose, $m = 50,000$. So, we can do large sample approximation here. We want to test at 5% level of significance. Then, the test statistic will be,

$$T = \frac{\sqrt{m}\,(\overline{Y_{n,t,k}} - c(n,t,k))}{s_{n,t,k}},$$

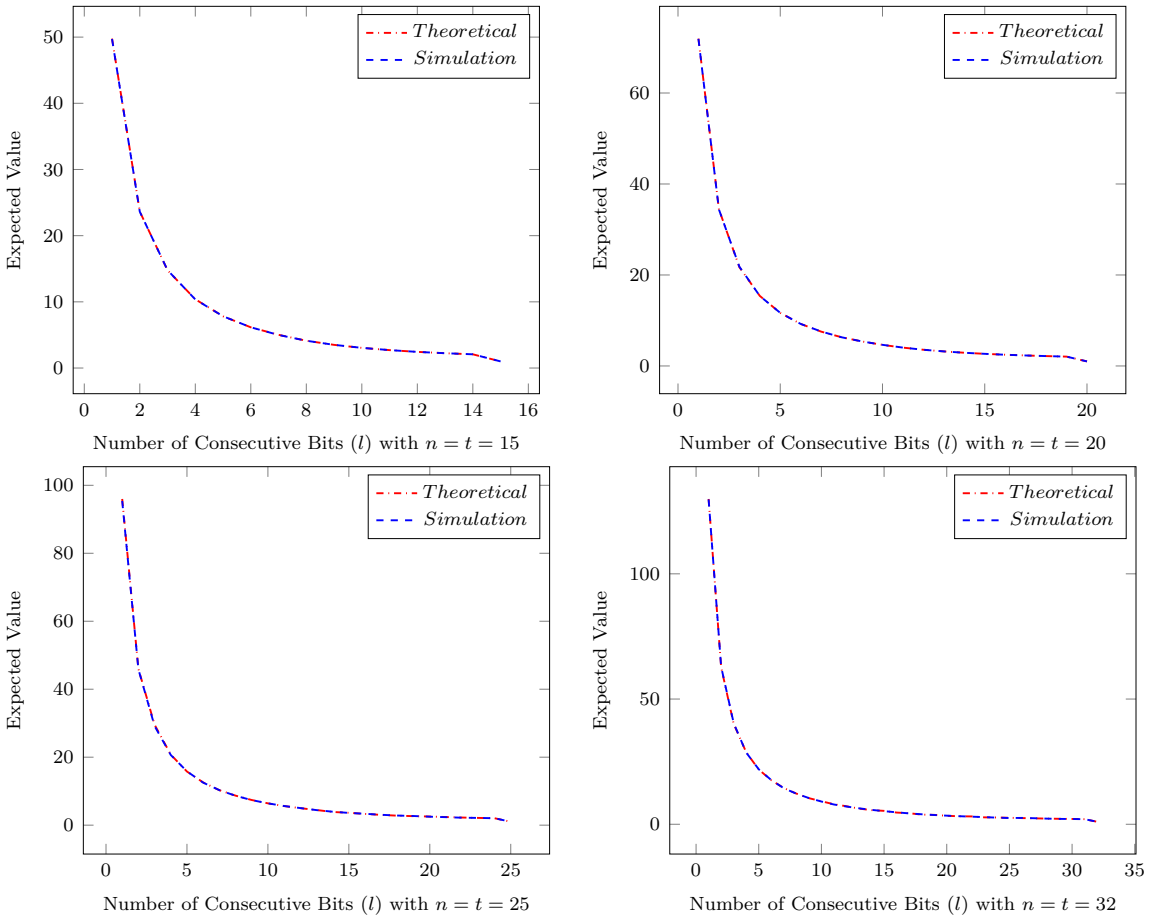


Figure 3-6: Graphical comparison of expected values for $n = t = 15, 20, 25, 32$

where,

$$\overline{Y_{n,t,k}} = \frac{1}{m}\sum_{i=1}^{m} Y^i_{n,t,k}, \quad \text{and} \quad s_{n,t,k} = \frac{1}{m-1}\sum_{i=1}^{m}(Y^i_{n,t,k} - \overline{Y_{n,t,k}})^2.$$

Under $\mathcal{H}_0$, $T \sim \mathcal{N}(0,1)$. Since we are doing a both-sided test, our decision rule will be,

$$\boxed{\text{Reject } \mathcal{H}_0 \text{ if } |T_{obs}| > 1.96}.$$

Now, observe that some of the observations must be accurate to the theoretical values. So, those observations totally coincides with their theoretical values and do not qualify for testing. To be very specific,

1. For some fixed n and t, we need to consider k's only from 1 to $t-1$ because if $k \geq t$, then, t many positions get marked only in one go.
2. For any n, when $t = 1$, that means we want to see how many trials are required to mark at least one of the entries checked and that value is always 1.
3. Since the calculation of theoretical values is computationally expensive, we are testing upto $n = 23$

3.3.2.3 Total Number of Hypothesis Testing

For each fixed n, we are taking t from 2 to n, that is $n-1$ values and for each fixed t value, we are considering k values from 1 to $t-1$. So for $n = 1, 2, \ldots, 23$, we are testing a total of

$$\sum_{n=2}^{23}\sum_{t=2}^{n}(t-1) = \sum_{n=2}^{23}\left[\sum_{t=2}^{n} t - (n-1)\right] = \sum_{n=2}^{23}\left[\frac{(n+1)n}{2} - 1 - n + 1\right] = \sum_{n=2}^{23}\frac{n^2+n-2n}{2}$$

$$= \frac{1}{2}\left[\sum_{n=2}^{23} n^2 - \sum_{n=2}^{23} n\right] = \frac{1}{2}\left[\sum_{n=1}^{23} n^2 - \sum_{n=1}^{23} n\right] = \frac{1}{2}\left[\frac{23\times 24\times 47}{6} - \frac{23\times 24}{2}\right] = 2024.$$

Out of these 2024 many testing procedures, only 125 many have been rejected which is $\frac{125}{2024}\times 100 = 6.1\%$ of the total testing problems, which is significantly small. So, we can expect that our theoretical values are correct.

3.4 Some observations

Now, it is of interest not how many of them gets rejected but which of them are getting rejected. We have done this testing procedure multiple times and have observed that, in each case, the anomaly includes observations for some particular values of n. In the above case, there were 3 observations for $n = 8$ and 7 observations for $n = 16$.

We are including another set of simulated values for which the testing procedure was performed. Here in total 26 hypothesis got rejected, which is $\frac{26}{276} \times 100\% = 9.4\%$ of the total values which is less than 10% but interestingly enough, out of these, 4 were from $n = 8$ and 9 were from $n = 16$. We can also look at the corrplot 3-4 and compare it with the previous one presented above and immediately identify some similarities where the simulated and theoretical values are not close enough.

3.5 Conclusion

This chapter focuses on two new variants of the generalized coupon collector problem, namely the non-circular and circular consecutive coupon collector problems. We have provided theoretical solutions to both of these variants, and our testing procedures have validated our calculations for both problems.

Our results have demonstrated a high level of accuracy, with over 90% of the observed values closely matching the simulated values for both non-circular and circular coupon collector problems. However, as previously mentioned, further investigation is needed to address the issue of fluctuations in the data when the number of seats, n, is a power of 2. It appears that the speed at which seats are checked out may vary for these values of n, which could potentially impact the accuracy of our results.

To address this issue, we need to conduct additional tests with higher powers of 2 to determine if the fluctuations persist. By identifying and analyzing the factors contributing to the variability in the checkout speed for power-of-2 values of n, we can refine our model and improve the accuracy and reliability of our calculations. Overall, our work on the non-circular and circular consecutive coupon collector problems provides new insights and solutions to these important problems in probability theory.

4 DIFFERENTIAL FAULT ANALYSIS OF NORX USING VARIANTS OF COUPON COLLECTOR PROBLEM

4.1 Introduction

Fault analysis has always been one of the most popular physical attacks. It is primarily attributed to the ease of mounting such attacks in practice and secondarily due to the difficulty in inhibiting them. The basic idea of fault-based attacks is to maliciously modify intermediate states of a cryptosystem, while in operation, leading to cryptanalytically exploitable side-channel information. Differential Fault Attacks (DFA) is the most well-studied type of fault attack and was first demonstrated by Biham and Shamir [119]. In general, DFA is based on inducing faults in the state of a cipher while studying the diffusion in the state and leveraging the relationship between the faulty ciphertext and its fault-free counterpart.

The initiation of CAESAR competition [22] on authenticated encryption (AE) schemes generated a lot of interest in the crypto community to analyze these ciphers which try to combine the goals of authenticity and confidentiality under a single unified primitive. Researchers also tried to look at authenticated ciphers from the fault analysis perspective. Authenticated ciphers submitted to CAESAR presented many interesting problems due to their diverse design strategies and paradigms as well as an array of useful features like Nonce-Misuse Resistance (NMR), Online Authenticated Encryption (OAE), Inverse Free, Release of Unverified Plaintexts (RUP), Par-

allelizable Encryption/Decryption and others (a good account of which is available in [147]). Eventually, it was found that some of these desirable features lead to previously nonexistent vulnerabilities with regards to fault attacks [41, 40, 42]. One of many ways that AE schemes can be classified is based on the use of *nonces* giving us two types of authenticated ciphers: one that prohibits reusing the nonce and the other that provides *some* security under nonce-reuse. In CHES 2016 [42] and later in JCEN'17 [148], Saha and Roy Chowdhury, using a demonstration on nonce-based authenticated cipher PAEQ [149], highlighted the importance of the *nonce-barrier* in the context of automatic prevention of DFA. The basic problem seems to be the fact that nonce-based schemes inhibit replaying of the algorithm which is a premise to DFA, thereby implicitly thwarting them. The authors show how parallelism in PAEQ could be exploited to mount a DFA with a single faulty ciphertext using an internal difference between parallel branches, thereby completely avoiding the nonce constraint. The attack requires two faults: one for colliding the branch inputs, while another to mount a classical DFA. The authors also generalized the attack, showcasing the threat it poses to parallelizable ciphers that employ the counter-mode.

In this chapter, we target a nonce-respecting CAESAR submission called NORX [150] which survived up to the third round. NORX constitutes a family of sponge-based authenticated encryption with associated data (AEAD) algorithms designed by Aumasson, Jovanovic and Neves. The original submission NORX v1 [151] proposes versions of NORX with 32 and 64−bit words called NORX 32 and NORX64 respectively. Subsequently, two more versions, namely, NORX-v2.0 [152] and NORX v3.0 [150], were proposed with the same word-size variants. Two additional designs with 8 and 16-bit words were proposed in [153] as lightweight variants, but were not part of the CAESAR competition. Our work is applicable to all the versions v1, v2.0, v3.0. However, we describe the analysis on NORX v3.0 only, as this is the latest revised version of CAESAR third round candidates.

Our interest in NORX stems from the fact that it has a unique *parallel* architecture based on the *MonkeyDuplex* construction [66] and supports an arbitrary parallelism degree, based on LRX (Logical-Operations Rotation XOR) [154] primitives. The way NORX instantiates parallelism deviates from the classical parallelizable ciphers as

stated in [42], where it is assumed that the branches are processed independently. However, though NORX uses a variant of the counter-mode to separate the branches, it merges all the bran-ches at the end. This is the reason why the idea of using a fault-induced internal difference between branches proposed in [42] cannot be directly extended to get an attack on NORX. Hence, NORX presents an interesting case for investigation in the light of fault-based attacks. Finally, though NORX has been eliminated from the CAESAR competition, still there are many NORX-like constructions [155, 156, 157] that continue to motivate this kind of analysis.

In this work, we additionally focus on two aspects. On the one hand, we perform extensive theoretical analysis using variants of the Coupon Collector Problem (CCP) for estimating the expected number of faults required using four different fault models for reducing the key-space to practically acceptable limits. The second aspect we address is to look at FORK-256 [158] which is a hash function having a structure loosely similar to NORX (as it also uses parallel branches to compute the compression function) in the light of the fault attack strategy developed here. We found that in some particular types of implementation strategies that might be adopted for the resourced constrained environment, FORK-256 may be vulnerable to DFA. The idea is similar to the one used for NORX and consists of colliding the parallel branches which results in the chaining-value being passed unaltered to the output. This *might* pose a threat if FORK-256 is used in the keyed mode.

4.1.1 Summary of The Chapter

The first contribution comes in the form of generating fault-based internal state collisions on NORX with a level of parallelism $p \in \{2, 4\}$, leading to an *all-zero* state at the end of message processing. Once the collision is generated in *any*[1] NORX instantiation, if the trailing data is identical then all such NORX instances will produce the same (faulty) tag. In other words, the tags would also collide and this can be interpreted as a *faulty forgery*. This mimics the replay of the NORX which makes the threat of classical DFA pertinent.

Devising the attack constitutes our second contribution. We find that if we can inject

[1]We stress that this is true for any nonce and messages with consecutive identical blocks.

Fault Model	Expected # Faults	Reduced Key-Space
Random Bit-Flip	1384	2^{32}
	1544	2^{16}
Random Byte-Flip	136	1
Random Byte with Known Fault	332	2^{32}
	372	2^{16}
Random Consecutive Bit-Flip	302	2^{32}
	340	2^{12}

Table 4.1: Attacks on NORX reported in this work

faults in the internal state of permutation in the last iteration of the round function, then based on the fault model adopted, one or multiple bits of the internal state is revealed by the `XOR` of the tags.

We perform extensive theoretical analysis using variants of the Coupon Collector Problem [137] for estimating the expected number of faults required using *four* different fault models for reducing the key-space to practically acceptable limits. The summary of these results is furnished in Table 6.1.

In the Consecutive Bit-flip Fault Model, an attacker flips l consecutive bits in a register of size n bits. This scenario can be modeled as a Consecutive Coupon Collector Problem, where instead of drawing l random coupons, l successive coupons are drawn from a set of n coupons in each random sampling. Theoretical solutions to this problem have been presented in Chapter 3, and our analysis demonstrates that these theoretical calculations closely match the simulated values.

We additionally, do a DFA vulnerability assessment for the FORK-256 hash function in the light of the fault model developed for NORX.

4.1.2 Related Works

In [154], Aumasson et al. thoroughly analyzed the differential and the rotational properties of the core permutation of NORX. They gave upper bounds on the differential probability for the reduced permutation. More precisely, they demonstrated that using a classical differential analysis with an input difference in the nonce during initialization, they could devise a single round differential characteristic with probabilities less than 2^{-60} (for NORX 32) and 2^{-53} (for NORX 64). Furthermore, they have

$$S = \left(\begin{array}{cccc} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ \hline s_{12} & s_{13} & s_{14} & s_{15} \end{array}\right) \qquad S^{init} = \left(\begin{array}{cccc} n_0 & n_1 & u_2 & u_3 \\ K^{(1)} & K^{(2)} & K^{(3)} & K^{(4)} \\ u_8 & u_9 & u_{10} & u_{11} \\ \hline u_{12} & u_{13} & u_{14} & u_{15} \end{array}\right)$$

(a) State representation (b) State initialization

Figure 4-1: Matrix form of the state

found the best characteristics for four rounds with probabilities of 2^{-584} and 2^{-836} for NORX 32 and NORX 64, respectively. In [159], Das et al. describe statistical variants of zero-sum distinguishers that allow distinguishing the full-round permutation of NORX-64 and 3.5 rounds of the permutation of NORX-32 from random permutations. These results cover more rounds compared to the first-order differential analysis provided in [154]. The used approach is similar to zero-sum distinguishers [160], but it is probabilistic rather than deterministic. Later in [161], Bagheri et al. showed a state/key recovery attack for both variants for a reduced version of NORX v2.0 where the underlying permutation applies half the rounds (2 out of 4). After that, in [162] Dwivedi et al. analyzed the state-recovery resistance of several submitted CAESAR candidates, including NORX, using a SAT solver. They have also analyzed modified versions of these algorithms, including round reduced variants. Later in [163], using non-random properties of the underlying permutation, they showed a ciphertext-only forgery with time and data complexity 2^{-66} (resp. 2^{-130}) for the variant of NORX v2.0 of 128-bit (resp. 256-bit) keys and broke the designer's claim of a 128-bit (resp. 256-bit) security. Furthermore, they showed that this forgery attack can be extended to a key-recovery attack on the full NORX v2.0 with the same time and data complexities. Also, for NORX v3.0, the resulting attack enables an adversary to generate forgeries with data complexities $2^{2w}, w = 32, 64$ for 128, 256-bit keys respectively. It is interesting to see that despite a lot of cryptanalytic results reported in the literature, there is no side-channel attack or physical attack developed on NORX. This forms one of the initial motivations of this work.

Notation	Description
$x\|\|y$	Concatenation of bitstrings x and y
$^-, \wedge, \vee, \oplus$	Bitwise negation, `AND`, `OR`, `XOR`
$x \ll n, x \ggg n$	Left-shift/Right-rotation of bitstring x by n bits
$\leftarrow$	Variable assignment
$right_r(x)$	Truncation of bitstring x to its r right-most bits
$w \in \{32, 64\}, l \in [1, 63]$	Word size, Round number
$p \in [0, 255], t(\leq 4w)$	Parallelism degree, Tag size
$S = s_0\|\|s_1\|\|\cdots\|\|s_{15}$	NORX state, where $s_i, i \in [0, 15]$ represents a word
K, N	Key ($4w$ bits), Nonce ($2w$ bits)
A, Z	Associated data with header A and trailer Z
M, C, T	Message, Ciphertext, Tag (t bits)
F, G	Round and quarter-round function of NORX core permutation
$col(S), diag(S)$	Column and diagonal steps of the round function F in NORX permutation
$ldiag(S)$	The final diagonal step ($diag(S)$ call) before computing the tag T in NORX permutation
(a, b, c, d)	$4w$-bits input to the G function with d as the capacity word
$(G1), \cdots, (G8)$	Consecutive steps to evaluate the G function in NORX
$x[i]$	i-th bit of any word x
$x_i, x_{[i,i+j-1]}$	$x[i]$, $x[i]\|\|\cdots\|\|x[i+j-1]$
$x_{i+j}, 0 \leq i, j \leq w-1$	$x_{(i+j) \mod w}$, for any word x
x^i	i-th byte of the word x
$\tilde{x}$	Faults injected on word x at the time of execution among any steps from $(G1)$ to $(G5)$ inside the G function at the $ldiag(S)$ operation
$\tilde{b}$	Faults are induced on word b at Step $(G5)$ inside the G function
$\tilde{c}$	Faults are induced on word c at Step $(G3)$ inside the G function
$T, T^{\tilde{x}}$	Fresh tag, faulty tag due to $\tilde{x}$
$\tilde{x}_i, \tilde{x}^i, \tilde{x}_{[i,i+j-1]}$	i-th bit fault, i-th byte fault, consecutive bits fault from i to $i+j-1$ of x
$T^{\tilde{x}_i}, T^{\tilde{x}^i}, T^{\tilde{x}_{[i,i+j-1]}}$	Faulty tags correspond to bit, byte, consecutive bits fault on x
$\Delta z^{\tilde{x}_i}, \Delta z^{\tilde{x}^i}, \Delta z^{\tilde{x}_{[i,i+j-1]}}$	`XOR` difference of z and $\tilde{z}$ due to $\tilde{x}_i$, $\tilde{x}^i$, $\tilde{x}_{[i,i+j-1]}$
$\Delta T^{\tilde{x}_i}, \Delta T^{\tilde{x}^i}, \Delta T^{\tilde{x}_{[i,i+j-1]}}$	`XOR` difference of T and $T^{\tilde{x}_i}$, T and $T^{\tilde{x}^i}$, T and $T^{\tilde{x}_{[i,i+j-1]}}$

Table 4.2: Notations

4.2 Specification of NORX

We start by defining the notations used throughout the chapter in Table 4.2. Next, we give a brief description of the NORX family of Authenticated Encryption with Associated Data (AEAD) algorithms, mainly the description of NORX v3.0.

A NORX instance is denoted by (w, l, p, t). Table 4.3 proposes five NORX instances for different use cases. The state S is viewed as a chain of 16 words, i.e., $S = s_0||s_1||....||s_{15}$, where $s_0, ..., s_{11}$ are called the rate words (where data blocks are injected) and $s_{12}, ..., s_{15}$ are called the capacity words (which remain untouched). The state S can be viewed as a 4×4 matrix as shown in Figure 6-2a. More information on the constants can be found in [150]. The encryption algorithm takes as inputs a key K of k-bits, a nonce N of n-bits, a plaintext M and an associated data formed by a header A and a trailer Z. The header, the plaintext and the trailer are three optional strings. The encryption algorithm computes an authentication tag T of t-bits, and a ciphertext C of the same bit-length as the plaintext M. Similarly, the decryption algorithm takes as inputs (K, N, A, C, Z, T) and returns either $\perp$ or M depending on whether the tag verification succeeds or not. Both encryption and decryption algorithms begin by an initialization phase that sets the internal state to S^{init}, consists of $4w$-bit key $K = K^{(1)}||K^{(2)}||K^{(3)}||K^{(4)}$, the $2w$-bit nonce $N = n_0||n_1$ and some initialization constants (u_i) in the internal state, is given in Figure 6-2b. The basic

Serial Number	Word Size (w)	Round Number (l)	Parallelism Degree (p)	Tag Size (t)	Key Size (k)	Nonce Size (n)
1.	64	4	1	256	256	256
2.	32	4	1	128	128	128
3.	64	6	1	256	256	256
4.	32	6	1	128	128	128
5.	64	4	4	256	256	256

Table 4.3: NORX instances [150]

building block of NORX is a permutation F, also called a round, and F^l is l consecutive applications of F. The permutation F over the state S transforms its columns with

$$G(s_0, s_4, s_8, s_{12})\ \ G(s_1, s_5, s_9, s_{13})\ \ G(s_2, s_6, s_{10}, s_{14})\ \ G(s_3, s_7, s_{11}, s_{15})$$

and then transforms its diagonals with

$$G(s_0, s_5, s_{10}, s_{15})\ G(s_1, s_6, s_{11}, s_{12})\ G(s_2, s_7, s_8, s_{13})\ G(s_3, s_4, s_9, s_{14}).$$

Those two operations are denoted by $col(S)$ and $diag(S)$ respectively. Similarly, a round function F can be viewed as $F = diag \circ col(S)$. The complete pseudo-code for the NORX core permutation F^l is given in Figure 6-3. The G function uses cyclic rotations $\ggg$ and a non-linear operation H interchangeably to update its four input words a, b, c, d. The rotation offsets r_0, r_1, r_2, and r_3 for the cyclic rotations of 32- and 64-bit NORX are specified in Table 4.4. The designers proposed certain configurations of the mode to process the payload in parallel. The parallel mode is controlled by the parameter $0 \leq p \leq 255$. For $p = 1$, the design of NORX corresponds to the sequential duplex construction and is shown in Figure 4-3. For $p > 1$, NORX achieves a parallelism of degree p. For example, the design of NORX with a parallelism degree of 2 ($p = 2$) is illustrated in Figure 4-4. The parameter combinations of the NORX variants are given in [164, Table 1].

w	r_0	r_1	r_2	r_3
32	8	11	16	31
64	8	19	40	61

Table 4.4: Rotation offsets for NORX 32 and NORX 64

4.3 Attack Scenario

In this section, we illustrate the fault injections required to first create a replay in the nonce-respecting scenario, i.e., create identical states by a counter fault to get the all-zero state after merging, and then perform DFA. Our differential fault attack on NORX is applicable for any parallelism of even degree, in particular for $p = 2$ and 4. Note that $p = 4$ is a valid instance as per NORX specification. However, we will discuss our approach with $p = 2$ for simplicity. Further, in NORX, the operations in the internal permutation are word-wise, i.e., the internal permutation of NORX works on the word-level w, where w is either 32 or 64. Here, we describe our attack for 32-bit version, but, it still works for 64-bit version also.

Algorithm 1: $col(S)$

c1. $(s_0, s_4, s_8, s_{12}) \leftarrow G(s_0, s_4, s_8, s_{12})$;
c2. $(s_1, s_5, s_9, s_{13}) \leftarrow G(s_1, s_5, s_9, s_{13})$;
c3. $(s_2, s_6, s_{10}, s_{14}) \leftarrow G(s_2, s_6, s_{10}, s_{14})$;
c4. $(s_3, s_7, s_{11}, s_{15}) \leftarrow G(s_3, s_7, s_{11}, s_{15})$;
c5. return S;

Algorithm 2: $diag(S)$

d1. $(s_0, s_5, s_{10}, s_{15}) \leftarrow G(s_0, s_5, s_{10}, s_{15})$;
d2. $(s_1, s_6, s_{11}, s_{12}) \leftarrow G(s_1, s_6, s_{11}, s_{12})$;
d3. $(s_2, s_7, s_8, s_{13}) \leftarrow G(s_2, s_7, s_8, s_{13})$;
d4. $(s_3, s_4, s_9, s_{14}) \leftarrow G(s_3, s_4, s_9, s_{14})$;
d5. return S;

Algorithm 3: $H(x, y)$

H1. return $(x \oplus y) \oplus ((x \wedge y) \ll 1)$;

Algorithm 4: $F^l(S)$

F1. **for** $i = 0$ *to* $l - 1$ **do**
  $S \leftarrow diag(col(S))$;
F2. return S;

Algorithm 5: $G(a, b, c, d)$

G1. $a \leftarrow H(a, b)$;
G2. $d \leftarrow (a \oplus d) \ggg r_0$;
G3. $c \leftarrow H(c, d)$;
G4. $b \leftarrow (b \oplus c) \ggg r_1$;
G5. $a \leftarrow H(a, b)$;
G6. $d \leftarrow (a \oplus d) \ggg r_2$;
G7. $c \leftarrow H(c, d)$;
G8. $b \leftarrow (b \oplus c) \ggg r_3$;
G9. return a, b, c, d;

Figure 4-2: The NORX permutation F^l

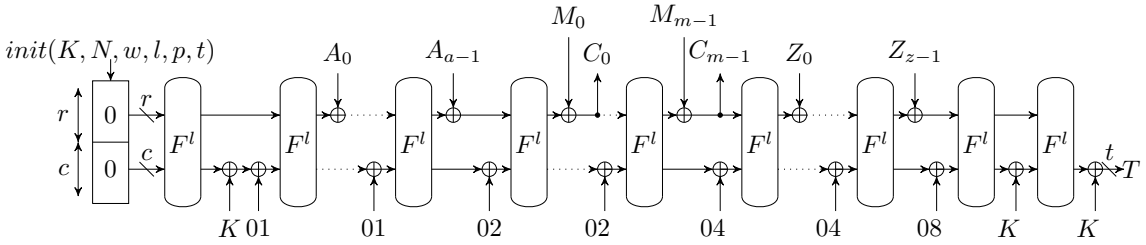


Figure 4-3: Layout of NORX with $p = 1$

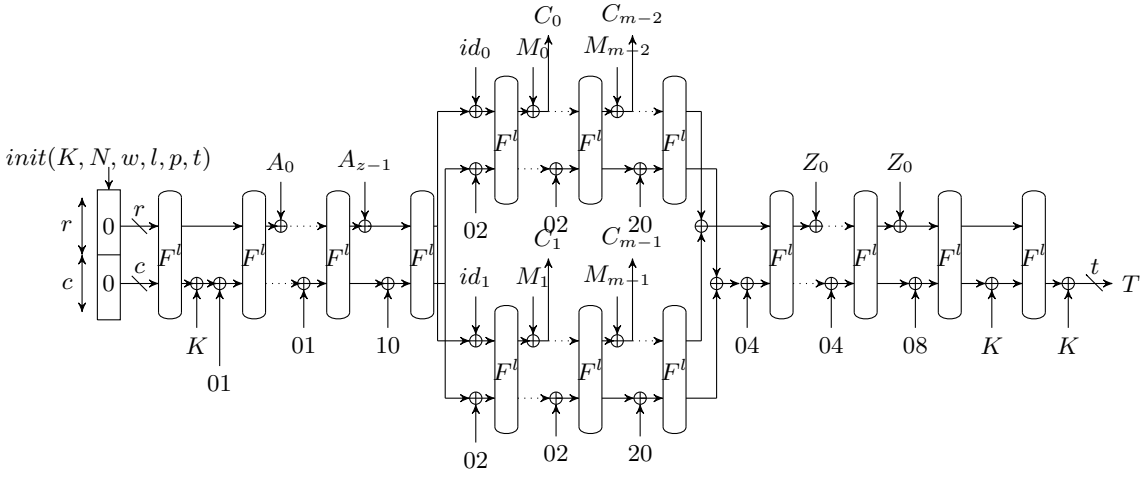


Figure 4-4: Layout of NORX with $p = 2$

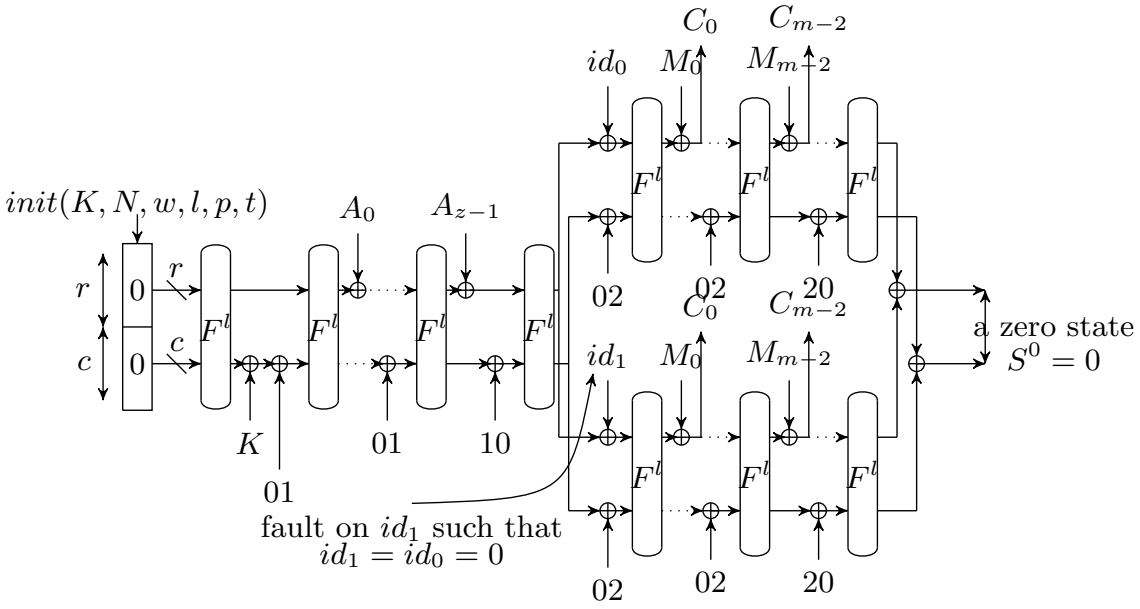
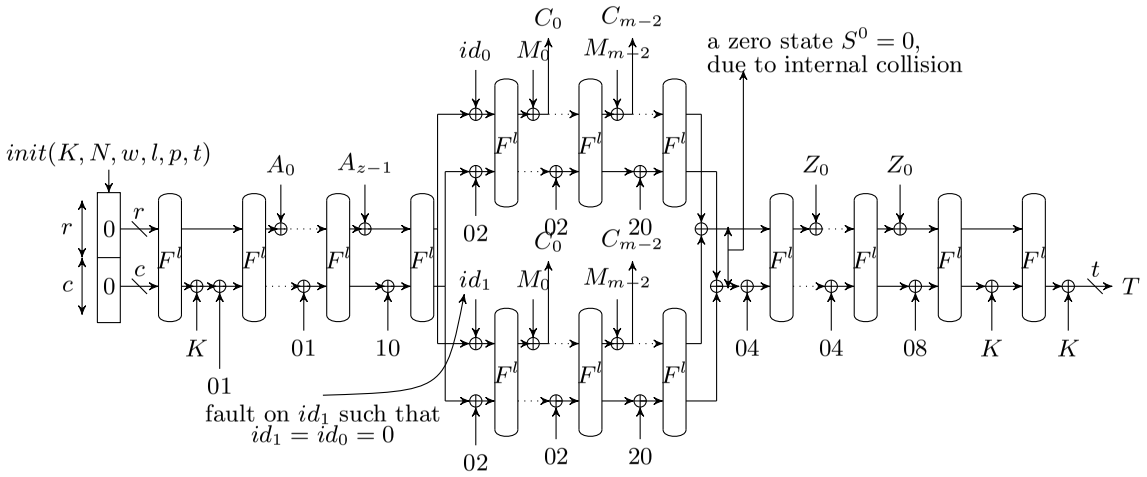


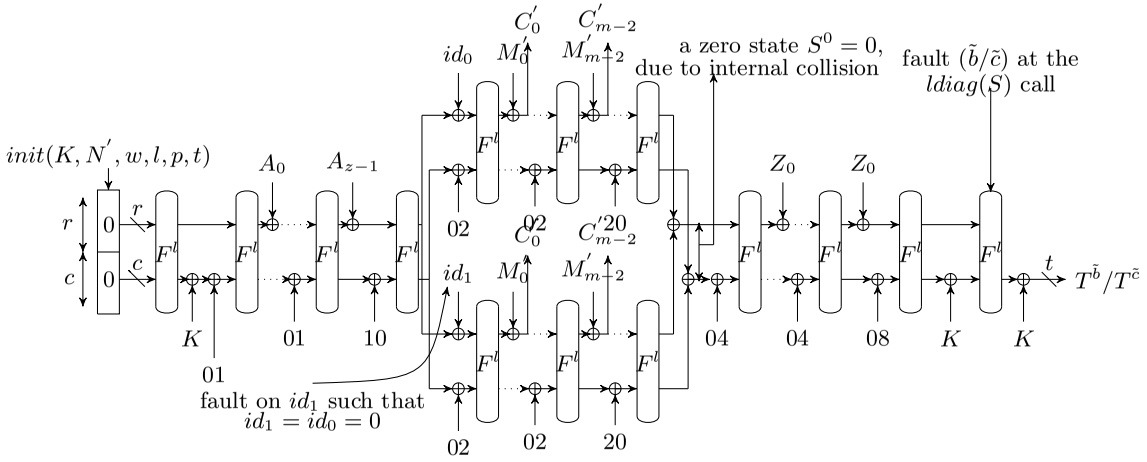
Figure 4-5: Counter fault on NORX with $p = 2$

To perform differential analysis on NORX with parallelism degree $p = 2$, for each message query, the attacker will inject two faults where the first fault is a counter fault to create a replay and the second fault is on the words $(\tilde{b}/\tilde{c})$ inside the G function at the $ldiag(S)$ call. Thus, it shows that the attacker moves from a single-fault model to a double-fault one. However, in this particular context, it is clear that performing the double faults [42, 33, 165] is not too difficult compared to the single one because of the following reasons.

1. The attacker can send a long enough message so that the second fault occurs far enough from the first one
2. Thus, we can observe the success of the first fault so that the attacker can tune the two fault settings independently.



(a) A non-faulty tag T using an internal collision



(b) A faulty tag T^{Δ} using an internal collision

Figure 4-6: Replay on NORX ($p = 2$)

4.3.1 Creating a Replay on NORX with Parallel Encryption

An important requirement of DFA is the ability to replay the execution of this cipher in order to exploit the difference between faulty and non-fault outputs. The DFA in a cryptosystem requires an attacker to be able to inject faults by replaying a previous fault-free run of the algorithm, which is known as the replaying criterion of DFA [41]. The definite structure of nonce-based encryption proposed by Rogaway [38] expects the uniqueness of the nonce in every instantiation of the cipher and the security claims rely on this premise. Thus, the use of a unique nonce contradicts the ability to replay a cipher and thereby results in automatic protection from DFA. So it is quite clear that the DFA is not applicable for the standard NORX with $p = 1$. But for NORX with parallelism degree $p = 2$, we can make an exception by inducing a counter fault and processing the messages in a specific way for each query. For $p = 2$, in each message query, we can produce a replay, i.e., create a fixed state $S^0 = \mathbf{0}$ after the merging phase (i.e., before processing the trailing data) by injecting a fault on the counter id_1 such that $id_1 = id_0 = 0$ and processing the message $M = M_0||M_1||\ldots||M_{m-1}$ such that $M_i = M_{i+1}, i = 0, 2, \cdots, m-2$. This description is outlined in Figure 4-5.

4.3.2 Feasibility of the Counter Fault

According to the branch algorithm in [152], for $p \geq 2$, the branching is done by `XOR`ing the variable i at 12 different positions (i.e., at the rate words) in the state. To produce a replay, an attacker can inject a random fault at the variable i (or may induce a bit-fault on the most significant bit (`msb`) at the variable i) so that it will skip the for-loop for branching. Moreover, using the laser bit sets technique [166], an attacker can practically induce faults on the data part of the ADD instruction to prematurely escape a for loop. As a result, it leaves all the states identical after the branching phase. Also, according to software implementation given by the NORX designers in [167], a variable i is used to `XOR` with all the rate words of the state for branching. For $p = 2$, we induce a bit fault at the `lsb` of the variable $i = 1$, at the time when i will be `XOR`ed to all rate words of the state. This equalizes the two branches of NORX in the branching phase. For $p = 4$, we need two faults to collide

all four branches to a zero state. The first one is to inject a bit flip at the `lsb` of the second branch variable $i = 1$ and the second one is to inject a bit flip at the `lsb` of the fourth branch $i = 3$. It can further be noted that if it is a parallelized hardware or software implementation with no explicit counter, then we target the particular bits in the bus, so that the inputs to each parallel branch is the same, thereby leading to the same output values to be `XOR`ed for $p = 2$ and 4 which in turn leads to an all-zero state after merging the branches.

4.3.3 Injecting Fault at the $ldiag(S)$ Call

In NORX, an authentication tag $T = F^l(F^l(S \oplus 08) \oplus K) \oplus K$ is generated by first injecting the domain separation constant `08`, then transforming the state S twice with the permutation F^l interleaved by two key additions to the capacity, and finally extracting the $t(\leq 4w)$ rightmost capacity bits from S, given in Figure 4-8. Thus, we can think of the tag T as the `XOR`ing of the capacity bits, after the $ldiag(S)$ operation with the master key K. The tag T can be viewed as $T_1||T_2||T_3||T_4$ and the key K as $K^{(1)}||K^{(2)}||K^{(3)}||K^{(4)}$ respectively, where $|K^{(i)}| = |T_i| = w, i = 1, 2, 3, 4$. The core permutation F of NORX has a natural parallelism of 4 independent G applications. The four diagonal words $(s_0, s_5, s_{10}, s_{15})$, $(s_1, s_6, s_{11}, s_{12})$, (s_2, s_7, s_8, s_{13}) and (s_3, s_4, s_9, s_{14}) are fed into the G-function independently inside the $diag(S)$ call. So after the $ldiag(S)$ operation, the tag $T = T_1||T_2||T_3||T_4$ can be viewed as independent `XOR`ing of the outputs of 4 parallel executions of the G function with the keys $K^{(1)}, K^{(2)}, K^{(3)}, K^{(4)}$ as depicted in Figure 4-7.

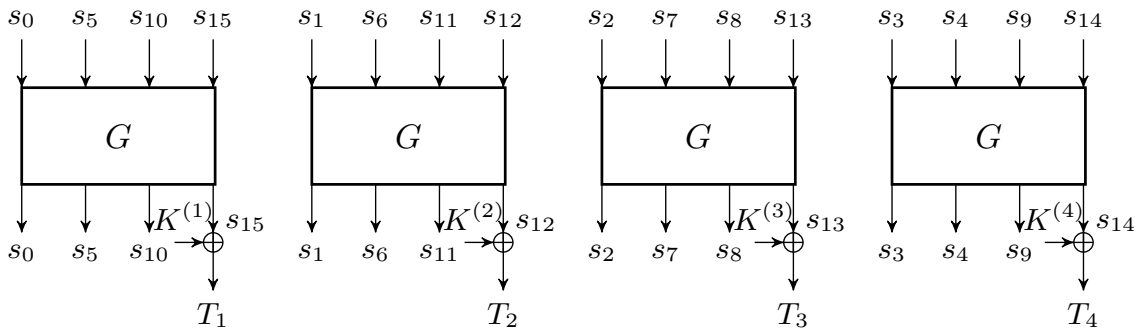


Figure 4-7: Parallel execution of G-functions at the last $diag(S)$ call to generate the tag T

If we look at the input (a, b, c, d) to the G application, it is clear that the word d is nothing but a word that belongs to the capacity part of the state. Inside one of the four independent G applications during $ldiag(S)$ operation at Step (G6) in Algorithm 5, the updated word d produces the corresponding tag word $T_i, i = 1, 2, 3, 4$ by `XOR`ing with the corresponding word of the key K. Inside the G application, H is the only non-linear operation that helps to increase the degree of a boolean function. Our aim is to recover the bits of d by inducing faults on a word $(\tilde{b}/\tilde{c})$ and observe the `XOR`ed relation of the faulty and non-faulty tags[2]. The following steps can recover the bits of the word d as also outlined in Figure 4-9.

1. Recover the corresponding bits of a by injecting a fault $\tilde{b}$ and repeat this step until we recover all the bits of a.

2. Based on the recovered bits of a, we can recover the bits of d by injecting a fault $\tilde{c}$ and repeat this step until we recover all the bits of d.

The overall fault induction process using primary (counter) faults and secondary faults is furnished in Figure 4-6.

[2]Here non-faulty tag means, the tag where we do not induce any fault at the last $diag(S)$ call, after the internal collision in the state by giving a counter fault. But physically, all tags are faulty due to the prerequisite of fault-based internal state collision.

Algorithm: $finalise(S, K)$

1. $s_{15} \leftarrow s_{15} \oplus 0x08$;
2. $S \leftarrow F^l(S)$;
3. $(s_{12}, s_{13}, s_{14}, s_{15}) \leftarrow (s_{12}, s_{13}, s_{14}, s_{15}) \oplus (K^1, K^2, K^3, ^4)$;
4. $S \leftarrow F^l(S)$;
5. $(s_{12}, s_{13}, s_{14}, s_{15}) \leftarrow (s_{12}, s_{13}, s_{14}, s_{15}) \oplus (K^1, K^2, K^3, ^4)$;
6. $T \leftarrow right_t(S)$;
7. return S, T;

Figure 4-8: Tag generation algorithm of NORX [150]

4.4 Our General Strategy to Perform DFA on NORX

Before going to discuss several fault models, we need to build the bit relations of the words a, b, c, d from Step $(G5) - (G6)$ due to faults injection in the word b at Step $(G5)$. Additionally, the bit relations of the words a, b, c, d from Step $(G3) - (G6)$ due to faults injected in the word c at Step $(G3)$ are also required.

Fault injection in b at Step $(G5)$: We assume that the words a, b, c, d are unknown at the Step (G4) inside the G function. Then the update of the words a, b, c, d from Step $(G5)$ and $(G6)$ inside the G application can be written in bit-level relations as below.

$$
\begin{aligned}
& \left(a_0^{'}, a_1^{'}, \cdots, a_{30}^{'}, a_{31}^{'}\right) \\
G5. \quad & \leftarrow \Big((a_0 \oplus b_0, a_1 \oplus b_1, \cdots, a_{31} \oplus b_{31}) \oplus ((a_0 \wedge b_0, a_1 \wedge b_1, \cdots, a_{31} \wedge b_{31}) \ll 1)\Big) \\
& = \Big(a_0 \oplus b_0 \oplus (a_1 \wedge b_1), a_1 \oplus b_1 \oplus (a_2 \wedge b_2), \cdots, a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), a_{31} \oplus b_{31}\Big).
\end{aligned}
$$

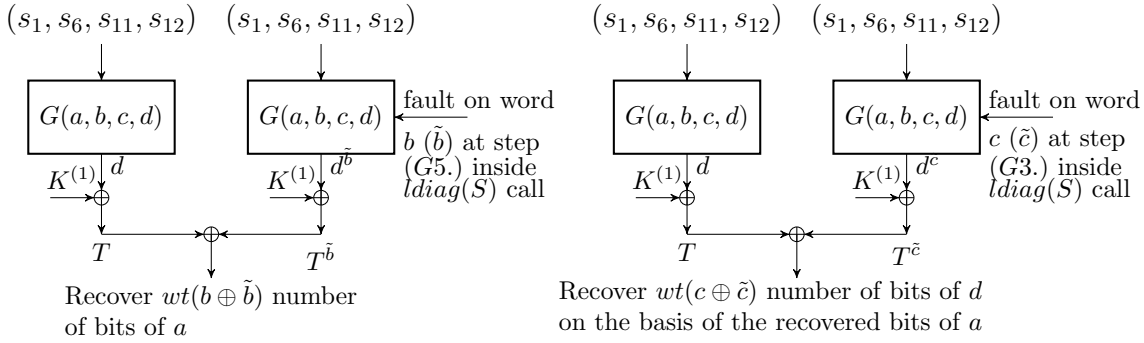


Figure 4-9: Fault inducing steps to recover the capacity d of the state S inside the G-function

$$
\begin{aligned}
&\Big(d_0, \cdots, d_{14}, d_{15}, d_{16}, \cdots, d_{30}, d_{31}\Big) \\
&\leftarrow \Big((d_0 \oplus a_0^{'}, \cdots, d_{14} \oplus a_{14}^{'}, d_{15} \oplus a_{15}^{'}, d_{16} \oplus a_{16}^{'}, \cdots, d_{30} \oplus a_{30}^{'}, d_{31} \oplus a_{31}^{'}) \ggg 16\Big) \\
&= \Big((d_0 \oplus a_0 \oplus b_0 \oplus (a_1 \wedge b_1), \cdots, d_{14} \oplus a_{14} \oplus b_{14} \oplus (a_{15} \wedge b_{15})), d_{15} \oplus a_{15} \oplus b_{15} \oplus \\
G6. \quad &(a_{16} \wedge b_{16}), d_{16} \oplus a_{16} \oplus b_{16} \oplus (a_{17} \wedge b_{17}), \cdots, d_{30} \oplus a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), d_{31} \\
&\oplus a_{31} \oplus b_{31}) \ggg 16\Big) \\
&= \Big(d_{16} \oplus a_{16} \oplus b_{16} \oplus (a_{17} \wedge b_{17}), \cdots, d_{30} \oplus a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), d_{31} \oplus a_{31} \oplus b_{31}, \\
&d_0 \oplus a_0 \oplus b_0 \oplus (a_1 \wedge b_1), \cdots, d_{15} \oplus a_{15} \oplus b_{15} \oplus (a_{16} \wedge b_{16})\Big).
\end{aligned}
$$

In short, the capacity word d at Step ($G6$) can be written in bit-level relations as below.

$$
d_i = \begin{cases} d_{i+16} \oplus a_{i+16} \oplus b_{i+16} \\ \oplus (a_{i+17} \wedge b_{i+17}), & \text{if } i \in \{0, 1, \cdots, 31\} \setminus \{15\}; \\ \\ d_{31} \oplus a_{31} \oplus b_{31}, & \text{if } i = 15. \end{cases} \tag{4.1}
$$

Fault injection in c at Step ($G3$): Similarly, we assume that the words a, b, c, d are unknown at the Step ($G2$) inside the G function. The update of the words a, b, c, d from Step ($G3$) to ($G6$) inside the G application can be written in bit-level relations as below.

$$
\begin{aligned}
&\Big(c_0^{'}, c_1^{'}, \cdots, c_{30}^{'}, c_{31}^{'}\Big) \\
&\leftarrow \Big(c_0 \oplus d_0, c_1 \oplus d_1, \cdots, c_{30} \oplus d_{30}, c_{31} \oplus d_{31}) \oplus ((c_0 \wedge d_0, c_1 \wedge d_1, \cdots, c_{30} \wedge d_{30}, \\
G3. \quad &c_{31} \wedge d_{31}) \ll 1\Big) \\
&= \Big(c_0 \oplus d_0, c_1 \oplus d_1, \cdots, c_{30} \oplus d_{30}, c_{31} \oplus d_{31}) \oplus (c_1 \wedge d_1, c_2 \wedge d_2, \cdots, c_{31} \wedge d_{31}, 0\Big) \\
&= \Big(c_0 \oplus d_0 \oplus (c_1 \wedge d_1), c_1 \oplus d_1 \oplus (c_2 \wedge d_2), \cdots, c_{30} \oplus d_{30} \oplus (c_{31} \wedge d_{31}), c_{31} \oplus d_{31}\Big).
\end{aligned}
$$

$$
\begin{aligned}
&\Big(b_0^{'}, \cdots, b_9^{'}, b_{10}^{'}, b_{11}^{'}, \cdots, b_{31}^{'}\Big) \\
&\leftarrow \Big((b_0 \oplus c_0^{'}, \cdots, b_9 \oplus c_9^{'}, b_{10} \oplus c_{10}^{'}, b_{11} \oplus c_{11}^{'}, \cdots, b_{31} \oplus c_{31}^{'}) \ggg 11\Big) \\
G4. \quad &= \Big(b_{21} \oplus c_{21} \oplus d_{21} \oplus (c_{22} \wedge d_{22}), \cdots, b_{30} \oplus c_{30} \oplus d_{30} \oplus (c_{31} \wedge d_{31}), b_{31} \oplus c_{31} \oplus d_{31}, \\
&b_0 \oplus c_0 \oplus d_0 \oplus (c_1 \wedge d_1), \cdots, b_{20} \oplus c_{20} \oplus d_{20} \oplus (c_{21} \wedge d_{21})\Big).
\end{aligned}
$$

$$\Big(a_0^{'}, \cdots, a_9^{'}, a_{10}^{'}, a_{11}^{'}, \cdots, a_{30}^{'}, a_{31}^{'}\Big)$$

$$\leftarrow \Big(a_0 \oplus b_0^{'}, \cdots, a_9 \oplus b_9^{'}, a_{10} \oplus b_{10}^{'}, a_{11} \oplus b_{11}^{'} \cdots, a_{30} \oplus b_{30}^{'}, a_{31} \oplus b_{31}^{'}) \oplus$$

$G5.$ $\qquad (a_0 \wedge b_0^{'}, \cdots, a_9 \wedge b_9^{'}, a_{10} \wedge b_{10}^{'}, a_{11} \wedge b_{11}^{'}, \cdots, a_{30} \wedge b_{30}^{'}, a_{31} \wedge b_{31}^{'}) \ll 1\Big)$

$$= \Big(a_0 \oplus b_0^{'} \oplus (a_1 \wedge b_1^{'}), \cdots, a_9 \oplus b_9^{'} \oplus (a_{10} \wedge b_{10}^{'}), a_{10} \oplus b_{10}^{'} \oplus (a_{11} \wedge b_{11}^{'}),$$

$$a_{11} \oplus b_{11}^{'} \oplus (a_{12} \wedge b_{12}^{'}), \cdots, a_{30} \oplus b_{30}^{'} \oplus (a_{31} \wedge b_{31}^{'}), a_{31} \oplus b_{31}^{'}\Big).$$

$$\Big(d_0, \cdots, d_{14}, d_{15}, d_{16}, \cdots, d_{25}, d_{26}, d_{27}, \cdots, d_{31}\Big)$$

$$\leftarrow \Big((d_0 \oplus a_0^{'}, \cdots, d_{14} \oplus a_{14}^{'}, d_{15} \oplus a_{15}^{'}, d_{16} \oplus a_{16}^{'}, \cdots, d_{25} \oplus a_{25}^{'}, d_{26} \oplus a_{26}^{'},$$

$$d_{27} \oplus a_{27}^{'}, \cdots, d_{31} \oplus a_{31}^{'}) \ggg 16\Big)$$

$$= \Big(d_{16} \oplus a_{16}^{'}, \cdots, d_{30} \oplus a_{30}^{'}, d_{31} \oplus a_{31}^{'}, d_0 \oplus a_0^{'}, \cdots, d_9 \oplus a_9^{'}, d_{10} \oplus a_{10}^{'},$$

$G6.$ $\qquad d_{11} \oplus a_{11}^{'}, \cdots, d_{15} \oplus a_{15}^{'}\Big)$

$$= \Big(d_{16} \oplus a_{16} \oplus b_{16}^{'} \oplus (a_{17} \wedge b_{17}^{'}), \cdots, d_{30} \oplus a_{30} \oplus b_{30}^{'} \oplus (a_{31} \wedge b_{31}^{'}), d_{31} \oplus a_{31}$$

$$\oplus b_{31}^{'}, d_0 \oplus a_0 \oplus b_0^{'} \oplus (a_1 \wedge b_1^{'}), \cdots, d_9 \oplus a_9 \oplus b_9^{'} \oplus (a_{10} \wedge b_{10}^{'}), d_{10} \oplus a_{10} \oplus b_{10}^{'}$$

$$\oplus (a_{11} \wedge b_{11}^{'}), d_{11} \oplus a_{11} \oplus b_{11}^{'} \oplus (a_{12} \wedge b_{12}^{'}), \cdots, d_{15} \oplus a_{15} \oplus b_{15}^{'} \oplus (a_{16} \wedge b_{16}^{'})\Big).$$

In short, the capacity word d at Step ($G6$) can be shown as follows.

$$d_i = \begin{cases} d_{i+16} \oplus a_{i+16} \oplus b_{i+16}^{'} & \\ \oplus (a_{i+17} \wedge b_{i+17}^{'}), & \text{if } i \in \{0, 1, \cdots, 31\} \setminus \{15\}; \\ & \\ d_{31} \oplus a_{31} \oplus b_{31}^{'}, & \text{if } i = 15, \end{cases} \tag{4.2}$$

where

$$b_i^{'} = \begin{cases} b_{i+21} \oplus c_{i+21} \oplus d_{i+21} & \\ \oplus (c_{i+22} \wedge d_{i+22}), & \text{if } i \in \{0, 1, \cdots, 31\} \setminus \{10\}; \\ & \\ b_{31} \oplus c_{31} \oplus d_{31}, & \text{if } i = 10. \end{cases} \tag{4.3}$$

For the DFA on NORX, two faults are injected for each encryption query. One corresponds to creating an internal state collision (replay) and another one corresponds to inducing faults at the last $ldiag(S)$ operation. So for the DFA, analyzing the relation among the faulty and the non-faulty tags by injecting faults on a word inside

$G(s_0, s_5, s_{10}, s_{15})$ will be sufficient to count the total number of faults required to recover the full key K, where $(s_0, s_5, s_{10}, s_{15})$ is the first diagonal input to the $ldiag(S)$ operation. Let n be the number of faults (except the counter faults) required, by analyzing the relation among the faulty and the non-faulty tags of $G(s_0, s_5, s_{10}, s_{15})$. Thus, the total number of faults (except the counter faults) will be $4n$. In the subsequent sections, we will discuss several fault models with their theoretical analysis. Then we will demonstrate a close comparison of the simulation result with the theoretical values.

4.4.1 Random Byte with Known Fault Model

In this fault model, a random byte fault ($\tilde{x}^i = x^i \oplus \delta, i = 0, 1, \cdots, \frac{w}{8} - 1$) is injected on a word with a known fault value, i.e., δ is known. If $T, T^{\tilde{x}^i}$ represent the fresh and the faulty tags due to a random byte with known fault δ in a word respectively, then we can easily rectify the corresponding faulty byte of the word from the relation $\Delta T^{\tilde{x}^i}$ by observing 1 on the corresponding byte position. However, if we know δ, then we can recover exactly $wt(\delta)$ bits from their `XOR`ed tag relation, where $wt(\delta)$ represents the hamming weight of δ. Another restriction is that, for the first byte-fault δ such that the 0-th bit is flipped, we can recover exactly $wt(\Delta) - 1$ number of bits from their `XOR`ed tag relation. This happens because the bit information $(x_0 \wedge y_0)$ is lost due to one left shift inside the H-function of NORX core permutation. For $w = 32$, $\Delta^{x^i}, i = 0, 1, 2, 3$ represents the corresponding byte-fault on the word x, where $x = (x_0, \cdots, x_{31})$ and $x^j = (x_{8j}, x_{8j+1}, \cdots, x_{8j+7})$, $j = 0, 1, 2, 3$.

Let $b = b^0||b^1||b^2||b^3$ be a non-faulty word. Now, for any byte-fault $\tilde{b}^i$ on the word b leading to the faulty word $\tilde{b}$, there is a one-to-one correspondence between $\Delta\tilde{b}$ and $\Delta T^{\tilde{b}}$. This means that for a byte-fault $\tilde{b}^0$ on the word b, there exists at least one bit position $i \in \{16, 17, \cdots, 23\}$ such that $\Delta T_i^{\tilde{b}} = 1$ and $\Delta T_i^{\tilde{b}} = 0$ for the remaining bit positions. Thus in this fault model we can easily detect which byte is faulted due to $\tilde{b}$ by observing $\Delta T^{\tilde{b}}$. As we know the value of $\tilde{b}^i$, we extract the bit information of a^i corresponding to the non-zero bits of $\tilde{b}^i$ from their `XOR`ed tag $\Delta T^{\tilde{b}^i}$. Similarly, we recover the word d by injecting $\tilde{c}^i$ and extract the bits information of d^i corresponding to the non-zero bits of $\tilde{c}^i$ from $\Delta T^{\tilde{c}^i}$. For better understanding, we give an example of two random byte differences with known values as $\delta_1 =$ `0x9B000000`, $\delta_2 =$ `0x005C0000`

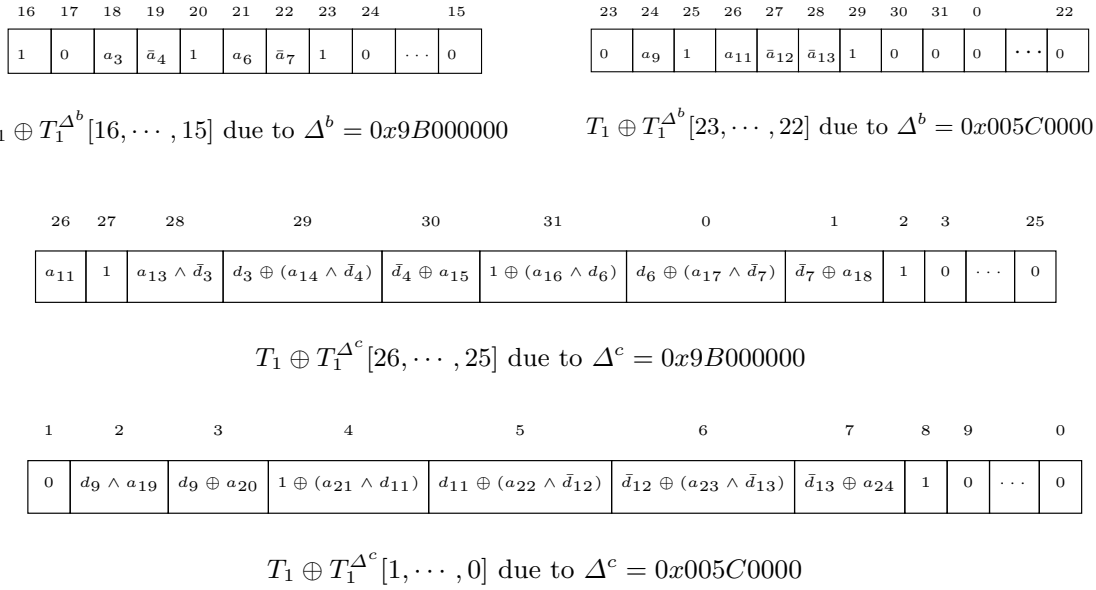


Figure 4-10: Two different cases of byte-flips on the word b as well on c

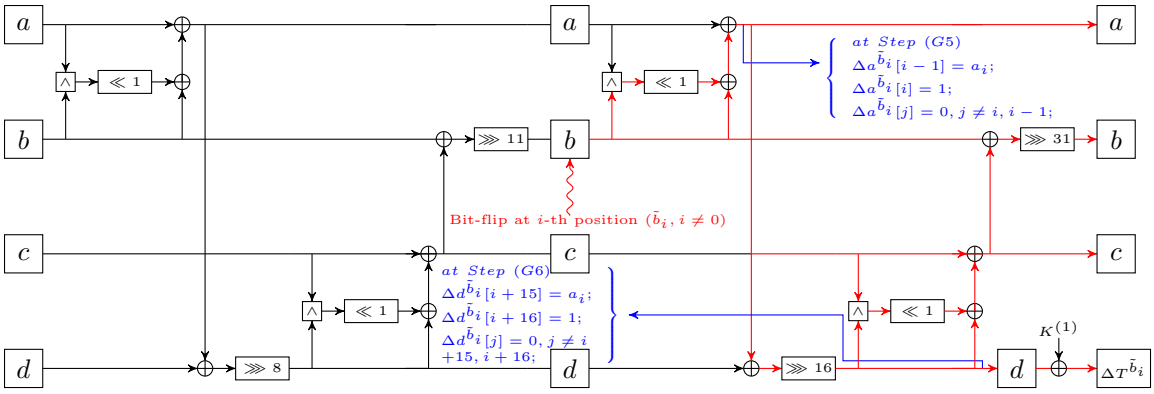
(in hexadecimal notation) on both the words b and c in Figure 4-10. So we have to estimate how many random byte-faults are required such all the bits of the word will be flipped at least once.

4.4.2 Random Bit-flip Fault Model

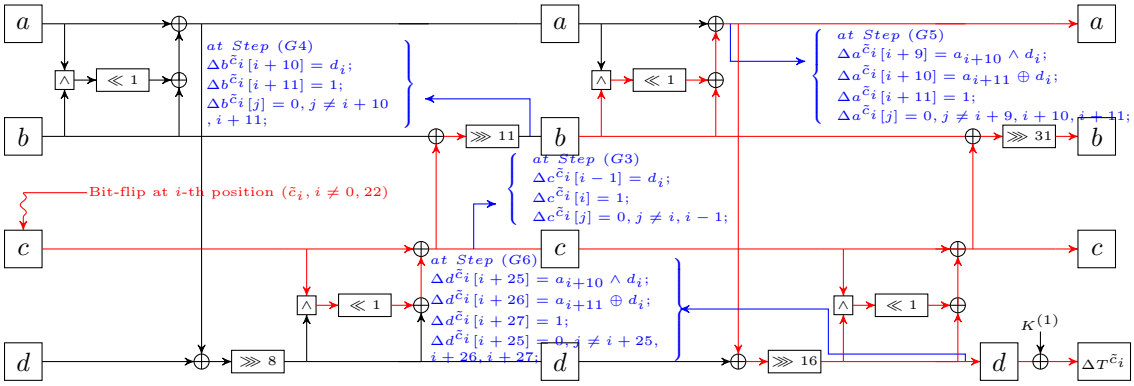
The fault model, we are considering here is that the effect of an induced fault is to flip one bit of the word in a state. In most of the devices, bit-flip faults are feasible with precise corruptability [168, 169, 170, 171]. Here we show some interesting relations among the words a, b, c, d by inducing a single bit random fault either on the word b or on c, by formalizing their respective `XOR`ed relations of the fresh tag and the faulty tag. We categorize this fault model into two cases depending on whether a fault is injected in the word b or in c.

4.4.2.1 Case 1: Faults on word b

According to the Equation (4.1), we can see that the bit $b_i, i \neq 0$ is present at the two consecutive postions in d_{i+15} and d_{i+16}. The bit b_0 is only present at d_{16}. Thus, due to $\tilde{b}_i, i \neq 0$, the XORed tag relation only has two-bit differences at the positions $i + 15$ and $i + 16$ respectively. Whereas, for $\tilde{b}_0$, the XORed tag relation only has



(a) Bit fault $\tilde{b}_i$ inside the G-function at the $ldiag(S)$ call



(b) Bit fault $\tilde{c}_i$ inside the G-function at the $ldiag(S)$ call

Figure 4-11: A bit-flip scenarios in the G circuit.

one-bit difference at the position 16. Figure 4-11a illustrates the location of the bit-fault injection to retrieve the corresponding bit of the word a. Thus, for any $\tilde{b}_i$, the generalized form of the `XOR`ed tag is as follows:

$$(\Delta T^{\tilde{b}_i})[j] = \begin{cases} \begin{cases} a_i, & \text{if } \ j = i+15 \\ 1, & \text{if } j = i+16 \\ 0, & otherwise \end{cases}, & \text{if } i \in \{0, \cdots, 31\} \setminus \{0\}; \\ \begin{cases} 1, & \text{if } j = 16 \\ 0, & otherwise \end{cases}, & \text{if } i = 0. \end{cases} \tag{4.4}$$

It is clear from Equation (4.4) that, for any $\tilde{b}_i$, we can find the corresponding faulty bit position in the word b by observing the non-zero value 1 from the relation $\Delta T^{\tilde{b}_i}$). So using this relation, we retrieve all the bit values of the word a except a_0 by injecting

random bit-faults in the word b.

4.4.2.2 Case 2: Faults on word c

According to Equation (4.2), the bit c_i is present at three consecutive positions as $i+25$, $i+26$, $i+27$ in d respectively due to $\tilde{c}_i, i \neq 0, 22$. But for $\tilde{c}_i, i = 0, 22$, the bit c_i is present at two consecutive positions as $i+26, i+27$ in d. Figure 4-11b illustrates the location of the bit-fault injection to retrieve the corresponding bit of the word d. Thus, for any $\tilde{c}_i$, the generalized form of the XORed tag is as below.

$$(\Delta T^{\tilde{c}_i})[j] = \begin{cases} \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25 \\ a_{i+11} \oplus d_i, & \text{if } j = i+26 \\ 1, & \text{if } j = i+27 \\ 0, & otherwise \end{cases}, & \text{if } i \neq 0, 21, 22; \\ \begin{cases} a_{31} \wedge d_{21}, & \text{if } j = 14 \\ d_{21}, & \text{if } j = 15 \\ 1, & \text{if } j = 16 \\ 0, & otherwise \end{cases}, & \text{if } i = 21; \\ \begin{cases} a_1 \oplus d_{22}, & \text{if } j = 16 \\ 1, & \text{if } j = 17 \\ 0, & otherwise \end{cases}, & \text{if } i = 22; \\ \begin{cases} a_{11}, & \text{if } j = 26 \\ 1, & \text{if } j = 27 \\ 0, & otherwise \end{cases}, & \text{if } i = 0. \end{cases} \tag{4.5}$$

After retrieving the whole word a, we can recover all the bits in the word d, except d_0 from Equation (4.5), by injecting random bit faults in c such that all bit positions are faulted at least once. If we can recover d (except d_0), the secret key $K^{(1)}$ (except $K_0^{(1)}$) will be recovered by XORing the respective word d and the non-faulty tag. For the unknown key bit $K_0^{(1)}$, we guess that bit and check the corresponding tag for a given query.

4.4.2.3 Theoretical Analysis of the Model

To estimate the total number of independent parallel faults, we first calculate the number α of single-bit faults on a word ($w = 32$) such that each bit of that word is flipped at least once. This problem is equivalent to the well-known Coupon Collector Problem [172]. Therefore, following Equation 3.1, we can directly estimate the required number of faults to retrieve any number of bits of a word using the following Corollary.

Corollary 1. For any word of length n, let X be a discrete random variable that represents the number of random bit-flips on the word such that l different bit positions will be flipped at least once. Then the expected number of random bit-flips required in order to hit at least l different bit positions of the word will be given by $E[X] = n \cdot (H_n - H_{n-l})$, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} harmonic number.

4.4.3 Random Byte-flip Fault Model

In this fault model, we consider that the effect of an induced fault can flip all the bits in a byte of the word in the state. Now, any word x of size $w \in \{32, 64\}$ can be viewed as sequential bytes $x^0, x^1, \cdots, x^{\frac{w}{8}-1}$ respectively, where $x^j = (x_{8j}, \cdots, x_{8j+7})$, $j = 0, 1, \cdots, \frac{w}{8} - 1$. In this model, we flip a byte randomly out of all the bytes of the word and collect the corresponding faulty tag.

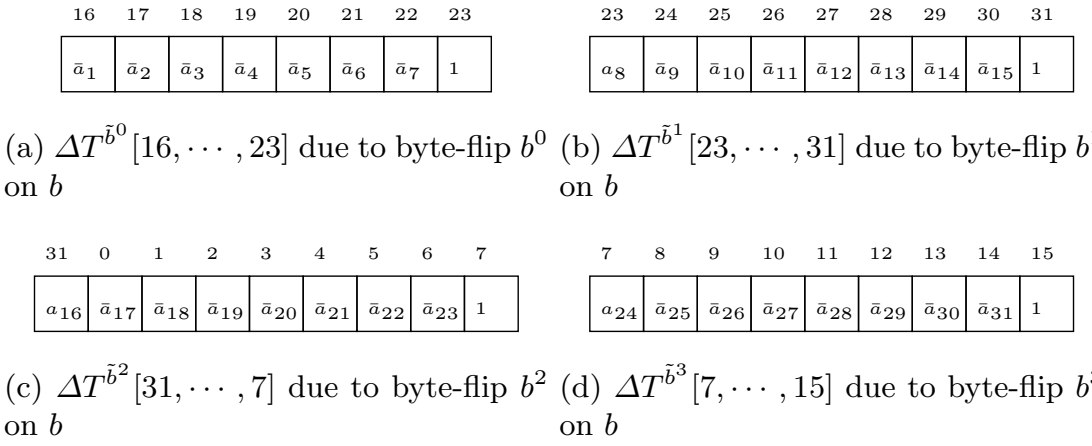
4.4.3.1 Case 1: Byte-Flip on the word b

Let us assume that, $\tilde{b}^i, i = 0, 1, \cdots, \frac{w}{8} - 1$, i.e., a byte b^i is flipped out of the word b. According to Equation (4.1), the XORed relation of the two tags $\Delta T^{\tilde{b}^i}$ is given in Figure 4-12.

4.4.3.2 Case 2: Byte-flip on the word c

We assume that, $\tilde{c}^i$, $i = 0, 1, \cdots, \frac{w}{8} - 1$, i.e., a byte c^i is flipped in the word c. According to Equation (4.2) and (4.3), the XORed relation of the two tags $\Delta T^{\tilde{c}^i}$ is given in Figure 4-13.

From Case 1, it is clear that we can recover all the bits of a except a_0, by injecting some random byte-flips $\tilde{b}^i$ in the word b such that all the four bytes of b are flipped



(a) $\Delta T^{\tilde{b}^0}[16, \cdots, 23]$ due to byte-flip b^0 on b

(b) $\Delta T^{\tilde{b}^1}[23, \cdots, 31]$ due to byte-flip b^1 on b

(c) $\Delta T^{\tilde{b}^2}[31, \cdots, 7]$ due to byte-flip b^2 on b

(d) $\Delta T^{\tilde{b}^3}[7, \cdots, 15]$ due to byte-flip b^3 on b

Figure 4-12: Four different cases of byte-flip on the word b

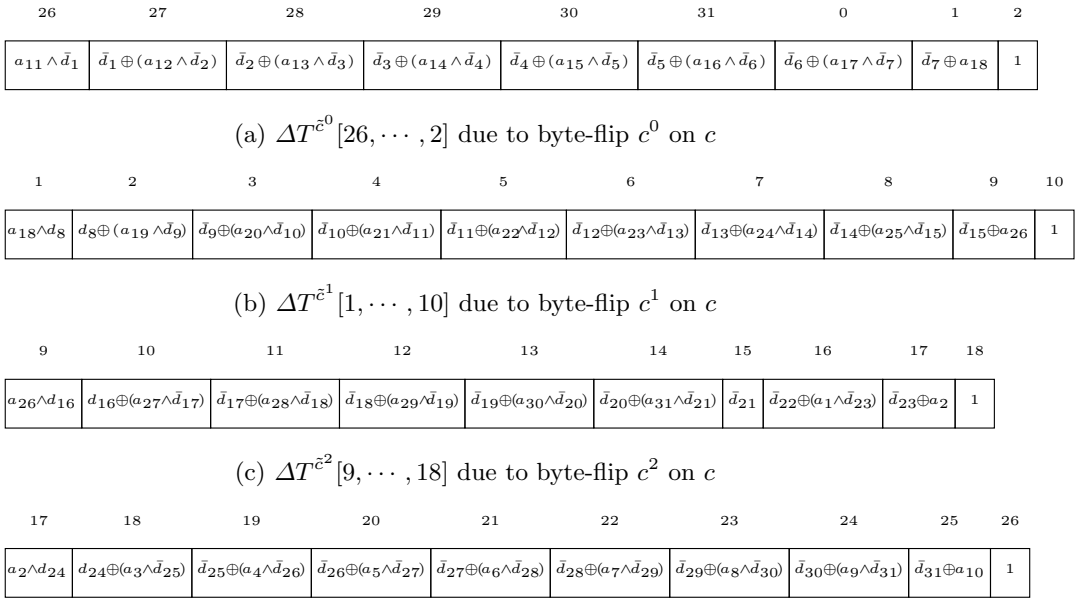
at least once. Similarly, from Case 2, we can recover all the bits of d except d_0 by injecting some random byte-flips $\tilde{b}^i$ on the word c. So we recover the key $K^{(1)}$ except $K_0^{(1)}$ due to the unrecoverable bit d_0.

4.4.3.3 Theoretical Analysis of the Model

To estimate the number of byte-flips in a word so that all the 4 bytes are flipped at least once, we will use Equation 3.1. As the word contains four bytes, the expected number of byte-flips needed to cover all the four bytes are $4 \cdot H_4 = 8.3$. Hence, for both of the cases, the expected number of byte-flips is $2 \cdot 4 \cdot H_4 \approx 17$. Since the capacity of the state has four words, so the total number of byte faults at the $ldiag(S)$ operation is $4 \cdot 17 = 68$. Therefore, the total number of faults (including counter faults) is $2 \cdot 68 = 136$.

4.4.4 Consecutive Bit-Flip Fault Model

Laser systems have proven to be highly effective at injecting faults into semiconductor devices. Indeed, laser systems combine high power with fine control over the surface and time to achieve finer disruptions. It can cause multiple bit errors as well as controlled and reproducible single bit-flip by choosing the appropriate laser parameters [171, 136, 134, 135, 173]. In this fault model, we assume that the induced fault can change either a single bit or more than one adjacent bits on a word. Now, we



(a) $\Delta T^{\tilde{c}^0}[26, \cdots, 2]$ due to byte-flip c^0 on c

(b) $\Delta T^{\tilde{c}^1}[1, \cdots, 10]$ due to byte-flip c^1 on c

(c) $\Delta T^{\tilde{c}^2}[9, \cdots, 18]$ due to byte-flip c^2 on c

(d) $\Delta T^{\tilde{c}^2}[17, \cdots, 26]$ due to byte-flip c^2 on c

Figure 4-13: Four different cases of byte-flips on the word c

discuss the generalization of the XORed relation of the non-faulty and the faulty tags, due to l ($1 \leq l \leq 32$) consecutive bit-flips in a word. For $l = 1$, we already discuss it in Section 4.4.2. In this model, there are two scenarios.

1. For each replay, the adversary can flip a random but fixed number of consecutive bits in a word, i.e., l is fixed for each replay, but the starting position of the consecutive bit-flip injected by the adversary is random.
2. Another scenario is where both l and the starting position of the consecutive bit-flip are random.

4.4.4.1 Case 1: Consecutive Bits Fault on b

Assume that $\tilde{b}_{[i,i+l-1]}$ happens, i.e., any l consecutive bits flip in the word b starting from the bit position i and $T^{\tilde{b}_{[i,i+l-1]}}$ is the corresponding faulty tag. According to Equation (4.1), the bit b_0 appears in one position in d as d_{16}, whereas the remaining b_i's, $i \neq 0$ appears at two consecutive positions in d as d_{i+15}, d_{i+16} respectively. Thus, for any l consecutive bit-flips, if 0^{th} bit of the word b is flipped, then there are several cases to their XORed tag relation, otherwise if $0 \notin \{i, \cdots, i+l-1\}$, then it is quite

easy to follow the bit relation of $\Delta T^{\tilde{b}_{[i,i+l-1]}}$. Using Equation (4.1), the XORed relation $\Delta d^{\tilde{b}_{[i,i+l-1]}}$, including all the corner cases, are given as follows.

Case 1: For any $\tilde{b}_{[i,i+l-1]}$ such that $0 \notin \{i, \cdots, i+l-1\}$, we have,

$$\Delta d^{\tilde{b}_{[i,i+l-1]}}[j] = \begin{cases} a_i, & \text{if } j = i+15; \\ \bar{a}_{i+j}, & \text{for } j \in \{i+16, \cdots, i+14+l\}; \\ 1, & \text{if } j = i+15+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Case 2: $\tilde{b}_{[i,i+l-1]}$ such that $0 \in \{i, \cdots, i+l-1\}$.

Sub-case 1: For $\tilde{b}_{[i,i+l-1]}$, i.e., for $i = 0$, we have,

$$\Delta d^{\tilde{b}_{[0,l-1]}}[j] = \begin{cases} \bar{a}_j, & \text{for } j \in \{16, \cdots, 14+l\}; \\ 1, & \text{if } j = 15+l; \\ 0, & \textit{otherwise} \end{cases}$$

Sub-case 2: For any $\tilde{b}_{[i,i+l-1]}$ such that $0 \in \{i+1, \cdots, i+l-2\}$, we have,

$$\Delta d^{\tilde{b}_{[i,i+l-1]}}[j] = \begin{cases} a_i, & \text{if } j = i+15; \\ \bar{a}_{i+12}, & \text{for } j \in \{i+16, \cdots, i+14+l\}/\{15\}; \\ 1, & \text{if } j = i+15+p = 15, 1 \leq p \leq l-1; \\ 1, & \text{if } j = i+15+l; \\ 0, & \textit{otherwise}. \end{cases}$$

From the above relations, we can retrieve all the bits of a (except a_0) by inducing random $\tilde{b}_{[i,i+l-1]}$ in the word b so that all the bits in the word b will be flipped at least once.

4.4.4.2 Case 2: Consecutive Bits Fault on c:

Similarly, assume that $\tilde{c}_{[i,i+l-1]}$ happens, i.e., any l consecutive bits flip starting from the bit position i in the word c and let $T^{\tilde{c}_{[i,i+l-1]}}$ be the corresponding faulty tag.

According to Equations (4.2) and (4.3), c_0 appears at two positions in d as d_{26}, d_{27} respectively, c_{21} appears at three consecutive positions in d as d_{14}, d_{15}, d_{16} respectively and c_{22} appears at two consecutive position in d as d_{16}, d_{17} respectively. Whereas the remaining c_i's, $i \neq 0, 21, 22$ will appear at three consecutive positions in d as $d_{i+25}, d_{i+26}, d_{i+27}$ respectively. Using Equations (4.2) and (4.3), the XORed relation of $\Delta d^{\tilde{c}_{[i,i+l-1]}}$, including all the corner cases, are given as follows.

Case 1. For any $\tilde{c}_{[i,i+l-1]}$ such that $0, 21, 22 \notin \{i, \cdots, i+l-1\}$, we have,

$$\Delta d^{\Delta^c_{[i,i+l-1]}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (\bar{d}_{i+1} \wedge a_{i+11}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus (\bar{d}_{i+j} \wedge a_{i+12}), & \text{for } j \in \{2 \cdots, l-1\}; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Case 2. For any $\tilde{c}_{[i,i+l-1]}$ such that $0 \in \{i, \cdots, i+l-1\}$ and $i = 0$, we have,

$$\Delta d^{\tilde{c}_{[0,l-1]}}[j] = \begin{cases} a_{11} \wedge \bar{d}_1, & \text{if } j = 26; \\ \bar{d}_j \oplus & \text{for } j \in \{26+1, \\ (a_{11+j} \wedge \bar{d}_{j+1}), & \cdots, 26+l-1\}; \\ \bar{d}_l \oplus a_{11+l}, & \text{if } j = 26+l; \\ 1, & \text{if } j = 27+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Case 3. $\tilde{c}_{[i,i+l-1]}$ such that $21, 22 \in \{i, \cdots, i+l-1\}$.

Sub-case 3.1. $21, 22 \in \{i, \cdots, i+l-1\}$ such that $i = 21, 22$.

If $i = 21$, then we have,

$$\Delta d^{\tilde{c}_{[21,20+l]}}[j] = \begin{cases} a_{31} \wedge d_{21}, & \text{if } j = 14; \\ d_{21}, & \text{if } j = 15; \\ \bar{d}_{21+j-1} \oplus & \text{for } j \in \{16, \\ (a_{31+j} \wedge d_{21+j}), & \cdots, 13 + l\}; \\ \bar{d}_{21+l-1} \oplus a_{31+l}, & \text{if } j = 14 + l; \\ 1, & \text{if } j = 15 + l; \\ 0, & \textit{otherwise.} \end{cases}$$

If $i = 22$, then we have,

$$\Delta d^{\tilde{c}_{[22,21+l]}}[j] = \begin{cases} d_{22} \oplus (a_1 \wedge d_{23}), & \text{if } j = 16; \\ \bar{d}_{22+j} \oplus & \text{for } j \in \{17, \\ (a_{j+1} \wedge d_{23+j}), & \cdots, 15 + l\}; \\ \bar{d}_{22+l} \oplus a_{l+1}, & \text{if } j = 16 + l; \\ 1, & \text{if } j = 17 + l; \\ 0, & \textit{otherwise.} \end{cases}$$

Sub-case 3.2. For $21, 22 \in \{i + 1, \cdots, i + l - 2\}$, we have,

$$\Delta d^{\tilde{c}_{[i,i+l-1]}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i + 25; \\ d_i \oplus (a_{i+11} \wedge \bar{d}_{i+1}), & \text{if } j = i + 26; \\ \bar{d}_{i+j-1} \oplus & \text{for } j \in \{i + 27, \\ (a_{i+10+j} \wedge \bar{d}_{i+j}), & \cdots, i + 24 + l\}; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i + 25 + l; \\ & \text{if } \exists s \in \{1, \cdots, l\} \\ \bar{d}_{21}, & \ni j = i + 25 + s = 15; \\ 1, & \text{if } j = i + 26 + l; \\ 0, & \textit{otherwise.} \end{cases}$$

Sub-case 3.3. $21, 22 \in \{i, \cdots, i + l - 1\}$ such that $i + l - 1 = 21, 22$.

If $i+l-1=21$, then we have,

$$\Delta d^{\tilde{c}_{[i,i+l-1]}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (a_{i+11} \wedge \bar{d}_{i+1}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus & \text{for } j \in \{i+27, \\ (a_{i+10+j} \wedge \bar{d}_{i+j}), & \cdots, i+24+l\}; \\ d_{i+l-1}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise}. \end{cases}$$

If $i+l-1=22$, then we have,

$$\Delta d^{\tilde{c}_{[i,i+l-1]}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (a_{i+11} \wedge \bar{d}_{i+1}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus & \text{for } j \in \{i+27, \\ (a_{i+10+j} \wedge \bar{d}_{i+j}), & \cdots, i+23+l\}; \\ \bar{d}_{i+l-2}, & \text{if } j = i+24+l = 15; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Based on the retrieved bits of a, for any $\tilde{c}_{[i,i+l-1]}$, we can recover all the bits of d except d_0 by following one of the above tag relations. Hence, we can recover $K^{(1)}$ (except $K_0^{(1)}$) by inducing $\tilde{c}_{[i,i+l-1]}$ in the word c such that all the bits of c flips at least once. Instead of recovering all the bits of $K^{(1)}$, we can recover partial bits of $K^{(1)}$ and use the brute-force approach for the remaining key bits.

4.4.4.3 Theoretical Analysis of the Model 1

In this model, we have two cases. The first one is where an attacker randomly flips the l number of consecutive bits to the register, i.e., an attacker chooses a random bit position i and then flips bits from i to $i+l-1(\leq n)$. In Chapter 3, we have provided an exact theoretical solution to this problem.

4.4.5 Experimental Results

In this section we present a set of simulations of our fault models and compare the expected number of required faults with the corresponding theoretical estimates. The simulations were performed on a Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz computer.

4.4.5.1 Random Byte with Known Fault Model

By simulating the above problem for 10^5 times, the expected number of byte-faults to cover the whole bit positions of the word is 29. So to recover the word a and then d, the total number byte faults required will be approximately $2 \cdot 29 = 58$. But instead of recovering all the bits of the word d, we can recover r (≥ 16) number of bits of the secret key $K^{(1)}$ and the remaining bits are recovered by guessing them. Our simulation result is given in Table 4.5.

No. of Recovered Bits (r)	Expected No. of $\tilde{b}^i$ ($= Z_1$)	Expected No. of $\tilde{c}^i$ ($= Z_2$)	Total No. of faults ($= Z_1 + Z_2$)	Brute-force complexity
16	126	20	292	2^{64}
20	126	32	316	2^{48}
24	126	40	332	2^{32}
28	126	60	372	2^{16}
30	126	76	404	2^{8}
31	126	92	436	2^{4}

Table 4.5: Expected number of random byte with known faults to recover the key K

4.4.5.2 Random Bit-flip Fault Model

As there are four independent diagonal inputs (a, b, c, d) to the G function, the total number of faults for each of $\tilde{b}_i$ or $\tilde{c}_i$ is $4 \times \alpha$. The estimation of the total number of faults (including counter faults) corresponding to both the theory and simulation is furnished in Table 4.6.

4.4.5.3 Consecutive Bit-Flip Fault Model

The total number of faults required to recover some partial bits (r) of the key for different l is given in Table 4.7 according to our simulation and the theoretical estimations. Also, we have done this simulation by running the procedure 1,00,000 times

Recovered Bits($4 \times t$)	Simulation			Theoretical			Brute-force complexity
	Expected No. of $\tilde{b}_{[i,i+l-1]}$ (z_1)	Expected No. of $\tilde{c}_{[i,i+l-1]}$ (z_2)	Total faults $2 \cdot (z_1 + z_2)$	Expected No. of $\tilde{b}_{[i,i+l-1]}$ (z_1)	Expected No. of $\tilde{c}_{[i,i+l-1]}$ (z_2)	Total faults $2 \cdot (z_1 + z_2)$	
64 (4×16)	519.5 (4×129.8)	87.2 (4×21.8)	1213.4	520 (4×67.8)	88 (4×22)	1216	2^{64}
96 (4×24)	519.5 (4×129.8)	171.2 (4×42.8)	1381.4	520 (4×67.8)	172 (4×43)	1384	2^{32}
116 (4×29)	519.5 (4×129.8)	284.4 (4×71.1)	1607.8	520 (4×67.8)	285 (4×71.3)	1609	2^{12}
124 (4×31)	519.5 (4×129.8)	391.3 (4×97.8)	1821.5	520 (4×67.8)	392 (4×98)	1824	2^{4}

[a] The feature in the parenthesis indicates the number of registers × number of faults on a register

Table 4.6: Expected number of single-bit random faults to recover the key K

for some values of n, k, and t and taken averages over those values. For the second case, our simulated values are given in Table 4.8.

Number of consecutive bits(l)	Recovered Bits($4 \times t$)	Simulation			Theoretical			Brute-force complexity
		Expected No. of $\tilde{b}_{[i,i+l-1]}$ (z_1)	Expected No. of $\tilde{c}_{[i,i+l-1]}$ (z_2)	Total faults $2 \cdot (z_1 + z_2)$	Expected No. of $\tilde{b}_{[i,i+l-1]}$ (z_1)	Expected No. of $\tilde{c}_{[i,i+l-1]}$ (z_2)	Total faults $2 \cdot (z_1 + z_2)$	
2	64 (4×16)	270 (4×67.5)	44.8 (4×11.2)	629.6	271.2 (4×67.8)	46.4 (4×11.6)	635.2	2^{64}
	96 (4×24)	270 (4×67.7)	88 (4×22)	716	271.2 (4×67.8)	89.2 (4×22.3)	720.8	2^{32}
	116 (4×29)	270 (4×67.5)	145.2 (4×36.3)	830.4	271.2 (4×67.8)	143.2 (4×35.8)	828.8	2^{12}
	124 (4×31)	270 (4×67.5)	201.2 (4×50.3)	942.4	271.2 (4×67.8)	202 (4×50.5)	946.4	2^{4}
3	64 (4×16)	197.6 (4×49.4)	30.8 (4×7.7)	456.8	196.8 (4×49.2)	32 (4×8)	457.6	2^{64}
	96 (4×24)	197.6 (4×49.4)	60 (4×15)	515.2	196.8 (4×49.2)	61.2 (4×15.3)	516	2^{32}
	116 (4×29)	197.6 (4×49.4)	100 (4×25)	595.2	196.8 (4×49.2)	99.6 (4×24.9)	592.8	2^{12}
	124 (4×31)	197.6 (4×49.4)	139.6 (4×34.9)	674.4	196.8 (4×49.2)	140.8 (4×35.2)	675.2	2^{4}
7	64 (4×16)	139.2 (4×34.8)	15.6 (4×3.9)	309.6	140.4 (4×35.1)	14.8 (4×3.7)	310.4	2^{64}
	96 (4×24)	139.2 (4×34.8)	28.8 (4×7.2)	336	140.4 (4×35.1)	29.2 (4×7.3)	339.2	2^{32}
	116 (4×29)	139.2 (4×34.8)	50.4 (4×12.6)	379.2	140.4 (4×35.1)	50 (4×12.5)	380.8	2^{12}
	124 (4×31)	139.2 (4×34.8)	80.8 (4×20.2)	440	140.4 (4×35.1)	81.2 (4×20.3)	443.2	2^{4}
19	64 (4×16)	128.8 (4×32.2)	7.6 (4×1.9)	272.8	128 (4×32)	8 (4×2)	272	2^{64}
	96 (4×24)	128.8 (4×32.2)	16 (4×4)	289.6	128 (4×32)	15.6 (4×3.9)	287.2	2^{32}
	116 (4×29)	128.8 (4×32.2)	33.6 (4×8.4)	324.8	128 (4×32)	32.8 (4×8.2)	321.6	2^{12}
	124 (4×31)	128.8 (4×32.2)	65.2 (4×16.3)	388	128 (4×32)	64.4 (4×16.1)	384.8	2^{4}

[a] The feature in the parenthesis indicates the number of registers × number of faults on a register

Table 4.7: Expected number of consecutive faults for different l, t, and $n = 32$

4.5 Discussion

In the literature on physical attacks on cryptographic protocols, the most powerful and effective fault analyses are DFA and Statistical Fault Analysis (SFA) [31]. In DFA, the input is encrypted twice with a fault being induced in the last rounds of the second run, and then, the difference between the correct and the faulty ciphertexts is used to retrieve the master key. On the other hand, SFA requires only a collection of

No. of consecutive bits (l)	Recovered Bits ($4 \times t$)	Expected No. of $\Delta^{b_{[i,i+l-1]}}$ (z_1)	Expected No. of $\Delta^{c_{[i,i+l-1]}}$ (z_2)	Total No. of faults $2 \cdot (z_1 + z_2)$	Brute-force complexity
random	64 (4×16)	132	10.24	284.48	2^{64}
	96 (4×24)	132	19	302	2^{32}
	116 (4×29)	132	38	340	2^{12}
	124 (4×31)	132	69.4	402.8	2^{4}

Table 4.8: Expected number of consecutive faults for random l

faulty ciphertexts to recover the correct key but does not require correct and faulty ciphertext pairs. However, two conditions must be satisfied for SFA: the inputs to the block cipher are different from each other, and the faulty ciphertexts are the direct outputs of the block cipher [174, 175]. More precisely, the attacker collects biased faulty ciphertexts (distributed non-uniformly), computes backward to the target byte corresponding to different key guesses, and tries to discard the wrong key guesses that would lead to approximately a uniform distribution of the biased target byte.

So, this type of attack is very useful where a key-whitening mechanism is used in any kind of iterated cipher or in any kind of mode/AEAD scheme where the underlying block cipher/permutation uses a key-whitening mechanism. But in NORX design, the master key is used only in the initial phase and at the end of the tag generation phase. Also, the message is XORed with the rate part of the state to output the ciphertext and then the permutation is applied to the state to process another message. So if we generate biased fault on a specific word/byte of a state just before the message processing at the last round of the permutation F and collect the faulty ciphertexts by XORing the message with the rate part of the state, then there will be no way to guess the key such that after one round inverse we can check whether that specific word/byte values behave non-uniformly or not. Similarly, at the last diagonal round of the tag generation phase, collecting faulty tags by injecting biased faults on a specific byte/word inside the state does not make it vulnerable to SFA. Since here we only know the capacity of the state (tag) and the whole rate part is unknown to us. So, by guessing the key byte/word corresponding to the faulty tag, we can not lead to the target byte to check its non-uniform behavior by inverting one diagonal round.

In our attack, we make use of Differential Fault Attacks (DFA) on NORX, by creating a replay in the nonce respecting scenario. Also, we discuss several random fault models

and recover the secret key bits by building a mathematical relation between the faulty and the non-faulty tags. Moreover, in the precise control fault model, it is quite easy to recover the secret key bits with very few faults based on the mathematical relation between the faulty and the non-faulty tags. Another important point in our attack is that instead of counter fault for branching to get a state collision in the encryption query, if the fault modifies the round iteration counter or some other variable, then we can easily detect this situation at the faulty tag output after injecting the second fault. For example, in the bit-fault model, if the XOR of the fresh tag and the faulty tag have a pattern different from what is expected as per Equation (4.4) or (4.5), then we remove that faulty encryption query from our analysis.

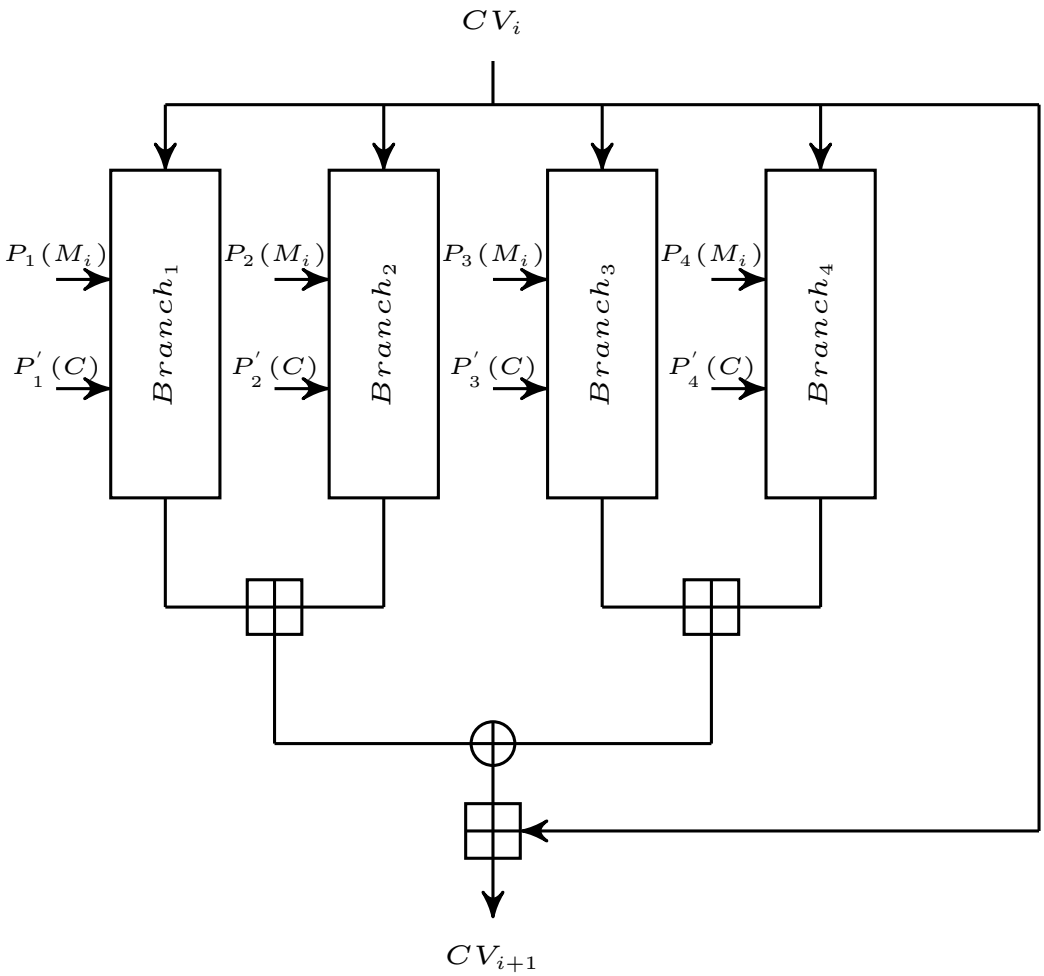


Figure 4-14: FORK-256 Compression function

s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(s)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_2(s)$	14	15	11	9	8	10	3	4	2	13	0	5	6	7	12	1
$\sigma_3(s)$	7	6	10	14	13	2	9	12	11	4	15	8	5	0	1	3
$\sigma_4(s)$	5	12	1	8	15	0	13	11	3	10	9	2	7	14	4	6

Table 4.9: Message word ordering in FORK-256

4.5.1 Fault Attack on FORK-256

FORK-256 is a 256-bit hash function that consists of four parallel branches used inside the compression function. These four branches take the previous chaining value CV_i as an input. The output of these branches is different due to the message word ordering and the constant ordering. These are given in Table 4.9, and Table 4.10 respectively. Figure 4-14 captures the compression function of FORK-256. The iterative process of FORK-256 uses the Merkle-Damgård construction [176, 177] to hash inputs of arbitrary length. More details about FORK-256 are given in [158].

Our observation is that FORK-256, when implemented in the serial mode for resource-constrained devices, can have its compression function behave like an identity function. This is under the assumption that the message word and constant orderings are implemented using look-up tables. This is done by inducing two faults separately on the row indexes of the look-up tables so that the message word ordering and constant ordering remain the same for all four branches. As an effect of faults on the look-up tables, we will always have $CV_{i+1} = CV_i$. It looks similar to the one used for NORX where all the branches collide to a zero state. For FORK-256, it results in the chaining value being passed unaltered to the final hash output. It might pose a threat if FORK-256 is used in the keyed mode. For example, if someone constructs an NMAC using the FORK-256 hash function, we can recover the secret key by injecting faults on the look-up tables of message word ordering and constant ordering. For NMAC, the secret key is used as an IV for the compression function. Since the chaining value always remains unaltered for FORK-256 by keeping the message and constant ordering identical to all the branches, we will get the secret key in the tag output.

s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_1(s)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_2(s)$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$\rho_3(s)$	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
$\rho_4(s)$	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1

Table 4.10: Constant ordering in FORK-256

4.5.2 Countermeasures

The attacks demonstrated in this work can be thwarted or made difficult to mount using two approaches. The first is related to the implementation of the counter in software or hardware. In software, the counter may be implemented in Single Instruction Multiple Data (SIMD) to make the attack harder, as one has to induce separate faults for each branch to create the collision. The same is true when the counter values are hardwired separately for each branch in case of a hardware implementation. A more effective strategy is to implement a random number generator or to use pre-generated random numbers to separate the branches, instead of using a counter. This would make the branch collision highly improbable thereby violating the initial premise of the attack. Apart from these, other standard mitigation techniques like duplicating parts of the cipher and using redundancy can also be used, though the overhead of implementing them would be, as already known, quite high.

4.6 Conclusion

We show the first fault attack that uses both internal and classical differentials to mount a differential fault analysis on the nonce-based authenticated cipher NORX. Our fault analysis is applicable to all three versions of NORX proposed in the CAESAR competition for any degree of parallelism $p > 1$. We show that the faults introduced in NORX in parallel mode can be used to collide the internal state to produce an all-zero state and this can be used to replay NORX despite being instantiated by different nonces and messages. Once replayed, we show how the internal state of NORX can be recovered using secondary faults. We use four different fault models. Under the random bit-flip model, around 1384 faults are to be induced to reduce the key-space from 2^{128} to 2^{32}. Whereas for the reduced key space of size 32, under random byte

with known fault and random consecutive bit-flip models, approximately 332 and 302 faults are to be induced respectively. For the random byte-flip model, our analysis shows that, by injecting approximately 136 faults, we can recover the full key.

It is interesting to note that our attack might apply to other ciphers which are similar in structure to NORX like FORK-256 [156], RIPEMD-160 [155] Keyak [157] etc. We address a similar kind of fault attack on a hash function FORK-256 that forces the compression function to act like an identical function. Further, it leads to full key recovery if FORK-256 is used in the keyed mode as NMAC.

5

DIFFERENTIAL FAULT ATTACK ON FEISTEL-BASED SPONGE AE SCHEMES

5.1 Introduction

Block ciphers can be primarily classified into one of two constructions: Substitution-Permutation Networks (SPN) and Generalized Feistel Networks (GFN). Furthermore, the Add-Rotate-XOR (ARX) structure represents another class of symmetric-key algorithms, designed using only the following simple operations: modular addition, bitwise rotation, and exclusive-OR. Traditionally, a classical Feistel Network (FN) has been used to design a block cipher, typically consisting of only two branches, making it the most popular variant. Further research has generalized FN to Generalized Feistel Network (GFN), which can be considered a special case of GFN. Some well-known examples of block ciphers are AES [62], PRESENT [65] (SPN), DES [52] (FN) and SPECK [178] (a variant of ARX).

Feistel is one of the popular structures among the designers as it can be implemented with a primitive of smaller state size. Moreover, in the decryption module, there is no need to invert the Feistel round function for the implementation circuit.

Feistel was first introduced with the design Lucifer [179], designed by Feistel et al.. It has been adopted in several block ciphers (with some variations) such as DES [52], FEAL [53], GOST [54], Khufu & Khafre [55], LOKI [56], CAST [57], Blowfish [58], and RC5 [59], DES being the most popular among them.

Later, Zheng et al. in [60], first described the general transformations of Feistel

Networks as type-1, type-2, type-3, which are provably secure as well as can be easily implemented with current technology. They also proved that generalized type-2 transformation is an excellent building block for cryptosystems. Context wise, we use the notation $\mathsf{GFN}^{\mathbf{f}}$, to denote an internal function f is used inside the GFN. The designs are categorized as (all of these Feistel variants are described in Section 2.1)

- Unbalanced Feistel Networks [180] consists of either expanding or contracting round functions,
- Alternating Feistel Networks [61, 181] with the rounds alternate between contracting and expanding steps and
- Generalized Feistel Networks of type-1, -2, -3 [60] using an n-bit to n-bit round function to create a kn-bit block cipher for $k \geq 2$

Some well-known GFN are Skipjack [182] (unbalanced Feistel), BEAR/LION [61] (the only alternating Feistel), CAST-256 [57] (type-1), RC6 [183] (type-2), and MARS [184] (type-3). Several generalizations of Feistel networks and their analyses [185, 180, 186, 187, 188] have been used in the new block cipher, MAC, and AE proposals.

Authenticated encryption, also known as AE, is a type of encryption that provides both encryption and authentication of data in a single process. Its significance can be observed in numerous applications, including SSL/TLS, IPSEC, SSH, and hard-disk encryption. Various modes are used to design AE schemes such as BC-based feedback modes, OCB-based modes, SIV-based modes, and Permutation-based Sponge modes. Sponge modes and their variants are the popular ones among the designers due to their low state size. It consists of a sequential application of a permutation F on a state of b bits. This state is partitioned into an r-bit rate (or outer part) and a c-bit capacity (or inner part), where $b = r + c$. It can also be visualized as stream cipher modes with the block of b-bits injected to the state and the key is only used to initialize the state. Some Sponge designs including one of its variant Beetle [189] only use the key to initialize the state. Thus, the implementation of such designs does not need any storage for the key. However, Sponge based AE does not provide security bound more than $\frac{c}{2}$ bits in the state size, and later several variants like ASCON [190]

(with Key XOR at the end), Beetle (with combined feedback) have been proposed to increase the security bound.

Sponge modes are based on public permutation, and $\mathsf{GFN}^{\mathbf{f}}$ have been used as the internal permutation in the sponge AE construction. There are several choices to design $\mathbf{f}$ function, where SPN or ARX is a popular choice among them. If we increase the number of branches, the length of each branch will become smaller. Thus, we can use the round function $\mathbf{f}$ with a smaller state size. This is advantageous for hardware-constraint lightweight applications.

In our contemporary world, Lightweight designs are necessary for the resource-constraint devices in the research community due to the growing popularity of IoT (Internet of things) applications. Recently, NIST has invited researchers to submit their designs that must be suitable for use in resource-constrained environments. A significant number of the NIST LWC submissions are based on Sponge (including GFN based permutation). Along with the must-have properties, the submissions have also been encouraged to be side-channel resistant. Most of the sponge-based designs have been analyzed under the black-box model. However, they have not been analyzed significantly in the grey-box model. Observing the growing popularity of Sponge based designs, we attempt to provide grey box fault attacks on generic sponge modes with GFN based permutations. We follow the reasonable approach of using a GFN with SPN-like structure. A few examples of fault attacks on AE schemes are available in [191, 192].

To perform DFA on sponge AE scheme, the attacker needs to create a replay in each encryption query. But due to the use of a unique nonce in each encryption, it may become a difficult task to create a faulty state collision to satisfy the replaying criterion. Whereas, the effects of this unique nonce are no longer valid in the decryption query. Further, to create a replay of sponge AE in the decryption query, we might need to create faulty forgery. Here, faulty forgery means *an attacker repeatedly makes queries by choosing a new tag $T^{'}$, and then injects faults in the intermediate rounds of the state at the time of the last permutation call until he successfully gets at least one tag forgery $T^{'}$*. Now, the immediate questions are

1. First question: what will be the probability of getting a faulty forgery?

2. Second question: can we efficiently recover the full state by performing faulty forgery?

Our research reveals that if the internal permutation of the sponge construction is SPN-based GFN, then faulty forgery can be achieved with less number of queries compared to other GFN-based structures. Then, to recover the state, we do the offline computations based on the collected forgeries. Note that, in the case of a sponge, the key is used to initialize the state. So, the state recovery is sufficient to recover the secret key if there are no extra key injections in the sponge. Though it seems to be a special case, there exist several designs that follow this case [193, 194, 195]. Our attack can work on several other examples. We have also taken the use case of CiliPadi [196] AE from NIST LWC to verify our generic proof.

5.1.1 Summary of The Chapter

This chapter makes three significant contributions to the field of cryptanalysis. Firstly, we investigate the applicability of the differential fault attack (DFA) on sponge-based authentication schemes that utilize generalized Feistel networks (GFNs) as their permutation function. Specifically, we focus on the scenario where the Generalized Feistel Network (GFN^f) internally uses a substitution-permutation network (SPN). In other words, the random function/permutation f inside the GFN follows an SPN structure. By analyzing this setup, we demonstrate the potential of DFA in compromising the security of such schemes. Additionally, we provide a comprehensive estimation of the required number of faults, as well as the associated time and memory complexities, to successfully recover the full state. Furthermore, we explore the feasibility of extending state recovery to achieve full key recovery, opening up new possibilities for exploiting vulnerabilities.

Secondly, we apply the DFA technique to recover the internal state and subsequently extract the secret key of a specific cipher known as CiliPadi authenticated encryption. Through this practical application, we not only demonstrate the effectiveness of the attack, but also provide detailed insights into the number of faults, time, and space complexities involved in the recovery process. Lastly, we contribute to the field of cryptographic security by proposing general countermeasures to mitigate the risks

associated with fault attacks. These countermeasures are designed to enhance the overall resilience of cryptographic systems against DFA.

5.2 Preliminaries

In this chapter, we will introduce a set of notations to facilitate the understanding and clarity of the concepts discussed. These notations will be used consistently throughout the chapter to represent various variables, parameters, and operations. These notations are as follows.

- $\mathsf{GFN}^{\mathbf{f}}$ denotes an internal permutation in the sponge AE, whereas $\mathbf{f}$ denotes an internal function in the GFN.
- $\mathbf{f}_{i,j,k}$ denotes an internal function $\mathbf{f}$ in GFN, where i, j represent the round number of $\mathbf{f}$ and the GFN respectively, and k represents the branch number of GFN.
- $\mathbf{f}^{'}$ denotes the internal permutation used inside the GFN at the last permutation call to output the final tag in the sponge AE, i.e., in short, we denote it as $\mathsf{GFN}^{\mathbf{f}^{'}}$.
- $T^{'}$ denotes a new tag formed by the adversary before a fault is injected at $\mathsf{GFN}^{\mathbf{f}^{'}}$ during decryption.
- m denotes the total bytes/nibbles in the $\mathbf{f}$ state. If m is a square number, we can represent the $\mathbf{f}$ state as a $\sqrt{m} \times \sqrt{m}$ matrix, i.e., $(i, j), i, j = 0, 1, \ldots, \sqrt{m} - 1$. Otherwise, the $\mathbf{f}$ state can be represented as a rectangular matrix.
- λ denotes the all possible byte/nibble differences for the SPN state, i.e.,

$$\lambda = \begin{cases} 2^8, & for\ byte\ case; \\ 2^4, & for\ nibble\ case;. \end{cases}$$

5.2.1 Sponge based Authenticated Encryption using GFN

Authenticated encryption (AE) and authenticated encryption with associated data (AEAD) are used to assure both the confidentiality and authenticity of data. AE schemes are designed using an efficient combination of a semantically secure encryption scheme and an unforgeable message authentication code (MAC) under chosen-plaintext attack. Sponge mode has been introduced with the Keccak hash func-

CiliPadi-	Algorithm	Length of				Number of rounds	
	$[n, r, a, b]$	Key	Nonce	Tag	Block	P_n^a	P_n^b
Mild	$[256, 64, 18, 16]$	128	128	64	64	18	16
Medium	$[256, 96, 20, 18]$	128	128	96	96	20	18
Hot	$[384, 96, 18, 16]$	256	128	96	96	18	16
ExtraHot	$[384, 128, 20, 18]$	256	128	128	128	20	18

Table 5.1: CiliPadi parameters with the primary member as CiliPadi-Mild

tion [197]. Later several AEAD schemes have been proposed supporting Sponge mode. Several sponges-based AE schemes have also been submitted to the ongoing lightweight cryptography competition, initiated by NIST. We observed that 25 out of 56 NIST lightweight submissions are sponge-based, where the majority of the underlying permutations inside the sponge are SPN/Feistel-like structures. In this chapter, we only focus on the sponge-based AE schemes, where the internal permutations are Feistel-like structures. Following are the sponge-based AE with GFN as the underlying public permutation, submitted to the ongoing NIST lightweight competition: ACE [198], CiliPadi [196], Oribatida [199], SPARKLE [200], SPIX [201], SpoC [202].

5.2.2 Description of CiliPadi

CiliPadi [196] is a family of lightweight authenticated encryption with associated data (AEAD). It is submitted to the NIST lightweight competition with four flavors as CiliPadi-Mild, CiliPadi-Medium, CiliPadi-Hot, and CiliPadi-ExtraHot. The CiliPadi $[n, r, a, b]$ mode of operation is based on the MonkeyDuplex [203, 157] construction and is depicted in Figure 5-1, where n denotes the state size in bits. The bitrate is r bits and the capacity is $c = n - r$ bits. The permutation (P_n^a) for the initialization and finalization phases has a rounds while the permutation (P_n^b) for the associated data and message encryption/decryption phases has b rounds, where $b > a$. All the four variants and their security parameters are listed in Table 5.1.

The internal permutation function P_n makes use of an unkeyed 2-round lightweight block cipher LED [63] as the round and branch dependent f function used inside the type-2 GFN. It consists of l branches with $\frac{n}{l}$ bits each and uses $\frac{l}{2}$ calls to f in each round. If $X_1||\cdots||X_l$ be the input to the i-th round type-2 GFN, then they are

updated by the f-function as follows:

$$X_j \leftarrow \begin{cases} X_j & \text{if} \quad j = 1, 3, \cdots, d-1 \\ X_j \oplus f^i_{j/2}(X_{j-1}) & \text{if} \quad j = 2, 4, \cdots, d \end{cases}$$

After applying f-functions to the odd-numbered branches, all the branches are shuffled by the permutation function π. For $l = 4$, $\pi = \{2, 3, 4, 1\}$. Type-2 GFN employed in CiliPadi with $l = 4$ and $l = 6$ are given in Figure 5-2. In particular, the designers have employed Type-2 GFN with $l = 4$ when the (CiliPadi) state size is 256 bits, whereas it employed Type-2 GFN with $l = 6$ for 384 bits state. For CiliPadi, the permutations π for different l are given in Table 5.2. The input to the LED cipher is 64 bits, which can be partitioned into 16 4-bit cells. Let $x = x_1||\cdots||x_{16}$ denote this input which can be viewed as a 4×4 matrix and entered row-wise. A single LED round (see Figure 5-3) consists of the following four operations: AddConstants (AC), SubCells (SC), ShiftRows (SR), and MixColumnSerial (MCS). The first two branches are used to output the 128-bit tag $T = T_1 \,||\, T_2$. More details about CiliPadi are given in [196].

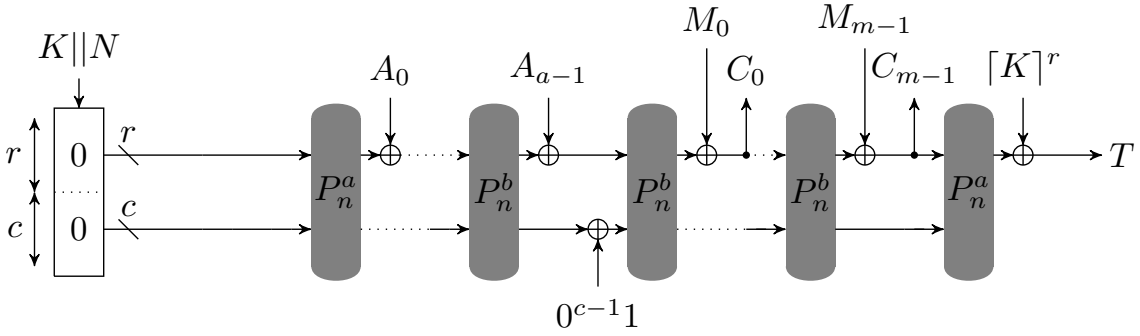
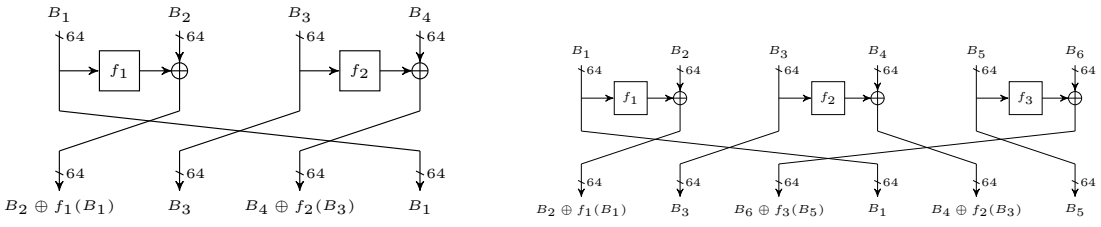


Figure 5-1: CiliPadi mode of operation



(a) $l = 4$ (b) $l = 6$

Figure 5-2: Type-2 GFN employed in CiliPadi for $l = 4$ and $l = 6$

Input length (n)	Number of branches (l)	Shuffle (π)
128	2	{2,1}
256	4	{4,1,2,3}
384	6	{4,1,2,5,6,3}

Table 5.2: Shuffling used in the Type-2 GFN

5.3 Fault Attack on CiliPadi

To perform DFA for any sequential sponge AE schemes, the attacker needs to create a replay which is actually a very difficult task. This is because a different nonce is used to initialize the state in each encryption query so that an attacker can not create an identical state to make a differential attack. Therefore, to perform DFA, we need to repeat the nonce in each query. However, in decryption performing DFA is an easy task as the attacker has no restrictions to make several queries for a fixed nonce. In the decryption query, an attacker can repeatedly make faulty queries with a new tag $T^{'}$ but with the same nonce, associated data, and message, where faults are injected at the last permutation call just before the tag verification is checked. We repeat the queries to collect some valid tag forgeries $T^{'}$. Therefore, based on the collected forgeries on sponge-based AE, we can perform DFA to recover its internal state. We now give a general idea to perform DFA on CiliPadi, which follows GFN-based sponge AE scheme where an SPN has internally used inside the GFN. The attack targets the last permutation call during a decryption query. A short description to recover the internal state is given as follows.

1. First, collect multiple forgeries using several fault-injected forging attempts (a fraction of them are valid) in CiliPadi.

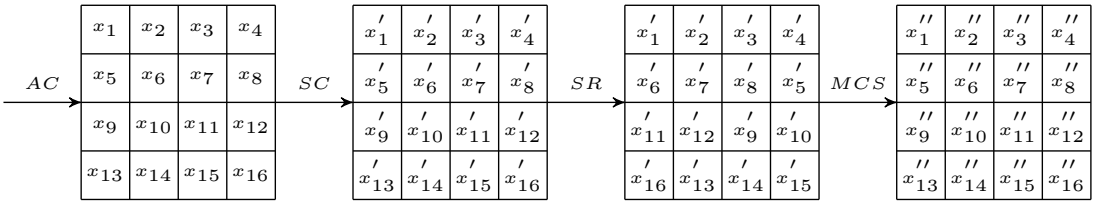


Figure 5-3: A single round of LED

2. Then, we recover the SPN state (i.e., one branch of GFN) by collecting multiple forgeries at different locations in the CiliPadi state.
3. Finally, we will retrieve other branches of GFN separately by performing DFA sequentially.

5.3.1 Random Fault Model

The fault model, we are considering here is that the effect of an induced fault is to change one nibble of the state to a random one. The injected faults can be controlled to corrupt the data in a specific round. In the context of DFA attacks, it is practical and, in fact, the minimal assumption. For example, an attacker could attempt to use a clock glitch/EM/laser to create a fault at the input of a particular round with a certain probability.

5.3.1.1 The Fault Attack Description

In this attack, we first describe how we can recover the internal branches (i.e., an SPN state of Type-2 GFN) of GFN in CiliPadi one by one by performing faulty forgery in the decryption query. Then, we again perform faulty forgery to recover other branches of GFN (i.e., a full CiliPadi state) and finally, we retrieve the master key. We present the attack on CiliPadi-Mild with 256 bits state, which employed a GFN with branch number $l = 4$. This attack can be further applied in the same way for other CiliPadi variants with $l = 4, 6$.

5.3.1.2 The Forging Attack

To attack CiliPadi-Mild, we induce faults at the first round of LED state (before the MCS operation). Let, Δ_{in} represents the input difference correponds to faults at any nibble position (i, j) in the LED state and Δ_{out} denotes the corresponding output difference after the 2nd round of LED. Also, $\Delta_{in}^{'}, \Delta_{out}^{'}$ represent the internal diffrences in the LED rounds (see Figure 5-4), where $\Delta_{in}^{'} = \mathsf{AC} \circ \mathsf{MCS}(\Delta_{in})$ and $\Delta_{out}^{'} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta_{out})$. We further divide the differential states as $\Delta_{in}^{'} = \delta_0 \,||\, \delta_1 \,||\, \delta_2 \,||\, \delta_3$ and $\Delta_{out}^{'} = \Delta_0 \,||\, \Delta_1 \,||\, \Delta_2 \,||\, \Delta_3$, where both $\delta_i, \Delta_i, \forall i \in \{0, 1, 2, 3\}$ denotes the i-th column difference. Further, δ_i can be divided into four nibbles as

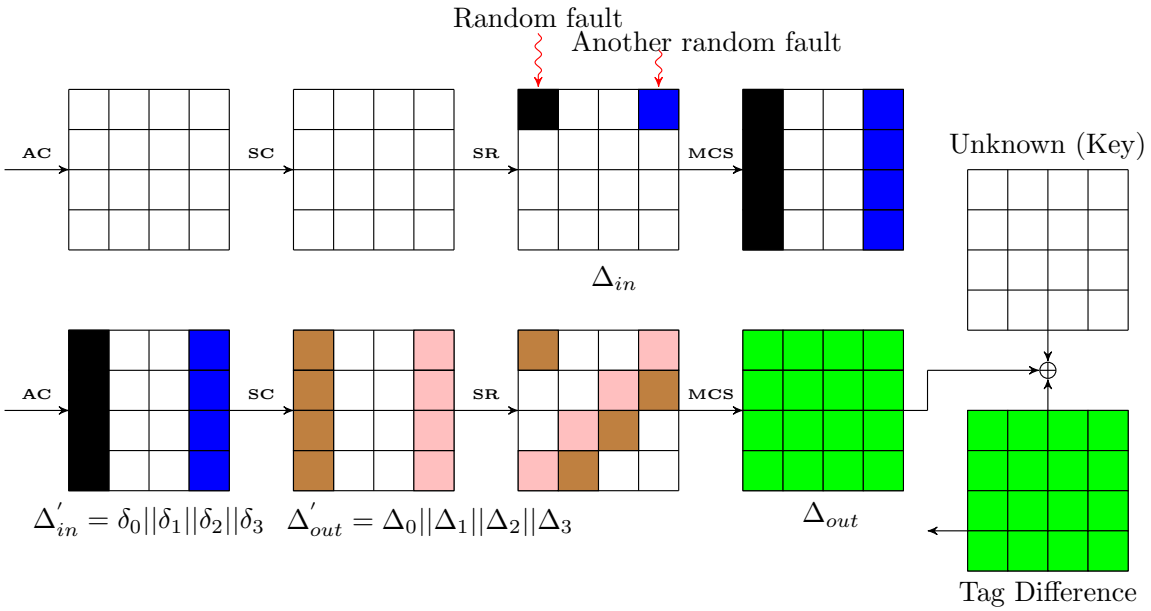


Figure 5-4: LED state recovery using faults

$\delta_i = \delta_{0,i} \,||\, \delta_{1,i} \,||\, \delta_{2,i} \,||\, \delta_{3,i}$ and similarly, $\Delta_i = \Delta_{0,i} \,||\, \Delta_{1,i} \,||\, \Delta_{2,i} \,||\, \Delta_{3,i}$. In particular, for a nibble fault at the j-th column, we use another notations as $\Delta'_{in,j}$ instead of Δ'_{in} and $\Delta'_{out,j}$ instead of Δ'_{out}. Like f', LED$'$ represents the internal permutation used inside the Type-2 GFN at the last permutation call in the sponge AE. These notations are followed throughout this section.

A brief overview of how to create a faulty forgery is as follows. At first, we fix a nibble (i, j) in the j-th column before the MCS operation at the first round of LED$'$ where we repeatedly induce faults during decryption, i.e., a random Δ_{in} is injected during decryption. Since we know which nibble position has a difference in Δ_{in}, then we can easily observe which nibbles get affected in Δ'_{out} (see Figure 5-4) after the second round SC operation in LED$'$. Here, $\Delta_{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta'_{out})$ can be easily obtained because SR, MCS are linear operations. Further, due to the last round MCS operation, it is clear from the Figure 5-4 that the number of active differences in Δ'_{out} are much less than in Δ_{out}. Therefore in each query, we first fill the non-zero nibble differences in Δ'_{out} (according to Δ_{in}) by random values (other nibble positions will be filled by zeros) and form a new tag $T' = T \oplus \Delta_{out}$. We repeat the queries (N, A, C, T') to collect q_j different forgeries and store them in a list. Also, we have further estimated the number of faulty decryptions to get more than one forgery in Theorem 1. We

continue to make faulty forgery for other columns also. The following steps to make faulty forgery in CiliPadi are described in Algorithm 1.

Theorem 1. For CiliPadi sponge AE, let χ denote faulty decryption queries to collect q different forgeries (i.e., to retrieve the i^{th} column of LED state uniquely) at the i^{th} phase in Algorithm 1. Then, the expected number of faulty queries to collect q distinct forgeries is less than $2^{20} \cdot \left[1 + \log\left(\frac{2^4}{2^4-q+1}\right)\right]$.

Proof. To make a valid forgery (according to Algorrithm 10), we have to satisfy this condition: $\mathsf{SC}(\Delta_{in}^{'}) = \Delta_{out}^{'}$. Now, for any phase i,

$$\Pr[\mathsf{SC}(\Delta_{in}^{'}) = \Delta_{out}^{'}] = \frac{2^4 - 1}{(2^4 - 1) \times (2^{16} - 1)} \approx \frac{1}{2^{16}} = p(say).$$

Let χ denote the trials to get a success, i.e., a valid forgery. So, for each trial, $\Pr[success] = p$ and $\Pr[failure] = q = 1 - p$. Therefore, $\Pr[\chi = j] = (1-p)^{j-1} \cdot p = (q)^{j-1} \cdot p$. It shows that χ follows a geometric distribution with probability p. Hence, $E(\chi) = \frac{1}{p}$.

For $1 \leq j \leq q$, let χ_j be the trials needed to collect j^{th} forgery after $j-1$ forgeries have been collected. Since χ represents the independent trials to collect q distinct successful forgeries, we can write $\chi = \chi_1 + \cdots + \chi_q$. Observe that the probability of collecting

Algorithm 1 Forging Strategy in CiliPadi

1. Make an encryption query (N, A, M) and get (C, T) pair.
2. Choose a branch (LED$^{'}$ function) of GFN at the last permutation call in CiliPadi, which directly outputs one of the 64-bits tag T_1 or is related to T_1.
3. For each value of the phase counter $j \in \{0, 1, 2, 3\}$:
   - 3a. Fix a nibble position $(i, j), 0 \leq i \leq 3$ in the j-th column, where faults are injected at the (i, j)-th position in the first round (before MCS operation) of LED$^{'}$.
   - 3b. Make several faulty decryption queries $(N, A, C, T^{'})$ by randomly choosing the j-th column to form $\Delta_{out,j}^{'}$ such that $T^{'} = T \oplus \mathsf{MCS} \circ \mathsf{SR}(\Delta_{out,j}^{'}) = T \oplus \Delta_{out}$ becomes a valid forgery. Store $T^{'}$ and the fault position (i, j) in a list $\mathcal{H}_j$.
   - 3c. Continue the step 3b to collect q_j different forgeries.
4. Do offline computation (refer to Algorithm 2) to recover the LED$^{'}$ state, i.e., a branch of type-2 GFN in CiliPadi.

j^{th} forgery is $p_j = \frac{2^4-j+1}{2^4 \times 2^{16}} = \frac{2^4-i+1}{2^{20}}$. Therefore, χ_j has geometric distribution with expectation $\frac{1}{p_j}$. By the linearity of expectations we have,

$$\begin{aligned} E(\chi) &= E(\chi_1) + \cdots + E(\chi_q) \\ &= \sum_{j=1}^{q} \frac{2^{36}}{2^4 - j + 1} \\ &= 2^{20}\left(\frac{1}{2^4} + \frac{1}{2^4 - 1} + \cdots + \frac{1}{2^4 - q + 1}\right) \\ &< 2^{20} \cdot \left[1 + \log\left(\frac{2^4}{2^4 - q + 1}\right)\right]. \end{aligned}$$

$$\left[\log(x+1) < \sum_{i=1}^{x} \frac{1}{i} < 1 + \log(x)\right]$$

5.3.1.3 LED State Recovery of Type-2 GFN in CiliPadi

For CiliPadi, two branches are directly outputed as the 128-bit tag $T = T_1 \mid\mid T_2$ by XORing with the master key $K = K_1 \mid\mid K_2$. Thus, the tag $T = T_1 \mid\mid T_2$ can be viewed as the function of two branches $(B_1 \mid\mid B_2)$ of the GFN and the master key K, i.e., $T_1 = B_1 \oplus K_1, T_2 = B_2 \oplus K_2$ (see Figure 5-5). Therefore, we can recover the state of LED$'$ by retrieving each columns separately based on the collected forgeries in the list $\mathcal{H}_j, j \in \{0, 1, 2, 3\}$. Based on the collected forgeries according to Algorithm 1, the LED$'$ state recovery is described in Algorithm 2.

[2] $\mathcal{L}_{*,j,h} = (\mathcal{L}_{0,j,h}, \mathcal{L}_{1,j,h}, \mathcal{L}_{2,j,h}, \mathcal{L}_{3,j,h})$

Algorithm 2 State Recovery of LED$'$

1. Initialize several empty lists $\mathcal{L}_{u,v,h}, 0 \leq u, v \leq 3$ and $0 \leq h <\mid \mathcal{H}_v \mid$.
2. For each value of phase counter $j \in \{0, 1, 2, 3\}$:
   - 2a. for each $h \in \{0, 1, \cdots, \mathcal{H}_j - 1\}$:
     - 2a.1. Take the forgery T' and the fault position (i, j) from the list $\mathcal{H}_j$.
     - 2a.2. Make all 2^4 j-th column differences at the output of $\Delta'_{in,j} = \mathsf{AC} \circ \mathsf{MCS}(\Delta_{in})$ (see Figure 5-4), where Δ_{in} has a nibble hamming weight of 1.
     - 2a.3. Compute $\Delta_{out} = T \oplus T'$, and then, $\Delta'_{out,j} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta_{out})$. This $\Delta'_{out,j}$ has a j-th column difference.
     - 2a.4. For each (2^4 number of) j-th column differences $\Delta'_{in,j}$:
       - 2a.4.1. **ListUpdate**$(j, \Delta'_{in,j}, \Delta'_{out,j}, \mathcal{L}_{*,j,h})$[2].

5.3.1.4 Branch Recovery of CiliPadi

To recover each branch of GFN using faulty forgeries, we need some strategy to do that. A brief description to retrieve all l branches of GFN is as follows:

1. Target one $\mathbf{f}^{'}$ = LED$^{'}$ function (one branch) at the last round of GFN which directly outputs the tag part. Make faulty forgeries on this $\mathbf{f}^{'}$ function and retrieve its state as described in Section 5.3.1.3, i.e., one branch of GFN.
2. Then, target another $\mathbf{f}^{'}$ function that either directly outputs the tag part, or it is directly related to the $\mathbf{f}^{'}$ function whose state value has already been retrieved using faulty forgery. In either case, make faulty forgery on this $\mathbf{f}^{'}$ function and retrieve the full state.
3. In a similar fashion, we repeat step 2 until we recover all the l branches of GFN$^{\mathbf{f}}$.

The strategy to recover the internal permutation in CiliPadi-Mild is to perform faulty forgery to all the branches separtely which is depicted in Figue 5-5. In Figure 5-5, both the functions $\mathbf{f}_1^{'}$, $\mathbf{f}_2^{'}$ represent two round LED ciphers. We use this notation to present branch recovery of CiliPadi in a simple way. In the following way, we will recover each of the four branches of CiliPadi-Mild. At first, we can recover a branch (shown in red color in Figure 5-5) using the forging strategy and the state recovery as discussed in Algorithms 1 and 2. We can also recover another branch (shown in blue color in Figure 5-5) using Algorithms 1 and 2 because the output of $\mathbf{f}_2^{'}$ is directly produces the tag by XORing with the key K_2. Since we have retrieved both the branches shown in red and blue colors in Figure 5-5, the branch shown in green color is now known to us. Next, we target the third branch shown in violet color in Figure 5-5. We inject faults at the first round of the SPN ($\mathbf{f}_1^{'}$) state in the third last round of the type-2 GFN. In each query, we choose a random state difference $\Delta_{out}^{'}$ in

Algorithm 3 ListUpdate$(j, \Delta_{in,j}^{'}, \Delta_{out,j}^{'}, \mathcal{L}_{*,j,h})$

1. Check $\forall i \in \{0, \cdots, 3\}$, DDT$[\delta_{i,j}][\Delta_{i,j}] > 0$ or not? If yes, then do the following:
   1a. For each nibble position i in j-th column:
      1a.1. For each $x \in \{0, 1, \cdots, 15\}$:
         1a.1.1. If $S(x) \oplus S(x \oplus \delta_{i,j}) == \Delta_{i,j}$ holds, then store x in the list $\mathcal{L}_{i,j,h}$.

$\mathbf{f}_1^{'}$ state and produce $T^{'}$ as $T^{'} \leftarrow T \oplus \mathbf{f}_1^{'} \circ \mathbf{f}_1^{'}(\Delta_{out})$, where $\Delta_{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta_{out}^{'})$. This computation $\mathbf{f}_1^{'} \circ \mathbf{f}_1^{'}(\Delta_{out})$ is possible as we have already know both the internal branches, shown in green and red colors in Figure 5-5. Thus, using Algorithms 1,2, we can recover the branch shown in violet color.

The fourth branch shown in yellow color in Figure 5-5 can be recovered in the same way. First, we inject faults at the second to last round of the $\mathbf{f}_1^{'}$ function in the fourth last round of the type-2 GFN. In each decryption query, we choose a random difference $\Delta_{out}^{'}$ in the $\mathbf{f}_1^{'}$ state and make $T^{'} = T \oplus \mathbf{f}_1^{'} \circ \mathbf{f}_1^{'} \circ \mathbf{f}_1^{'}(\Delta_{out})$, where $\Delta_{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta_{out}^{'})$. Then, by peforming faulty forgery as described in Algorithm 1, we can retrieve the $\mathbf{f}_1^{'}$ state using Algorithm 2.

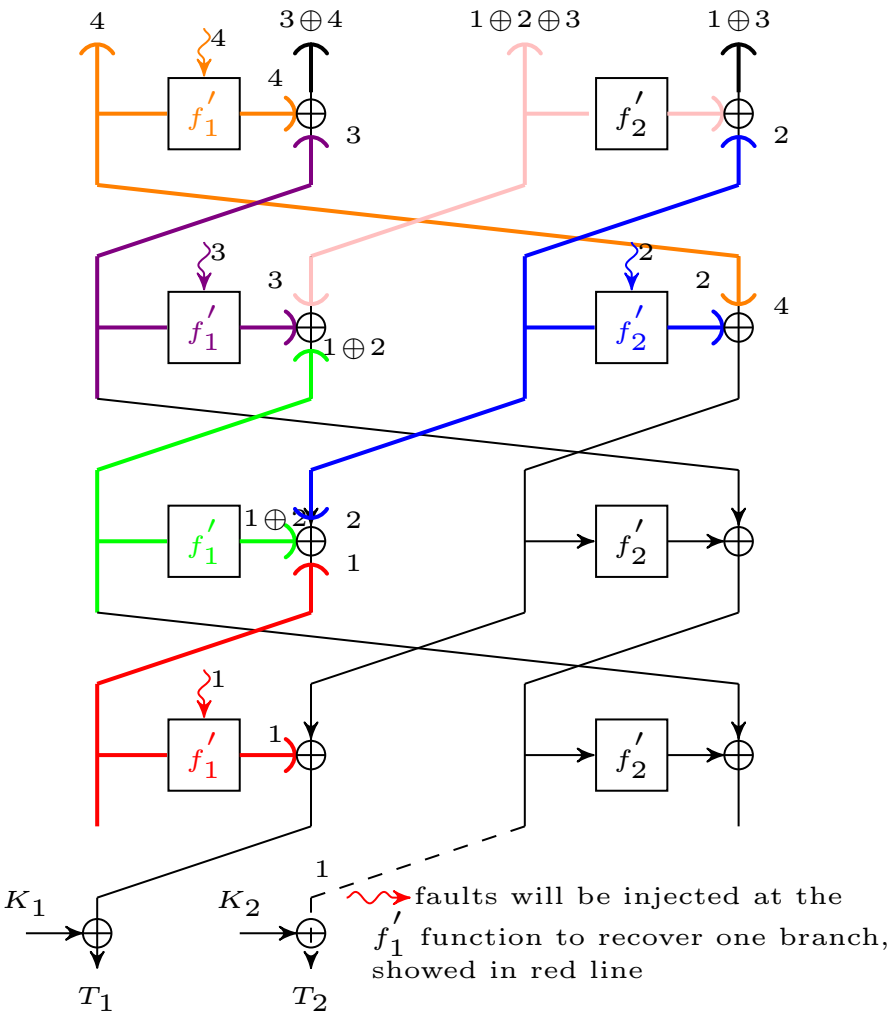


Figure 5-5: Branch Recovery of Type-2 GFN

5.3.1.5 Key Recovery of CiliPadi AEAD

We have discussed in the previous section how we can recover all the branches of the GFN using faults. Moreover, we have outlined a branch recovery of type-2 GFN in Figure 5-5. Thus, in the same way, we can retrieve all the branches of the type-2 GFN in CiliPadi, and finally, the secret key directly retrieved by XORing the 128-bits internal state with tag T. Also, in CiliPadi mode, there are no extra key injections used either after the initialization or before the finalization calls. Therefore, we can go back up to initializing state by inverting the internal permutations and recovering its secret key.

5.3.1.6 Attack Complexity

Practically, we have implemented[3] the state recovery of CiliPadi and checked that if $q = 6$, we can retrieve a column of LED state uniquely. So, according to Theorem 1, we need to perform around $2^{18.8}$ faulty queries to retrieve a column of LED state. Also, for the key recovery in CiliPadi AE, we need to recover all the l different branches of GFN. Therefore, the required faulty queries to retrieve all the branches of GFN will be $l \times 2^{18.8} = 4 \times 2^{18.8} \approx 2^{21}$ (for CiliPadi-Mild). Also, according to Algorithm 2, the time complexity will be $2^2 \times 2^{2.5} \times 2^4 \times 2^2 \times 2^4 \approx 2^{14.5}$. Further, in Algorithm 2, there are several lists used to store the nibbles. Thus, the memory complexity will be $2^2 \times 2^2 \times 2^{2.5} \times 4 \approx 2^{8.5}$ (as the maximum DDT entry is 4) nibbles.

5.3.1.7 Theory vs. Experiment.

In this section we present a set of simulations of our random fault model and compare the expected number of required faults with the corresponding theoretical estimates. The simulations were performed on a Intel(R) Core(TM) i5-8250U CPU @1.60GHz computer. According to our simulation and the theoretical estimations, the total number of faulty queries required to retrieve a column of the LED state for random fault model is given in Table 5.3. Also, we have done this simulation by running the procedure 1,00,000 times for this model.

[3] https://github.com/janaamit001/State-Recovery-of-LED-Permutation-in-the-CiliPadi-AEAD

Number of forgeries (q)	Random Fault Model	
	Theory	Simulation
1	2^{16}	$2^{15.5}$
2	2^{17}	$2^{16.7}$
3	$2^{17.7}$	$2^{17.3}$
4	$2^{18.2}$	$2^{17.9}$
5	$2^{18.5}$	$2^{18.3}$
6	$2^{18.8}$	$2^{18.6}$

Table 5.3: Expected Number of Faulty Queries corresponding to Theorem 1

5.4 Generalized Fault Attack on GFN-based Sponge AE

In the previous section, we have introduced how an attacker can perform DFA to make faulty forgery in CiliPadi. The CiliPadi family of AE is a GFN-based sponge AE where LED is employed inside the Type-2 GFN. We observe that, for duplex sponge mode with any GFN-based permutation, we might be able to recover the full permutation state by performing faulty forgery in the decryption query. This observation also makes the reasonable assumption that the round function $\mathbf{f}$ used in GFN follows SPN-like structure, and at least one branch of GFN is output as the final tag.

We now give a general idea to perform faulty forgery by inducing byte/nibble differences on the $\mathsf{GFN}^{\mathbf{f}'}$-based sponge AE. The attack targets the last permutation call during a decryption query. To recover its internal state, a short description is given as follows.

1. First, collect multiple valid forgeries using several fault-injected forging attempts (a fraction of them are valid). The number of forgeries depends on the underlying construction. To be precise, random faults are injected at a fixed byte/nibble position in the $\mathbf{f}'$ state (typically, at the second to last round in $\mathbf{f}'$) in GFN.
2. Then, we recover the full $\mathbf{f}'$ state of GFN (i.e., one branch of $\mathsf{GFN}^{\mathbf{f}'}$) by collecting multiple forgeries to different locations in the state.
3. Finally, we will retrieve other branches of $\mathsf{GFN}^{\mathbf{f}'}$ separately by performing DFA sequentially.

In general, a single SPN round consists of the following four operations: AddConstants (AC), SubCells (SC), ShiftRows (SR) and MixColumnSerial (MCS). Like CiliPadi, We first assume that the SPN inside the GFN have at least two rounds. In this case, we inject random faults at the second to last round of the SPN and perform faulty forgery on different byte/nibble positions to retrieve its state. still, there will be another issue if a single round SPN is employed in GFN. In this case, we can recover the state if we switch from the random fault model to the known fault model. We inject known faults at the last round (before SC operation) in SPN and make faulty forgery concerning different positions in the SPN state. Then, recover its state based on the collected faulty forgery. Thus, we need two different fault models to overcome the situation. In the first fault model, faults are induced at the 2nd last round in the state. A state recovery of GFN (where SPN has at least two rounds) under this fault model is described in Section 5.4.1. The second fault model is a known fault model, where the fault is random, but its value is known to the attacker. Under this fault model, we describe a state recovery of GFN (where SPN has either only one round or more rounds) in Appendix 5.4.2.

5.4.1 Fault Attack on SPN-based GFN Sponge AE

In this attack, we first describe SPN state recovery (i.e., a branch of GFN) by performing faulty forgery in the decryption query. Then, we repeatedly perform this faulty forgery to recover other branches of GFN. Further, we discuss the possibilities of the key recovery to this scheme.

5.4.1.1 The Fault Model

Here, we have considered a random fault model whre the effect of an induced fault is to change one byte/nibble of the internal state to a random one. For example, an attacker could attempt to use a clock glitch/EM/laser to create a fault at the specific byte/nibble position of a particular round with hign accuracy.

5.4.1.2 The Fault Attack Description

Assuming we have a duplex sponge with a permutation $\mathsf{GFN}^{\mathbf{f}}$ ($\mathbf{f}$ represents an SPN based structure), where at least one full branch of $\mathsf{GFN}^{\mathbf{f}}$ is output as the final tag

T. Assuming $\mathbf{f}^{'}$ has at least two rounds. In this attack, we inject byte/nibble faults at the internal state (before the SR operation) during the second to last round of $\mathbf{f}^{'}$, i.e., at the $\mathbf{f}^{'}_{\mathbf{a-2,b,k}}$ state. For the sake of simplicity, we describe the attack with AES-like internal permutation $\mathbf{f}^{'}$, which seems practically relevant. Later, we will explain that this attack can be extended to most of the SPN structures. One round $\mathbf{f}^{'}$ permutation can be viewed as a composition of four layers: AddConstants (AC), SubCells (SC), ShiftRows (SR), and MixColumnsSerial (MCS).

A short description of forging and state recovery attacks is as follows. At the first phase, we make an encryption query (N, A, M) to get (C, T). We choose a byte/nibble position e_0 inside the $\mathbf{f}^{'}_{\mathbf{a-2,b,k}}$ state. We make several decryptions as $(N, A, C, T^{'})$ by repeatedly injecting faults on the byte/nibble position e_0 and collect one forgery $T^{'}$. We again repeat this faulty decryption by injecting faults at the e_0-th byte/nibble position to collect q_0 (say) different tag forgeries $T^{'}$. The last round SR, MCS do not have any impact on the state recovery attack of $\mathbf{f}^{'}$ function. Let us assume

[1]Let us take the $\mathbf{f}^{'}$ state difference as $\Delta_{in} = (0, 0, \ldots, \delta_e, 0)$, $\delta_e > 0$ and $0 \leq e \leq m-1$. Then, $\mathrm{MCS} \circ \mathrm{SR}(\Delta_{in}) = (\delta_0^{'}, \delta_1^{'}, \cdots, \delta_{m-1}^{'})$, where $\delta_j^{'} > 0$, $\forall j \in \{o_1, \cdots, o_\sigma\}$ and $\delta_j^{'} = 0$ for the remaining $m-\sigma$ positions. Based on the above example, we define a new function $\mathrm{MC} : \{1, 2, 3, \cdots, m\} \rightarrow S \subset \{1, 2, 3, \cdots, m\}$ such that $\mathrm{MC}(e) = \{o_1, o_2, \cdots, o_\sigma\}$, where e represent a byte/nibble fault position and $o_1, o_2, \cdots, o_\sigma$ are different byte/nibble (non-zero difference) positions after applying SR, MCS operations to Δ_{in}.

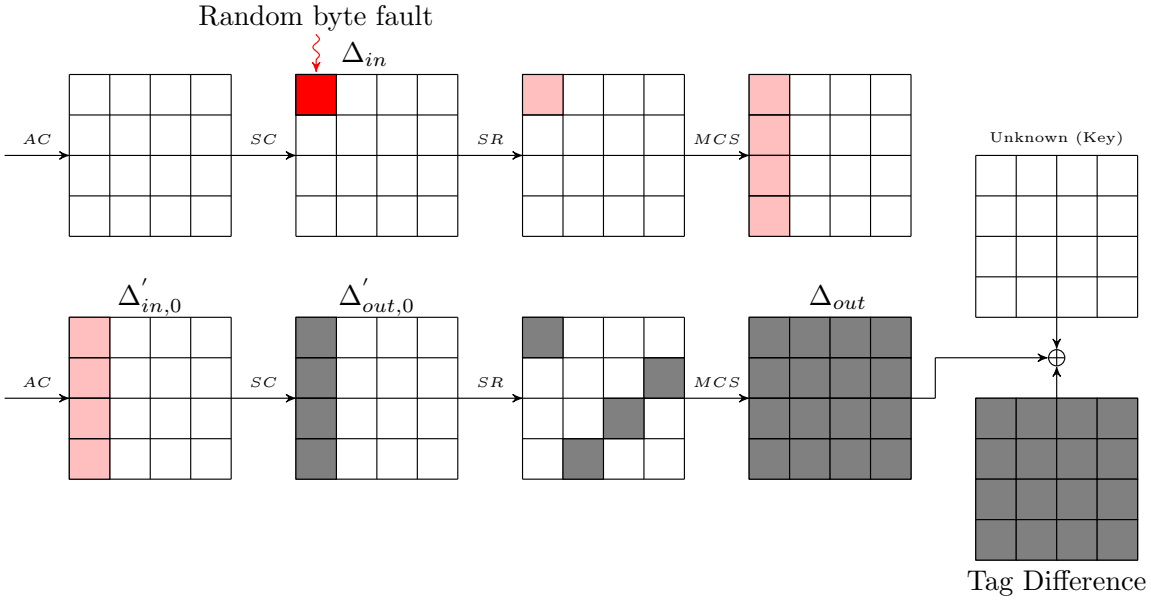


Figure 5-6: An Example of Forgery for AES-like structure

that the difference at the byte/nibble position e_0 diffused to r_0 bytes/nibbles after going through the second to last round operations SR and MCS. Now, we aim to recover these r_0 bytes/nibbles uniquely by guessing those bytes/nibbles and invert them up to 2nd last round's SR^{-1} operation and then check whether it leads to a single byte/nibble difference at the position e_0.

Next in the second phase, we choose another faulty byte/nibble position e_1 at the $\mathbf{f}'_{\mathbf{a-2,b,k}}$ state, and this difference propagated to r_1 bytes/nibbles after the SR and MCS operations. These r_1 bytes/nibbles differences do not coincide with the previous r_0 bytes/nibbles positions at the first phase. Then, We repeat this process to collect q_1 (say) tag forgeries T' and recover r_1 bytes/nibbles of the state uniquely. Similarly, we will continue to recover other bytes/nibbles uniquely in the next phases until the full $\mathbf{f}$ state is recovered uniquely. To perform faulty forgery, the following steps are described in Algorithm 4.

We showcase an example of the above attack steps by taking AES-like $\mathbf{f}'$, where the

Algorithm 4 Forging Strategy

1. Make an encryption query (N, A, M) and get (C, T).
2. Choose an $\mathbf{f}'_{\mathbf{a-2,b,k}}$ function at the last permutation call of the sponge function where the k-th branch either directly ouputs the tag at the last round of GFN or is related to the tag at the intermediate round of GFN.
3. Initialize a phase counter $PhC = 0$ and a set $\mathcal{A} = \emptyset$ which store the byte/nibble positions of $\mathbf{f}'$ state.
4. If $\mid \mathcal{A} \mid = m$, then goto step 8, else, goto step 5.
5. At the PhC^{th} phase, fix a byte/nibble position e $(0 \leq e \leq m-1)$ from the PhC^{th} column at the $\mathbf{f}'_{\mathbf{a-2,b,k}}$ state, where faults are repeatedly induced at this position in each decryption query. Let Δ_{in} and Δ_{out} denote the corresponding input and output differences respectively. Then, compute $\Delta'_{out,PhC} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta_{out})$ (see Figure 5-6). This position e will be choosen in such a way that, $\mathrm{MC}(e)^1 \cup \mathcal{A}$, maximizes. Update $\mathcal{A} \leftarrow \mathrm{MC}(e) \cup \mathcal{A}$.
6. In each query, choose $\Delta'_{out,PhC}$ randomly. Make faulty decryption queries of the form (N, A, C, T') with $T' = T + \Delta_{out}$ $(= \mathsf{MCS} \circ \mathsf{SR}(\Delta'_{out,PhC}))$ such that T' becomes a valid forgery. Store the forging tag T' in the list $\mathcal{H}_{PhC}$. Continuing this step to collect q_{PhC} tag forgeries.
7. Increment PhC by 1 and repeat steps 4-6.
8. Do offline computation to recover the full f' state, i.e., one branch of GFN.

final tag is generated by XORing with the key and the output of $\mathbf{f}^{'}$ function. The byte (/nibble) fault will be injected at the second to last round of $\mathbf{f}^{'}$ before the SR operation. At the 0-th phase, we fix the fault position as $(0,0)$-th byte (/nibble) of the $\mathbf{f}^{'}_{\mathbf{a-2,b,k}}$ function. Therefore, the input difference Δ_{in} has only one byte/nibble difference at the $(0,0)$-th position, whereas the output difference Δ_{out} has full state difference. Now, the internal state diffrence $\Delta^{'}_{out,0}$ only has a (1st) column difference (see Figure 5-6), if we apply the $\mathsf{SR}^{-1}, \mathsf{MCS}^{-1}$ operations on Δ_{out}. Then, perform several faulty decryption queries $(N, A, C, T^{'})$ by choosing a random column difference to form $\Delta^{'}_{out,0}$ such that $T^{'} = T \oplus \mathsf{MCS} \circ \mathsf{SR}(\Delta^{'}_{out,0})$ becomes a valid forgery. Repeat this procedure to collect q_0 different forging tags, which helps to recover the first column uniquely. Then, we change the fault position to $(0,1)$ and make several faulty decryption queries with $T^{'} = T \oplus \mathsf{MCS} \circ \mathsf{SR}(\Delta^{'}_{out,1})$ and expect one $\mathsf{faulty\ forgery}$. Repeat this procedure to collect q_1 different forging tags, which helps to recover the second column uniquely. In this way, we can recover other columns uniquely and, finally, retrieve the full state of $f^{'}$ state.

Therefore, we need to estimate the required number of faulty decryptions to get one forgery. Then, we have to estimate the number of faulty decryptions to get more than one forgeries. The following theorems will give the answer.

Theorem 2. For any SPN $(\mathbf{f}^{'})$ based $\mathsf{GFN}^{\mathbf{f}^{'}}$ with sponge duplex AE, by attempting λ^{σ} faults with respect to a fixed byte/nibble position in the decryption query of $\mathbf{f}^{'}_{\mathbf{a-2,b,k}}$ function (an associated branch that immediately outputs the tag part) and choosing $T^{'} = T \oplus \Delta_{out}$ randomly in each query (according to Algorithm 4), we expect one tag forgery.

Proof. In the decryption query, a byte/nibble fault Δ_{in} will be induced at the fixed byte/nibble position of the $\mathbf{f}^{'}_{\mathbf{a-2,b,k}}$ state. Let Δ_{out} denote the corresponding output (tag) difference after the last round of $\mathbf{f}^{'}$ function. Due to the linear operations (SR,MCS), the byte/nibble difference will be propagated to some number of different byte/nibble positions. Let $\Delta^{'}_{in}$ denotes the output state difference before the SC operation at the last round of $\mathbf{f}^{'}$, i.e., $\Delta^{'}_{in} = \mathsf{AC} \circ \mathsf{MCS} \circ \mathsf{SR}(\Delta_{in})$ (see Figre 5-6 for AES-like $\mathbf{f}^{'}$). In each decryption query, we randomly choose a $\Delta^{'}_{out} = SR^{-1} \circ MCS^{-1}(\Delta_{out})$ such that $T^{'} = T \oplus \Delta_{out}$ becomes a valid forgery. Let us assume that, $\Delta^{'}_{out}$ have σ

(say) non-zero byte/nibble differences. Now, the total number of byte/nibble faults for Δ_{in} is λ. Hence, the total number of $\Delta_{in}^{'}$ is λ. Again, the total number of byte/nibble differences of $\Delta_{out}^{'}$ will be λ^{σ}.

Therefore,

$$\Pr\left(SC(\Delta_{in}^{'}) = \Delta_{out}^{'}\right) = \frac{\lambda - 1}{(\lambda - 1) \times (\lambda^{\sigma} - 1)} \approx \frac{1}{\lambda^{\sigma}} = p(say).$$

Let $\mathcal{X}$ denote the independent trials to get a success, where $\Pr(success)$ for each trial is $p = \frac{1}{\lambda^{\sigma}}$. Then,

$$\Pr(X = i) = (1 - p)^{i-1} \cdot p = q^{i-1} \cdot p, q + p = 1.$$

It shows that $\mathcal{X}$ follows a geometric distribution with probability p. Hence, $E(X) = \frac{1}{p}$.

Theorem 3. For any SPN (**f**) based GFN with sponge duplex AE, we can expect q tag forgeries by attempting at most $\lambda^{\sigma+1} \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q}\right)\right]$ faulty decryption queries.

Let $\mathcal{X}$ be the independent trials needed to collect q successful forgeries. Let $\mathcal{X}_i$ be the trials needed to collect i-th forgery after $i - 1$ forgeries have been collected. Then $\mathcal{X} = \mathcal{X}_1 + \cdots + \mathcal{X}_q$. Observe that the probability of collecting a new forgery is $p_i = \frac{\lambda - i + 1}{\lambda^{\sigma+1}}$.

Therefore, $\mathcal{X}_i$ has geometric distribution with expectation $\frac{1}{p_i}$. By the linearity of expectations we have,

$$\begin{aligned}
E(\mathcal{X}) &= E(\mathcal{X}_1) + \cdots + E(\mathcal{X}_q) \\
&= \frac{\lambda^{\sigma+1}}{\lambda} + \frac{\lambda^{\sigma+1}}{\lambda - 1} + \cdots + \frac{\lambda^{\sigma+1}}{\lambda - q + 1} \\
&= \lambda^{\sigma+1}\left(\frac{1}{\lambda} + \frac{1}{\lambda - 1} + \cdots + \frac{1}{\lambda - q + 1}\right) \\
&\leq \lambda^{\sigma+1} \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q}\right)\right]. \\
&\left[\log(\lambda + 1) \leq \sum_{i=1}^{\lambda} \frac{1}{i} \leq 1 + \log(\lambda)\right]
\end{aligned}$$

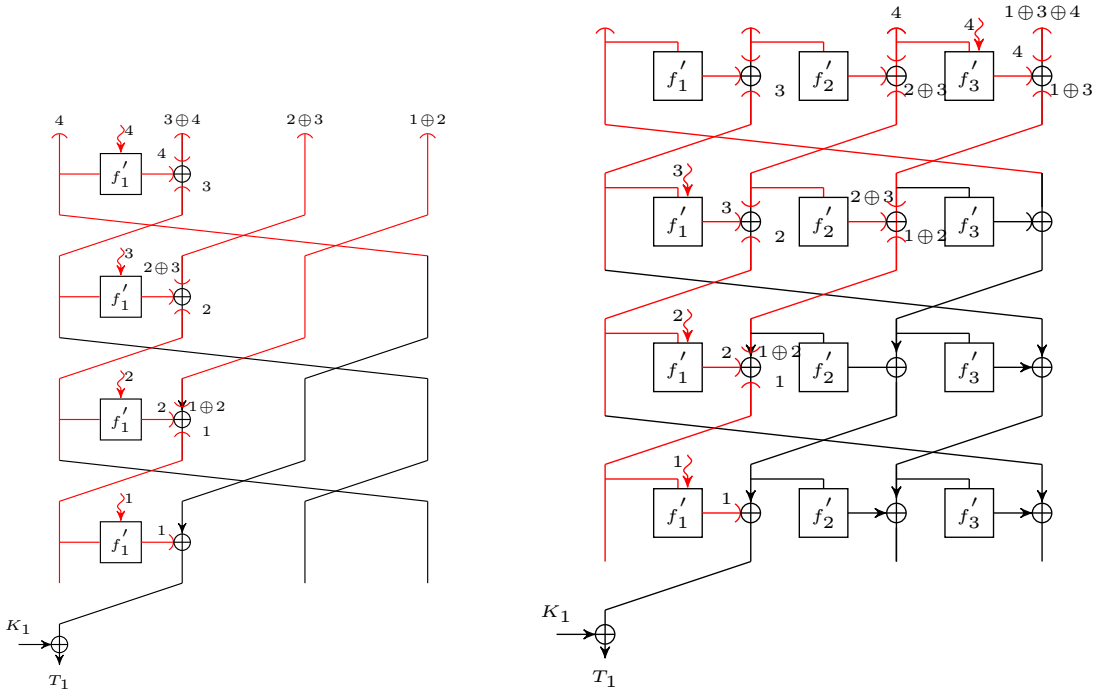
5.4.1.3 SPN State Recovery in GFN

According to the Algorithm 4, at the PhC^{th} phase, let r_{PhC} denotes the total byte/nibble differences in $\Delta'_{out,PhC}$. Also, let $\mid \mathcal{H}_{PhC} \mid = q_{PhC}$ represents the total different forging tags T'. To recover the $\mathbf{f}'$ state (i.e., a branch of the GFN), the following steps are in Algorithm 5 (see Figure 5-6 presents a differential to follow this Algorithm 5).

5.4.1.4 Branch Recovery of SPN-based GFN

We have demonstrated that how to perform DFA that leads to a faulty forgery. In this section, we give a concise description of how we can retrieve the state of the balanced GFN (i.e., BFN) that internally uses SPN-based function $\mathbf{f}$. We also show that at least one branch as a part of the tag is sufficient to recover the GFN state. We can use the same idea to recover all its l branches which we have already discussed in Section 5.3.1.4.

Therefore, by injecting faults at the four different branches of the type-2 GFN$^{\mathbf{f}'}$, we can



(a) Type-1 GFN (b) Type-3 GFN

Figure 5-7: State Recovery of Type-1,3 GFNs using Faults

retrieve all the four branches separately. In a similar fashion, we can recover all the branches of type-1, and type-3 Feistel designs, shown in Figure 5-7a, 5-7b respectively. The above analysis, especially for the branch recovery using faulty forgery at the second to last round of the SPN structure of GFN-based sponge AE is summarized in the following lemma.

Lemma 1. For any SPN-based BFN with branch number $l \geq 2$, we can retrieve each of the l branches by making faulty forgeries on l different $\mathbf{f}$ functions separately in the decryption query.

Similarly, instead of BFN, if UFN is used as the internal permutation of the sponge AE scheme, where the underlying $\mathbf{f}^{'}$ function inside UFN is the composition of SPN and expanding/contracting function, then the same attack is also applied to recover the branch for the UFN.

5.4.1.5 Key Recovery of Sponge AE

The sponge state will be retrieved when we successfully recover all the branches of $\mathsf{GFN}^{\mathbf{f}^{'}}$. Thus, the state recovery will help to retrieve the secret key for the sponge AE if there are no extra key injections used in the sponge design except to initialize the sponge. Further, if there is an extra key injection to output the final tag (by XORing the rate part of the last permutation call with the secret key), then we don't need to recover all the branches of GFN. Because, in this case, we can recover the secret key if we can recover only those branches which directly outputs the tag part. Therefore, in this case, we can significantly reduce all the complexities (data, time, and memory) of this attack.

5.4.1.6 Attack Complexity

According to Theorem 3, μ byte/nibble faults are required to get at least one successful forgery. According to Algorithm 4, the $\mathbf{f}^{'}$ (SPN) state has $(PhC-1)$ phases to retrieve its full state, where $(PhC-1)$ represents the columns in the $\mathbf{f}^{'}$ state. So, for any $\mathbf{f}^{'}$ state, let $q = q_1 + q_2 + \cdots + q_{\text{PhC-1}}$ $(q_1 = \ldots = q_{\text{PhC-1}})$ represents total number of different forged tags to retrieve the $\mathbf{f}^{'}$ state uniquely. Thus, we need at least $(PhC-1) \cdot \lambda^{\sigma} \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q_1}\right)\right]$ byte/nibble faults to recover the full $\mathbf{f}^{'}$ state

uniquely. Again, there are l different branches for the $\mathsf{GFN}^{\mathbf{f}'}$ inside the sponge-based AE. Hence, we need at least $l \cdot (PhC - 1) \cdot \lambda^{\sigma} \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q_1}\right)\right]$ byte/nibble faults to recover the full sponge state.

5.4.2 Fault Attack on Single Round GFN-based Sponge AE

In this attack, we first describe the fault model. Then, give a description of the state recovery by repeatedly performing faulty forgery in the decryption query.

Algorithm 5 State Recovery of $\mathbf{f}'$

1. Compute all $\lambda \cdot m$ possible differences at the output of $\mathsf{MCS} \circ \mathsf{SR}(\Delta_{in})$, where the internal state difference Δ_{in} of $\mathbf{f}'_{\mathbf{a-2,b,k}}$ has a byte/nibble hamming weight of 1. Store them in a list $\mathcal{D}$. Initialize a phase counter $PhC_1 = 0$.
2. At the PhC_1^{th} phase, initialize two lists as $\mathcal{L}'_{PhC_1}, \mathcal{L}_{PhC_1} \leftarrow \emptyset$.
3. Consider the tag T and a faulty tag T' from the list $\mathcal{H}_{PhC_1}$ out of q_{PhC} faulty tags, where $\Delta_{out} = T \oplus T'$. Then, Compute $\Delta'_{out,PhC_1} = \mathsf{SR}^{-1} \circ \mathsf{MCS}(\Delta_{out})$ (see Figure 5-6).
4. If the list $\mathcal{L}_{PhC_1} == \emptyset$, goto step 4a, else goto step 5.

4a. Make a list $\mathcal{L}_{PhC_1}$ which contains set of all possible r_{PhC_1} byte/nibble values. These r_{PhC_1} bytes/nibbles are corresponding to the non-zero differences of Δ'_{out,PhC_1}.

5. For all $X \in \mathcal{L}_{PhC_1}$:

   5a. Create states X, X', where $X' = X \oplus \Delta'_{out,PhC_1}$.

   5b. Compute the difference $\mathsf{AC}^{-1} \circ \mathsf{SC}^{-1}(X) \oplus \mathsf{AC}^{-1} \circ \mathsf{SC}^{-1}(X')$ and check wheather it is in $\mathcal{D}$.

   5c. If yes, then add X to the list $\mathcal{L}'_{PhC_1}$.

7. Update $\mathcal{L}_{PhC_1} \leftarrow \mathcal{L}'_{PhC_1}$ and $\mathcal{L}'_{PhC_1} \leftarrow \emptyset$.
8. Repeat step 3-7 for all pairs in $\mathcal{H}_{PhC_1}$ until the list $\mathcal{L}_{PhC_1}$ contains a unique value.
9. Increment the phase counter $PhC_1 = PhC_1 + 1$.
10. Repeat steps 2-8 to retrieve the remaining byte/nibble values of the $\mathbf{f}'_{\mathbf{a,b,k}}$ state uniquely.

5.4.2.1 The Fault Model

We are considering a known byte/nibble fault model in which an attacker intentionally injects a fault into a specific byte or nibble of a system or algorithm. This fault is assumed to cause a fully biased distribution of faulty values, and the attacker has complete knowledge of this statistical distribution.

5.4.2.2 The Fault Attack Description

In this attack model, we will inject a byte/nibble fault at the last round of SPN ($\mathbf{f}$) structure (before SC operation). Here, we assume that the last round of $\mathbf{f}$ permutation has the MixColumnsSerial operation. The following steps, by an attacker to get a faulty forgery using faults in the decryption query are given in Algorithm 6.

For better understanding, the above steps are explained for AES-like $\mathbf{f}'$, where MCS is used in the last round of $\mathbf{f}'$ permutation. In this attack, faults will be injected at the last round of $\mathbf{f}'$ just before the SC operation. In the 1st phase, select i_0-th byte/nibble as the fault position. Then, repeatedly make faulty decryption queries by injecting faults at the i_0-th position to collect q_0 tag forgeries. Next in the 2nd phase, choose the i_1-th byte/nibble as the fault position so that $\mathrm{MC}(i_1) \cap \mathrm{MC}(i_0)$ will be minimized or an empty set. Then, repeat the faulty decryption queries (with faults at the i_1-th position) to collect q_1 tag forgeries. Finally, continue this for other

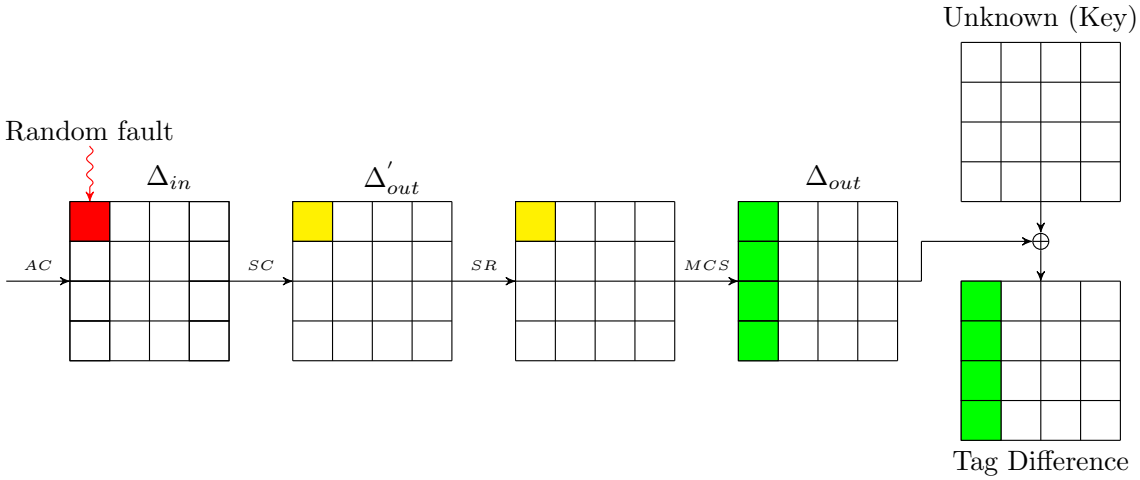


Figure 5-8: Forgery in Sponge-based AE with AES-based GFN as Underlying Permutation

faulty positions until we have $\mathrm{MC}(i_0) \cup \mathrm{MC}(i_1) \cup \cdots \cup \mathrm{MC}(i_{PC-1}) = \{1, 2, \cdots, m\}$. The number of required faulty decryptions to get one forgery is summarized in the following theorem.

Theorem 4. Let χ denote the faulty decryption queries to collect q distinct tag forgeries, i.e., repeatedly induce nibble (known) faults at the fixed position in the state (last round) until we get q different forgeries. Then, $E(\chi) < \lambda^2 \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q + 1}\right)\right]$.

Proof. To make a valid forgery, we have to satisfy this condition: $\mathsf{SC}(\Delta_{in}) = \Delta'_{out}$. Therefore, at any phase $i, 0 \leq i < 4$,

$$\Pr[\mathsf{SC}(\Delta_{in}) = \Delta'_{out}] = \frac{\lambda - 1}{(\lambda - 1) \times (\lambda - 1)} \approx \frac{1}{\lambda} = p(say).$$

Let, $\chi_j, 1 \leq j \leq q$ be the trials needed to collect j^{th} forgery after $j - 1$ forgeries have been collected. As χ represents the independent trials needed to collect q successful forgeries, we have, $\chi = \chi_1 + \cdots + \chi_q$. Further, the probability of collecting j^{th} forgery

Algorithm 6 Forging Strategy

1. Make an encryption query (N, A, M) and get (C, T).
2. Choose an $\mathbf{f}'_{\mathbf{a-1,b,k}}$ function at the last permutation call of the sponge function where the k-th branch either directly ouputs the tag at the last round of GFN or is related to the tag at the intermediate round of GFN.
3. Initialize a phase counter $PC = 0$ and $\mathcal{A}_1 = \emptyset$.
4. If $\mid \mathcal{A}_1 \mid = m$, then goto step 8, else goto step 5.
5. At the PC^{th} phase, fix a byte/nibble position $e, 0 \leq e \leq m-1$ from PC^{th} column at the $\mathbf{f}'_{\mathbf{a-1,b,k}}$ state, where faults (δ (say) is random but known to the attacker) are repeatedly induced before the SC operation at the $\mathbf{f}'_{\mathbf{a-1,b,k}}$ state. Let $\Delta_{in}, \Delta_{out}$ denote the corresponding input and output differences (see Figure 5-8). Then, compute $\Delta'_{out} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta_{out})$. This position e will be choosen in such a way that, $\mathrm{MC}(e)^1 \cup \mathcal{A}_1$, maximizes. Update $\mathcal{A}_1 = \mathrm{MC}(e) \cup \mathcal{A}_1$.
6. Choose Δ'_{out} randomly and make faulty decryption queries of the form (N, A, C, T') with $T' = T + \mathsf{MCS} \circ \mathsf{SR}(\Delta_{out}^{-1})$ and expect one forgery. Store T' and the faulty value δ in the list $\mathcal{H}_{PC}$. Continue this step to collect q_{PC} tag forgeries.
7. Increment PC by 1 and repeat steps 4-6.
8. Do offline computation to recover the full f' state, i.e., one branch of GFN.

is $p_j = \frac{\lambda-j+1}{\lambda\times\lambda} = \frac{\lambda-j+1}{\lambda^2}$. Therefore, χ_j follows geometric distribution and $E(\chi_j) = \frac{1}{p_j}$. By the linearity of expectations, we have,

$$\begin{aligned} E(\chi) &= \sum_{j=1}^{q} \frac{\lambda^2}{\lambda - j + 1} \\ &= \frac{\lambda^2}{\lambda} + \frac{\lambda^2}{\lambda - 1} + \cdots + \frac{\lambda^2}{\lambda - q + 1} \\ &< \lambda^2 \cdot \left[1 + \log\left(\frac{\lambda}{\lambda - q + 1}\right)\right] \\ &\left[\log(x+1) < \sum_{i=1}^{x} \frac{1}{i} < 1 + \log(x)\right] \end{aligned}$$

5.4.2.3 State Recovery of Sponge AE

Based on the collected lists $\mathcal{H}_{PC}$ according to the Algorithm 6, we have to recover the $\mathbf{f}^{'}$ state byte/nibble-wise since there is only one difference in $\Delta^{'}_{out}$ (see Figure 5-8). Let, r_{PC} (=1) denote byte/nibble differences in $\Delta^{'}_{out}$. The state recovery of $\mathbf{f}^{'}$ is described in Algorithm 7.

Algorithm 7 State Recovery of $\mathbf{f}^{'}$

1. Initialize a phase counter $PC_1 = 0$.
2. At the PC_1^{th} phase, initialize two lists $\mathcal{L}^{'}_{PC_1}, \mathcal{L}_{PC_1} \leftarrow \emptyset$.
3. Consider the tag T and a faulty tag $T^{'}$ from the list $\mathcal{H}_{PC_1}$ out of q_{PC_1} faulty tags. Compute $\Delta^{'}_{out} = \mathsf{SR}^{'} \circ \mathsf{MCS}^{-1}(\Delta_{out})$, where $\Delta_{out} = T \oplus T^{'}$.
4. If the list $\mathcal{L}_{PC_1} == \emptyset$, goto step 4a, else goto step 5.
   4a. Make a list $\mathcal{L}_{PC_1}$ which contains set of all possible r_{PC_1} byte/nibble values. These r_{PC_1} bytes/nibbles are corresponding to the non-zero differences of $\Delta^{'}_{out}$.
5. For $X \in \mathcal{L}_{PC_1}$, create two states $X, X^{'}(= X \oplus \Delta^{'}_{out})$. Compute the difference $\mathsf{AC}^{-1} \circ \mathsf{SC}^{-1}(X) \oplus \mathsf{AC}^{-1} \circ \mathsf{SC}^{-1}(X^{'})$ and check wheather it is equal to δ or not. If yes, then add X to the list $\mathcal{L}^{'}_{PC_1}$.
7. Update the list $\mathcal{L}_{PC_1} \leftarrow \mathcal{L}^{'}_{PC_1}$ and $\mathcal{L}^{'}_{PC_1} \leftarrow \emptyset$.
8. Repeat steps 3-7 for all elements in $\mathcal{H}_{PC_1}$ until the list $\mathcal{L}_{PC_1}$ have a unique byte/nibble value.
9. Increment the phase counter $PC_1 = PC_1 + 1$.
10. Repeat steps 2-8 to retrieve the remaining byte/nibble values of the $\mathbf{f}^{'}_{\mathbf{a},\mathbf{b},\mathbf{k}}$ state uniquely.

Further, we can recover other branches of $\mathsf{GFN}^{\mathbf{f}'}$ in the similar way that we have discussed in Section 5.4.1.4. Finally, the key can be recovered when either the key is directly used to output the tag by XORing with the state (output of the last permutation) or there are no extra key injections used after the initialization or before the finalization calls.

5.4.2.4 Attack Complexity

According to Theorem 4, To get atleast one successful forgery we need to perform $\mu(=\lambda)$ faulty decryption queries. Let us assume taht, the retrieval of $\mathbf{f}'$ (SPN) state has been done by z phases using Algorithm 6. Now, let $q = q_0 + q_1 + \cdots + q_{z-1}$ $(q_0 = \ldots = q_{z-1})$ represent the total different forging tags to retrieve the $\mathbf{f}'$ state uniquely. Thus, we need at least $z \cdot \lambda$ byte/nibble faults to recover the full $\mathbf{f}$ state uniquely. Again, there are l different branches for the $\mathsf{GFN}^{\mathbf{f}}$ inside the sponge-based AE. Hence, we need approximately $l \cdot z \cdot \lambda$ byte/nibble faults to recover the full $\mathsf{GFN}^{\mathbf{f}}$ (sponge) state.

5.5 Countermeasures

The attacks demonstrated in this work can be thwarted or made more difficult using two approaches. Our DFA attack can recover the state when at least one branch is released as the tag. To counter this, a designer can make a slight modification: instead of releasing a branch as the tag, form the tag by concatenating the partial state of each branch. This way, an attacker only has partial state information of each branch, making state recovery significantly harder. The first approach is to take partial bits from all branches and concatenate them to output the final tag. The second approach is to XOR the master key either after the initialization or before the finalization call to the sponge construction. This makes the backward computation of the sponge harder, even if the sponge state is recovered using DFA in the decryption query.

5.6 Conclusion

This work demonstrates that the CiliPadi family of AE schemes is susceptible to DFA attacks if the final tag directly outputs at least one of the branches in the underlying GFN structure. To perform a DFA on CiliPadi, a faulty forgery needs to be created in the decryption query. Under the random fault model, we describe a state recovery attack on CiliPadi based on the faulty forgery. The time, data, and memory complexities of this attack are respectively $2^{14.5}$, 2^{21}, and $2^{8.5}$ nibbles.

We can recover the key in CiliPadi either by XORing the state with the tag or by going back to the initialization through backward computations. This attack can be extended to any sponge-based AE scheme that employs an underlying permutation using an SPN-based GFN structure. Furthermore, we can also apply this attack to recover the key when extra key masking is used to output the tag, or the extra key masking is not used after initialization and before the finalization calls. This is the first time such an attack has been demonstrated, and it can efficiently perform DFA to recover the state of the sponge-based AE schemes, where the internal permutation of the sponge uses an SPN-based GFN structure. The GFN structure may contain an SPN with either one or more than two rounds. We describe two attacks to recover the state under two different fault models. Finally, we provide general countermeasures against these types of fault attacks.

6 DIFFERENTIAL FAULT ATTACK ON SPN-BASED SPONGE AND SIV-LIKE AE SCHEMES

6.1 Introduction

The Internet of Things [204] (IoT) is the evolution of the Internet in the modern era that builds a network of small objects to connect millions and millions of devices from various platforms. In IoT technology, the two most essential components, RFID (Radio Frequency and Identification) and WSN (Wireless Sensor Networks) are used in several applications such as traffic control and environmental surveillance, home automation, and many more. Practically, these devices have a limited number of gates for security, minimum storage, processing capacity, and limited power consumption. The conventional cryptography methods such as AES (encryption), SHA-256 (hashing) and many more work well within systems (like Servers, Desktops, Tablets, and, smartphones), but they struggle in an IoT/embedded world due to too much processing power, physical space, and consumption of battery power. These algorithms do not fit into a world with embedded systems and sensor networks. Recently, lightweight cryptography (LwC) methods have been proposed to overcome these problems of conventional cryptography, which include constraints related to physical size, processing requirements, memory limitation, and limited power consumption.

Nowadays, a large number of lightweight cryptography primitives have been proposed and used over resource-limited devices. Symmetric key cryptographic algorithms can be classified into Stream cipher, Block cipher, Hash function, Message Authentication

Codes (MAC), and Authenticated ciphers. Some of the well known lightweight block ciphers are PRESENT [65], PRINCE [205], LED [63], SKINNY [206], KATAN & KTANTAN [207], and GIFT [64]. An extensive review of lightweight block ciphers is available in the paper [208]. Also, in [209], the authors have systemized the knowledge to better understand what "lightness" is in the area of lightweight cryptography. Besides, the national (NIST) and international (ISO/IEC) organizations outline several methods for lightweight cryptography, which could also be useful in IoT and RFID devices [210]. In 2019, NIST initiated a process to solicit, evaluate, and standardize lightweight cryptographic algorithms suitable for use in constrained environments. Initially, it received 57 submissions out of which 56 were accepted for Round 1. Then, after the evaluation, 32 candidates were selected for Round 2 from which 10 finalists had recently been announced. Ultimately, Ascon [190] won the competition.

With the emergence of IoT and cloud computing, small-scale devices are becoming more ubiquitous. These devices often perform cryptographic operations, which are vulnerable to device-specific attacks. For example, the sensor nodes in WSN may be deployed in unattended and possibly hostile environments where they may be exposed to (un)intentional circumstances. This is an interesting implication of the IoT setting and hence for the LwC, which leads to a switch focus from the traditional black-box attack model that generally relies on classical cryptanalysis to the gray-box model where an attacker gets access to additional side-channel information. The leakage is passively observed via timing information, power consumption, electromagnetic radiations, etc.. The leaked information has been shown to result in a catastrophic breakdown of security.

One of the most popular attacks in the gray-box model is the Fault Attack (FA) is shown to be powerful against common ciphers. Among them, the Differential Fault Attack (DFA) is possibly the most commonly employed fault technique. It is an extension of conventional differential cryptanalysis but augmented with extra information in the form of the faulty output of a cipher. The NIST LwC Competition presents an interesting premise where it becomes necessary to study the submissions in the light of physical attacks like DFA which forms the main motivation of this work. Moreover, "fault attacks" is one of the specific requirements according to

design considerations in their LwC report in [210], which also motivates us to do this work.

Some other kinds of fault attacks are based on fault models in which the distribution of faults behaves non-uniformly. The non-uniform behavior is usually seen when faults are injected in the device where some faulty values occur more often than the other like a bit reset is more likely than a bit set when overclocking-based injection is used. Some of the effective classes of such biased fault analysis are Statistical Fault Analysis (SFA) [211] and Ineffective Fault Analysis (IFA) [212]. The most recent biased fault attack is the so-called Statistical Ineffective Fault Analysis (SIFA), which can break symmetric key cryptography [32, 33]. It combines the concepts of SFA attacks with that of the Ineffective Fault Analysis (IFA). SIFA exploits ineffective faults, which are those faults which, even if successfully injected, cannot make the output faulty and result in a correct ciphertext. Methodologically, SIFA differs from DFA in that, for DFA, the injected faults need to be effective, and the two states must be identical for faults to be injected into one of the states. Moreover, SIFA becomes a very powerful attack because it can bypass all existing fault attack countermeasures (both detection and infection-based) even while they are combined with Side-Channel (SCA) countermeasures like masking or threshold implementation (TI).

DFA on AE Schemes. An authenticated encryption (AE) is a cryptographic algorithm where it is usually assumed that the nonce N never repeats for encryptions E under the same key K that implicitly protects several classes of fault attacks [213, 26, 28]. In particular, during encryptions, it is almost impossible to perform DFA. Hence, an attacker could not generate the correct and faulty output for the same input. Some schemes claim a certain level of robustness even in misuse settings such as repeated nonces, or release of unverified plaintext. Moreover, in contrast to nonce-based encryption schemes, the decryption procedure (with a fixed nonce) is still susceptible to DFA. Recall the term faulty forgery (see Chapter 5) as the ability of the attacker to produce a plaintext $P \neq P^*$ after inducing faults such that the verification passes. This might lead to retrieving the internal state and then the master key. However, under this scenario, the probability of the attacker producing a faulty forgery is negligible. For sponge-based AE, to produce a state collision (faulty

forgery) by injecting faults at the state during plaintext processing or by choosing a plaintext $P^*(\neq P)$ is improbable. Because injecting fault at the state during plaintext absorption might lead to a negligible probability of forgery. Whereas, for SIV-based AE schemes with the non-empty messages, this kind of state collision never happens due to its two-pass design structure. Another approach to defining faulty forgery would be to inject faults at the last round permutation call just before outputting the final tag. Based on the collected forging tags, we can recover the internal state. Then we can recover the secret key, if no extra key is used in the sponge except at the initialization.

Choosing an internal permutation inside any sponge-based AE can be categorized into three types of structures as SPN, ARX (ADD Rotation XOR), and LRX (Logical-Operations Rotation XOR). In general, the operations in ARX/LRX are word-wise (32/64-bits), whereas it is byte/nibble-wise for the SPN structure. Moreover, another structure called Generalised Feistel Networks (GFN), can be used in the sponge-based AE as an internal permutation. It is evident that some sponge-based AEAD also use GFN as the internal permutations inside it. The round functions of GFN internally used SPN, ARX, or LRX-based structures. For faulty forgery, we usually inject faults at the last rounds of the cipher and randomize $T^{'}$ so that $T^{'}$ becomes a valid forgery. The attacker can choose $T^{'}$ according to the non-zero differences at the last round of the cipher. Thus, if an attacker injects a difference at the last/second-last round in the state, the affected bits for SPN might be much lesser compared to the ARX/LRX-based structures. Moreover, some LRX-based permutations are not invertible. Therefore, faulty forgery will generally have an advantage with SPN designs compared to ARX/LRX designs. This is because, structurally, if a GFN uses an SPN internally and the tag is derived from one of its SPN states, fewer faulty queries will be required to produce a forgery. It is true because, inside the sponge, the state size of SPN inside GFN will be lesser than the full SPN-based permutation. Another advantage is that the state recovery using differential analysis for lesser state-sized SPN is much easier among other structures. A generalized fault attack on SPN-based GFN permutation in the sponge AE can be found in [214].

6.1.1 Summary of The Chapter

This work focuses on the PHOTON-BEETLE [193] AE scheme, which has been selected as one of the ten finalists in the NIST Lightweight Cryptography (LwC) competition. The PHOTON-BEETLE scheme internally utilizes a permutation function called PHOTON, which was originally used to design a lightweight hash function known as PHOTON-hash [215]. The PHOTON-BEETLE scheme employs the BEETLE [189] mode, which is a lightweight sponge-based AE scheme. One key observation in our research is that the internal permutation used in the PHOTON-BEETLE scheme exhibits a structure similar to a substitution-permutation network (SPN). This observation motivates us to explore the possibility of performing a differential fault attack (DFA) in the decryption query and subsequently recovering the internal state by leveraging differential analysis techniques. Since the key is used to initialize the PHOTON-BEETLE state, we can recover the secret key by inverting the permutations after retrieving the internal state. Building upon this finding, we extend our analysis to other NIST LwC schemes, namely ORANGE [194], SIV-TEM-PHOTON [216], and ESTATE [217]. Although ORANGE and ESTATE were selected as round-2 candidates in the NIST LwC standardization process, they did not make it to the finalist stage. Meanwhile, the SIV-TEM-PHOTON scheme was reached to the round-1 candidate stage. For all of these schemes, we apply the faulty forgery strategy in the decryption query and recover their respective keys by retrieving the internal state based on the collected forgeries.

To perform the faulty forgery in the decryption query, we propose a forging strategy under three different fault models. The first model is the Random Fault Model, where random nibble/byte faults are injected in the second last round before the tag is produced. In the second model, the Random Bit-flip Fault Model, random bit faults are injected in the last round during the final permutation call. The third model, the Known Fault Model, assumes that the faults are random but known to the attacker. For each of these fault models, we provide the attack complexities and validate the attacks using software platforms. Additionally, we conduct a theoretical analysis to estimate the number of faults required to reduce the keyspace to a practically acceptable level. We summarize these results in Table 6.1. Furthermore, we

Table 6.1: Fault Attacks on Four Schemes PHOTON-BEETLE, ORANGE, SIV-TEM-PHOTON, and ESTATE reported in this work

Fault Model	**Schemes**	**Expected # Faults**	**Reduced Key-Space**	**Reference**
RFM	PHOTON-BEETLE	$2^{37.15}$	2^{0}	Section 6.3.1
RBFM	PHOTON-BEETLE	$2^{11.5}$	2^{20}	Section 6.3.2
	ORANGE	$2^{11.5}$	2^{20}	
	SIV-TEM-PHOTON	$2^{11.5}$	2^{20}	
	ESTATE	$2^{13.1}$	2^{5}	
KFM	PHOTON-BEETLE	$2^{11.05}$	2^{0}	Section 6.3.3
	ORANGE	$2^{11.05}$	2^{0}	
	SIV-TEM-PHOTON	$2^{11.05}$	2^{0}	
	ESTATE	$2^{13.01}$	2^{0}	
KBFM	PHOTON-BEETLE	$2^{9.32}$	2^{0}	Section 6.3.3.6

Table 6.2: The PHOTON S-box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S-box	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

propose countermeasures specifically designed for SPN-based sponge AE schemes such as PHOTON-BEETLE and ORANGE. These countermeasures aim to increase the difficulty of retrieving the master key through our proposed attacks, thereby enhancing the overall security of these schemes.

Overall, this work makes significant contributions to the field of cryptanalysis by demonstrating the vulnerability of SPN-based sponge AE schemes to differential fault attacks. It provides insights into the attack complexities, key recovery processes, and potential countermeasures for various lightweight cryptographic schemes, ultimately contributing to the overall understanding of cryptographic security in the context of lightweight designs.

6.2 Notations and Cipher Specifications

Let us first introduce a set of notations that will be used consistently throughout the chapter. These are as follows.

– A PHOTON state S of 64 nibbles (i.e., 256 bits) can be viewed as 8×8 matrix,

i.e., $S = \left[s_{u,v}\right]_{8\times 8}, 0 \leq u, v < 8$.

- $|Z|$ denotes the cardinality of Z, where Z may be a set, list or a string.
- $Trunc(Z, u)$ is a function that returns the most significant u bits of Z.
- $Pad_r(Z) = Z||1||0^{r-|Z|-1}$, i.e., Pad denotes the padding function that applies 10^* padding on r bits.
- For any round in the PHOTON permutation, AC, SC, SR, MCS represent the AddConstants, SubCells, ShiftRows, MixColumnSerial operations respectively.
- Any PHOTON permutation is composed 12 rounds, i.e., $\sigma_{11} \circ \sigma_{10} \circ \cdots \circ \sigma_0$, where $\sigma_i = \mathsf{MCS} \circ \mathsf{SR} \circ \mathsf{SC} \circ \mathsf{AC}$.
- The last permutation call to the PHOTON-BEETLE primitive is denoted by LPHOTON, i.e., LPHOTON is the last permutation call to output the tag from its rate part.
- For $0 \leq u, v \leq 8$, let $\delta_{u,v}$ denotes the fixed nibble difference in the state S and $\Delta S_{u,v}$ is its corresponding differential state. For example, the differential state $\Delta S_{0,0}$ and its output corresponding to the following operations $\mathsf{AC}, \mathsf{SC}, \mathsf{SR}, \mathsf{MCS}$ are given in Figure 6-1.
- Any state difference ΔS can be written as $\Delta S = (\Delta C_0, \Delta C_1, \cdots, \Delta C_7)$, where ΔC_u represents the u-th column difference of ΔS. For example, $\Delta S_{0,0} = (\Delta C_0, 0, \cdots, 0)$.
- For a given u $(0 \leq u < 8)$ and a differential state Δ, we define $Col(u, \Delta)$ is a function which extracts the u-th column from Δ. E.g., $Col(0, \Delta S_{0,0}) = (\delta_{0,0}, 0, \cdots, 0)^\tau$, where τ is the matrix transpose.
- Throughout this chapter, faulty forgery means an attacker will inject a fault at the last rounds of the permutation call and choose a random T' such that the tag verification will succeed.

6.2.1 PHOTON Permutation

The PHOTON permutation [215] is first introduced to design a family of the lightweight hash function called PHOTON-hash function. It is a family of unkeyed AES-like

functions with five instances (denoted by P_t) corresponding to the state sizes $t = 100, 144, 196, 256$ and 288 bits respectively. In PHOTON-BEETLE AEAD mode, P_{256} is used as the underlying 256-bit permutation. It can be viewed as the state of 64 nibbles, which is represented as 8×8 matrix. P_{256} has 12 rounds with each round composed of four operations: AddConstants (AC), SubCells (SC), ShiftRows (SR), MixColumnSerial (MCS). Informally, AddConstants adds fixed constants to the nibbles of the internal state. SubCells applies a 4-bit S-box (see Table 6.2) to each of the 64 nibbles individually. ShiftRows rotates the position of the nibbles in each of the rows of the state matrix, i.e., for $0 \leq i < 8$, the i^{th} row will be rotated by i positions to the left and MixColumnSerial linearly mixes all the columns independently using serial matrix multiplication. In P_{256}, the MixColumnSerial matrix μ is a MDS matrix to achieve maximal diffusion, and hence it is invertible. A matrix is MDS if and only if every square sub-matrices of it are nonsingular. These matrices are used to provide perfect diffusion in block ciphers and hash functions. Both of μ, μ^{-1} are given in Figure 6-2. All the details of P_{256} and its formal operations are depicted in Figure 6-3.

$$\Delta S_{0,0} = \begin{pmatrix} \delta_{0,0} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ & & \vdots & \\ 0 & 0 & \cdots & 0 \end{pmatrix} \mathsf{MCS}(\Delta S_{0,0}) = \begin{pmatrix} \delta^0_{0,0} & 0 & \cdots & 0 \\ \delta^1_{1,0} & 0 & \cdots & 0 \\ & & \vdots & \\ \delta^7_{7,0} & 0 & \cdots & 0 \end{pmatrix} \mathsf{AC}, \mathsf{SR}/\mathsf{SC}(\Delta S_{0,0}) = \begin{pmatrix} \delta_{0,0}/\delta'_{0,0} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ & & \vdots & \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Figure 6-1: Differential States with respect to PHOTON Operations

$$\mu = \begin{pmatrix} 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \\ 12 & 9 & 8 & 13 & 7 & 7 & 5 & 2 \\ 4 & 4 & 13 & 13 & 9 & 4 & 13 & 9 \\ 1 & 6 & 5 & 1 & 12 & 13 & 15 & 14 \\ 15 & 12 & 9 & 13 & 14 & 5 & 14 & 13 \\ 9 & 14 & 5 & 15 & 4 & 12 & 9 & 6 \\ 12 & 2 & 2 & 10 & 3 & 1 & 1 & 14 \\ 15 & 1 & 13 & 10 & 5 & 10 & 2 & 3 \end{pmatrix} \quad \mu^{-1} = \begin{pmatrix} 4 & 7 & 9 & 10 & 12 & 12 & 3 & 15 \\ 13 & 13 & 10 & 10 & 7 & 13 & 10 & 7 \\ 14 & 2 & 3 & 14 & 4 & 10 & 5 & 11 \\ 5 & 4 & 7 & 10 & 11 & 3 & 11 & 10 \\ 7 & 11 & 3 & 5 & 13 & 4 & 7 & 2 \\ 4 & 15 & 15 & 6 & 1 & 14 & 14 & 11 \\ 5 & 14 & 10 & 6 & 3 & 6 & 15 & 1 \\ 2 & 1 & 12 & 1 & 4 & 11 & 3 & 9 \end{pmatrix}$$

(a) MixColumnSerial Matrix (b) Inverse MixColumnSerial Matrix

Figure 6-2: Matrix form μ, μ^{-1}

6.2.2 Photon-Beetle **AEAD**

Linear Functions ρ and ρ^{-1}. ρ is a linear function that receives two inputs: a state $S \in \{0,1\}^r$ and, an input data $U \in \{0,1\}^{\leq r}$. It produces an output data $V \in \{0,1\}^{|U|}$. ρ^{-1} is the inverse function of ρ, which takes the state S and the output data V to reproduce the input data U and update the state. The full description of

Algorithm 1: $\text{PHOTON}_{256}(X)$

for $i = 0$ *to* 11 **do**
  $X \leftarrow \text{AddConstant}(X, i)$;
  $X \leftarrow \text{SubCells}(X)$;
  $X \leftarrow \text{ShiftRows}(X)$;
  $X \leftarrow \text{MixColumnSerial}(X)$;
return X;

Algorithm 2: $AddConstant(X, k)$

$RC[12] \leftarrow \{1, 3, 7, 14, 13, 11, 6, 12, 9, 2, 5, 10\}$;
$IC[8] \leftarrow \{0, 1, 3, 7, 15, 14, 12, 8\}$;
for $i = 0$ *to* 7 **do**
  $X[i, 0] \leftarrow X[i, 0] \oplus RC[k] \oplus IC[i]$;
return X;

Algorithm 3: $MixColumnSerial(X)$

$\mu \leftarrow Serial[2, 4, 2, 11, 2, 8, 5, 6]$;
$X \leftarrow \mu^8 \odot X$;
return X;

Algorithm 4: $SubCells(X)$

for $i, j = 0$ *to* 7 **do**
  $X[i, j] \leftarrow \text{SC}(X[i, j])$;
return X;

Algorithm 5: $ShiftRows(X)$

for $i, j = 0$ *to* 7 **do**
  $X'[i, j] \leftarrow X[i, (i + j)\%8]$;
return X';

Algorithm 6: $\rho(S, U)$

$V \leftarrow Trunc(Shuffle(S), |U|) \oplus U$;
$S \leftarrow S \oplus Pad_r(U)$;
return (S, V);

Algorithm 7: $Shuffle(S)$

$S_1 || S_2 \xleftarrow{r/2} S$;
return $S_2 || (S_1 \ggg 1)$;

Algorithm 8: $\rho^{-1}(S, V)$

$U \leftarrow Trunc(Shuffle(S), |V|) \oplus V$;
$S \leftarrow S \oplus Pad_r(U)$;
return (S, U);

Figure 6-3: PHOTON_{256} Permutation

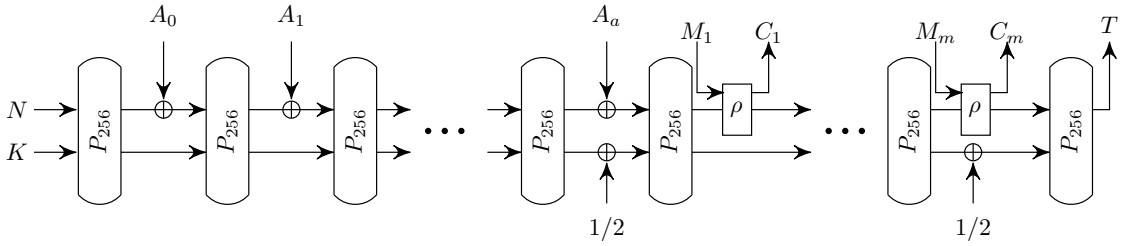


Figure 6-4: PHOTON-Beetle

ρ and ρ^{-1} can be found in Figure 6-3.

Specification. PHOTON-BEETLE [193] is a BEETLE [189] family of lightweight sponge-based authenticated encryption (AE) instantiated with the PHOTON permutation P_{256}. It take inputs as: an encryption key $K \in \{0,1\}^{128}$, a nonce $N \in \{0,1\}^{128}$, an associated data $A \in \{0,1\}^*$ and a message $M \in \{0,1\}^*$ and returns a ciphertext $C \in \{0,1\}^{|M|}$, and a tag $T \in \{0,1\}^{128}$. Similarly, the decryption algorithm of PHOTON-BEETLE take inputs as: a key $K \in \{0,1\}^{128}$, a nonce $N \in \{0,1\}^{128}$, an associated data $A \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$, and a tag $T \in \{0,1\}^{128}$ as inputs. It returns the plaintext $M \in \{0,1\}^{|C|}$ with respect to C if the tag T is verified.

An initial state of PHOTON-BEETLE is generated by simple concatenation of the nonce N and the master key K. Then, it absorbs the associated data A identical to the original sponge mode fashion, i.e., at each step the state is updated using P_{256} permutation and the first r output bits (i.e., the rate part) of the permutation is XORed with the next associated data block to produce the input (rate part) for the next permutation call. Then, it absorbs the message M to squeeze the ciphertext C in the following way.

1. Shuffle the rate part of the state using ρ after the permutation call P_{256}.
2. Then, XOR it with the corresponding message block.

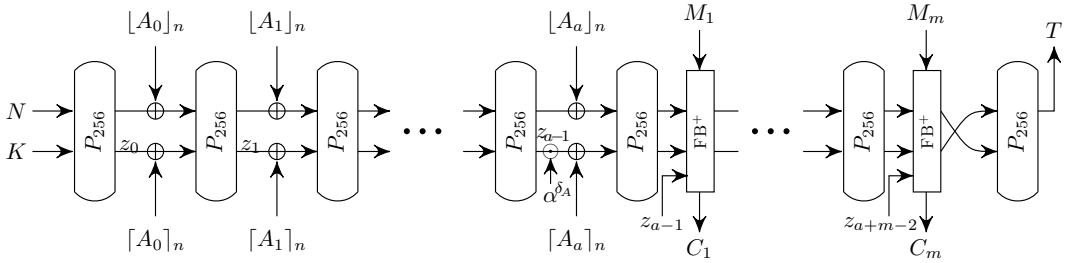
The first step differentiates this BEETLE mode from the Sponge Duplex mode [67]. Both the state update and the ciphertext generation during the message processing are handled by the function ρ. In the decryption, the state update and the message block computation using the ciphertext blocks are mixed up by ρ^{-1}. Also, 3-bit constants are added in the capacity part after the associated data and message processing for domain separation. Description of PHOTON-BEETLE encryption with non-empty associated data and message is depicted in Figure 6-4. Other descriptions for empty associated data and/or empty message processing are given in [193].

6.2.3 ORANGE AEAD

ORANGE [194] is a PHOTON permutation-based sponge-like AE scheme. This mode is very similar to the duplex construction except that it uses the full b-bit state

absorption and squeezes the full state during plain/cipher-text processing. Here, we only provide a summary of the details of ORANGE-ZEST encryptions for two different cases $|A| = |M| = 0$ and $|A| \neq 0, |M| = 0$ which are needed to understand our attack. In Figure 6-5, we have depicted the working principle of ORANGE-ZEST encryption for different cases of associated data and message processing. For processing the message, the feedback function FB^+ is used to absorb the full state-size message block and outputs the ciphertext. Whereas for the associated data process, each intermediate data blocks (of full state size) are absorbed by XORing with the state following a permutation call P_{256}. In the last associated data block process, first, α^{δ_A} multiplication[1] is applied to the most significant half of the state and then padding the last data block and XORed it with the state. Here, α^{δ_A} takes 1 or 2 according to the full or partial-sized block. More details about ORANGE can be found in [194].

[1] $\alpha.x$ denotes the α multiplication of an n-bit string $x = x_{n-1} \cdots x_1 x_0$. For $n = 128$, $\alpha \cdot x$ is defined as $(x \ll 1) \oplus 0^{120}10000111$, if $x_{127} = 1$ and $(x \ll 1)$, otherwise. Further, $\alpha^{\delta_A} \cdot x$ denotes δ_A times repeated α-multiplication of x.



(a) $|A| = |M \neq 0, |A_i| = 2n$

(b) $|A| \neq 0, |M| = 0, |A_i| = 2n$

(c) $|A| = |M| = 0$

Figure 6-5: Different Cases of ORANGE-ZEST Encryption

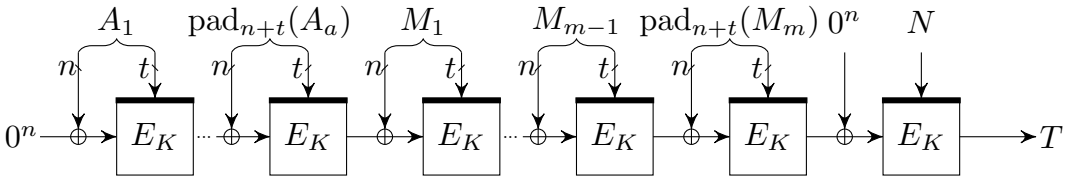
6.2.4 SIV-TEM-PHOTON AEAD

SIV-TEM-PHOTON [216] is a SIV construction which utilizes PHOTON-based tweakable block cipher, named TEM-PHOTON with 128-bit key and 128-bit tweak as the underlying building block. The SIV scheme [218] is a combination of an encryption scheme $\mathcal{E}$ and a pseudo-random function $\mathcal{F}$ to obtain an AEAD scheme. The encryption and decryption algorithms are given in Figure 6-7. TEM-PHOTON $E : \{0,1\}^k \times (\mathcal{I} \times \{0,1\}^t) \times \{0,1\}^n \rightarrow \{0,1\}^n$ is the underlying PHOTON-based tweakable block cipher (TBC), where $k = 128$ is the key length, $\mathcal{I}$ is the domain separation space, $t = 128$ ($\in \mathcal{T}$) is the tweak length, and $n = 256$ is the block length. The round function R_i of the original PHOTON P_{256} is defined as

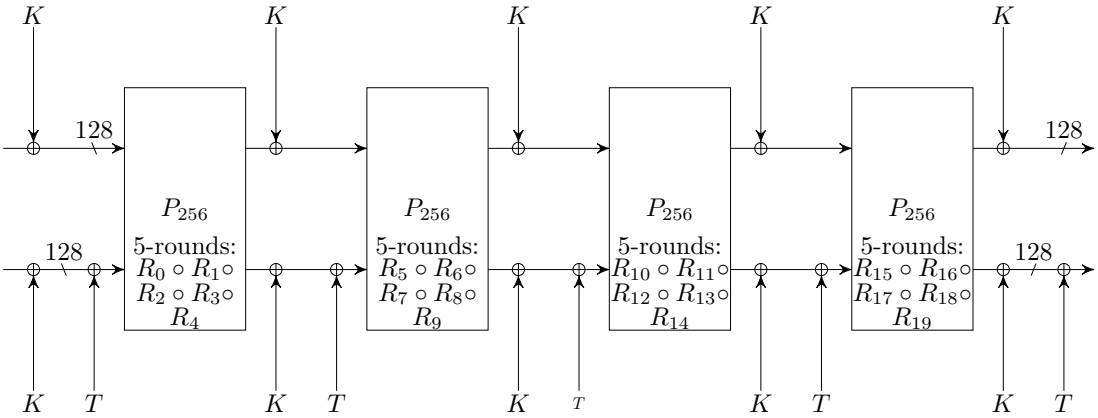
$$R_i = \mathsf{MixColumnSerial} \circ \mathsf{ShiftRows} \circ \mathsf{SubCells} \circ \mathsf{AddConstants}, i = 1, \cdots, 12.$$

Whereas for TEM-PHOTON, the round function is defined as

$$\mathcal{R}_i = \mathsf{MixColumnSerial} \circ \mathsf{ShiftRows} \circ \mathsf{SubCells} \circ \mathsf{AddDomain} \circ \mathsf{AddConstants}, i = 1, \cdots, 20.$$



(a) SIV-TEM-PHOTON



(b) TEM-PHOTON

Figure 6-6: SIV-TEM-PHOTON Encryption with TBC TEM-PHOTON

The AddDomain operation XORs the domain separator d to all cells of the second column at each round. The design of TBC TEM-PHOTON is depicted in Figure 6-6b. Also, the description of $\mathcal{F}$ with the last message block of size $\leq n+t$ to generate the tag $T, |T| = 256$ is presented in Figure 6-6a. More details about SIV-TEM-PHOTON can be found in [216].

6.2.5 ESTATE AEAD

ESTATE [217] is a tweakable block cipher based AE scheme based on the SIV paradigm [218]. Here the size of the key, nonce, and tag are 128 bits each. Also, this mode can be viewed as the tweakable block cipher-based variant of SUNDAE [219]. The designers have proposed two dedicated tweakable block ciphers TweGIFT-128 and TweAES-128 to instantiate the mode. In this chapter, we only discuss the designer's primary recommended scheme ESTATE-TweAES-128 which is instantiated by the ESTATE mode of operation with TweAES-128 block cipher. TweAES-128 is a 128-bit tweakable block cipher with 4-bit tweak and 128-bit key. It is identical to AES-128-128/128 except for the injection of a tweak value at intervals of 2 rounds. The pictorial description of ESTATE with associated data blocks and empty message is given in Figure 6-8. More details about the specification of ESTATE can be found in [217].

6.3 Differential Fault Based Forgery Attack

To recover the secret key of the NIST LwC schemes Photon-Beetle, ORANGE, SIV-TEM-PHOTON, and ESTATE, the attacker injects faults in the state during the decryption process and observes the resulting tag values. The attacker then uses

Algorithm 8 $\textbf{SIV.Enc}_K(N, A, M)$

1. $T \leftarrow \mathcal{F}_K(N, A, M)$;
2. $C \leftarrow \mathcal{E}_K^T(M)$;
3. *return* (C, T);

Algorithm 9 $\textbf{SIV.Dec}_K(N, A, C, T)$

1. $M \leftarrow \mathcal{D}_K^T(C)$;
2. $T^* \leftarrow \mathcal{F}_K(N, A, M)$;
3. *if* $T = T^*$ then return M;
4. *else return* $\perp$;

Figure 6-7: The encryption and decryption algorithms of SIV scheme

these tag values to recover the state and eventually the secret key. There are three different fault models that the attacker can use to inject faults into the state. In the first two models, faults are randomly injected at the intermediate rounds of the state during the tag generation call. In the third model, faults are injected at the last round in the state, but they are random and known to the attacker. For PHOTON-BEETLE and ORANGE, the tag generation process is similar (see Figure 6-9a), and the final tag value T is extracted from the rate part of the state. However, for SIV-TEM-PHOTON (see Figure 6-9b) and ESTATE, the final tag value is the state after the last TEM-PHOTON and TWEAES-128 permutation calls, respectively.

The attacker will use the same strategy to perform faulty forgery on all four schemes. The only difference is that for the PHOTON-BEETLE scheme, each query will consist of associated data blocks and a non-empty message, while for the other schemes, each query will consist of associated data blocks and an empty message. By inverting the rounds and comparing the resulting differences with the injected faults, the attacker can recover the state from the collected forgeries. Once the state is recovered, the attacker can use it to compute the secret key. The approach to recovering the state and secret key will be the same for all four schemes. Therefore, we will first provide a detailed discussion of how to perform fault-finding on the PHOTON-BEETLE scheme, and then we will apply similar techniques to the other three schemes: ORANGE, SIV-TEM-PHOTON, and ESTATE. After that, we will describe the process of state recovery for the PHOTON-BEETLE scheme based on the collected forgeries, and the same approach will be applied to recover the state of the other three schemes.

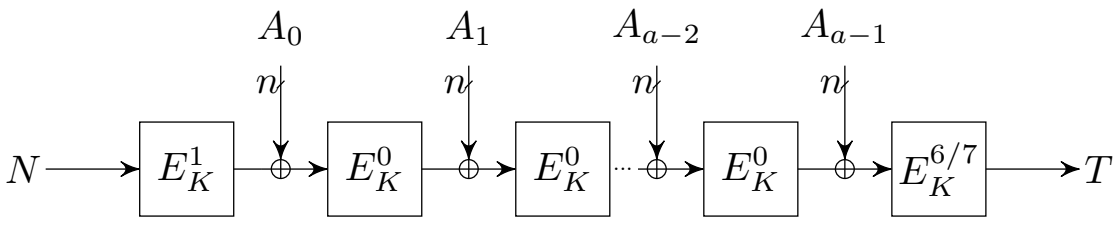


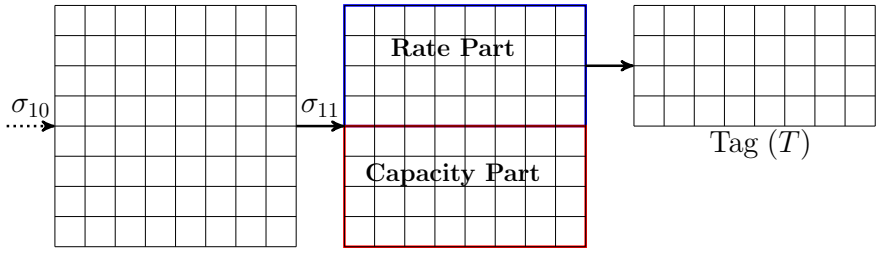
Figure 6-8: ESTATE with Associated Data Blocks and Empty Message

6.3.1 Random Nibble Fault Model

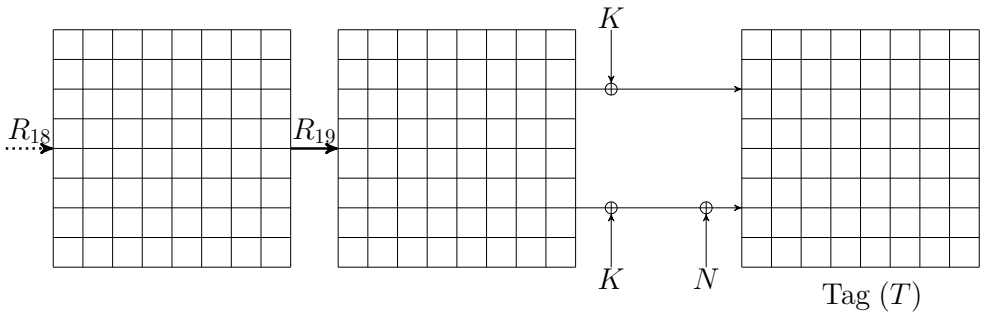
In this section, we first give a fault model, then discuss the state recovery using faulty forgery. Also, we give a theoretical estimation of faulty forgery. Finally, we validate this attack by implementing it in software.

6.3.1.1 The Fault Model

In this particular fault model, an attacker can induce a fault by changing one of the four-bit nibbles in the internal state of a cryptographic primitive to a random value. This can be accomplished using various techniques such as a clock glitch, voltage glitch, electromagnetic pulse (EMP), light pulse, or laser beam. For instance, an attacker may attempt to create a single-nibble fault by applying a clock glitch to the input of a particular round with a particular probability. Furthermore, attackers may use other methods such as voltage glitching, EMPs, light pulses, or laser beams to create faults at the input of a specific round. When a single-nibble fault is introduced, it can significantly impact the security of the cryptographic primitive. For example, it



(a) Tag Extraction of PHOTON-BEETLE and ORANGE



(b) Tag Extraction of SIV-TEM-PHOTON

Figure 6-9: The Last PHOTON Permutation Call before the Tag Extraction

could cause the primitive to produce incorrect output, which could then be exploited by an attacker to recover sensitive information like secret keys or plaintext.

6.3.1.2 The Fault Attack Description

In the decryption query of PHOTON-BEETLE, we make a faulty forgery by repeatedly inducing nibble faults at the last PHOTON permutation (LPHOTON) call and choose a random $T^{'} = T \oplus \Delta$. Precisely, we induce nibble faults at the second last round (before the MixColumnSerial operation) of LPHOTON in the decryption query.

A short description of forging and state recovery attacks is as follows. At the i^{th} $(0 \leq i < 8)$ phase, we make an encryption query (N, A, M) to get (C, T). We choose a nibble position (j, i) inside the LPHOTON state. Due this nibble difference $\Delta^{in}(= \Delta S_{j,i})$ at the 11^{th} round of LPHOTON, the output difference after the last (12^{th}) round SC will be of the form $\mathsf{MCS} \circ \mathsf{AC} \circ \mathsf{SC}(\Delta^{in}) = (0, \cdots, 0, \Delta C_i, 0, \cdots, 0) = \Delta_i^{out}$. Now, the total number of chosen differences of ΔC_i is 2^{32}. Therefore, we need to perform 2^{32} decryption queries as $(N, A, C, T^{'})$ to expect one faulty forgery with a high probability. After collecting one faulty forgery, we repeat the process by injecting faults at the same nibble position until we collect q_i different tag forgeries T' with high probability. Since the last round operations SR and MCS do not affect state recovery, our objective is to uniquely recover the nibbles corresponding to ΔC_i (i.e., Δ_i^{out}). We can accomplish this by guessing those nibbles, inverting them up to the second last round's MCS^{-1} operation, and then checking whether it leads to a single nibble difference at the position (j, i). To collect several faulty forgeries, the attacker performs multiple encryption and decryption queries, and the steps for collecting these faulty forgeries are given in Algorithm 10.

For better understanding, we explain the above algorithm only for the 0^{th} phase and the remaining phases will work similarly. In this phase, we first fix the nibble position as $(0, 0)$, i.e., faults will be injected at the $(0, 0)^{th}$ position (any nibble position in the first column) of the second last round of LPHOTON permutation in the decryption query. Let, Δ^{in} be the input difference due to a fault injected at $(0, 0)^{th}$ position, i.e., $\Delta^{in} = \Delta S_{0,0}$. This input difference Δ^{in} now propagates to the final tag. Let Δ^{out} be the final state difference after the last round σ_{11} of LPHOTON[1]. The final tag T is the

rate part of this LPhoton permutation. Thus, we can write $T \oplus T^{'} = Rate(\Delta^{out})$, where $T^{'}$ is a valid forgery. This scenario is depicted in Figure 6-10.

In this figure, to forge, we have to randomly choose a $Rate(\Delta^{out})$ in each decryption query. As $Rate(\Delta^{out}) = Rate(\mathsf{MCS} \circ \mathsf{SR}(\Delta_0^{out}))$, we have to randomly choose Δ_0^{out} in each decryption query instead of choosing $Rate(\Delta^{out})$. Continue this process until we collect q_0 forgeries and save them in the list $\mathcal{H}_0$. In the i^{th} phase, choose a nibble position from the i^{th} column and proceed similarly according to the Algorithm 10. Each list $\mathcal{H}_i$ contains q_i forging tags. This same strategy as discussed in Algorithm 10 can be applied to ORANGE scheme to collect faulty forgery. But, in this case, we will consider the message (/ciphertext) to be empty, i.e., $|M| = |C| = 0$. This is because we can not invert the feedback function (FB^{+}) although we recover the final LPhoton's state. Similarly, the same strategy as discussed in Algorithm 10 can be applied with empty message query for other schemes SIV-TEM-PHOTON and ESTATE.

Now the immediate question is how many number of faulty decryption queries are required to get at least one forgery. We estimate the number of required faulty decryption queries to get q_i tag forgeries. The following theorems give an estimation of the number of faults to make forgeries and finally, show that around $2^{34.15}$ faulty queries are necessary to produce $q_i(= 4)$ tag forgeries.

Theorem 5. For Photon-based AE schemes (Photon-Beetle, ORANGE, and SIV-TEM-PHOTON), let χ denote the faulty queries to collect a successful forgery at the i^{th} phase in Algorithm 10. Then, $E(\chi) = 2^{32}$.

Proof. To make a valid forgery (according to Algorrithm 10), we have to satisfy this condition: $\mathsf{SC} \circ \mathsf{MCS}(\Delta^{in}) = \Delta_i^{out}$. Now, for any phase i, $\Pr[\mathsf{SC} \circ \mathsf{MCS}(\Delta^{in}) = \Delta_i^{out}] = \frac{2^4-1}{(2^4-1)\times(2^{32}-1)} \approx \frac{1}{2^{32}} = p$.

Let χ denote the trials to get a success, i.e., a valid forgery. So, for each trial, $\Pr[success] = p$ and $\Pr[failure] = q = 1 - p$. Therefore,

$$\Pr[\chi = j] = (1-p)^{j-1} \cdot p = (q)^{j-1} \cdot p.$$

[1]Note that the tag difference is the rate difference of the state, but we can always map the tag difference to its corresponding full state difference.

It shows that χ follows a geometric distribution with probability p. Hence, $E(\chi) = \frac{1}{p}$. □

Theorem 6. For PHOTON-based AE schemes, let χ denote faulty decryption queries to collect q different forgeries (i.e., to retrieve the i^{th} column uniquely by offline computation) at the i^{th} phase in Algorithm 10. Then, the expected faulty queries to collect q distinct forgeries is less than $2^{36} \cdot \left[1 + \log\left(\frac{2^4}{2^4-q+1}\right)\right]$.

Proof. For $1 \leq j \leq q$, let χ_j be the trials needed to collect j^{th} forgery after $j-1$ forgeries have been collected. Since χ represents the independent trials to collect q distinct successful forgeries, we can write $\chi = \chi_1 + \cdots + \chi_q$. Observe that the probability of collecting j^{th} forgery is $p_j = \frac{2^4-j+1}{2^4 \times 2^{32}} = \frac{2^4-j+1}{2^{36}}$.

Therefore, χ_j has geometric distribution with expectation $\frac{1}{p_j}$. By the linearity of expectations, we have,

$$\begin{aligned} E(\chi) &= \sum_{i=1}^{q} E(\chi_i) = \sum_{j=1}^{q} \frac{2^{36}}{2^4 - j + 1} \\ &= 2^{36}\left(\frac{1}{2^4} + \frac{1}{2^4-1} + \cdots + \frac{1}{2^4-q+1}\right) \\ &< 2^{36} \cdot \left[1 + \log\left(\frac{2^4}{2^4-q+1}\right)\right]. \end{aligned}$$

□

6.3.1.3 State Recovery of Photon-based AE schemes

According to Algorithm 10, at the i^{th} $(0 \leq i \leq 7)$ phase, take a list $\mathcal{H}_i$ which store the fault position as well as the forging (PHOTON-BEETLE) state differences Δ^{out}. Now at this phase, computing $\mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$ leads to Δ_i^{out}, a differential state only have the differences at the i^{th} column (see Figure 6-10). Also, Compute all state differences $\Delta' = \mathsf{MCS}(\Delta^{in})$ corresponding to all 2^4 values of Δ^{in} in PHOTON-BEETLE state and store them in a list $\mathcal{D}$. The formal approach to recover the state is as follows. First, Take all possible (i^{th}) column values and form two states $X, X' = X \oplus \Delta_i^{out}$. Then, invert the states X, X' up to the second round MCS and check whether their difference is in $\mathcal{D}$ or not. If yes, then store the possible values to the list $\mathcal{L}$. Repeat

this for all q_i Δ^{out} in the list $\mathcal{H}_i$ to get a unique (i^{th}) column value, i.e., recover the actual column (i^{th}) value of PHOTON-BEETLE state after the last round SC during the decryption process. In this way, other columns can be recovered from the lists

Algorithm 10 Forging Strategy

Input: Associated data A and Plaintext M.
Output: A List $\mathcal{H}_i$ and a fixed number q_i, $0 \leq i \leq 7$.

1. Make an encryption query (N, A, M) and get (C, T).
2. Choose the 11^{th} round σ_{10} at the last permutation call LPHOTON of the PHOTON-BEETLE sponge function where the nibble faults will be injected before the MCS operation. It is denoted by $\sigma_{10}^{\text{LPHOTON}}$. Initialize a phase counter $i = 0$.
3. For $i = 0$ to 7:
   4. At the i^{th} phase, fix a nibble position (j, i)[1] at the $\sigma_{10}^{\text{LPHOTON}}$ state, where faults will be injected repeatedly at this state position. Therefore, the input state difference will be $\Delta^{in} = \Delta S_{j,i}$ and MCS (Δ^{in}) gives a i^{th} column difference.
   5. For each decryption query as $(N, A, C, T^{'})$:
      5a. Randomly fill the i^{th} column difference to form Δ_i^{out} (see Figure 6-10) and compute $\Delta^{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta_i^{out})$.
      5b. Make $T^{'} = T + Rate(\Delta^{out})$ and check whether $T^{'}$ is a valid forgery or not? If this happens, then store both the fault position (j, i) and final state difference Δ^{out} in the list $\mathcal{H}_i$.
      5c. Continue this until we collect q_i tag forgeries.
7. Do offline computation to recover the full LPHOTON state, i.e., the PHOTON-BEETLE state.

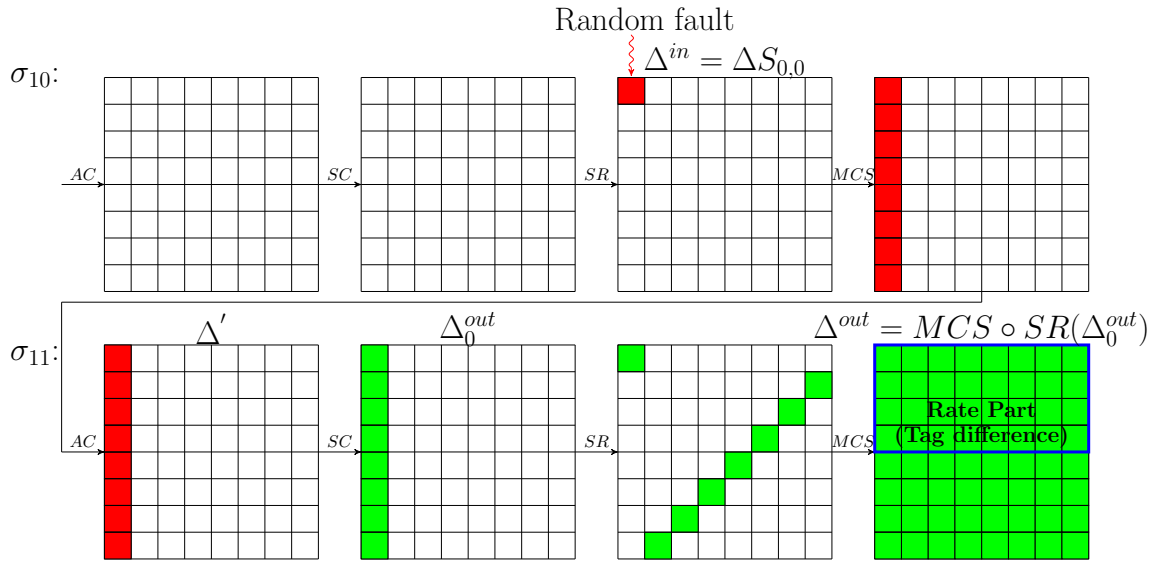


Figure 6-10: Faulty Forgery: Inducing Faults at the Second Last Round

$\mathcal{H}_i$. Thus, the time complexity of this approach will be greater than 2^{32}. Now, the question is whether we can improve the time complexity. The time complexity can be enhanced by processing columns nibble-wise instead of processing the entire column at once. This approach, detailed in Algorithm 11, is used to recover the full state of PHOTON-BEETLE more efficiently. In the same fashion, we can recover the states of ORANGE and SIV-TEM-PHOTON schemes. The same approach can also be applied to the AES-based scheme ESTATE to recover its state.

6.3.1.4 Attack Complexity

We verify that $\forall i \in \{0, 1, \cdots, 7\}$, $q_i = 4$ is sufficient to recover the i^{th} column uniquely. To retrieve the i^{th} column uniquely, the attacker will have to perform approximately $2^{34.15}$ faulty decryption queries (according to Theorem 6) to collect $q_i(= 4)$ forgeries. Therefore, the total faulty decryption queries to retrieve the state is $8 \times 2^{34.15} = 2^{37.15}$. Now, to estimate the complexity of the above state recovery attack,

Algorithm 11 State Recovery of PHOTON-based AEAD

Input: List $\mathcal{H}_{i_1}, 0 \leq i_1 < 8$.
Output: A PHOTON state $st[8][8]$.

1. Initialize a phase counter $i_1 = 0$. Initialize several lists $\mathcal{L}_{i_1,j,l} \leftarrow \emptyset$, where $0 \leq i_1, j < 8$ and $0 \leq l < |\mathcal{H}_{i_1}|(= q_{i_1})$.
2. For each nibble fault position (j, i_1) and the final state difference Δ^{out} in the list $\mathcal{H}_{i_1}$:
   - 2a. Compute all 2^4 possible i_1^{th} column differences at the output of $\Delta^{'} = \mathsf{MCS}(\Delta^{in})$, where $\Delta^{in} = \Delta S_{j,i}$ has a nibble hamming weight of 1. Store them in a list $\mathcal{D}$. (Let us denote these i_1^{th} column differences as $\Delta_0^{'}, \Delta_1^{'}, \cdots, \Delta_{15}^{'}$, i.e., $Col(i, \Delta_m^{'}) = (\delta_0, \delta_1, \cdots, \delta_7)^{\tau}$, where $m \in \{0, 1, \cdots, 15\}$ and τ is the matrix transpose.)
   - 2b. Compute $\Delta_{i_1}^{out} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$. This $\Delta_{i_1}^{out}$ is a i_1^{th} column difference, i.e., $Col(i, \Delta_i^{out}) = (\delta_0^{'}, \delta_1^{'}, \cdots, \delta_7^{'})^{\tau}$, $m \in \{0, 1, \cdots, 15\}$.
   - 2c. For each (2^4 number of) i_1^{th} column differences $\Delta^{'}$:
     - 2c.1. Check $\forall\, p \in \{0, 7\}$, $\mathsf{DDT}[\delta_p][\delta_p^{'}] > 0$ or not? If yes, then do the following:
     - 2c.1.1. For each nibble position j in the i_1^{th} column:
     - 2c.1.1.1. For each $x \in \{0, 1, \cdots, 15\}$:
     - 2c.1.1.1.1. Store x in the list $\mathcal{L}_{i_1,j,l}$ if $S(x) \oplus S(x \oplus \delta_j) == \delta_j^{'}$ holds.
   - 2d. For each nibble position j in the i_1^{th} column:
     - 2d.1. Compute $\cap_l \mathcal{L}_{i_1,j,l}$, which gives a unique nibble value and stores it in the state st, i.e., $st[j][i] \leftarrow \cap_l \mathcal{L}_{i_1,j,l}$.
3. Increment $i_1 \leftarrow i_1 + 1$ and continue steps 2-3 up to $i_1 = 7$.

we need first to calculate complexities step by step. According to Algorithm 11, the phase counter (step 1) runs for 8 times. Step 2 runs up to $|\mathcal{H}_{i_1}|$, i.e., q_{i_1} times. Finally, step 2c runs up to $2^4 \times 2^3 \times 2^4 = 2^{11}$ times. Therefore, the overall time complexity will be $2^3 \times 2^2 \times 2^{11} = 2^{16}$. Also, the memory complexity of this algorithm (maximum entry of the DDT table is 4) is $4 \times 2^4 + (i_1 \times j \times l \times 4) = 2^6 + (8 \times 8 \times 4 \times 4) \approx 2^{10}$ nibbles.

6.3.1.5 Software Implementation

Practically, we have implemented both our proposed algorithms (Algorithm 10, Algorithm 11) to collect several forgeries and then, retrieved the sponge state uniquely. The forgery verification in Algorithm 10 was performed on a Intel(R) Core(TM) i5-8250U CPU @1.60GHz machine. Finally, the secret key was successfully recovered approximately with the above-mentioned complexities. The source code of this attack is available in [220].

6.3.2 Random Bit-flip Fault Model

The previous section discussed the difficulty of injecting a large number of faults (around 2^{32}) to make one faulty forgery using nibble faults, making it impractical in practice. This is because the faults have to be injected at the second to last round of the PHOTON state (before MCS operation). Injecting random nibble faults at the last round (before the SC operation) reduces the number of fault injections significantly (around 2^4 injections to make one faulty forgery). However, the required faulty forgery will become much higher to recover the state because there is no significant information about the injected faults. This may result in the state not being uniquely retrievable by collecting several faulty forgery corresponding to all the nibbles in the PHOTON state. Therefore, instead of nibble faults, random bit-flip faults are considered as they are better suited in this scenario.

This section will introduce a fault model based on random bit-flips for the PHOTON-BEETLE scheme, which can be extended to the ORANGE, ESTATE, and SIV-TEM-PHOTON schemes to recover their states through faulty forgery during decryption. Additionally, we will provide a theoretical analysis of the required faulty forgery and validate our attack using software implementations of all four schemes.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
a	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
b	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
c	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
d	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
e	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

Table 6.3: Difference Distribution Table

6.3.2.1 The Fault Model.

In this fault model, an attacker can induce a fault by precisely flipping a single bit, thereby changing one nibble of the internal state. For instance, the attacker can use a Laser beam to perform such precise faults [171, 170, 168] with high accuracy in both space and time. Additionally, EM is another method that can be used to inject such precise bit-level faults, without requiring chip de-packaging. In practice, an EM fault injection setup such as the one described in [169] can be used to achieve this.

6.3.2.2 The Fault Attack Description

This attack model differs from the first one discussed in Section 6.3.1. In this model, we induce nibble faults at the last round, just before the SC operation, in the LPhoton call. For each query, we randomly select a tag $T^{'}$ and repeatedly induce nibble faults until we obtain a forgery. Additionally, we individually inject faults in each nibble of the state to collect several forgeries and recover the nibbles individually.

A short description to make faulty forgery is as follows. At any phase i, $(0 \leq i \leq 7)$, we make an encryption query (N, A, M) and get (C, T). We choose a nibble position (j, i), $j \in \{0, 1, \cdots, 7\}$ just before the SC operation inside the LPhoton state. Due

to this nibble difference $\Delta^{in}(= \Delta S_{i,j})$ at the 12^{th} round of LPhoton, the output difference after the last (12^{th}) round SC operation will be of the form $\mathsf{SC}(\Delta^{in}) = \Delta^{out}_{j,i}$ (see Figure 6-11).

Algorithm 12 Forging Strategy

Input: Associated data A and Plaintext M.
Output: A List $\mathcal{H}_{i,j}$ and a number $q_{j,i}$, $0 \leq i, j \leq 7$.

1. Make an encryption query (N, A, M) and get (C, T).
2. Choose the 12^{th} round (σ_{11}) at the last permutation call LPhoton of the Photon-Beetle sponge function where the nibble faults will be injected before the SC operation. We denote this as $\sigma_{11}^{\text{LPhoton}}$.
3. Initialize a phase counter $i = 0$ and a variable j to denote the row position.
4. For each $j \in \{0, 1, \cdots, 7\}$ at the i^{th} phase:
   - 4a. Fix a nibble position (j, i) at the $\sigma_{11}^{\text{LPhoton}}$ state, where a random bit fault $\delta \in \{1, 2, 4, 8\}$ will be injected repeatedly at this state position. Therefore, the input state difference will be $\Delta^{in} = \Delta S_{j,i}$, i.e., SC (Δ^{in}) has only a non-zero nibble difference at the position (j, i) (see Figure 6-12).
   - 4b. For each decryption query as (N, A, C, T'):
     - 4b.1. Randomly fill the $(j, i)^{th}$ nibble difference to form Δ^{out}_i (see Figure 6-12) and compute $\Delta^{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta^{out}_i)$.
     - 4b.2. Make $T' = T + Rate(\Delta^{out})$ and check whether T' is a valid forgery or not? If this happens, then store both the fault position (j, i) and the final state difference Δ^{out} in the list $\mathcal{H}_{j,i}$.
     - 4b.3. Continuing this to collect $q_{j,i}$ tag forgeries.
5. Increment $i \leftarrow i + 1$ and repeat steps 4-5 up to $i = 7$.
6. Do offline computation to recover the full LPhoton state, i.e., the Photon-Beetle state.

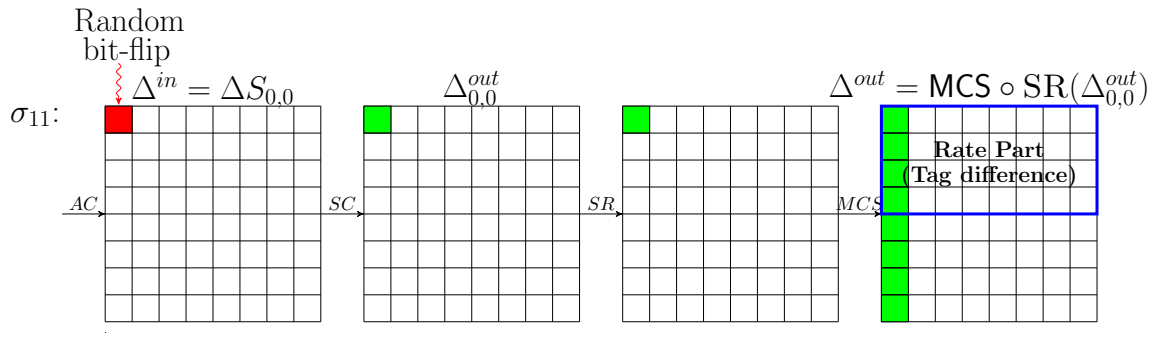


Figure 6-11: Faulty Forgery: Inducing Random Bit Faults at the Last Round

If the (bit) faults are injected repeatedly at the (j,i)-th nibble position in that case the values of the (j,i)-th nibble difference can be either $1, 2, 4$, or 8, i.e., $\Delta^{in} = \{1, 2, 4, 8\}$. Meanwhile, according to the DDT table (Table 6.3), there are 11 different output differences after the SC operation, i.e., $\Delta^{out}_{j,i} = \{3, 5, 6, 7, 9, a, b, c, d, e, f\}$. Thus, out of the 11 differences, the attacker can randomly choose one difference $\Delta^{out}_{j,i}$ in each faulty decryption query. Also, the last round SR, MCS operations do not have any impact on this attack. Therefore, by making 11 faulty (at the nibble position (j,i)) decryption queries (N, A, C, T') with $T' = T \oplus \Delta^{out}$ $(= \mathsf{MCS} \circ \mathsf{SR}(\Delta^{out}_{j,i}))$ we can expect one forgery. We repeat queries by injecting faults at the same nibble position to collect $q_{j,i}$ different forgeries and store them in the list $\mathcal{H}_{j,i}$. In this way, we individually perform faulty forgery for other nibbles also. The whole steps are illustrated in Algorithm 12 to get faulty forgery for all the nibbles in the LPHOTON state. A similar technique to Algorithm 12 can be employed to perform fault injections and create forgeries for other SPN-based schemes like ORANGE, SIV-TEM-PHOTON, and ESTATE, and subsequently recover their states. The only difference lies in the fact that for these schemes, the attacker needs to choose queries with the empty message.

Theorem 7. Let χ denote the faulty decryption queries to collect q distinct tag forgeries, i.e., repeatedly induce random bit faults at the fixed nibble position in the state (in the last round) until we get q different forgeries. Then, $E(\chi) < 44 \cdot \left\lceil 1 + \log\left(\frac{4}{4-q+1}\right)\right\rceil$.

Proof. To make a valid forgery, we have to satisfy this condition: $\mathsf{MCS}\circ\mathsf{SR}\circ\mathsf{SC}(\Delta^{in}) = \Delta^{out}$. Therefore, for each nibble position $(j,i), 0 \leq i, j < 8$, the attacker chooses the difference $\Delta^{out}_{j,i}$ randomly from the set $\{3, 5, 6, 7, 9, a, b, c, d, e, f\}$. Thus, $\Pr[\mathsf{MCS} \circ \mathsf{SR} \circ \mathsf{SC}(\Delta^{in}) = \Delta^{out}] = \frac{4}{4\times 11} = \frac{1}{11} = p$.

Let, $\chi_j, 1 \leq j \leq q\ (= 4)$ be the number of trials needed to collect j^{th} forgery after $j-1$ forgeries have been collected. As χ represents the number of independent trials needed to collect q successful forgeries, we have, $\chi = \chi_1 + \cdots + \chi_q$. Further, the probability of collecting j^{th} forgery is $p_j = \frac{4-j+1}{4\times 11} = \frac{5-j}{44}$. Therefore, χ_j follows geometric distribution and $E(\chi_j) = \frac{1}{p_j}$. By the linearity of expectations, we have,

tiny

$$E(\chi) = \sum_{j=1}^{q} \frac{44}{5-j} = \frac{44}{4} + \cdots + \frac{44}{4-q+1}$$
$$< 44 \cdot \left[1 + \log\left(\frac{4}{4-q+1}\right)\right].$$

□

6.3.2.3 State Recovery of Photon-based AE Schemes

According to the Algorithm 12, for each nibble position (j, i), we have a list $\mathcal{H}_{j,i}$ which contains $q_{j,i}$ number of pairs. Each pair contains the nibble position (j, i) and the corresponding forging state difference Δ^{out}. Further, the state difference Δ^{out} only has a column difference (i^{th} column) and the remaining columns have no differences. To, recover the state, we invert Δ^{out} up to SR operations and get $\Delta^{out}_{j,i} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$ (see Figure 6-11). As a result, we have only a non-zero nibble difference $\delta^{'}$ ($(j, i)^{th}$ position in the state) in $\Delta^{out}_{j,i}$ and the input difference $\Delta^{in} \in \{1, 2, 4, 8\}$. We try through all the nibble values x (with the associated difference $\delta^{'}$) to go backward by inverting the SC operation and store them in a list if the input difference δ ($= S^{-1}(x) \oplus S^{-1}(x \oplus \delta^{'})$) is equal to 1 or 2 or 4 or 8. Repeat this for other pairs in the list $\mathcal{H}_{j,i}$ and we can get a list of possible nibble values or might get a unique nibble value. Thus, based on the collected forgeries from the list $\mathcal{H}_{j,i}$, we can get a list of possible values for other nibbles in the state. In Algorithm 13, we have proposed a search procedure to retrieve all nibbles of the state for PHOTON-based schemes. The same strategy can also retrieve all bytes of the AES state for ESTATE.

6.3.2.4 Attack Complexity

We have observed that for PHOTON-BEETLE, ORANGE, and SIV-TEM-PHOTON schemes, if $q_{j,i} = 3$, then the number of possible values corresponding to each nibble position in the state becomes at most 2. Hence, the reduced state space for these schemes is around 2^{18} when $q = 3$. To recover the full state of PHOTON-BEETLE using faulty forgery, the attacker needs to collect three different forgeries at each (j, i)-th phase in Algorithm 12. To estimate the total number of faulty forgery, the attacker will

have to perform $2^{5.6}$ faulty decryption queries for each phase. Therefore, the total number of faulty decryption queries required to retrieve the full PHOTON-BEETLE state is $64 \times 2^{5.6} = 2^{11.5}$.

Similarly, for ESTATE, we verify that the list of possible values corresponding to each byte position in the AES state becomes at most 2 when $q_{j,i} = 2$. Based on the DDT table of AES, there are 254 ($\approx 2^8$) possible output differences corresponding to $\Delta^{in} \in 1, 2, 4, 8, 16, 32, 64, 128$. Using Theorem 7, the attacker needs to perform around $2^{9.1}$ faulty decryption queries to collect two different forgeries for a byte position. Therefore, the total number of faulty decryption queries required to retrieve the full ESTATE state is $16 \times 2^{9.1} = 2^{13.1}$.

To estimate the complexity of the above state recovery attack, we need to first calculate complexities step by step. According to Algorithm 13, both the phase counter i_1 and step 2 run 8 times separately. The step 2a. runs up to $|\mathcal{H}_{j,i_1}|$ number of times. Finally, step 2a.1.1. runs up to 2^4 times. Therefore, the overall time complexity is $8 \times 8 \times 3 \times 2^4 = 2^{11.5}$. Since the maximum DDT entry of PHOTON S-box is 4 and for

Algorithm 13 State Recovery of PHOTON-based AEAD

Input: List $\mathcal{H}_{j,i_1}, \forall\ 0 \leq j, i_1 < 8$.
Output: A PHOTON-BEETLE state $st[8][8]$ (two dimensional list of size 8×8).

1. Initialize a phase counter $i_1 = 0$ and another variable $j = 0$. Initialize several lists $\mathcal{L}_{j,i_1,l} \leftarrow \emptyset$, where $0 \leq i_1, j < 8$ and $0 \leq l < |\mathcal{H}_{j,i_1}| (= q_{j,i_1})$.
2. For each $j \in \{0, 1, \cdots, 7\}$:
   - 2a. For each state difference Δ^{out} corresponding to the fault position (j, i_1) in the list $\mathcal{H}_{j,i_1}$:
     - 2a.1. Compute $\Delta^{out}_{j,i_1} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$. This Δ^{out}_{j,i_1} has only one non-zero nibble difference at the position (j, i_1) as δ' (say).
       - 2a.1.1. For each $x \in \{0, 1, \cdots, 15\}$:
         - 2a.1.1.1. Compute $\delta = S^{-1}(x) \oplus S^{-1}(x \oplus \delta')$.
         - 2a.1.1.2. If $(\delta == 1)$ or $(\delta == 2)$ or $(\delta == 4)$ or $(\delta == 8)$ hold, then store x in the list $\mathcal{L}_{j,i_1,l}$.
   - 2b. Compute $\bigcap_l \mathcal{L}_{j,i_1,l}$, which gives a list of possible nibble values and stores it in the state list st, i.e., $st[j][i] \leftarrow \bigcap_l \mathcal{L}_{j,i_1,l}$.
3. Increment $i_1 \leftarrow i_1 + 1$ and continue steps 2-3 up to $i_1 = 7$.

Schemes	# Nibble Forgeries ($q_{j,i}$)	Expected # Faults	Reduced Key-Space
PHOTON-BEETLE	3	$2^{11.5}$	2^{20}
ORANGE	3	$2^{11.5}$	2^{20}
SIV-TEM-PHOTON	3	$2^{11.5}$	2^{20}
ESTATE	2	$2^{13.1}$	2^{5}

Table 6.4: Results on Expected Faults and Reduced Key-Space under Bit Faults

a given output difference, there are at most 8 possible nibble values x which leads to the input differences either 1 or 2 or 4 or 8. Thus, the space complexity of this state recovery algorithm is $8 \times 8 \times 8 = 2^9$ nibbles.

6.3.2.5 Software Implementation

We successfully implemented the faulty forgery attack (Algorithm 14) and the state recovery attack (Algorithm 15) on all schemes and recovered their secret keys. Our implementation used an Intel(R) Core(TM) i5-8250U CPU @1.60GHz machine, and the secret key was recovered with the complexities listed in Table 6.4. The source code for both attacks is available in [221, 222, 223, 224].

6.3.3 Known Fault Model

In this model, we first discuss the fault model, where we inject (nibble) faults that are random but known to the attacker. Then, we retrieve the secret key by repeatedly performing faulty forgery for each nibble and then recover its secret. Further, we additionally give a software implementation of these algorithms to validate this attack.

6.3.3.1 The Fault Model

In this fault model, a random fault is induced to change one nibble of the internal state but, the faulty value is known to the attacker. For example, an attacker could attempt to create a nibble fault at the input of a particular round such that the attacker perfectly knows the statistical distribution of the faulty value. The attacker can use a Laser beam to induce such faults with high accuracy (space and time).

6.3.3.2 The Fault Attack Description

In this model, the attacker induces nibble faults at the last round (before the SC operation) in the LPhoton call. Then, in each query, repeatedly choose a random tag T' and induce nibble faults until it becomes a forgery. Further, in this case, we inject faults in each nibble of the state individually to collect several forgeries and then, recover nibbles individually.

A short description to make faulty forgery is as follows. At any phase i, $(0 \leq i \leq 7)$, make an encryption query (N, A, M) and get (C, T). Choose a nibble position (j, i), $j \in \{0, 1, \cdots, 7\}$ just before the SC operation inside the LPhoton state. Due to this nibble difference $\Delta^{in} (= \Delta S_{i,j})$ at the 12^{th} round of LPhoton, the output difference after the last (12^{th}) round SC operation will be of the form $\mathsf{SC}(\Delta^{in}) = \Delta^{out}_{j,i}$ (see Figure 6-12). Therefore, the total number of chosen differences of Δ^{out}_{i} is 2^4. Also, the last round SR, MCS do not have any impact in this attack. So, choosing 2^4 faulty (with faulty nibble position (j, i)) decryption queries (N, A, C, T') by $T' = T \oplus \Delta^{out}$ $(= \mathsf{MCS} \circ \mathsf{SR}(\Delta^{out}_{j,i}))$ and expect one faulty forgery. Then, repeat queries by injecting faults at the same nibble position to collect $q_{j,i}$ different forgeries and store them in the list $\mathcal{H}_{j,i}$. This approach can be repeated for the other nibbles in the LPhoton state to induce faults and obtain faulty forgery for each nibble. The complete steps are presented in Algorithm 14 to obtain faulty forgery for all nibbles in the LPhoton state. Therefore, Algorithm 14 can be used to perform faulty forgery for Photon-Beetle, ORANGE, and SIV-TEM-PHOTON schemes. The approach to performing faulty forgery for the ESTATE scheme is similar to Algorithm 14 and can be applied accordingly.

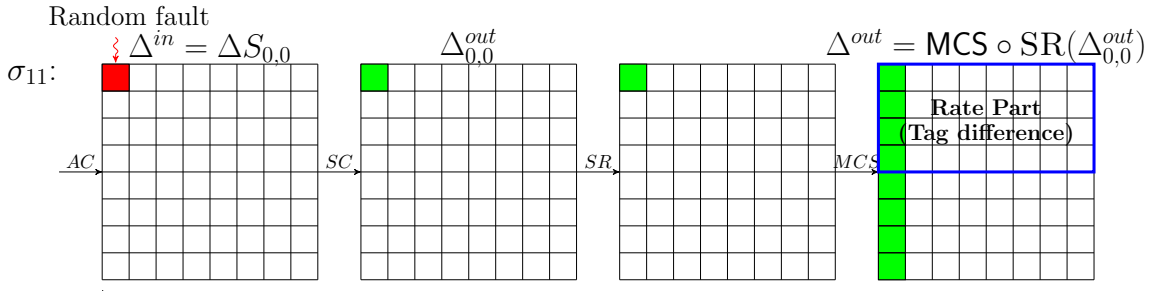


Figure 6-12: Faulty Forgery: Inducing Faults at the Last Round

Theorem 8. Let χ denote the faulty decryption queries to collect q distinct tag forgeries, i.e., repeatedly induce nibble (known) faults at the fixed position in the state (last round) to get q different forgeries. Then, $E(\chi) < 2^8 \cdot \left[1 + \log\left(\frac{2^4}{2^4-q+1}\right)\right]$.

Proof. To make a valid forgery, the attacker has to satisfy this condition: $\mathsf{MCS} \circ \mathsf{SR} \circ \mathsf{SC}(\Delta^{in}) = \Delta^{out}$. Therefore, at any phase $i, 0 \leq i < 8$, $\Pr[\mathsf{MCS} \circ \mathsf{SR} \circ \mathsf{SC}(\Delta^{in}) = \Delta^{out}] = \frac{2^4-1}{(2^4-1)\times(2^4-1)} \approx \frac{1}{2^4} = p$.

Let, $\chi_j, 1 \leq j \leq q$ be the trials needed to collect j^{th} forgery after $j-1$ forgeries have been collected. As χ represents the number of independent trials needed to collect q successful forgeries, we have, $\chi = \chi_1 + \cdots + \chi_q$. Further, the probability of collecting j^{th} forgery is $p_j = \frac{2^4-j+1}{2^4\times 2^4} = \frac{2^4-j+1}{2^8}$. Therefore, χ_j follows geometric distribution and $E(\chi_j) = \frac{1}{p_j}$. By the linearity of expectations, we have, tiny

$$\begin{aligned} E(\chi) &= \sum_{j=1}^{q} \frac{2^8}{2^4-j+1} = \frac{2^8}{2^4} + \frac{2^8}{2^4-1} + \cdots + \frac{2^8}{2^4-q+1} \\ &< 2^8 \cdot \left[1 + \log\left(\frac{2^4}{2^4-q+1}\right)\right]. \end{aligned}$$

□

6.3.3.3 State Recovery of Photon-based AE Schemes

According to the Algorithm 14, for each nibble position (j,i), we have a list $\mathcal{H}_{j,i}$ which contains $q_{j,i}$ number of tripplets, which contains the nibble position (j,i), faulty value δ, and the corresponding forging state difference Δ^{out}. Further, this differential state Δ^{out} only has a column difference (i^{th} column) and the remaining columns have no differences. To, recover the state, invert Δ^{out} up to SR operations and get $\Delta^{out}_{j,i} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$ (see Figure 6-12). As a result, there is only one nibble difference ($(j,i)^{th}$ position in the state) in $\Delta^{out}_{j,i}$ as well as in Δ^{in}. Moreover, in this case, both nibble differences in the differential states $\Delta^{out}_{j,i}$ and Δ^{in} are known. Thus, try for all the nibble values and store them in a list if it pass through the S-box. Do this for other tripplets in the list $\mathcal{H}_{j,i}$, and finally, the $(j,i)^{th}$ nibble value of the LPhoton state will be retrieved. In this way, the attacker retrieves all the other nibbles from the list $\mathcal{H}_{j,i}$. The procedure described in Algorithm 15 can be used to recover all nibbles of the Photon-Beetle state. The same approach can

also be applied to recover the state for other schemes such as ORANGE, SIV-TEM-PHOTON, and ESTATE.

6.3.3.4 Attack Complexity

For all schemes, we verify that $\forall i, j \in \{0, 1, \cdots, 7\}$, $q_{j,i} = 2$ is sufficient to recover the $(j, i)^{th}$ nibble uniquely. According to Theorem 8, the attacker will have to perform $2^{5.05}$ faulty decryption queries to collect two different forgeries at the i^{th} phase in the Algorithm 14. Therefore, the total faulty decryption queries to retrieve the full PHOTON-BEETLE state is $64 \times 2^{5.05} = 2^{11.05}$. Whereas for AES-based scheme ESTATE, the attacker will have to perform around $2^{9.01}$ faulty decryption queries

Algorithm 14 Forging Strategy

Input: Associated data A and Plaintext M.
Output: A List $\mathcal{H}_{i,j}$ and a number $q_{j,i}$, $0 \leq i, j \leq 7$.

1. Make an encryption query (N, A, M) and get (C, T).
2. Choose the 12^{th} round (σ_{11}) at the last permutation call LPHOTON of the PHOTON-BEETLE sponge function where the nibble faults will be injected before the SC operation. Denoting this as $\sigma_{11}^{\text{LPHOTON}}$.
3. Initialize a phase counter $i = 0$ and a variable j to denote the row position.
4. For each $j \in \{0, 1, \cdots, 7\}$ at the i^{th} phase:
   - 4a. Fix a nibble position (j, i) at the $\sigma_{11}^{\text{LPHOTON}}$ state, where a fault δ (known to the attacker) will be injected repeatedly at this state position. Therefore, the input state difference will be $\Delta^{in} = \Delta S_{j,i}$, i.e., SC (Δ^{in}) has only a non-zero nibble difference at the position (j, i) (see Figure 6-12).
   - 4b. For each decryption query as (N, A, C, T'):
     - 4b.1. Randomly fill the $(j, i)^{th}$ nibble difference to form Δ_i^{out} (see Figure 6-12) and compute $\Delta^{out} = \mathsf{MCS} \circ \mathsf{SR}(\Delta_{j,i}^{out})$.
     - 4b.2. Make $T' = T + Rate(\Delta^{out})$ and check whether T' is a valid forgery or not? If this happens, then store both the fault position (j, i), fault value δ, and the final state difference Δ^{out} in the list $\mathcal{H}_{j,i}$.
     - 4b.3. Continue this to collect $q_{j,i}$ many tag forgeries.
5. Increment $i \leftarrow i + 1$ and repeat steps 4-5 up to $i = 7$.
6. Do offline computation to recover the full LPHOTON state, i.e., the PHOTON-BEETLE state.

Schemes	# Nibble Forgeries ($q_{j,i}$)	Expected # Faults	Reduced Key-Space
PHOTON-BEETLE	2	$2^{11.05}$	2^0
ORANGE	2	$2^{11.05}$	2^0
SIV-TEM-PHOTON	2	$2^{11.05}$	2^0
ESTATE	2	$2^{13.01}$	2^0

Table 6.5: Results on Expected Faults and Reduced Key-Space under Known Faults

to collect two different forgeries. Thus, the total faulty decryption queries to retrieve the full ESTATE state is $16 \times 2^{9.01} = 2^{13.01}$. Now, to estimate the complexity of the above state recovery attack, we need first to calculate complexities step by step. According to Algorithm 15, both the counters i_1 and j runs for 8 times. The step 2a. runs up to $|\mathcal{H}_{i_1}|$, i.e., q_{i_1} number of times. Finally, step 2a.2.1. runs up to 2^4 times. Therefore, the overall time complexity is $8 \times 8 \times 2 \times 2^4 = 2^{11}$. Since the maximum DDT entry of PHOTON S-box is 4, the space complexity of this state recovery algorithm is $8 \times 8 \times 2 \times 4 = 2^9$ nibbles.

6.3.3.5 Software Implementation

We have implemented both the faulty forgery attack (in Algorithm 14) and the state recovery attack (in Algorithm 15) to recover the secret key. For all the schemes, the implementation was performed on an Intel(R) Core(TM) i5-8250U CPU @1.60GHz machine, and their secret key was successfully recovered uniquely with different fault complexities(given in Table 6.5). The source code of these attacks is available in [221, 222, 223, 224].

6.3.3.6 Further Improvement over Faults

To reduce the number of faults, an attacker can induce a precise bit fault to change one nibble (at the input of SC operation) in the internal state instead of injecting known nibble faults in the PHOTON-BEETLE state. For example, the attacker can use a Laser beam to induce such faults [171, 170, 168] with high accuracy (space and time). Further, EM is also a good way of injecting such precise bit faults, and it does not require any chip de-packaging. Practically, precise bit-level fault injections can be achieved by EM fault injection setup in [169].

According to the DDT (given in Table 6.3) of the S-box in the PHOTON-BEETLE AE, the input difference 1 has only 4 non-zero output differences. Also, the input difference 8 has only 6 non-zero output differences, whereas the other two input differences 2 and 4 have 7 non-zero output differences. According to Algorithm 11, the attacker can repeatedly inject a bit fault $\delta = 1$ (at the LSB) in the nibble position (j, i) (i.e., Δ^{in}) at the last round of LPHOTON. In this case, the total number of chosen differences of Δ_i^{out} is 4. Therefore, the attacker needs to repeat the bit fault injection in the position (j, i) 4 times and expect one forgery. Further, according to Algorithm 12, the attacker needs $q_{j,i} = 2$ forgeries to uniquely retrieve the nibble. So, to reduce faults, the attacker needs bit fault $\delta = 8$ (at the MSB) in the nibble position (j, i) (i.e., Δ^{in}) at the last round of LPHOTON. In this case, the attacker needs to repeat the bit fault injection in the position (j, i) 6 times and expect one forgery. Finally, the attacker needs to perform $(6 + 4) = 10$ faulty queries to collect two different forgeries. Hence, the total number of faulty decryption queries to retrieve the full PHOTON-BEETLE state is $64 \times 10 = 640$.

Algorithm 15 State Recovery of PHOTON-BEETLE

Input: List $\mathcal{H}_{j,i_1}, \forall\ 0 \leq j, i_1 < 8$.
Output: A PHOTON-BEETLE state $st[8][8]$.

1. Initialize a phase counter $i_1 = 0$ and another variable $j = 0$. Initialize several lists $\mathcal{L}_{j,i_1,l} \leftarrow \emptyset$, where $0 \leq i_1, j < 8$ and $0 \leq l < |\mathcal{H}_{j,i_1}| (= q_{j,i_1})$.
2. For each $j \in \{0, 1, \cdots, 7\}$:
   - 2a. For each (faulty) difference value δ, and the state difference Δ^{out} corresponding to the fault position (j, i_1) in the list $\mathcal{H}_{j,i_1}$:
     - 2a.1. Compute $\Delta_{i_1}^{out} = \mathsf{SR}^{-1} \circ \mathsf{MCS}^{-1}(\Delta^{out})$. This $\Delta_{i_1}^{out}$ has only one nibble difference at the position (j, i_1) as δ'.
     - 2a.2. If $\mathsf{DDT}[\delta][\delta'] > 0$:
       - 2a.2.1. For each $x \in \{0, 1, \cdots, 15\}$:
         - 2a.2.1.1. If $S(x) \oplus S(x \oplus \delta) == \delta'$ holds, then store x in the list $\mathcal{L}_{j,i_1,l}$.
   - 2b. Compute $\bigcap\limits_l \mathcal{L}_{j,i_1,l}$, which gives a unique nibble value x and store it in the state st, i.e., $st[j][i] \leftarrow \bigcap\limits_l \mathcal{L}_{j,i_1,l}$.
3. Increment $i_1 \leftarrow i_1 + 1$ and continue steps 2-3 up to $i_1 = 7$.

Table 6.6: Expected Number of Faulty Queries for Different Fault Models

Schemes	Number of forgeries (q)	KFM		RFM		RBFM	
		Theory	Simulation	Theory	Simulation	Theory	Simulation
PHOTON-BEETLE	1	2^{4}	$2^{3.92}$	2^{32}	$2^{31.8}$	$2^{3.47}$	$2^{3.3}$
	2	$2^{5.05}$	$2^{4.92}$	$2^{33.04}$	$2^{32.85}$	$2^{4.7}$	$2^{4.38}$
	3	$2^{5.7}$	$2^{5.53}$	$2^{33.68}$	$2^{33.49}$	$2^{5.58}$	$2^{5.1}$
ESTATE	1	2^{8}	$2^{7.88}$	2^{32}	$2^{31.8}$	$2^{7.99}$	$2^{7.84}$
	2	$2^{9.01}$	$2^{8.71}$	$2^{33.04}$	$2^{32.85}$	$2^{9.1}$	$2^{8.8}$

Note: "KFM", "RFM", and "RBFM" stands for Known Fault Model, Random Fault Model, and Random Bit-flip Fault Model respectively.

6.3.4 Theory vs. Experiment

We present a set of simulations for the PHOTON-BEETLE and ESTATE schemes to evaluate the expected number of required faults and compare them with theoretical estimates. The simulations were conducted on an Intel(R) Core(TM) i5-8250U CPU @1.60GHz computer, and the software implementations are available in [225]. We executed the simulation procedure 100,000 times for the second model, while for the first model, we performed it 100 times due to the low complexity of making one forgery. Table 6.6 summarizes the total number of faulty queries necessary to recover a nibble/byte of the LPHOTON/TWEAES-128 state for different fault models, as obtained from both the simulations and theoretical estimations.

6.4 Discussion and Countermeasures

In this section, we will first provide a brief description of the challenges in directly applying the statistically Ineffective Fault Attack (SIFA) to SPN-based sponge AE ciphers compared to Differential Fault Analysis (DFA) on such schemes. Next, we will demonstrate that SIFA requires a significantly larger number of faults compared to the DFA attack. Finally, we will propose potential countermeasures to prevent DFA on SPN-based sponge AE schemes.

SIFA is a powerful technique that exploits the distribution of faults such that the faults do not affect the outcome of a computation (ineffective faults). This distribution of the values where an ineffective fault does not change the computation often behaves non-uniformly in practice. Most of the active fault attacks thwart detection or infection-based countermeasures, where faults are effective during computation. For SIFA,

ineffective faults can easily bypass these countermeasures. Further, SIFA can break the countermeasure (detection and infection-based) even while they are combined with SCA countermeasures like masking. The basic concept of this technique is as follows:

- Make several encryption/decryption queries by repeatedly injecting faults at a bit/nibble/byte during the computation and collect some plain- or ciphertexts where the fault was ineffective (i.e., for correct computations).
- Apply the SFA to retrieve the key bits.

Further, SFA can not be directly applied at the finalization phase in the sponge AE even if the key is XORed before the tag is obtained. Because, in the finalization phase, some partial bits of the internal state are output as the final tag. As a result, we can not directly apply SIFA at the finalization phase. Therefore, to apply this technique on nonce-based (sponge or stream cipher-based) AE, the attacker needs to target the initialization phase. To do this, we first target the decryption of valid messages (N, A, C, T) in the presence of faults (at the intermediate rounds) that are induced during the initialization phase. If the fault is ineffective, the tag verification will succeed. Next, based on the collected forgeries (respective nonces), the attacker finds the key bits that directly influence the value (the position where the faults are induced). The wrong key guess leads to a uniform distribution, whereas the right key guess leads to a non-uniform distribution.

In [192], Dobraunig et al. first applied this technique to attack sponge-based AE Keyak and Ketje. In this paper, they choose to induce faults at the second round (before the non-linear operation) in the initialization phase. Finally, by performing around $24 \times 250 \approx 6,000$ faulty decryptions, they recovered 82 bits and 152 bits of the key for Keyak and Ketje respectively. The remaining key bits can be determined either by brute force or repeating key recovery for a different fault location. Recently, in [212], Gruber et al. further applied this technique on GIMLI which is one of the second-round candidates in NIST LwC competition. They induce faults at three different rounds (respectively 23rd, 22nd, and, 21st rounds) at the initialization phase in the decryption query. They used three different fault models random-And, Stuck-at-0, and probabilistic bit-flip model to simulate a fault in software implementation,

which can further apply to hardware implementations with the same reasoning. This simulation shows that the ineffectiveness rate for random-And (for byte faults) is approx 10^{-1} is much higher in comparison with the other two given fault models. Under the random-And byte fault model, the attack needs around 180 and 340 ineffective faults to recover up to 48 (for the 22nd round) and 176 (for the 21st round) key bits respectively. Hence, the total number of faulty decryption queries to retrieve 48 and 176 key bits are around $180 \times 10 \approx 1800$ and $340 \times 10 \approx 3400$ respectively.

Countermeasures. To counter the attacks demonstrated in this work on SPN-based sponge authentication schemes, the following approach can be employed. The idea is to rearrange the nibble positions within the state from which the tag is generated, whether it originates from the rate part or the capacity part. In general, the final tag in SPN-based schemes is produced from the rate part of the state. For square values of m, such as when the state of an SPN cipher is represented as an $m \times m$ nibble/byte matrix, where $m = 4, 8$, the rate part of these permutation-based schemes consists of the first few rows in the $m \times m$ state, while the capacity part occupies the remaining rows. Similarly, for an SPN state represented as a rectangular matrix, the rate part consists of the first few rows, while the remaining rows constitute the capacity part. By applying fault injection, we can accurately reproduce the complete state differences based on the differences observed in faulty and non-faulty tags. However, if the tag is generated from the first four columns of the state, it may be challenging to deduce the full state difference from the tag differences caused by faulty and non-faulty computations.

Therefore, if the tag is directly produced from the first four columns of the state, meaning 64 bits from the rate part and the remaining 64 bits from the capacity part, an attacker can only perform fault injection attacks on the first four columns. Consequently, the attacker can only recover the state information corresponding to these columns, while the other four (last) columns remain unknown. This significantly reduces the attacker's ability to fully recover the state. By rearranging the nibble positions in the state, especially when the tag is generated from the first four columns, the attacker's task becomes more difficult. In such cases, the attacker can only recover the state information for the first four columns, while the remaining columns remain

unknown. This effectively increases the complexity of retrieving the full state for an attacker attempting to exploit the device.

6.5 Conclusion

We first show the differential fault attacks on PHOTON-BEETLE, ORANGE, SIV-TEM-PHOTON, and ESTATE under three different fault models. We do this by performing faulty forgery in the decryption query to collect some forging tags. Then, we recover the state by doing offline computation based on the collected forgeries. Once the state is recovered, the master key can be retrieved either by inverting the sponge function or by retrieving the subkey from the state. Also, we give a theoretical estimation to collect different forgeries corresponding to a fixed nibble/byte position for the given fault models.

Under the first fault model, approximately $2^{37.15}$ faulty forgery are required to retrieve the state. Also, the time and the space complexities to recover the state are respectively 2^{16} and 2^{10} nibbles. Then, to retrieve the state for PHOTON and AES-based AE schemes under the second fault model, the number of faulty forgery are $2^{11.5}$ and $2^{13.1}$ respectively. The time and the space complexities of the given attack are respectively $2^{11.5}$ and 2^{9} nibbles. Whereas, under the known fault model, we need to perform approximately $2^{11.05}$ faulty forgery to recover the state. Also, the time and space complexities of this attack are 2^{11} and 2^{9} nibbles respectively. However, we have reduced the number of faulty queries under the precise bit-fault model instead of using known nibble faults. Under this bit-fault model, we have shown that around $2^{9.32}$ faults are sufficient to recover the state. For validation purposes, we provided software implementations of the complete attacks for all the proposed fault models.

Finally, in this work, we have shown that how we can recover the internal state by performing DFA on SPN-based sponge and SIV-like AE schemes PHOTON-BEETLE, ORANGE, SIV-TEM-PHOTON, and ESTATE. Also, in the SPN-based sponge-like constructions, if some designer uses an extra key injection just before the final permutation call, still we can retrieve its secret key by performing the attack in two steps. Because, in the first DFA attack we recover the internal state, then we repeat the DFA attack again to recover the state before the final permutation and then

retrieve its secret key. Also, for any SPN-based sponge AE, it is quite straightforward that the number of faulty queries for the SIFA technique might be much higher in comparison to our proposed DFA. However, we conjecture that our attack can also be applied to any SPN-based sponge and SIV-like schemes.

7 DEPEND ON DEEPAND: CRYPTANALYSIS OF NLFSR-BASED LIGHTWEIGHT CIPHERS TINYJAMBU AND KATAN

7.1 Introduction

The well-established block cipher design principle constitutes the construction of a so-called *weak* round-function which is iterated (*sufficiently*) many times to create a secure construction. One of the fundamental decisions in such an iterative block cipher design is the number of rounds. This decision is a trade-off between security and efficiency and plays an even more critical part in the context of Lightweight Cryptography which is referred to as crypto tailored for resource-contained environments. A typical way to decide this is to take into account the penetration of the best attack available and then add some more rounds as the so-called *security-margin*. Traditionally, designers try to prove how many rounds are sufficient to resist a certain kind of attack. This in general is a rigorous task and primarily limited to a specific construction. For instance, resistance against differential cryptanalysis [226] relies on the number of active sboxes in the best available differential characteristic. It has been a long-standing question if these seemingly critical tasks of cryptanalysis could be automated or aided in some generic way. Though there have been initial attempts in this direction but the first major breakthrough in this direction is attributed to Mouha *et al.* [48] who was one of the first to demonstrate how the cryptanalytic problems of determining the minimum number of active sboxes could be modeled as an optimiza-

tion problem which could in turn be solved by automated solvers. In particular, the authors showcased how Mixed Integer Linear Programming (MILP) can be leveraged as an ingenious cryptanalysis aid. This seminal work spawned an entirely new line of research where the goal is one hand to increase the breadth of the strategy with new modelings (applications to linear, division, and impossible differential cryptanalysis). On the other hand, the idea is to improve upon the existing models to capture the underlying crypto property as closely as possible. The current work aims to add to state-of-art with *better* MILP modeling.

Interestingly, researchers have shown that there are mechanisms to precisely model valid transition for many crypto properties [115, 116, 114]. However, the catch is that this results in models becoming over-constrained thereby infeasible to be solved in reasonable time. On the other end of the spectrum is an oversimplified model which might lead to invalid transitions. There is a rich body of work that tries to reach a middle ground by what can perhaps be referred to as *balanced* modeling [227, 117]. In FSE 2020, Saha *et al.* made an interesting observation in this line of balanced modeling for the NIST-LWC [228] competition initial version TinyJAMBU [229] (version 1). The authors pointed out that the correlation between multiple AND gates could lead to them becoming dependent leading to joint propagation of differential characteristics. Our research pushes the boundaries to reveal that further refinement is possible and a generalized model can be devised to extend the findings to a class of Non-Linear Feedback Shift Register (NLFSR) based lightweight block ciphers with specific results on KATAN [230], NIST-LWC [228] competition finalist TinyJAMBU [231] (version 2) and its previous version TinyJAMBU [229] (version 1).

7.1.1 Summary of The Chapter

Generalizes AND Modeling Framework. The current work proposes a generalized model to capture first-order correlation in single as well as *multiple* AND gates. This is a direct improvement over the recent work [49] by Saha *et al.* where a new MILP model was developed leveraging AND gate correlations. To be precise, the analysis by Saha *et al.* exploits two subsequent AND computations with a common input position (for e.g the middle bit position b out of three inputs a, b, c to the subsequent ANDs). The present work provides further insight into this interesting correlation by

extending it to *multiple* AND gates. The findings show a significant impact on the actual probabilities of the differential characteristics. More specifically, the common input position in the two subsequent ANDs will be revealed when a particular difference pattern. For instance if $(\Delta a, \Delta b, \Delta c) = (1, 0, 1)$ one has to pay a probability for only the first AND whereas the second AND will pass freely. We further re-investigate this case and observe that due to the difference $(\Delta a, \Delta b) = (1, 0)$, the output difference (Δz_1) of the first AND directly reveals the bit b, i.e., $\Delta z_1 = b$. Once Δz_1 is fixed, passage through the second AND is for free. From another perspective, for an AND gate with two inputs a, b, if we know the bit value of a, then for a given difference pattern $(\Delta a, \Delta b) = (0, 1)$, the output difference $\Delta z = a$ will become deterministic. We would like to emphasize that the distribution of differences in AND gates under conditionally known inputs might be well-known. However, in the current work, we revisit this in the light of correlations that develop and can hence be exploited in MILP modelings.

Improved Bounds for TinyJAMBU and KATAN. Our research constitutes a comprehensive study of all correlations that develop (and were perhaps *missed* in earlier attempts) with or without conditionally known inputs. These correlations when incorporated in MILP models lead to the best characteristics known on NLFSR-based ciphers TinyJAMBU and KATAN which in turn can be exploited to mount distinguishing and forgery attacks. It is worth noting that correlated AND, though not a new observation, earlier results were only restricted to the single AND gates. For NLFSR-based ciphers employing multiple ANDs, the current work adds newer cases there also bettering the size of the differential characteristic clusters generated. All findings are consolidated into a new generalized *refined* model. Finally, we apply this new model in the keyed-permutation of TinyJAMBU AE and to all the KATAN -variants and show that our model captures all possible correlations between ANDs and provides better optimal differential characteristics in comparison to previous models.

Notion of Conditionally Free Rounds. The work builds upon two fundamental observations which are looked at from an information-theoretic way in terms of con-

ditionally known inputs and how much reduction it leads to in terms of the overall entropy of the values and differences that related to one/multiple AND gates. The Observations 2 and 3 help identify the underlying principle behind the chained (a term introduced in [49]) AND gates by introducing the notion of what we refer as *conditionally free* rounds. The primary motivation is to redefine the notion of correlated AND operations (Lemma 2) using these observations referred to above. The proof idea stems from the fact that for some specific differential inputs and output of the AND gate, we gain some extra information about the actual bits of the internal state thereby reducing the entropy. The observations are further exploited to develop a generalized MILP model for differential cryptanalysis, referred to as DEEPAND, which has the potential to capture all possible correlations between multiple ANDs in NLFSR-based (AND based) block ciphers. Consider a NLFSR-based block cipher with h AND gates. Suppose, the AND gates compute $(a_1^1 \cdot b)$, $(b \cdot a_1^2)$, $\cdots$, $(a_h^1 \cdot b)$, $(b \cdot a_h^2)$ across some rounds. We will show that these $2h$ AND operations are correlated. Essentially, if $b = 0$ and out of these $2h$ AND computations, m AND gates are active, then due to the correlated nature between the AND gates the output of these AND computations can be fixed with probability 2^{-1} instead of 2^{-m}. The proposed DEEPAND model employs the following properties to find differential characteristics

1. Captures all possible correlations between several AND computations
2. Exploits observations made to gain advantage thereby penetrating some extra rounds freely in the differential characteristics.

As an immediate application, the DEEPAND model is applied on the KATAN block cipher to find differential characteristics that are better than existing ones. We have explicitly shown characteristics where the dependency between several AND computations are captured. The model is also able to improve the related-key boomerang attacks on KATAN. To show the versatility of the model, it is also employed on keyed permutation of TinyJAMBU (version 2). The model can converge faster while searching for differential characteristics for TinyJAMBU in comparison to the refined model which we believe is due to the notion of complete and conditionally free rounds introduced in this work. Finally, a forgery attack on TinyJAMBU (version 1) is mounted with a probability of $2^{-67.88}$.

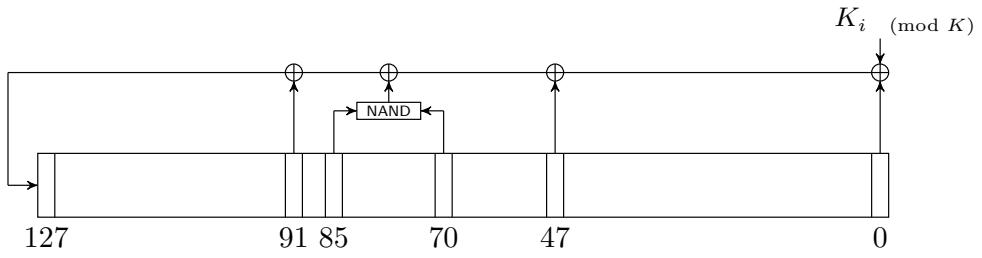


Figure 7-1: The Permutation P^{k_i}

7.2 Specifications of TinyJAMBU and KATAN

In this section, we provide a brief description of two cryptographic algorithms: TinyJAMBU and KATAN .

Table 7.1: TinyJAMBU Variants

AEAD Variants of TinyJAMBU Mode	Size in bits				Number of Rounds in	
	State	Key	Nonce	Tag	P_l	$\hat{P}_l$
TinyJAMBU-128	128	128	96	64	640	1024
TinyJAMBU-192	128	192	96	64	640	1152
TinyJAMBU-256	128	256	96	64	640	1280

7.2.1 TinyJAMBU

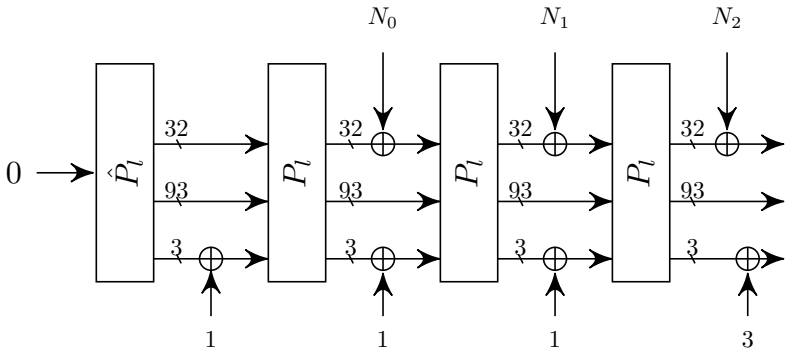


Figure 7-2: The Initialization of TinyJAMBU [49]

TinyJAMBU [231] (version 2) is a variant of JAMBU that was selected as a finalist in the NIST Lightweight Cryptography competition. It uses a 128-bit NLFSR-based keyed permutation with 128-bit state size and 32-bit message block size. It offers better security than JAMBU and duplex mode with nonce reuse. The permutation, denoted

by P_l^K, has l rounds and supports key sizes (|) K|of 128, 192, or 256 bits. In short, we use $\mathcal{P}_l$ to denote an l-round keyed permutation of TinyJAMBU throughout the chapter. The i^{th} round of the $\mathcal{P}_l$ permutation transforms a 128-bit state to another 128-bit state. The transformation is defined by $sf = s_0 \oplus s_{47} \oplus \overline{s_{70} s_{85}} \oplus s_{91} \oplus k_{i \mod |K|}$, where s_f is the transformed state and $k_{i \mod |K|}$ is the secret key. The permutation is shown in Figure 7-1. The TinyJAMBU mode has three variations named TinyJAMBU-128, TinyJAMBU-192, and TinyJAMBU-256, with specifications listed in Table 7.1.

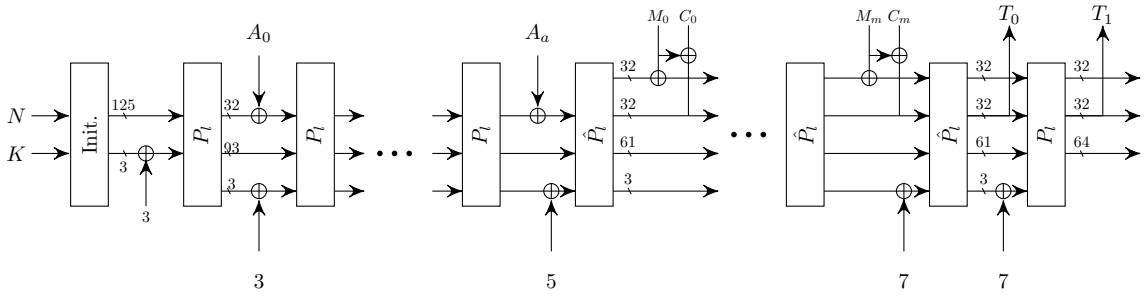


Figure 7-3: The Description of TinyJAMBU Mode [49]

The TinyJAMBU encryption process has four stages: Initialization, Associated Data Processing, Encryption, and Finalization. In the Initialization stage, the state is initialized through key and nonce setup. In the Associated Data Processing stage, each data block is processed by XORing with the state, updating the state with P_l, and XORing the associated data block with the updated state. In the Encryption stage, each message block is encrypted by XORing with the state, updating the state with $\hat{P}_l$, injecting the message block into the first block of the state, and producing the ciphertext by XORing the message block with the second block of the state. In the Finalization stage, the authentication tag $T = T_0||T_1$ is generated by XORing with the state, updating the state with $\hat{P}_l$, extracting T_0 from the state, XORing again with the state, updating the state with P_l, and extracting T_2 from the resulting state. The overall structure of the TinyJAMBU mode is depicted in Figure 7-3, where the permutations P_l and $\hat{P}_l$ are specified in Table 7.1. The only difference between the initial version of TinyJAMBU [229] (version 1) compared to version 2 is the number of rounds of P_l. In version 1, the number of rounds in P_l is 384.

7.2.2 KATAN

The KATAN family is a very efficient NLFSR-based hardware-oriented block cipher with three variants, namely KATAN32, KATAN48, KATAN64 correspond to 32, 48, and 64-bit block sizes. All these variants have 254 rounds and use the non-linear functions $\mathcal{NF}_1$ and $\mathcal{NF}_2$. Also, they use the same LFSR-based key schedule which takes an 80-bit key as an input. The general structure of the KATAN cipher is as follows.

First, the plaintext is loaded into two registers L_1 and L_2. In each round, several bits are taken from the registers to fed into the non-linear functions, and finally, the output of $\mathcal{NF}_1$ and $\mathcal{NF}_2$ is loaded to the least significant bits (LSB) to the registers L_2 and L_1 respectively. The key schedule function expands an 80-bit user-provided key k_i $(0 \leq i < 80)$ into a 508-bit subkey sk_i $(0 \leq i < 508)$ by the following linear operations,

$$sk_i = \begin{cases} k_i, & 0 \leq i < 80 \\ k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13}, & 80 \leq x < 508. \end{cases}$$

Also, the two non-linear functions are defined as follows:

$$\mathcal{NF}_1(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a$$

$$\mathcal{NF}_2(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b,$$

where IR is the pre-defined round constant value (see the specification in [230]), and k_a, k_b are the two subkey bits. The selection of the bits $x_i, 1 \leq i \leq 5$ and $y_i, 1 \leq i \leq 6$ are defined for each variant independently, and are listed in Table 7.2. For KATAN32, the i-th round function is depicted in Figure 7-4, where $k_a \leftarrow k_{2i}$ and $k_b \leftarrow k_{2i+1}$. Finally, after 254 rounds, the values of registers are output as a ciphertext. For KATAN48, the non-linear functions $\mathcal{NF}_1$ and $\mathcal{NF}_2$ are applied twice in one round of the cipher, i.e., the first pair of $\mathcal{NF}_1$ and $\mathcal{NF}_2$ is applied, and then after the update of the registers, they have applied again using the same subkeys. Similarly, in KATAN64, each round applies $\mathcal{NF}_1$ and $\mathcal{NF}_2$ three times with the same key bits. More details about the specification of KATAN-family of ciphers can be found in [230].

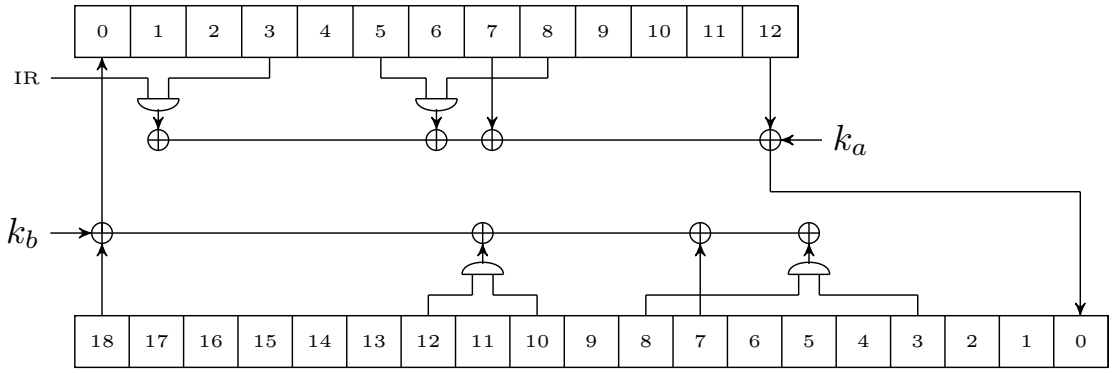


Figure 7-4: Round Function of KATAN32

Table 7.2: Parameters of KATAN Variants

KATAN Variants	$\mid L_1 \mid$	$\mid L_2 \mid$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

7.3 Introducing DEEPAND Modeling

In this section, we introduce the basic idea behind DEEPAND which attempts to generalize the way AND gates are modeled by proposing a systematic way to capture the correlation between AND gates. We first revisit the difference distribution of the output of an AND gate under certain restrictions on the inputs. We then show how the refined model given in [49] can be interpreted as a special case of DEEPAND. We later show how DEEPAND can better capture correlations in both single and multiple AND based NLFSRs.

Our first goal is to look at the difference distribution of an AND gate. Consider an AND gate A_1 with (a, b) as its input, $(\Delta a, \Delta b)$ as its input difference, and Δz as its output difference. Then, the output difference Δz can be expressed as shown in Equation 7.1.

$$\begin{aligned} \Delta z &= A_1(a, b) \oplus A_1(a \oplus \Delta a, b \oplus \Delta b) \\ &= (a \cdot b) \oplus (a \oplus \Delta a) \cdot (b \oplus \Delta b) \\ &= (a \cdot \Delta b) \oplus (b \cdot \Delta a) \oplus (\Delta a \cdot \Delta b) \end{aligned} \tag{7.1}$$

Table 7.3: Difference Distribution Table of AND Gate

a	b	Δa	Δb	Δz
0	0	0	0	0
		0	1	0
		1	0	0
		1	1	1
0	1	0	0	0
		0	1	0
		1	0	1
		1	1	0
1	0	0	0	0
		0	1	1
		1	0	0
		1	1	0
1	1	0	0	0
		0	1	1
		1	0	1
		1	1	1

The distribution of Δz corresponding to all values of (a, b) and $(\Delta a, \Delta b)$, is shown in Table 7.3 from where it is evident that for a given non-zero input difference $(\Delta a, \Delta b)$ of A_1, $\Pr(\Delta z = 0) = \Pr(\Delta z = 1) = 2^{-1}$, i.e., it behaves uniformly. However, under certain conditions, Δz behaves non-uniformly An example of this non-uniform behavior is shown in Example 2. From Table 7.3, the following observations have been made.

Example 2. $\Pr[\Delta z = 0|(a = 0, \Delta a = 0, \Delta b = 1)] = 1$

Observation 1. *If the value of a, b, Δa, and Δb are known, then Δz becomes deterministic*[1]*.*

Observation 2. *If $\Delta a = 0$, $\Delta b = 1$, and the value of a is known, then Δz can be determined with probability 1. Similarly, if $\Delta a = 0$, $\Delta b = 1$, and the value of Δz is known, then 'a' can be guessed deterministically.*

Remark. If $\Delta a = 0$, $\Delta b = 1$, then from Equation 7.1, $\Delta z = a$. This means, for an input difference $(\Delta a, \Delta b) = (0, 1)$, if a is known, then Δz is also known and vice versa.

[1]Observation 1 may seem trivial but it has been included for the sake of completeness.

Observation 3. *If $\Delta a = 1$, $\Delta b = 0$ and the value of b is known then Δz can be determined with probability 1. Similarly, if $\Delta a = 1$, $\Delta b = 0$ and the value of Δz is known, then b can be guessed deterministically.*

Remark. The explanation is similar to the explanation of Observation 2.

Based on the above observations from Table 7.3, it is evident that the distribution Δz directly depends on the input bits a, b when the input difference $(\Delta a, \Delta b)$ is fixed. According to Equation (7.1), the Δz can be rewritten in the following way.

$$\Delta z = \begin{cases} 0, & \text{if } (\Delta a, \Delta b) = (0,0), \\ a, & \text{if } (\Delta a, \Delta b) = (0,1), \\ b, & \text{if } (\Delta a, \Delta b) = (1,0), \\ a \oplus b \oplus 1, & \text{if } (\Delta a, \Delta b) = (1,1), \end{cases} \tag{7.2}$$

We refer to the view captured by Equation (7.2) as DEEPAND. With the DEEPAND view of Δz in place, we are in a position to revisit the refined model proposed by Saha *et al.* [49] for TinyJAMBU.

7.3.1 Refined Modeling as a Special Case of DEEPAND

We start by restating the observation made by Saha *et al.* in the so-called *Refined* Model. Consider two AND gates A_1 with (a, b), and A_2 with (b, c) as their inputs, i.e., they both share a common input as b and hence referred to as *correlated*. Also, let $(\Delta a, \Delta b)$, $(\Delta b, \Delta c)$ are the input differences, and $\Delta z_1, \Delta z_2$ are the output differences of A_1, A_2 respectively. The primary observation in [49] was that when $(\Delta a, \Delta b) = (1, 0)$ and $(\Delta b, \Delta c) = (0, 1)$ then $\Delta z_1 = \Delta z_2 = b$. This implies that, for two correlated AND gates A_1 and A_2, when $(\Delta a, \Delta b, \Delta b) = (1, 0, 1)$, then both the output differences are 0 with probability 2^{-1} or 1 with probability 2^{-1}. Whereas, for two un-correlated AND gates this figure would have been 2^{-2}. Lemma 2 gives a separate perspective on the two correlated AND gates based on the Observations 2 and 3.

Lemma 2. Let the input difference to two correlated AND gates be $(\Delta a, \Delta b)$ and $(\Delta b, \Delta c)$ respectively, and corresponding output differences be Δz_1 and Δz_2 respectively. If $\Delta a = 1$, $\Delta b = 0$, $\Delta c = 1$, then $\Pr[\Delta z_1 = \Delta z_2] = 1$.

Proof. First, the value of Δz_1 is computed. For $(\Delta a, \Delta b) = (1, 0)$, it follows from Observation 3 that $\Delta z_1 = b$. Also, for the second AND gate with $(\Delta b, \Delta c) = (0, 1)$, $\Delta z_2 = b$ (from Observation 2). Hence, we have, $\Pr[\Delta z_1 = \Delta z_2] = \Pr(b) = 1$. □

Remark. It is worth mentioning that despite being one of the first attempts In [49], the authors do not explicitly give a systematic way to capture the correlation between two AND gates. Moreover, the authors have not considered the Observations 2 and 3 in their refined model. In this work, these two observations along with Observation 1 are exploited to penetrate more rounds for NLFSR-based ciphers.

7.4 DEEPAND Modeling of NLFSR-based Ciphers

A NLFSR is a shift register whose input bit, often called a *feedback bit*, is a non-linear function of its previous state. In this section, we will first review some different classes of NLFSRs based on the number of AND gates that are used to define a non-linear feedback function. We will then state the explicit form of these NLFSRs. Finally, we will describe how DEEPAND leads to a general attack framework to capture correlations among single and multiple AND gates.

7.4.1 Case-1: Single AND Based NLFSR

Any n-bit cipher based on the NLFSR-based keyed permutation with single AND gate can be further classified into two cases. In each round of the cipher, the first one is to feed the feedback bit using the non-linear function to the most significant bit (MSB) in the state and then shift each bit towards the least significant bit (LSB) (see Figure 7-1). Similarly, for the second one, compute the feedback bit and then feed it into the LSB, and then shift each bit towards MSB. We now give the explicit form of these two NLFSRs.

7.4.1.1 Computing Forward Differential

Consider an n-bit NLFSR-based cipher $\overleftarrow{\mathcal{C}}$ with s^0 being its initial state value, where $s^0 = (s_0^0, s_1^0, \cdots, s_{n-1}^0)$. Then, for each round number $i, 1 \leq i \leq l$, the feedback bit f^i is computed first, in the following way:

$$f^i \leftarrow s_0^{i-1} \oplus s_{j_1}^{i-1} \oplus \cdots \oplus s_{j_m}^{i-1} \oplus s_{u_1}^{i-1} s_{v_1}^{i-1} \oplus K_{(i-1) \bmod |K|}.$$

where $0, j_1, \cdots, j_m$ are the tap bit positions of the NLFSR and u_1, v_1 $(u_1 < v_1)$ are the input bits to the AND gate. Then, the state bits in the next round (round $i+1$) are updated as follows:

$$s_j^i = \begin{cases} s_{j+1}^{i-1}, & \text{for } 0 \leq j \leq (n-2) \\ f^i, & \text{for } j = n-1 \end{cases}$$

Consider a similar cipher $\overrightarrow{\mathcal{C}}$, whose tap bits are the same as that of $\overleftarrow{\mathcal{C}}$. The only difference is that the bits are shifted in opposite direction as that of $\overleftarrow{\mathcal{C}}$ and in the feedback function s_{n-1}^{i-1} is XOR-ed instead of s_0^{i-1}. The cipher $\overrightarrow{\mathcal{C}}$ is called *reverse-fed* cipher of $\overleftarrow{\mathcal{C}}$. The feedback bit f^i for $\overrightarrow{\mathcal{C}}$ is computed as follows:

$$f^i = s_{j_1}^{i-1} \oplus \cdots \oplus s_{j_m}^{i-1} \oplus s_{n-1}^{i-1} \oplus s_{u_1}^{i-1} s_{v_1}^{i-1} \oplus K_{(i-1) \bmod |K|}.$$

and

$$s_j^i = \begin{cases} s_{j-1}^{i-1}, & \text{for } 1 \leq j \leq (n-1) \\ f^i, & \text{for } j = 0 \end{cases}$$

To find the differential characteristics for such ciphers $\overleftarrow{\mathcal{C}}$, $\overrightarrow{\mathcal{C}}$, the probability is only paid for the active AND gates through rounds. Thus, given an l round differential characteristic, the overall probability can be calculated by counting only the total active ANDs in the characteristic. Also, it is to be noted that, the whole state bits become unknown after n rounds. In another way, we can say that exactly $n-i$ state bits are still known for the initial i $(1 \leq i \leq n)$ rounds. Therefore, in the chosen plaintext scenario, we can deterministically bypass some of the active AND gates by fixing the message bits for up to some initial i $(\leq n)$ rounds. This characteristic of any NLFSR-based ciphers $\overleftarrow{\mathcal{C}}$,$\overrightarrow{\mathcal{C}}$ is described in the following lemma.

Lemma 3. For cipher $\overrightarrow{\mathcal{C}}$, forward differential characteristic for the first $(u_1 + 1)$ rounds is completely free. For the next $(v_1 - u_1)$ rounds, if the input differential to the AND gate is 0 and 1 (i.e., $\Delta s_{u_1} = 0, \Delta s_{v_1} = 1$) then the output of the AND gate

can be determined with probability 1 (**conditionally free**[2]). Similarly, for a cipher $\overleftarrow{\mathcal{C}}$, $(n - v_1)$ rounds are completely free and $(v_1 - u_1)$ rounds are conditionally free.

Proof. As both the inputs to AND gate are known for the first $(u_1 + 1)$ rounds, the output difference of the AND gate can be computed with probability 1. For the next $(v_1 - u_1)$ rounds, the u_1-th bit in the state, i.e., s_{u_1} is still known to us from the given input message. Therefore, at the intermediate rounds i $(u_1 + 1 < i \leq v_1)$ if the input difference corresponding to the AND gate becomes $(0, 1)$, i.e., $\Delta s_{u_1} = 0$ and $\Delta s_{v_1} = 1$, then by Observation 2 the output difference of the AND gate can be deterministically bypassed. The proof for the cipher $\overleftarrow{\mathcal{C}}$ follows a similar approach. □

Note that, in the chosen plaintext attack model (CPA), Lemma 3 can be exploited by carefully choosing the message bits. This, in turn, reduces the *degrees of freedom* of the message space.

7.4.1.2 Computing Backward Differential

While computing the backward differential for a cipher $\overleftarrow{\mathcal{C}}$, the feedback function remains almost the same except only the index of the bits is changed. Consider that the initial state is t^0 and the intermediate state after the i^{th} round is t^i. Then the feedback bit, f^i for the i^{th} round is computed in the following way:

$$f^i \leftarrow t^{i-1}_{j_1-1} \oplus \cdots \oplus t^{i-1}_{j_m-1} \oplus t^{i-1}_{n-1} \oplus t^{i-1}_{u_1-1} t^{i-1}_{v_1-1} \oplus K_{(i-1) \bmod |K|}$$

and the state bits are updated as follows:

$$t^i_j = \begin{cases} t^{i-1}_{j-1}, & \text{for } 1 \leq j \leq (n-1) \\ f^i, & \text{for } j = 0. \end{cases}$$

Similarly, for cipher $\overrightarrow{\mathcal{C}}$, the feedback bit is computed as

$$f^i = t^{i-1}_{j_1+1} \oplus \cdots \oplus t^{i-1}_{j_m+1} \oplus t^{i-1}_0 \oplus t^{i-1}_{u_1+1} t^{i-1}_{v_1+1} \oplus K_{(i-1) \mod |K|}$$

[2]Conditionally free AND is a scenario where, if the input difference to the AND gate is either (0,1) or (1,0), the output difference can be fixed by an attacker. This is done by selecting a message bit that will be one of the inputs to the AND gate.

and

$$t_j^i = \begin{cases} t_{j+1}^{i-1}, & \text{for } 0 \leq j \leq (n-2) \\ f^i, & \text{for } j = n-1. \end{cases}$$

Lemma 4. For cipher $\overrightarrow{\mathcal{C}}$, backward differential characteristic for first $(n - v_1 - 1)$ rounds is completely free. The next $(v_1 - u_1)$ rounds are conditinally free. Similarly, for cipher $\overleftarrow{\mathcal{C}}$, the first $(u_1 - 1)$ rounds are completely free whereas the next $(v_1 - u_1)$ rounds are conditionally free.

Proof. The proof is quite similar to that of Lemma 3 □

7.4.2 Case-2: Multiple AND Based NLFSR

Consider an n-bit NLFSR-based block cipher $\overrightarrow{\mathcal{D}}$ with the initial state value as $s^0 = (s_0^0, s_1^0, \cdots, s_{n-1}^0)$. At each round i, the feedback bit f^i is computed in the following way.

$$f^i \leftarrow s_{j_1}^{i-1} \oplus \cdots \oplus s_{j_m}^{i-1} \oplus s_{n-1}^{i-1} \oplus s_{u_1}^{i-1} s_{v_1}^{i-1} \oplus \cdots \oplus s_{u_h}^{i-1} s_{v_h}^{i-1} \oplus K_{i-1},$$

where

- k^{i-1} is the key bit used in the i^{th} round,
- $j_1, \cdots, j_m, n-1$ are the taps of the NLFSR,
- u_j, v_j are the inputs to the AND gate A_j such that $u_j < v_j \leq n-1$, $1 \leq j \leq h$,
- $j_1 < j_2 \implies u_{j_1} < u_{j_2}$.

Also, the state in the next round is updated in the following way.

$$s_j^i = \begin{cases} s_{j-1}^{i-1}, & \text{for } 1 \leq j \leq (n-1) \\ f^i, & \text{for } j = 0. \end{cases}$$

Lemma 5. For a cipher $\overrightarrow{\mathcal{D}}$, in the forward differential, the output of gate A_j is deterministic for the first $(u_j + 1)$ rounds. For the next $(v_j - u_j)$ rounds, the output of the AND gate is conditionally free. Similarly, for a cipher $\overleftarrow{\mathcal{D}}$, the *reverse-feed* cipher of $\overrightarrow{\mathcal{D}}$, the output of gate A_j is deterministic for the first $(n - v_j)$ rounds and conditionally free for the next $(v_j - u_j)$ rounds.

Proof. For cipher $\overrightarrow{\mathcal{D}}$, as $s^i_{u_j}$ and $s^i_{v_j}$ are known for $0 \leq i \leq u_j$, so ΔA_j can be deterministically computed for the first $(u_j + 1)$ rounds as both inputs to the AND gate are known.

Suppose, during the intermediate rounds, $s^i_{v_j}$ is known and $s^i_{u_j}$ is unknown for $u_j+1 \leq i \leq v_j$ (round number $u_j + 2$ to $v_j + 1$). If $\Delta s^i_{v_j} = 0$ and $\Delta s^i_{u_j} = 1$, then by Observation 3, $\Delta A_j = s^i_{v_j}$. Hence, for round $u_j + 1$ to v_j, ΔA_j can be determined with probability 1 when such conditions are met.

For cipher $\overleftarrow{\mathcal{D}}$, $s^i_{u_j}$ and $s^i_{v_j}$ are known for $0 \leq i \leq (n - v_j - 1)$. Hence, ΔA_j can be determined completely free for the first $(n - v_j)$ rounds. $s^i_{u_j}$ is known and $s^i_{v_j}$ is unknown for $(n - v_j) \leq i \leq (n - u_j - 1)$ (round number $(n - v_j + 1)$ to $(n - u_j)$). If $\Delta s^i_{v_j} = 1$ and $\Delta s^i_{u_j} = 0$, then by Observation 2, $\Delta A_j = s^i_{u_j}$. Therefore, for the next $(v_j - u_j)$ rounds, ΔA_j can be determined with probability 1 when such conditions are met. □

In the same fashion, computing the backward differential, the feedback bit f^i for i^{th} round is computed as

$$f^i \leftarrow t^{i-1}_{l_1+1} \oplus \cdots \oplus t^{i-1}_{l_m+1} \oplus t^{i-1}_0 \oplus t^{i-1}_{u_1+1} t^{i-1}_{v_1+1} \oplus \cdots \oplus t^{i-1}_{u_h+1} t^{i-1}_{v_h+1} \oplus k'^{i-1}$$

and the state in the next round is updated as

$$t^i_j = \begin{cases} t^{i-1}_{j-1}, & \text{for } 0 \leq j \leq (n-2) \\ f^i, & \text{for } j = n-1. \end{cases}$$

Lemma 6. For cipher $\overrightarrow{\mathcal{D}}$, in the backward differential, the output of gate A_j is deterministic for first $(n - v_j - 1)$ rounds. For the next $(v_j - u_j)$ rounds, the output of the gate is conditionally free. Similarly, for a cipher $\overleftarrow{\mathcal{D}}$, the *reverse-feed* cipher of $\overrightarrow{\mathcal{D}}$, in the backward differential the output of gate A_j is deterministic for first (u_j) rounds and conditionally free for next $(v_j - u_j)$ rounds.

Proof. For cipher $\overrightarrow{\mathcal{D}}$, as $t^i_{u_j+1}$ and $t^i_{v_j+1}$ are known for $0 \leq i \leq n - v_j - 2$, so ΔA_j can be deterministically computed for first $(n - v_j - 1)$ rounds as both inputs to the AND gate are known.

Also, $t^i_{v_j+1}$ is known and $t^i_{u_j+1}$ is unknown for $n - v_j - 1 \leq i \leq n - u_j - 2$ (round number $n - v_j$ to $n - u_j - 1$). If $\Delta t^i_{u_j+1} = 0$ and $\Delta t^i_{v_j} = 1$, then by Observation 2, $\Delta A_j = t^i_{u_j+1}$. Hence, for round $n - v_j$ to $n - u_j - 1$, ΔA_j can be determined with probability 1 when such conditions are met.

In similar way, it can be proved for $\overleftarrow{\mathcal{D}}$. □

7.4.3 Generalization of Chained ANDs

Consider an n-bit cipher $\mathcal{C}$ with $(s_{u_1}, s_{u_2}), (s_{u_2}, s_{u_3})$ and $(\Delta s_{u_1} = 1, \Delta s_{u_2} = 0), (\Delta s_{u_2} = 0, \Delta s_{u_3} = 1)$ are respectively two sequential inputs and their differences to the AND gate. Suppose we have differential characteristics and at the round i, we see that the internal difference $\Delta s_{u_1} = 1, \Delta s_{u_2} = 0$ happens at the AND gate and Δz be the corresponding output difference. Then, according to Observation 3, the internal state bit s_{u_2} will be revealed due to the relation $\Delta z = s_{u_2}$. Thus, after the $(u_2 - u_1 - 1)$ rounds, i.e., at the round $i + (u_2 - u_1 - 1)$, $\Delta s_{u_2} = 0, \Delta s_{u_3} = 1$ becomes the input difference to the AND gate. In this case, by Observation 2, this active AND gate will be freely bypassed as we know the bit value s_{u_2}. Therefore, if the subsequent input differences to the AND gate are $1, 0, 1$ then instead of paying the probability of $\frac{1}{4}$, we only have to pay the probability of $\frac{1}{2}$. In another way, we can say that when this subsequent $1, 0, 1$ bit difference arises in the AND gate, we will count it as one active AND. Because, out of two subsequent active ANDs, we only pay the probability for the first one (i.e., when $\Delta s_{u_1} = 1, \Delta s_{u_2} = 0$) whereas the second (where $\Delta s_{u_2} = 0, \Delta s_{u_3} = 1$) one will pass with probability 1.

In the refined modeling paper [49] introduced for TinyJAMBU, the authors added some extra constraints in the simple MILP model and recorded all the two subsequent ANDs with $1, 0, 1$ bit differences which helps to increase the overall probability of the differential characteristic. We named this kind of two subsequent ANDs with $1, 0, 1$ bit differences as Chained AND Bit Pattern (BAND). Now, if we consider a NLFSR with multiple ANDs-based cipher, then there might arise more than two subsequent ANDs with various bit difference patterns that might significantly increase the overall probability of the characteristic and we named it as Multiple AND Bit Pattern (MAND). Before going to define it, we give one example to show how MAND increases

the probability in the characteristic.

Example 3. Suppose, we have an n-bit cipher $\overrightarrow{\mathcal{D}}$ with two ANDs, where $n = 32$ and $(3, 8), (10, 12)$ are the two different AND's input positions in the NLFSR state. At the round i, we assume that a particular bit difference $\Delta s_8 = 1$, $\Delta s_5 = 1$, $\Delta s_3 = 0$, $\Delta s_1 = 1$, and $\Delta s_{-2} = 0$[3] happens in the state. Also, we choose the bit difference 0 at the third position in the state as a pivot. In the subsequent rounds, this pivot will activate some related AND gates, and then it helps to freely pass some subsequent ANDs in the following way.

1. At round i, since $\Delta s_8 = 1, \Delta s_3 = 0$ happens, we get the information of the state bit at the pivotal position according to Observation 2.

2. Then, at the round $i+7$, the pivot goes to the bit position 10 and activates the second AND gate as $\Delta s_{12} = 1, \Delta s_{10} = 0$. Thus, according to the Observation 3, this active AND will be freely passed.

3. Similarly, when the pivot goes to the 12th position in the state at the round $i+9$, the AND will be passed deterministically according to the Observation 2.

The above steps are summarized in the Table 7.4a. In this example, we have to only pay the probability of 2^{-1} instead of 2^{-3}, as the total number of active ANDs subject to the pivot is 3.

Table 7.4: Examples of MAND and BAND

(a) An Example of MAND

Round	NLFSR State Bit Positions[4]						
	Δs_{12}	Δs_{10}	Δs_8	Δs_5	Δs_3	Δs_1	Δs_{-2}
i	0	0	1	1	0	1	0
$i+5$	0	1	0	-	0	-	-
$i+7$	1	0	1	0	-	-	-
$i+9$	0	1	-	-	-	-	-

(b) An Example of BAND

Round	NLFSR State Bit Positions[5]		
	Δs_{21}	Δs_{24}	Δs_{27}
i	1	0	1
$i+3$	-	1	0

[3]This is an imaginary position. This bit difference value $\Delta s_{-2} = 0$ is shifted to the position at 0, i.e., $\Delta s_0 = 0$ after two rounds.

[4]The bit values in $\Delta s_8, \Delta s_5, \Delta s_3, \Delta s_1$, and Δs_{-2} are shown in orange, green, red, violet, and brown colors respectively.

[5]The bit values in $\Delta s_{21}, \Delta s_{24}$, and Δs_{27} are shown in blue, red, and green colors respectively.

Let us denote Δs_j^i to be the state difference Δs_j at round i and s_j^i to be the state value s_j at round i. Also, for ciphers like $\overrightarrow{\mathcal{C}}$ and $\overrightarrow{\mathcal{D}}$, we use $\Delta s_{u_1}^i$ to be the the pivotal bit difference at the position u_1, the first AND bit position. We now furnish the formal definitions of BAND and MAND for ciphers $\overrightarrow{\mathcal{C}}$ and $\overrightarrow{\mathcal{D}}$ respectively. They can be defined similarly for ciphers like $\overleftarrow{\mathcal{C}}$ and $\overleftarrow{\mathcal{D}}$.

Definition 8 (Bi-AND Bit Pattern - BAND)**.** Consider the cipher $\overrightarrow{\mathcal{C}}$ with (u_1, v_1) as its input position of the AND gate. BAND of a pivotal bit difference ($\Delta s_{v_1}^i = 0$) is denoted by $\mathcal{B}_i^{\overrightarrow{\mathcal{C}}}$ and is defined as a bit string in the following way.

$$\mathcal{B}_i^{\overrightarrow{\mathcal{C}}} = l_1 \overbrace{\Delta s_{u_1}^i}^{\text{pivot}} r_1, \text{ where } \begin{cases} l_1 = \Delta s_{u_1}^i & \text{when } \Delta s_{u_1}^i \text{is at } u_1 \\ r_1 = \Delta s_{u_1}^{i+(v_1-u_1)} & \text{when } \Delta s_{u_1}^i \text{is at } v_1 \end{cases}$$

Example 4. Consider an NLFSR-based block cipher $\overrightarrow{\mathcal{C}}$ with $n = 32$ and $(24, 27)$ as the inputs to the AND gate A_1. Let us assume that, at round $i(> 42)$, particular bit differences of $\Delta s_{21} = \mathbf{1}$, $\Delta s_{24} = \mathbf{0}$, and $\Delta s_{27} = \mathbf{1}$ occur in the state (see Table 7.4b). Then the BAND of the pivot $\Delta s_{24}^i = \mathbf{0}$, $\mathcal{B}_i^{\overrightarrow{\mathcal{C}}}$ is given as below.

$$\begin{aligned} \mathcal{B}_i^{\overrightarrow{\mathcal{C}}} = l_1 \boxed{\Delta s_{24}^i} r_1 &= \Delta s_{24}^{i+3} \boxed{\Delta s_{24}^i} \Delta s_{27}^i \\ &= \Delta s_{21}^i \boxed{\Delta s_{24}^i} \Delta s_{27}^i, \quad [\because \Delta s_b^{i+a} = \Delta s_{b-a}^i] \end{aligned}$$

Definition 9 (Multiple AND Bit Pattern - MAND)**.** Consider the cipher $\overrightarrow{\mathcal{D}}$ with (u_1, v_1), $\cdots$, (u_h, v_h) denoting respectively input positions to h AND gates. The MAND of a pivotal bit difference ($\Delta s_{u_1}^i = 0$) is denoted by $\mathcal{M}_i^{\overrightarrow{\mathcal{D}}}$ and is defined as a $(2h+1)-$bit string in the following way:

$$\mathcal{M}_i^{\overrightarrow{\mathcal{D}}} = l_h\ l_{h-1}\ \cdots\ l_1 \overbrace{\Delta s_{u_1}^i}^{\text{pivot}} r_1\ \cdots\ r_{h-1}\ r_h, \text{ where, } \exists \text{ some } p \in \{1, \cdots, h\}$$

$$\text{such that } \begin{cases} l_p = \Delta s_{v_p}^{i+(u_p-u_1)} & \text{when } \Delta s_{u_1}^i \text{ is at } u_p \\ r_p = \Delta s_{u_p}^{i+(v_p-u_1)} & \text{when } \Delta s_{u_1}^i \text{ is at } v_p \end{cases}$$

When there is exactly a single $p \in \{1, \cdots, h\}$ such that $l_p = r_p = 1$, $\mathcal{M}_i^{\overrightarrow{\mathcal{D}}}$ collapses to a BAND which can hence be interpreted as a specific instance of a MAND. With

the above formalisms in place, we can now revisit Example 3 where the MAND of the pivot $\Delta s_3^i = \mathbf{0}$ can be captured as below.

$$\begin{aligned}\mathcal{M}_i^{\vec{\mathcal{D}}} = l_2 l_1 \boxed{\Delta s_3^i} r_1 r_2 &= \Delta s_{10}^{i+9} \Delta s_3^{i+5} \boxed{\Delta s_3^i} \Delta s_8^i \Delta s_{12}^{i+7} \\ &= \Delta s_1^i \Delta s_{-2}^i \boxed{\Delta s_3^i} \Delta s_8^i \Delta s_5^i, \quad [\because \Delta s_b^{i+a} = \Delta s_{b-a}^i]\end{aligned}$$

We can demonstrate that the probability of a particular characteristic in an AND-based cipher ($\vec{\mathcal{D}}$) can be significantly increased due to the occurrence of MANDs. To detect the MANDs, we need to introduce variables that represent the output differences of the AND gates in the intermediate rounds. For $p \in \{1, \cdots, h\}$, we define $\Delta A_p^{i,u_p}$ and $\Delta A_p^{i,v_p}$ as the output differences of AND gate A_p when the pivotal bit difference $\Delta s_i^{u_1}$ moves to positions u_p and v_p, respectively. When certain MANDs occur at the intermediate rounds (out of a total of $2h+1$ bit patterns), we can establish relationships among $s_{u_1}^i$, $\Delta A_p^{i,u_p}$, and $\Delta A_p^{i,v_p}$. These relationships can help us understand how the occurrence of MANDs affects the characteristic probability. We have already established how BAND is a special case of MAND. The following lemma captures the behavior of BAND with regards to variables $\Delta A_p^{i,u_p}$, and $\Delta A_p^{i,v_p}$ introduced above. Later we use the notion of $\mathcal{M}_i^{\vec{\mathcal{D}}}$-weight in subsequent lemmas to highlight the gain in characteristic propagation probability that ensues due to MANDs.

Lemma 7. Consider a BAND with $\mathcal{B}_i^{\vec{\mathcal{C}}} = l_1 \ \Delta s_{u_1}^i \ r_1$. If $l_1 = r_1 = 1$, then $\Delta A_1^{i,u_1} = \Delta A_1^{i,v_1}$.

Proof. According to the Observation 3, if $l_1 = 1$ and $\Delta s_{u_1}^i = 0$, then $\Delta A_1^{i,u_1} = s_{u_1}^i$. Similarly, as $r_1 = 1$ and $\Delta s_{u_1}^i = 0$, we have $\Delta A_1^{i,v_1} = s_{u_1}^i$. Hence, we can conclude that $\Delta A_1^{i,u_1} = \Delta A_1^{i,v_1}$. □

Definition 10 (MAND-weight)**.** The weight of a MAND $\mathcal{M}_i^{\vec{\mathcal{D}}}$, denoted by $wt(\mathcal{M}_i^{\vec{\mathcal{D}}})$ captures its Hamming weight.

Lemma 8. Consider a MAND with $\mathcal{M}_i^{\vec{\mathcal{D}}} = l_h \ \cdots \ l_1 \ \Delta s_{u_1}^i \ r_1 \ \cdots \ r_h$ and $wt(\mathcal{M}_i^{\vec{\mathcal{D}}}) =$

$p+q$. For $\{w_1, \cdots, w_p\}$ and $\{y_1, \cdots, y_q\} \subset \{1, \cdots, h\}$,

$$l_{w_1} = \cdots = l_{w_p} = r_{y_1} = \cdots = r_{y_q} = 1$$
$$\implies \Delta A_{w_1}^{i,u_{w_1}} = \cdots = \Delta A_{w_p}^{i,u_{w_p}} = \Delta A_{y_1}^{i,v_{y_1}} = \cdots = \Delta A_{y_q}^{i,v_{y_q}}$$

Proof. By Observation 3, if $l_{w_g} = 1$ and $\Delta s_{u_1}^i = 0$ then $\Delta A_{w_g}^{i,u_{w_g}} = s_{u_1}^i$ holds $\forall g \in \{1, \cdots, p\}$. Similarly, as $r_{y_g} = 1$ and $\Delta s_{u_1}^i = 0$, $A_{y_g}^{i,u_{y_g}} = s_{u_1}^i$ holds $\forall g \in \{1, \cdots, q\}$. Hence, we can conclude that $\Delta A_{w_1}^{i,u_{w_1}} = \cdots = \Delta A_{w_p}^{i,u_{w_p}} = \Delta A_{y_1}^{i,v_{y_1}} = \cdots = \Delta A_{y_q}^{i,v_{y_q}}$. □

Lemma 9. Let $wt(\mathcal{M}_i^{\vec{\mathcal{D}}}) = m$ and $m \geq 2$. Then the subsequent output differences of m active AND gates can be increased to probability 2^{-1} instead of 2^{-m}.

Proof. As $wt(\mathcal{M}_i^{\vec{\mathcal{D}}}) = m$, then Lemma 8 implies that output differences of m AND gates should be equal to $s_{u_1}^i$. Thus the output differences are correlated and the joint propagation probability increases from 2^{-m} to 2^{-1}. □

7.4.4 Experimental Evidence of MAND

The effect of MAND is observed in the 60-round related-key differential characteristic of KATAN48. The characteristic is listed in Table 7.14 with input difference `0x820031400000` and output difference `0x00018000c000`.

The feedback function $f_b(L_2)$ of KATAN48 consists of two AND gates. $L_2[6]$ and $L_2[15]$ are inputs to one AND gate whereas $L_2[13]$ and $L_2[21]$ are inputs to another AND gate. Using the NLFSR description from Section 7.4.2, the following values can be fixed.

$$u_1 = 6 \qquad v_1 = 15 \qquad u_2 = 13 \qquad v_2 = 21$$

Now we find the MAND with respect to the pivot $\Delta s_{u_1}^{103} (= \Delta s_6^{103})$. In particular, we are finding the expression for $\mathcal{M}_{103}^{\vec{\mathcal{D}}}$. ($\vec{\mathcal{D}} =$ KATAN48). From Definition 9,

$$\mathcal{M}_{103}^{\vec{\mathcal{D}}} = l_2 l_1 s_6^{103} r_1 r_2$$

The values for l_2, l_1, r_2, r_1 are needed to be computed. Again from Definition 9, in this case $p \in \{1, 2\}$. Thus,

$$l_2 = \Delta s_{v_2}^{103+(u_2-u_1)} = \Delta s_{21}^{103+(13-6)} = \Delta s_{21}^{110} = 0$$

$$r_2 = \Delta s_{u_2}^{103+(v_2-u_1)} = \Delta s_{13}^{118} = 1$$

$$l_1 = \Delta s_{v_1}^{103+(u_1-u_1)} = \Delta s_{15}^{103} = 0$$

$$r_1 = \Delta s_{u_1}^{103+(v_1-u_1)} = \Delta s_{6}^{112} = 1$$

Hence, $\mathcal{M}_{103}^{\vec{\mathcal{D}}} = 00011$. Now using Lemma 8, we have $y_1 = 1$ and $y_2 = 2$ and the following,

$$\Delta A_1^{103,v_1} = \Delta A_2^{103,v_2} \implies \Delta A_1^{103,15} = \Delta A_2^{103,21}$$

The above equality can also be verified from the characteristic given in Figure 7-5 (in both cases, the key difference is 0). The figure 7-5 shows the last few rounds characteristic of the 60-round (120 iterations) KATAN48 related-key distinguisher. In the figure, the left, middle, and right columns refer to the iteration number, and bit-differences in L_2 and L_1 register respectively. In the L_2 register, the red-colored bits denote the positions 6, 13, 15, and 21 (starting from left).

Now, as $wt(\mathcal{M}_{103}^{\vec{\mathcal{D}}}) \geq 2$, thus from Lemma 9 it can be concluded that MAND is able to deliver an advantage to increase the probability of the differential characteristic. *Note that, the above pattern 00011 can only be captured through MAND. Whereas BAND is* ***not*** *able to capture such patterns.*

In the next section, we showcase, how the advantage that MAND provides can be leveraged in the DEEPAND modeling of NLFSR-based ciphers using MILP.

7.5 MILP Based DEEPAND Modeling for NLFSR

For a given differential characteristic in NLFSR-based ciphers, the probability is calculated by counting the total active AND gates in each round. The objective is to find the optimal characteristic with the minimum active AND gates in a fixed round. In the simple MILP model, the goal is to minimize the non-zero input differences to the AND gates in each round. The authors of [49] studied the impact of AND gates

Figure 7-5: Experimental Demonstration of a MAND Occurrence.

on the characteristic probability for the single AND-based NLFSR cipher TinyJAMBU. They found that subsequent AND gates may depend on each other and form what we in the current work defined as a BAND, which has a significant effect on the characteristic probability. To capture such a BAND, they proposed a refined MILP model for TinyJAMBU. However, as per our investigations, it has been observed that further refinement is possible and a generalized model can be devised to extend the findings to a class of NLFSR-based ciphers. The refined model for capturing BANDs is described in Section 7.5.1. In Section 7.5.2, we describe how to capture MAND for multiple AND based NLFSR ciphers, which automatically includes BAND.

7.5.1 MILP Modeling of BAND

To model $\vec{\mathcal{C}}$ leveraging the BAND $\mathcal{B}_i^{\vec{\mathcal{C}}}$, we use γ_i to capture the correlation among two subsequent active AND gates. For each round, we compute γ_i as $\gamma_i = l_1 \ \overline{\Delta s^i_{u_1}} \ r_1$. According to Lemma 7, we have $\Delta A_1^{i,u_1} = \Delta A_1^{i,v_1}$. Thus for the pivot position at u_1 in the consecutive rounds of the state, the following constraints will be added to the MILP model to capture the correlation in BAND.

$$\gamma_i = l_1\ \overline{\Delta s^i_{u_1}}\ r_1, \qquad \Delta A_1^{i,u_1} - \Delta A_1^{i,v_1} \leq 1 - \gamma_i, \qquad \Delta A_1^{i,v_1} - \Delta A_1^{i,u_1} \leq 1 - \gamma_i$$

7.5.2 MILP Modeling of MAND

The valid patterns of MAND, which captures the dependency among the output differences of subsequent active AND gates, is described in the following Lemma 10.

Lemma 10. The valid patterns (λ) of a MAND $\mathcal{M}_i^{\vec{\mathcal{P}}}$ of an NLFSR-based cipher $\vec{\mathcal{D}}$ with h AND gates is equal to $\sum_{m=2}^{2h} \binom{2h}{m} = 4^h - 2h - 1$.

Proof. Consider a MAND with $wt(\mathcal{M}_i^{\vec{\mathcal{P}}}) = m$. There are $\binom{2h}{m}$ valid patterns of $\mathcal{M}_i^{\vec{\mathcal{P}}}$ which shows the dependency between m subsequent active AND gates. By Lemma 9, for a MAND, if $m \geq 2$, then we have shown a dependency between the output differences of AND gates. Therefore, the total valid MAND will be $\binom{2h}{2} + \binom{2h}{3} + \cdots \binom{2h}{2h} = 4^h - 2h - 1$. □

We would like to emphasize here that MAND captures all possible first-order correlations.[4] For modeling the dependency among the subsequent active AND gates, the approach is quite similar to the model given in [49]. To do so, first, a constraint is used to identify which AND gates are correlated and then pairs of AND gates are considered to model the dependency between them. So, to capture any bit difference

[4]Consider a MAND $\mathcal{M}_i^{\vec{\mathcal{P}}} = l_h \ \cdots \ l_1\ \Delta s^i_{u_1}\ r_1 \ \cdots \ r_h$ of an NLFSR-based cipher $\vec{\mathcal{D}}$ with h AND gates. A higher probability is achieved when pivot $\Delta s^i_{u_1} = 0$ and $wt(\mathcal{M}_i^{\vec{\mathcal{P}}}) \geq 2$. In other words, for the following two cases, there is no increase in the probability

(i) $\Delta s^i_{u_1} = 1$, or

(ii) $\Delta s^i_{u_1} = 0$ and $wt(\mathcal{M}_i^{\vec{\mathcal{P}}}) < 2$.

As discussed in Section 7.3, for a single AND-based NLFSR, only when the input differential pattern corresponding to (Δa, Δb, Δc,) is (1,0,1), an advantage on the output differential of two AND gates is gained. If Δb=1, then there is no advantage for first-order correlation. This can be naively extended for NLFSR with multiple AND gates and thus it can be concluded that when $\Delta s^i_{u_1} = 1$ there is no increase in the probability.

Case (ii) implies that there is at most one active AND gate, thus there is no possibility for first-order correlation between two or more active AND gates in such cases.

pattern in the MAND with $m \geq 2$, we have added some extra constraints corresponding to the chained active AND gates in the simple MILP modeling. As the MAND $\mathcal{M}_i^{\vec{\mathcal{D}}}$ has λ different valid patterns, we take γ_z, $1 \leq z \leq \lambda$ to capture the correlation among $wt(\mathcal{M}_i^{\vec{\mathcal{D}}})$ active AND gates.

Thus for the pivot position at u_1 in the consecutive rounds i of the state, we have λ γ_z and compute them in the following way.

$$\gamma_z = l_{w_1} \cdots l_{w_p} \overline{l_{w_1'}} \cdots \overline{l_{w_{p'}'}} \; \overline{\Delta s_{u_1}^i} r_{y_1} \cdots r_{y_q} \overline{r_{y_1'}} \cdots \overline{r_{y_{q'}'}}$$

$$\text{Where, } \begin{cases} l_{w_1} = \cdots = l_{w_p} = r_{y_1} = \cdots = r_{y_q} = 1 \\ l_{w_1'} = \cdots = l_{w_{p'}'} = r_{y_1'} = \cdots = r_{y_{q'}'} = 0 \\ \text{Such that} \begin{cases} \{w_1, \cdots, w_p\} \cup \{w_1', \cdots, w_{p'}'\} = \{u_1, \cdots, u_h\}, \\ \{w_1, \cdots, w_p\} \cap \{w_1', \cdots, w_{p'}'\} = \varnothing, \\ \{y_1, \cdots, y_q\} \cup \{y_1', \cdots, y_{q'}'\} = \{v_1, \cdots, v_h\}, \\ \{y_1, \cdots, y_q\} \cap \{y_1', \cdots, y_{q'}'\} = \varnothing \end{cases} \end{cases}$$

Lemma 8 implies that $\Delta A_{w_1}^{i,v_{w_1}} = \cdots = \Delta A_{w_p}^{i,v_{w_p}} = \Delta A_{y_1}^{i,v_{y_1}} = \cdots = \Delta A_{y_q}^{i,v_{y_q}}$. Therefore, for each of λ valid bit difference patterns of a MAND, the correlation is captured by the constraints given in Table 7.5. There constraints constitute the DEEPAND model for MILP that is used to find the better differentials for KATAN and TinyJAMBU leading to improved attacks on both the lightweight ciphers which are discussed in the subsequent sections.

Table 7.5: MILP Constraints Pertaining to DEEPAND

$\gamma_z = l_{w_1} \cdots l_{w_p} \overline{l_{w_1'}} \cdots \overline{l_{w_{p'}'}} \; \overline{\Delta s_{u_1}^i} r_{y_1} \cdots r_{y_q} \overline{r_{y_1'}} \cdots \overline{r_{y_{q'}'}}$	
$\left.\begin{array}{l} \Delta A_{w_t}^{i,u_{w_t}} - \Delta A_{w_x}^{i,u_{w_x}} \leq 1 - \gamma_z, \\ \Delta A_{w_x}^{i,u_{w_x}} - \Delta A_{w_t}^{i,u_{w_t}} \leq 1 - \gamma_z, \end{array}\right\}$	$1 \leq t < x \leq p$
$\left.\begin{array}{l} \Delta A_{w_t}^{i,u_{w_t}} - \Delta A_{y_x}^{i,v_{y_x}} \leq 1 - \gamma_z, \\ \Delta A_{y_x}^{i,v_{y_x}} - \Delta A_{w_t}^{i,u_{w_t}} \leq 1 - \gamma_z \end{array}\right\}$	$1 \leq t \leq p$, $1 \leq x \leq q$

7.6 Attacks on TinyJAMBU

The DEEPAND model has been applied to mount attacks on variants of *keyed* permutation $\mathcal{P}_l$, $\hat{\mathcal{P}}_l$ of TinyJAMBU. We start with a brief discussion of the relevant previous attacks before sharing the results obtained in this work to give a perspective on the degree of improvement.

7.6.1 Summary of Relevant Previous Attacks

In this section, we are going to describe the previous results regarding the differential properties of the keyed permutations $\mathcal{P}_l$, $\hat{\mathcal{P}}_l$ of TinyJAMBU and the forgery attack in the TinyJAMBU mode respectively.

7.6.1.1 Differential properties of the keyed permutation $\mathcal{P}_l$.

The designers of TinyJAMBU [232] have specified four different constraints regarding the input-output active-bit positions of $\mathcal{P}_l$ while searching for its differential characteristic.

> Type-I. Input differences only exist in the 32 MSBs. No restriction on the output.
>
> Type-II. No restriction on the input. Output differences only exist in the 32 MSBs.
>
> Type-III. Both the input and output differences only exist in the 32 MSBs.
>
> Type-IV. No restrictions on both the input and output.

Out of these four types, Type 3 is the most relevant to attack the TinyJAMBU mode, whereas Type 4 is useful to analyze the pseudo-randomness of the internal keyed permutations $\mathcal{P}_l$ and $\hat{\mathcal{P}}_l$. Note that, in 2019, the initial submission document of TinyJAMBU AEAD [229, 232] has 384 rounds in P_l. According to the designer's claim, using the simple MILP model, the best probability of a 384-round Type 3 characteristic is approx 2^{-78}. The designers have considered differential trails where each AND gates are treated independently. Then, in 2020, Saha *et al.* [49] have developed a new refined

MILP model, where the authors have formed a cluster differential characteristic on full 384 rounds with the probability of $2^{-70.64}$ by considering the correlations between two different AND gates. For different rounds, a comparison of the active AND gates of the best Type1, Type2, Type 3, and Type 4 characteristics corresponding to the simple model and refined model can be found in [49].

Also, for 320-round Type 4 differential characteristic of $\hat{P}_l$, the designers claimed probability is 2^{-13} (using simple model), whereas it is 2^{-12} according to the refined model.

7.6.1.2 Forgery Attacks on TinyJAMBU Mode.

In order to carry out a forgery in the TinyJambu mode, the attacker must first attempt to create an internal state collision by inducing differences in two consecutive data blocks. This can be achieved by either altering the nonce, associated data, plaintext, or ciphertext in such a way that after one permutation call, the resulting difference is canceled out by the next block difference. The second method involves more than two data blocks and requires at least two permutations. For example, consider two consecutive permutations ($\mathcal{P}l/\hat{\mathcal{P}}l$) using three nonce/associated data blocks. The attacker can first induce a Type 1 difference by altering the 32 most significant bits in the first data block. Then, in the next permutation, a Type 2 difference can be applied so that a specific 32-bit difference can cancel out the state difference. According to their previous submission [232], the probability of a differential forgery attack on nonce/associated data is at most 2^{-73}, while for plaintext/ciphertext it is at most 2^{-115}.

7.6.2 Attacks on Keyed Permutation $\mathcal{P}_l$

In their security analysis of the mode, the designers consider $\mathcal{P}_l$ to be an ideal keyed-permutations which means under a chosen plaintext attack, $\mathcal{P}_l$ cannot be distinguished from a random permutation. This gives us the motivation to evaluate the security of $\mathcal{P}_l$ against differential cryptanalysis as a stand-alone keyed permutation. Furthermore, based on our proposed DEEPAND model, we show that the keyed permutations $\mathcal{P}_l$ and $\hat{\mathcal{P}}_l$ do not behave as a pseudo-random permutations.

7.6.2.1 MILP Modeling for Finding Differential characteristic

As the design of TinyJAMBU is similar to the cipher described in Section 7.4.1, from Lemma 3 it can be concluded that the first $(128-85-1)=42$ rounds are *completely* free and the next $(85-70)=15$ rounds are *conditionally* free. For the rest of the rounds, refined modeling [49] is employed. It is worth mentioning that our findings with complete and conditionally free rounds lead to improvements in the results reported in [49].

To find the differential characteristics of $\mathcal{P}_l$, in addition to the refined model, the Observation 1 and Observation 2 are employed to improve the probability. By Lemma 3, it can be concluded that the first $(128-85-1)=42$ rounds is completely free, but some of the next $(85-70)=15$ rounds are conditionally free when a particular difference pattern $(\Delta s_{70}, \Delta s_{85})=(0,1)$ occurs in the input to the AND gate and s_{70} is completely known. This conditional free scenario is demonstrated in Table 7.6.

Consider the bits 70 and 85 in rounds numbers 43 to 57 of the characteristic given in Table 7.7. It is evident from the table that in rounds 49 and 52, $\Delta s_{70}^{49}=\Delta s_{70}^{52}=0$ and $\Delta s_{85}^{49}=\Delta s_{85}^{52}=1$. As s_{70}^{49} and s_{70}^{52} are known, the output difference of the corresponding AND gate is deterministic. Hence, this gives a factor of 2^2 advantage in the probability. Notice that, although it gives a factor of 2^2 advantage in the probability, parallely it also decreases the message space by the factor of 2^2. However, in general, in both the free and conditionally free cases, the characteristic probability can be increased by fixing some of the input message bit values. So, for the differential

Table 7.6: Part of Differential Characteristic of TinyJAMBU Showing the Effect of Observation 2.

#Rnd	$\Delta s_{70\cdots 85}$	Conditionally Free
42	0000000000000000	No
43	0000000000000000	No
⋮	⋮	⋮
48	0000000000000000	No
49	0000000000000001	Yes
50	0000000000000010	No
51	0000000000000100	No
52	0000000000001001	Yes
⋮	⋮	⋮
57	0000000100100000	No

attack, we need a trade-off between the probability and the message space (the data complexity of the attack).

Table 7.7: Type 4 Differential Characteristics of $\mathcal{P}_{384}$ with Probability 2^{-14}

Input:	$\Delta S_{127\cdots0}$	0x00000000	0x88040000	0x00000248	0x02000043
	$\Delta S_{255\cdots128}$	0x00000000	0x80000000	0x00010000	0x00000012
	$\Delta S_{383\cdots256}$	0x00000000	0x80000000	0x00000000	0x00000000
Output:	$\Delta S_{511\cdots384}$	0x04080000	0x80004000	0x00010200	0x00000010

It should be noted that the use of a single AND gate in TinyJAMBU means that the dependencies between the AND gates (BAND) will remain the same. Our analysis took into account the keyed permutation of TinyJAMBU, so these conditions will remain unaltered. A similar type of differential analysis was performed in [233] using a refined MILP model, which showed that the first 43 rounds are free when both inputs to the AND gate are known. Additionally, we have shown that even when only one input bit of the AND gate is known, the output difference of the AND gate can be deterministic (for rounds 43 to 57). This property was not captured in previous works [49, 233], but we have identified it as the underlying factor behind the DEEPAND model. This same property leads to the modeling of the correlation among multiple AND gates when used in a block cipher like KATAN. If we compare our model with [49], we need to omit the initial free rounds and our model will be similar to theirs. However, if we want to take advantage of the known plaintext scenario, then our model can be better or at least as good as that of [233].

Having said that it is worth highlighting that the completely/conditionally free gates lead to lesser constraints for the initial 57 rounds for the TinyJAMBU permutation which in turn allows the solver to converge faster towards the optimal solution for DEEPAND than the refined model for the same number of rounds. This is one of the possible reasons that DEEPAND model is able to find differential characteristics for more rounds in comparison to the refined model in a stipulated amount of time.

7.6.2.2 Cluster Differential characteristic of $\mathcal{P}_{384}$

By employing the DEEPAND model in MILP, we are able to find better differential characteristics. A comparison of these three models for both Type-IV and Type-I

differences with respect to different rounds is summarized in Table 7.8. For 320 rounds, our model gives a differential characteristic with probability 2^{-8} which is much better than previously reported results. For $\mathcal{P}_{384}$, a Type-IV differential characteristic with probability 2^{-14} is found. The characteristic is shown in Table 7.7. We obtained 4 differential characteristics with the same input and output difference as shown in Table 7.7 each with probability 2^{-14}, 2^{-15}, 2^{-16} and 2^{-17}. Thus the overall probability for the differential is $2^{-13.17}$.

Table 7.8: Best Results for Type-IV and Type-I characteristics of TinyJAMBU Correspond to Different MILP Models.

Number of Rounds	Simple Model [231]		Refined Model [49]		DEEPAND Model	
	Type-IV	Type-I	Type-IV	Type-I	Type-IV	Type-I
128	2	6	2	6	0	5
192	4	13	4	12	2	11
256	8	22	8	20	5	19
320	13	33	12	29	8	28
384	–	45	19	41	14	40?
480	–	–	29?	–	22	–
640	–	88	53?	–	42?	79?
1024	–	–	–	–	108?	–

"?" denotes that the solver has not stopped. Here each entry equals $-\log_2(\text{characteristic}_{\text{Probability}})$

Also, using the DEEPAND model, we have found a Type-III differential trai of $\mathcal{P}_{384}$ with probability 2^{-71}. The input and output differences are given in Table 7.9 that consists of a total of 84 active AND gates among which 6 gates are completely free, 0 gates are conditionally free, and 13 gates are correlated. Therefore to satisfy this Type-III characteristic with probability 2^{-65}, we need to fix precisely 6 bits in the input message. As a result, the message space will become reduced from 2^{128} to 2^{122}. We then evaluated its probability by finding multiple differential characteristics with the same input and output difference, given in Table 7.9. We found 50 distinct characteristics with probability 2^{-70} or more, whose distribution is listed in Table 7.10. By taking account of all these distinct characteristics, the overall probability to satisfy this Type-III differential will become $2^{-61.88}$

7.6.2.3 Differential characteristic of $\mathcal{P}_{640}, \mathcal{P}_{1024}$

The MILP model developed in this work has also been applied to the keyed permutations $\mathcal{P}_{640}$ and $\mathcal{P}_{1024}$ to find the best Type-IV differential characteristic. For, $\mathcal{P}_{640}$, we have found Type-IV and Type-I differential characteristics with probabilities of 2^{-42} and 2^{-79} respectively. We also searched for the best Type-III characteristic of $\mathcal{P}_{640}$ and were able to find a characteristic with probability 2^{-93} (see Table 7.9). However, for $\mathcal{P}_{1024}$, we could only find a differential characteristic with probability 2^{-108}. Note that the solver is unable to find the best characteristics due to a higher round in both the permutations $\mathcal{P}_{640}$,$\mathcal{P}_{1024}$.

7.6.2.4 Related-key Differential characteristic of $\mathcal{P}^{128}_{1024}, \mathcal{P}^{192}_{1152}$, and $\mathcal{P}^{256}_{1280}$

The designers have mentioned that if two related keys are available, then TinyJAMBU has the sliding property which can be prevented by adding the frame bits to the state. Although, for the keyed permutations $\mathcal{P}_l$ in the TinyJAMBU mode, the related-key differential attack is less practical compared to the single-key differential attack, we have applied our DEEPAND model for the keyed permutations $\mathcal{P}^{128}_{1024}, \mathcal{P}^{192}_{1152}$, and $\mathcal{P}^{256}_{1280}$ in the related key setting and found characteristics which are summarized in Table 7.11.

7.6.3 Fixing Saha *et al.*'s Forgery Attack [49]

In this subsection, we show that the forgery attack furnished in [49] has a flaw which makes it ineffective. To be precise, the flaw originates from the lack of entropy or degrees of freedom in generating sufficient messages to create a favorable event for the

Table 7.9: Differential Characteristics of the TinyJAMBU Keyed Permutation $\mathcal{P}_l$

Keyed Permutation	Differential characteristic		
	Type	probability	Masks
$\mathcal{P}_{384}$	Type-III	2^{-65}	Input Difference: 0x048a2000 0x00000000 0x00000000 0x00000000 Output Difference: 0x40800441 0x00000000 0x00000000 0x00000000
$\mathcal{P}_{640}$	Type-III	2^{-93}	Input Difference: 0xc3804381 0x00000000 0x00000000 0x00000000 Output Difference: 0x00000100 0x00000000 0x00000000 0x00000000
$\mathcal{P}_{640}$	Type-IV	2^{-42}	Input Difference: 0x00000204 0x10000080 0x00412000 0x01020800 Output Difference: 0x20409200 0x88000480 0x00001020 0x00024001
$\mathcal{P}_{1024}$	Type-IV	2^{-108}	Input Difference: 0x00308080 0x00002129 0x00000808 0x00420000 Output Difference: 0x40110000 0x02040920 0x00800048 0x00000102

Table 7.10: Multiple Type-III Characteristics and Their Probabilities of $\mathcal{P}_{384}$

Probability	2^{-65}	2^{-66}	2^{-67}	2^{-68}	2^{-69}	2^{-70}
Number of characteristics	3	3	7	10	13	14

Table 7.11: Related-key Differential Characteristics of the TinyJAMBU Keyed Permutations $\mathcal{P}_{1024}^{128}$, $\mathcal{P}_{1152}^{192}$, and $\mathcal{P}_{1280}^{256}$

Keyed Permutation	Differential characteristic		
	Type	probability	Masks
$\mathcal{P}_{1024}^{128}$	Type-IV	2^{-14}	Input Difference: 0x00000000 0x00000000 0x00000004 0x00000000 Output Difference: 0x00000000 0x00000000 0x00000004 0x00000000 Key Difference: 0x20000000 0x00020000 0x00000000 0x00000000
$\mathcal{P}_{1152}^{192}$	Type-IV	2^{-10}	Input Difference: 0x00000000 0x00000000 0x00000000 0x20000000 Output Difference: 0x00000000 0x00000000 0x00000000 0x20000000 Key Difference: 0x01000000 0x00001000 0x00000000 0x20000000 0x00000000 0x20000000
$\mathcal{P}_{1280}^{256}$	Type-IV	2^{-8}	Input Difference: 0x00000004 0x00000000 0x00000000 0x10000000 Output Difference: 0x00000000 0x00000000 0x00000000 0x10000000 Key Difference: 0x00800004 0x00000800 0x00000000 0x10000000 0x00000000 0x00000000 0x00000000 0x10000000

forgery. We restate the attack in order to highlight flaws in the arguments furnished in [49] followed by our fix. In their work Saha *et al.* discuss the forgery attack that can occur during the nonce setup or data processing phase. The attack involves injecting a 32-bit difference Δ_i into the i-th input block and then canceling the state differences by injecting another 32-bit state difference Δ_{i+1} into the $(i+1)$-th input block, which maps to Type-III difference. The attack is based on the existence of a differential characteristic that maps the state difference $(\Delta_i||0^{96})$ to $(\Delta_{i+1}||0^{96})$ through $\mathcal{P}_l$ with probability p.

There are two types of attacks mentioned in the paper. The first one is called the "*probabilistic nonce-reuse almost universal forgery*" where the length of the associated data must be at least two blocks. The attacker repeatedly makes queries to the encryption oracle with the same nonces to observe the tag T. If the observed tag T' is matched with the tag T, the attacker succeeds in making a forgery. This attack breaks the 64-bit security if the differential characteristic $\Delta_i \rightarrow \Delta_{i+1}$ of $\mathcal{P}_l$ has a probability $p \geq 2^{-64}$. The second attack is called the "*nonce-respect almost universal forgery with reforgeability*" where the attacker can choose the first 64 bits (out of 96 bits) of

the nonce $N = N_0||N_1||N_2$, and can make a forgery for any (A, M) immediately after finding N and T that satisfy the nonce-respect requirement. The attacker repeatedly makes queries to the encryption oracle with different nonces to observe the tag T. If the observed tag $T^{'}$ is equal to T, the attacker succeeds in making a forgery. The success probability of this attack is $D \times p$, where D is the number of distinct nonces examined by the attacker. Once the attacker finds a collision, they can obtain a valid tag for any (A, M) by choosing the last 32 bits of the nonce arbitrarily.

Now to satisfy the characteristic $\Delta_i \rightarrow \Delta_{i+1}$ for $\mathcal{P}_l$, the distinct state pairs (D) should be at least $\frac{1}{p}$. Equivalently, we can say that the expected number of state pairs to satisfy a given characteristic $\Delta_i \rightarrow \Delta_{i+1}$ will be $D \times p$. For the second forgery attack, by choosing different N_0, the distinct state pairs $(S, S^{'})$ with $S \oplus S^{'} = \Delta_i$ at the processing of the first nonce block will be $D = 2^{31}$. Note that, in this scenario, altering N_1 does not affect the quantity of state pairs (D). In [49], the authors found a differential characteristic $\Delta_i \rightarrow \Delta_{i+1}$ for $\mathcal{P}_{338}$ with probability $p = 2^{-62.68}$ (by considering multiple characteristics). Thus, for $\mathcal{P}_{338}$, using the second forgery attack, the attacker can find a state collision after exhausting the first two nonce blocks with probability $2^{31} \times 2^{-62.68} \approx 2^{-31.68} (\ll 1)$. Therefore, for $\mathcal{P}_{338}$, the proposed attack ***cannot*** *effectively find a state collision* to break the 64-bit authentication security, i.e., the probability to make a state collision at the first two nonce processing blocks will be $2^{-31.68}$ even though the attacker can make $2^{31} \times 2^{31} (= 2^{62})$ number of Q1 and Q2 queries.

In order to carry out a forgery attack in the nonce-respect scenario, the attacker needs to perform two queries repeatedly:

Q1: The attacker makes a query to the encryption oracle with inputs $(N_0||N_1||N_2, A^*, M^*)$ in order to observe the tag T.

Q2: The attacker makes a related query to the encryption oracle with inputs $(N_0||N_1 \oplus \Delta_i||N_2 \oplus \Delta_{i+1}, A^*, M^*)$ in order to achieve a successful forgery if the observed tag $T^{'}$ is equal to T.

In this scenario, the number of chosen state pairs at the input of the second nonce block for $\mathcal{P}_l$ would be $2^{32} \times 2^{31} = 2^{63}$. This means that if a given characteristic

$\Delta_i \rightarrow \Delta i+1$ has a probability $p \geq 2^{-63}$, then after making queries of Q1 and Q2 for all nonces N_0, N_1, it is expected that there will be at least one state collision at the third nonce block position, which will immediately lead to the forgery. Additionally, if $N_0||N_1||N_2$ and $N_0||N_1 \oplus \Delta_i||N_2 \oplus \Delta_{i+1}$ are two 96-bit nonces that result in a state collision, then the attacker can choose the last 32 bits of nonce $\tilde{N_2}$ ($\neq N_2, N_2 \oplus \Delta_{i+1}$) arbitrarily to obtain a tag T for $(N_0||N_1||\tilde{N_2}, A^*, M^*)$ through an encryption query. Then T will also be valid for $(N_0||N_1 \oplus \Delta_i||\tilde{N_2} \oplus \Delta_{i+1}, A^*, M^*)$ implying a forgery.

According to our analysis using the DEEPAND model for $\mathcal{P}_{384}$, we discovered a differential characteristic with a probability[5] of 2^{-65} when an attacker has the ability to manipulate 6 bits in the input message during encryption. After taking into account multiple characteristics for a differential, the probability increases to $2^{-61.88}$. However, in this forgery attack, the attacker has no control over the initial bits in the message and cannot freely bypass some initial AND gates. Therefore, by not considering the manipulation of the message bits at the initial 57 rounds of TinyJAMBU state, the overall probability decreases to $2^{-67.88}$, which is higher than the original estimations made by Saha *et al.* and the designers. Our DEEPAND model analysis for $\mathcal{P}_{384}$ suggests that the security margin against differential cryptanalysis is less than 4 bits.

7.7 Attacks on KATAN

In this section, to find the best differential characteristics of any rounds in the KATAN ciphers, we will show how the DEEPAND model efficiently captures the correlated ANDs and significantly increases their characteristic probability. First, we will show that the differential characteristics using our DEEPAND model for some initial rounds of KATAN give a much better probability than the designer's claims in [230]. Then, we show that the related key boomerang attack on KATAN in [234] can also be improved by employing this new model.

[5]The attack scenario does not allow for the attacker to control the input message bits in the encryption process, thus we have taken into account the cost of fixing the message bits by multiplying the overall probability by 2^{-6}.

7.7.1 Improved Differential Cryptanalysis of KATAN

7.7.1.1 MILP Modeling of Free Rounds.

In KATAN, there are three AND gates where the tuples $(y_3, y_4), (y_5, y_6)$, and (x_3, x_4) represent the input bit-positions to the AND gates A_1, A_2, and A_3 respectively. Then by Lemma 5, the differential output of the gates A_1, A_2 and A_3 in the forward differential characteristic are deterministic for the first (y_4+1), (y_6+1), and (x_4+1) rounds respectively. Also, they are conditionally free from the round number (y_4+2) to (y_3+1), (y_6+2) to (y_5+1), and (x_4+2) to (x_3+1) respectively.

Similarly, by Lemma 6, it can be concluded that in the backward differential characteristic, the output differences of the gates A_1, A_2 and A_3 are deterministic for the first $(n-y_3-1)$ rounds, first $(n-y_5-1)$ rounds and first $(n-x_3-1)$ rounds respectively and conditionally free from round number $(n-y_3)$ to $(n-y_4-1)$, $(n-y_5)$ to $(n-y_6-1)$ and $(n-x_3)$ to $(n-x_4-1)$ respectively. Here n denotes the state size of KATAN.

7.7.1.2 Modeling the Dependency Between AND Gates

For KATAN, there is only one AND gate in the L_1 register. In this case, a BAND can happen during intermediate rounds. To capture all the BANDs in rounds, we have to track the BAND for each round and then we add the respective constraints according to the MILP model discussed in Section 7.5.2.

In L_2 register, there are two AND gates and the dependency between two different AND gates is not captured in the refined model. Consider a bit s_3^i in register L_2. For KATAN32, the MAND of the pivotal difference $\Delta s_3^i = 0$ is

$$\mathcal{M}_i^{\mathsf{KATAN32}} = \Delta s_{10}^{i+9}\Delta s_3^{i+5}\boxed{\Delta s_3^i}\Delta s_8^i\Delta s_{12}^{i+7} = \Delta s_1^i\Delta s_{-2}^i\boxed{\Delta s_3^i}\Delta s_8^i\Delta s_5^i$$

Now by Lemma 10 there are $\binom{4}{4}+\binom{4}{3}+\binom{4}{2} = 11$ patterns for which output differential of several AND computations are inter-related. The MAND and its corresponding differential bit patterns with refined probabilities are shown in Table 7.12.

Table 7.12: MAND of Δs_3^i and The Corresponding Differential Value of Related Bits.

MAND	Δs_8^i	Δs_5^i	Δs_3^i	Δs_1^i	Δs_{-2}^i	Naive Prob.	Improved Prob.
11[0]11	1	1	[0]	1	1	2^{-4}	2^{-1}
11[0]10	1	1	[0]	1	0	2^{-3}	2^{-1}
11[0]01	1	1	[0]	0	1	2^{-3}	2^{-1}
10[0]11	1	0	[0]	1	1	2^{-3}	2^{-1}
01[0]11	0	1	[0]	1	1	2^{-3}	2^{-1}
11[0]00	1	1	[0]	0	0	2^{-2}	2^{-1}
10[0]10	1	0	[0]	1	0	2^{-2}	2^{-1}
01[0]10	0	1	[0]	1	0	2^{-2}	2^{-1}
10[0]01	1	0	[0]	0	1	2^{-2}	2^{-1}
01[0]01	0	1	[0]	0	1	2^{-2}	2^{-1}
00[0]11	0	0	[0]	1	1	2^{-2}	2^{-1}

7.7.1.3 DEEPAND Based New Differential characteristics for KATAN

In [230], the designers have claimed that for 42-round KATAN32, the best differential characteristic has probability 2^{-11}. However, for the initial 42 rounds, the DEEPAND MILP model is able to find two *identical* differential characteristics with probability 2^{-7}.

For 43-round KATAN48 and 37-round KATAN64, the best differential characteristic, as claimed by the designers, can be found with probability 2^{-18} and 2^{-20} respectively whereas for both variants our model finds differential characteristics with probability 2^{-14} and 2^{-17} respectively.

7.7.2 Related Key Differential Attack

In the related-key setting, the DEEPAND model was applied to the KATAN32 cipher and the best characteristic probabilities for various rounds are summarized in Table 7.14. This model outperforms previous simple and refined models in capturing multiple correlated ANDs. These correlated ANDs not only increase the trail probability but also aid in finding longer differential characteristics. As a result, this model can be used to identify better related-key differential characteristics for the KATAN48 and KATAN64 ciphers compared to the simple and refined models.

Table 7.13: Differentials of KATAN Variants.

Cipher	#R	Active Gates				Difference		Probability	
		t	fr	C_{fr}	C_A	Input	Output	p_α	p
KATAN32	42	7	4	1	0	0x08020040	0x00200420	2^{-7}	2^{-11}
	74	29	2	0	0	0x0000c010	0x40880101	2^{-29}	2^{-31}
	81†	29	5	0	4	0x10802004	0x00000800	2^{-29}	2^{-34}
KATAN48	43	14	10	0	0	0x000008442c10	0x040000000229	2^{-14}	2^{-24}
KATAN64	37	17	3	2	0	0x4000002001000800	0x0444200000001000	2^{-17}	2^{-20}

#R ← number of rounds, fr ← number of ANDs to be freely passed, C_{fr} ← number of conditionally free ANDs, C_A → number of correlated ANDs, t ← number of required ANDs where probability should be paid, p_α and p → refer to probabilities with and without bit-fixing.

†Note that, this characteristic has the probability 2^{-34} if we do not consider any message bit fixing. So, this 81-round characteristic can not be verified because the message space for KATAN32 is 2^{32}.

Table 7.14: Related-key Differentials of KATAN.

Cipher	#R	Active Gates				Difference[7]			Probability	
		t	fr	C_{fr}	C_A	Input	Output	Key	p_α	p
KATAN32	60	3	0	0	0	$0x00004000$	$0x00b80084$	$\Delta k[9,39,50,54,64]=1$	2^{-3}	2^{-3}
		7	0	0	0	$0x00042000$	$0x00880801$	$\Delta k[1,11,53,64,68,78]=1$	2^{-7}	2^{-7}
	70	6	1	0	0	$0x80031000$	$0x01200400$	$\Delta k[0,3,5,13,55,70,72]=1$	2^{-6}	2^{-7}
		4	3	0	0	$0xa4020010$	$0x00b80084$	$\Delta k[3,4,7,10,17,29,59,70,74]=1$	2^{-4}	2^{-7}
	84	16	0	1	1	$0xa0048000$	$0x01180263$	$\Delta k[1,4,23,31,42,61]=1$	2^{-16}	2^{-17}
KATAN48	50	0	7	0	0	$0x000000301800$	$0x000180000000$	$\Delta k[17]=1$	2^{0}	2^{-7}
	59	6	3	0	0	$0x000003018000$	$0x000000001460$	$\Delta k[13]=1$	2^{-6}	2^{-9}
		6	2	3	0	$0x820031400000$	$0x000060003000$	$\Delta k[5,24]=1$	2^{-6}	2^{-11}
	60	7	2	3	1	$0x820031400000$	$0x00018000c000$	$\Delta k[5,24]=1$	2^{-7}	2^{-12}
		6	14	0	0	$0xdb0000643018$	$0x180000000005$	$\Delta k[6,25]=1$	2^{-6}	2^{-20}
KATAN64	56	11	4	0	0	$0x0000001c00e00000$	$0x000020000001cce0$	$\Delta k[11]=1$	2^{-11}	2^{-15}
	57	13	3	0	1	$0x0000004801c00000$	$0x00000380001c0e00$	$\Delta k[1,7,20,26]=1$	2^{-13}	2^{-16}

#R ← number of rounds, fr ← number of ANDs to be freely passed, C_{fr} ← number of conditionally free ANDs, C_A → number of correlated ANDs, t ← number of required ANDs where probability should be paid, p_α and p → refer to probabilities with and without bit-fixing.

7.7.2.1 Improving Isobe *et al.*'s Related Key Boomerang Attack [234]

The related-key boomerang attack is a combination of the boomerang attack and the related-key differential attack. Such attacks are useful to build distinguishers when it consist of two shorter differential characteristics with high probabilities.

In [234] for KATAN32 ($= E_1 \circ E_0$), the authors devise a 140-round boomerang distinguisher, where both E_0 and E_1 have 70-rounds. Based on their efficient differential characteristics search for both E_0 and E_1, the authors provided maximum probability differential characteristics of each set in [234, Table 5,6]. In the construction of the boomerang distinguisher, the authors choose a differential characteristic of E_0 corresponding to the set 8 [234, Table 4] with probability 2^{-9} and of E_1 for the set 10 [234, Table 4] with probability 2^{-8}. Thus the probability to form a simple boomerang will be $(2^{-9})^2 \times (2^{-8})^2 = 2^{-34}$. Whereas for KATAN32, the attacker only has 2^{31} input message pairs with a fixed difference. To reduce the data complexity for this boomerang attack, the authors have considered multiple characteristics with the same input and output difference. As a result, the overall probability for the characteristics in E_0 and E_1 improves to $2^{-7.1}$ and $2^{-6.5}$ respectively. Therefore by combining these two differential characteristics, the overall probability of the above 140-round related-key boomerang distinguisher is increased to $(2^{-7.1})^2 \times (2^{-6.5})^2 = 2^{-27.2}$.

Table 7.15: Sets of Key Difference Considered in [234].

Set	0	1	2	3	4	5	6	7	8	9	10
Key Difference	0,19	1,20	2,21	3,22	4,23	5,24	6,25	7,26	8,27	9,28	10,29
Plaintext Difference	$L_2[9]$ $L_1[12]$	$L_2[18]$ $L_1[2,7,12]$	$L_2[8]$ $L_1[11]$	$L_2[17]$ $L_1[1,6,11]$	$L_2[7,18]$ $L_1[10]$	$L_2[16]$ $L_1[0,5,10]$	$L_2[6,17]$ $L_1[9]$	$L_2[15,18]$ $L_1[4,9]$	$L_2[5,16]$ $L_1[8]$	$L_2[14,17]$ $L_1[3,8]$	$L_2[4,15]$ $L_1[7,12]$

Using the DEEPAND model, we have verified all the characteristics corresponding to the differential characteristics of each set in [234, Table 5]. For set 0 and set 10, we have respectively identified three and one correlated AND gates in the characteristics with probabilities 2^{-12} and 2^{-10} (without considering free and conditionally free AND gates). Whereas, according to their search strategy, the characteristic probabilities

[7]Note that, for larger rounds, the DEEPAND model could not find the best characteristics due to too many constraints in the model.

Table 7.16: Verified Related-key Boomerang Distinguisher of KATAN32 .

KATAN32					
No.	Prob.	Input Diff	Output Diff	Key Difference	
		Upper	Lower	Upper	Lower
1.	2^{-22}	$0x00026000$	$0x48008b00$	$0xa0800000000002001504$	$0x52c0a267036154fc4c36$

In hexadecimal notation, the most significant bit (MSB) is placed on the right side and the least significant bit (LSB) is located on the left side.

for set 0 and set 10 are 2^{-15} and 2^{-12}. For other sets, the DEEPAND model did not find any extra advantage in the characteristics. Moreover, if we do not consider the predefined sets in Table 7.15, the DEEPAND model can find much better 70-round characteristics of probability 2^{-7} ($> 2^{-9}$). So, by choosing two characteristics of probabilities 2^{-7}, 2^{-7} for both E_0, E_1, we can form a boomerang distinguisher with probability $(2^{-7})^2 \times (2^{-7})^2 = 2^{-28}$. Also, in the similar fashion, we can further reduce the data complexity of this 140-round boomerang attack by choosing the multiple differentials that correspond to the same input/output difference. For the first boomerang in Table 7.16, the input and key difference of E_0 is represented by 4 characteristics of probability 2^{-7}, 8 characteristics of probability 2^{-8}, 16 characteristics of probability 2^{-9}, and 32 characteristics of probability 2^{-10}. Similarly, the output and key difference of E_1 is represented by 4 characteristics of probability 2^{-7}, 8 characteristics of probability 2^{-8}, and 32 characteristics of probability 2^{-9}. The overall probabilities of E_0 and E_1 are approximately $2^{-5.52}$ and $2^{-5.5}$, respectively. The overall probability of the boomerang distinguisher can be calculated as $(2^{-5.52})^2 \times (2^{-5.5})^2 = 2^{-22.04}$ which is greater than $2^{-27.2}$. Note that for the distinguishers given in Table 7.16, we have not considered any message-bit fixing in order to take advantage of the cluster of characteristics.

7.8 Conclusion

In this work, we have developed DEEPAND, a new generalized MILP model designed to capture first-order correlations in single and multiple AND-based (NLFSR) ciphers. This model is primarily based on three key observations: Observation 1, Observation 2, and Observation 3, and it introduces the concept of fully free and conditionally free rounds. Our model demonstrates that dependencies can exist among multiple

AND gates in NLFSR-based ciphers.

Saha *et al.* previously demonstrated the correlation between two subsequent AND gates sharing one common input for the NIST-LWC finalist TinyJAMBU, which follows a single AND-based NLFSR. To capture such correlations, which we have redefined as BAND, they built a MILP-aided tool and found improved differentials compared to the designer's claims. In our work, we expand on this by explaining the generalized view of when two subsequent ANDs will be correlated in a single AND-based NLFSR. Additionally, we systematize the multiple correlations of different subsequent ANDs to significantly increase the probability for multiple AND-based NLFSRs.

To accurately capture the dependencies of multiple ANDs in an NLFSR, we introduce MAND. Furthermore, we show that if one of the inputs of an AND gate is known, then for certain input differences, the output difference becomes deterministic. Using the DEEPAND model, we have primarily investigated the differential properties of TinyJAMBU's keyed permutations. For the full-round $\mathcal{P}_{1024}$, we found a differential characteristic (Type-IV) with a probability of 2^{-108}, highlighting its non-ideal nature. For $\mathcal{P}_{640}$, the probability is 2^{-42}.

For KATAN , we report the best differential characteristic for 42 rounds with a practical probability of 2^{-7}, challenging the designer's claim. We also improved the related-key boomerang attack by Isobe *et al.* using DEEPAND. Finally, we enhanced the differential characteristics for 43-round KATAN48 and 37-round KATAN64 , demonstrating the broad applicability of the DEEPAND model. The DEEPAND model developed in this work is an effective tool for probing into the correlations that develop during differential propagation, warranting further investigation.

8 CONCLUDING REMARKS

In conclusion, this thesis has focused on the cryptanalysis of a selection of SPN and NLFSR-based symmetric-key ciphers. Our main results and contributions are outlined in Chapters 3, 4, 5, 6, and 7. In this section, we provide a summary of the work presented in this thesis, highlighting the key findings and methodologies employed. We also provide a discussion offering insights into new design criteria for symmetric cryptography derived from our DFA attacks. Furthermore, we provide insights into how these findings integrate with existing research, emphasizing the connections between our DFA under the gray-box model and differential cryptanalysis on NLFSR-based ciphers under the black-box model across various cryptographic frameworks. Additionally, we discuss the future scope of this work, suggesting potential avenues for further research and exploration in the field of cryptanalysis and symmetric-key ciphers.

8.1 Summary

The thesis focuses on the cryptanalysis of various lightweight private-key ciphers based on SPN and NLFSR structures. The main objective of the thesis is to identify vulnerabilities in these ciphers and develop new techniques for analyzing their security. The thesis is organized into several chapters, each of which presents new results and contributions to the field of cryptanalysis. Chapter 3 introduces a new theoretical problem that is a variant of the generalized coupon collector problem. This problem is used to estimate the number of faults required for successful differential

fault attacks on several cryptographic schemes. Chapter 4 demonstrates differential fault attacks on the CAESAR and NORX ciphers with a level of parallelism of 2 and 4, respectively. The attacks demonstrate that these schemes are vulnerable to differential fault attacks even when a nonce is used. Chapter 5 presents differential fault attacks on several AE schemes based on GFN and SPN structures, using faulty forgery in the decryption query. These attacks highlight the vulnerability of these schemes to fault attacks and the importance of careful design and analysis of cryptographic schemes to ensure their resilience against such attacks. Chapter 6 extends the analysis to other SPN-based sponge/SIV-like AE schemes, including PHOTON-BEETLE, ORANGE, SIV-TEM-PHOTON, and ESTATE. Differential fault attacks are shown to be effective in breaking these schemes in the presence of faulty forgery in the decryption query. Finally, Chapter 7 proposes a new model called "DEEPAND" that captures correlations among multiple AND gates in NLFSR-based lightweight block ciphers. The model is applied to TinyJAMBU and KATAN and is shown to detect correlations that were missed by earlier models, leading to more accurate differential bounds for both ciphers. Overall, the thesis provides new insights into the design and analysis of secure lightweight cryptographic schemes in the presence of fault attacks and contributes to the ongoing research efforts in the field of cryptanalysis.

8.2 Discussion

In the realm of cryptographic security, differential fault analysis (DFA) has emerged as a critical method for assessing the robustness of cryptographic schemes, particularly those utilizing nonce-based authenticated encryption (AE) mechanisms. DFA involves deliberately introducing faults into a cryptographic system to analyze the resulting erroneous outputs, revealing sensitive information such as secret keys or internal states. Despite the added protection that nonces provide by ensuring unique outputs for each encryption session, recent research has demonstrated that nonce-based schemes are not invulnerable to differential fault attacks. These studies reveal that faults can still be exploited to compromise the security of such schemes, challenging the notion that nonces alone are sufficient to prevent DFA vulnerabilities.

Also, our thesis contributes to this understanding by exploring how nonce-based

Sponge AE schemes can be fortified against DFA attacks. Specifically, it addresses the complexities associated with retrieving keys when only partial state information is available. By analyzing how outputting tags from partial state information can complicate DFA attacks, this work highlights the challenges and provides strategies to mitigate these issues, such as selecting carefully chosen partial bits from the state to hinder effective key recovery.

Additionally, this research extends beyond fault attacks by investigating differential cryptanalysis within a black-box model applied to NLFSR-based ciphers. We have generalized correlations among AND operations in single and multiple AND-based NLFSR structures, revealing how these correlations can significantly enhance the probability of successful attacks. This differential cryptanalysis can complement DFA efforts by either reducing the number of faults required or directly aiding in key recovery due to the established correlations.

The integration of these findings shows a comprehensive approach to cryptographic security. By bridging the gap between DFA under a gray-box model and differential cryptanalysis in a black-box context, this work provides a unified perspective on the vulnerabilities and defense mechanisms applicable to modern cryptographic schemes. Future research can build upon these insights to develop more robust cryptographic solutions, leveraging the understanding of both fault and differential attacks to enhance overall security.

8.3 Future Research

Based on the research presented in the thesis, there are several interesting directions for future work that can be explored to further improve the security of lightweight cryptographic schemes against fault attacks.

Firstly, the differential fault attack on the NORX cipher demonstrated in Chapter 4 raises concerns about the vulnerability of ciphers with a high degree of parallelism to fault attacks. It would be interesting to investigate other ciphers with similar structures to NORX and explore whether the same attack can be applied to them. Additionally, further research can be conducted to design countermeasures that can protect ciphers from such attacks. Then, the use of faulty forgery in the decryption

query, as demonstrated in Chapters 5 and 6, can be extended to other types of authenticated encryption schemes, such as MACs and modes of hash functions. Further research can be conducted to evaluate the vulnerability of these schemes to differential fault attacks and explore new countermeasures to protect them. Finally, the proposed DEEPAND model in Chapter 7 can be further improved to effectively search for best trails for larger rounds. One direction for future research could be to minimize the constraints in the model to make it more efficient in searching for optimal trails. Additionally, the model can be extended to explore other types of differential-related attacks and identify previously unknown vulnerabilities in lightweight cryptographic schemes.

Overall, the findings presented in this thesis provide valuable insights into the security of lightweight cryptographic schemes against fault attacks, and the future research directions identified above can contribute to further strengthening the security of these schemes.

BIBLIOGRAPHY

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

[2] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[3] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.

[4] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[5] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[6] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[7] Auguste Kerckhoffs. La cryptographie militaire. In Walter Fumy, editor, *Journal des sciences militaires, vol. IX, pp. 5–38, Jan. 1883.*

[8] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[9] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[10] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1058–1069. ACM, 2015.

[11] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[12] Jean-Philippe Aumasson. *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, USA, 2017.

[13] Kazuo Sakiyama, Yu Sasaki, and Yang Li. *Security of Block Ciphers from Algorithm design to Hardware Implementation*. John Wiley and Sons Singapore Pte. Ltd., 2015.

[14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC Press, 2007.

[15] Kevin M. Igoe and Jerome A. Solinas. AES galois counter mode for the secure shell transport layer protocol. *RFC*, 5647:1–10, 2009.

[16] David A. McGrew and Kevin M. Igoe. AES-GCM authenticated encryption in the secure real-time transport protocol (SRTP). *RFC*, 7714:1–48, 2015.

[17] David A. McGrew and Daniel V. Bailey. AES-CCM cipher suites for transport layer security (TLS). *RFC*, 6655:1–8, 2012.

[18] Chunju Shao, Hui Deng, Rajesh S. Pazhyannur, Farooq Bari, Rong Zhang, and Satoru Matsushima. IEEE 802.11 medium access control (MAC) profile for control and provisioning of wireless access points (CAPWAP). *RFC*, 7494:1–13, 2015.

[19] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001.

[20] Thai Duong and Juliano Rizzo. Here come the XOR ninjas. In *White paper, Netifera, 2011*.

[21] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 526–540. IEEE Computer Society, 2013.

[22] CAESAR:. Competition for authenticated encryption: Security, applicability, and robustness.

[23] Kerry McKay, Lawrence E. Bassham, Meltem Sonmez Turan, and Nicky Mouha. Report on lightweight cryptography. 2017.

[24] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[25] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[26] Johannes Blömer and Volker Krummel. Fault based collision attacks on AES. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings*, volume 4236 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2006.

[27] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[28] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[29] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2010.

[30] Zhiqiang Liu, Dawu Gu, Ya Liu, and Wei Li. Linear fault analysis of block ciphers. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, volume 7341 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 2012.

[31] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118, 2013.

[32] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, pages 315–342. Springer, 2018.

[33] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

[34] Navid Vafaei, Sara Zarei, Nasour Bagheri, Maria Eichlseder, Robert Primas, and Hadi Soleimany. Statistical effective fault attacks: The other side of the coin. *IEEE Trans. Inf. Forensics Secur.*, 17:1855–1867, 2022.

[35] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 513–525, 1997.

[36] NIST Lightweight Cryptography Standardization Process, January 4, 2019.

[37] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks on symmetric key cryptosystems. *ACM Comput. Surv.*, 55(4), nov 2022.

[38] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[39] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[40] Dhiman Saha and Dipanwita Roy Chowdhury. Scope: On the side channel vulnerability of releasing unverified plaintexts. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015*, pages 417–438, 2015.

[41] Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. Escape: Diagonal fault analysis of APE. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 197–216, 2014.

[42] Dhiman Saha and Dipanwita Roy Chowdhury. Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 581–601, 2016.

[43] Amit Jana, Dhiman Saha, and Goutam Paul. Differential fault analysis of NORX. In Chip-Hong Chang, Ulrich Rührmair, Stefan Katzenbeisser, and Patrick Schaumont, editors, *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2020, Virtual Event, USA, November 13, 2020*, pages 67–79. ACM, 2020.

[44] Amit Jana, Anirban Nath, Goutam Paul, and Dhiman Saha. Differential fault analysis of NORX using variants of coupon collector problem. *J. Cryptogr. Eng.*, 12(4):433–459, 2022.

[45] Amit Jana. Differential fault attack on feistel-based sponge AE schemes. *J. Hardw. Syst. Secur.*, 6(1):1–16, 2022.

[46] Amit Jana and Goutam Paul. Differential fault attack on photon-beetle. In Chip-Hong Chang, Ulrich Rührmair, Debdeep Mukhopadhyay, and Domenic Forte, editors, *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*, pages 25–34. ACM, 2022.

[47] Amit Jana and Goutam Paul. Differential fault attack on spn-based sponge and siv-like AE schemes. *J. Cryptogr. Eng.*, 14(2):363–381, 2024.

[48] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[49] Dhiman Saha, Yu Sasaki, Danping Shi, Ferdinand Sibleyras, Siwei Sun, and Yingjie Zhang. On the Security Margin of TinyJAMBU with Refined Differential and Linear Cryptanalysis. *IACR Transactions on Symmetric Cryptology*, 2020(3):152–174, Sep. 2020.

[50] Amit Jana, Mostafizar Rahman, and Dhiman Saha. Deepand: In-depth modeling of correlated and gates for nlfsr-based lightweight block ciphers. Cryptology ePrint Archive, Paper 2022/1123, 2022. https://eprint.iacr.org/2022/1123.

[51] H. Feistel. Cryptography and computer privacy. In *Scientific American, v.228, n. 5, May 1973*, pages 15–23, 1973.

[52] NBS FIPS PUB 46 National Bureau of Standards. Data encryption standard. In *National Bureau of Standards, U.S. Department of Commerce, Jan 1977*.

[53] Akihiro Shimizu and Shoji Miyaguchi. Fast data encipherment algorithm FEAL. In *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, pages 267–278, 1987.

[54] Gosudarstvennyi Standard 28147-89 GOST. Cryptographic protection for data processing systems. In *Government Committee of the USSR for Standards, 1989*.

[55] Ralph C. Merkle. Fast software encryption functions. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 476–501, 1990.

[56] Lawrence Brown, Matthew Kwan, Josef Pieprzyk, and Jennifer Seberry. Improving resistance to differential cryptanalysis and the redesign of LOKI. In *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, pages 36–50, 1991.

[57] C.M. Adams and S.E. Tavares. Designing s-boxes for ciphers resistant to differential cryptanalysis. In *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography, Rome, Italy, 15-16 Feb 1993*, pages 181–190.

[58] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In Ross J. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[59] Ronald L. Rivest. The RC5 encryption algorithm. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer, 1994.

[60] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 461–480, 1989.

[61] Ross J. Anderson and Eli Biham. Two practical and provably secure block ciphers: BEARS and LION. In *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, pages 113–120, 1996.

[62] Joan Daemen and Vincent Rijmen. AES and the wide trail design strategy. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 108–109, 2002.

[63] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 326–341, 2011.

[64] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[65] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[66] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions (Version 0.1). https://keccak.team/files/CSF-0.1.pdf.

[67] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[68] Phillip Rogaway and Thomas Shrimpton. The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and Misuse-Resistant Nonce-Based Authenticated-Encryption, Aug 20, 2007.

[69] M. Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. https://www.nist.gov/publications/recommendation-block-cipher-modes-operation-galoiscounter-mode-gcm-and-gmac, NIST Special Publication 800-38D, November 2007.

[70] M. Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-38b.pdf, NIST Special Pulication 800-38B, May 2005.

[71] Dan Harkins. Synthetic initialization vector (SIV) authenticated encryption using the advanced encryption standard (AES). *RFC*, 5297:1–26, 2008.

[72] JunHyuk Song, Radha Poovendran, Jicheol Lee, and Tetsu Iwata. The AES-CMAC algorithm. *RFC*, 4493:1–20, 2006.

[73] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[74] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, 2008.

[75] Tatu Ylönen and Chris Lonvick. The secure shell (SSH) transport layer protocol. *RFC*, 4253:1–32, 2006.

[76] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. *RFC*, 5246:1–104, 2008.

[77] Stephen T. Kent. IP encapsulating security payload (ESP). *RFC*, 4303:1–44, 2005.

[78] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2014.

[79] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.

[80] Charanjit S. Jutla. Encryption modes with almost free message integrity. *J. Cryptol.*, 21(4):547–578, 2008.

[81] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2001.

[82] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 196–205. ACM, 2001.

[83] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[84] Doug Whiting, Russell Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). *RFC*, 3610:1–26, 2003.

[85] Mihir Bellare, Phillip Rogaway, and David A. Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[86] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer, 2004.

[87] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer, 2003.

[88] Philip Hawkes and Gregory G. Rose. Primitive specification for SOBER-128. *IACR Cryptol. ePrint Arch.*, page 81, 2003.

[89] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *Int. J. Wirel. Mob. Comput.*, 5(1):48–59, 2011.

[90] Christian Forler, Stefan Lucks, David McGrew, and Jakob Wenzel. Hash-CFB-authenticated encryption without a block cipher. https://hyperelliptic.org/DIAC/slides/hash-cfb-talk.pdf, Directions in Authenticated Ciphers (DIAC), 2012.

[91] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. https://keccak.team/files/KeccakDIAC2012.pdf, Directions in Authenticated Ciphers (DIAC), 2012.

[92] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002.

[93] Tetsu Iwata and Kan Yasuda. HBS: A single-key mode of operation for deterministic authenticated encryption. In Orr Dunkelman, editor, *Fast Software*

Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers, volume 5665 of *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009.

[94] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[95] Mitsuru Matsui. On correlation between the order of s-boxes and the strength of DES. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[96] Alex Biryukov and Ivica Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.

[97] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in ARX: application to block cipher speck. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 289–310. Springer, 2016.

[98] Joan Daemen and Gilles Van Assche. Differential propagation analysis of keccak. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 422–441. Springer, 2012.

[99] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in keccak. *IACR Trans. Symmetric Cryptol.*, 2017(1):329–357, 2017.

[100] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. Milp-based automatic search algorithms for differential and linear trails for speck. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2016.

[101] Yunwen Liu, Qingju Wang, and Vincent Rijmen. Automatic search of linear trails in ARX with applications to SPECK and chaskey. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve A. Schneider, editors, *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford,*

UK, June 19-22, 2016. Proceedings, volume 9696 of *Lecture Notes in Computer Science*, pages 485–499. Springer, 2016.

[102] Nicky Mouha and Bart Preneel. A proof that the ARX cipher salsa20 is secure against differential cryptanalysis. *IACR Cryptol. ePrint Arch.*, page 328, 2013.

[103] David A. Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[104] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and serpent. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

[105] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[106] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[107] Eli Biham, Orr Dunkelman, and Nathan Keller. New results on boomerang and rectangle attacks. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[108] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3g telephony. *J. Cryptol.*, 27(4):824–849, 2014.

[109] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714. Springer, 2018.

[110] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe. Bivium as a mixed-integer linear programming problem. In Matthew Geoffrey Parker, editor,

Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2009.

[111] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security analysis of SIMD. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010.

[112] Andrey Bogdanov. On unbalanced feistel networks with contracting MDS diffusion. *Des. Codes Cryptogr.*, 59(1-3):35–58, 2011.

[113] Shengbao Wu and Mingsheng Wang. Security evaluation against differential cryptanalysis for block cipher structures. Cryptology ePrint Archive, Paper 2011/551, 2011. https://eprint.iacr.org/2011/551.

[114] Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. Automatic security evaluation of block ciphers with s-bp structures against related-key differential attacks. In Dongdai Lin, Shouhuai Xu, and Moti Yung, editors, *Information Security and Cryptology - 9th International Conference, Inscrypt 2013, Guangzhou, China, November 27-30, 2013, Revised Selected Papers*, volume 8567 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2013.

[115] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

[116] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Cryptology ePrint Archive, Paper 2014/747, 2014.

[117] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017.

[118] Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. Improving the milp-based security evaluation algorithm against differential/linear cryptanalysis using A divide-and-conquer approach. *IACR Trans. Symmetric Cryptol.*, 2019(4):438–469, 2019.

[119] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 513–525, 1997.

[120] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, pages 77–88, 2003.

[121] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on A.E.S. In *Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003, Proceedings*, pages 293–306, 2003.

[122] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A diagonal fault attack on the advanced encryption standard. *IACR Cryptol. ePrint Arch.*, page 581, 2009.

[123] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, pages 224–233, 2011.

[124] Junko Takahashi and Toshinori Fukunaga. Differential fault analysis on CLEFIA with 128, 192, and 256-bit keys. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 93-A(1):136–143, 2010.

[125] Nasour Bagheri, Reza Ebrahimpour, and Navid Ghaedi. New differential fault analysis on PRESENT. *EURASIP J. Adv. Signal Process.*, 2013:145, 2013.

[126] Ling Song and Lei Hu. Differential fault attack on the PRINCE block cipher. In Gildas Avoine and Orhun Kara, editors, *Lightweight Cryptography for Security and Privacy - Second International Workshop, LightSec 2013, Gebze, Turkey, May 6-7, 2013, Revised Selected Papers*, volume 8162 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

[127] Wei Cheng, Yongbin Zhou, and Laurent Sauvage. Differential fault analysis on midori. In Kwok-Yan Lam, Chi-Hung Chi, and Sihan Qing, editors, *Information and Communications Security - 18th International Conference, ICICS 2016, Singapore, November 29 - December 2, 2016, Proceedings*, volume 9977 of *Lecture Notes in Computer Science*, pages 307–317. Springer, 2016.

[128] Fred Roberts and Barry Tesman. *Applied Combinatorics, Second Edition*. Chapman and Hall/CRC, 2nd edition, 2009.

[129] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discret. Appl. Math.*, 39(3):207–229, 1992.

[130] L. Holst. On birthday, collectors', occupancy and other classical urn problems. Internat. Statist. Rev., 54:15-27, 1986.

[131] D. C. Van Leijenhorst and Theo P. van der Weide. A formal derivation of heaps' law. *Inf. Sci.*, 170(2-4):263–272, 2005.

[132] Toshio Nakata. Coupon collecto's problem with unlike probabilities. *Journal of Classical Analysis*, Volume 14(Number 2):177–180, 2019.

[133] J. D. Newman and L. Shepp. The double dixie cup problem, 1960.

[134] Jakub Breier, Wei He, Shivam Bhasin, Dirmanto Jap, Samuel Chef, Hock Guan Ong, and Chee Lip Gan. Extensive laser fault injection profiling of 65 nm FPGA. *J. Hardw. Syst. Secur.*, 1(3):237–251, 2017.

[135] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Régis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018*, pages 1–6. IEEE Computer Society, 2018.

[136] Brice Colombier, Lilian Bossuet, Paul Grandamme, Julien Vernay, Emilie Chanavat, Lucie Bon, and Bruno Chassagne. Multi-spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks. In *20th Smart Card Research and Advanced Application Conference - CARDIS 2021*, Lübeck, Germany, November 2021.

[137] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley and New York, 1950.

[138] M. Sharif and B. Hassibi. Delay considerations for opportunistic scheduling in broadcast fading channels, 1960.

[139] Weiyu Xu and Ao Kevin Tang. A generalized coupon collector problem. *CoRR*, abs/1010.5608, 2010.

[140] H. von Schelling. Coupon collecting for unequal probabilities. American Mathematical Monthly, 61:306-311, 1954.

[141] Aristides Doumas and Vassilis Papanicolaou. The coupon collector's problem revisited: Asymptotics of the variance, 03 2012.

[142] Emmanuelle Anceaume, Yann Busnel, and Bruno Sericola. New results on a generalized coupon collector problem using markov chains. *J. Appl. Probab.*, 52(2):405–418, 2015.

[143] Petra Berenbrink and Thomas Sauerwald. The weighted coupon collector's problem and applications. In Hung Q. Ngo, editor, *Computing and Combinatorics, 15th Annual International Conference, COCOON 2009, Niagara Falls, NY, USA, July 13-15, 2009, Proceedings*, volume 5609 of *Lecture Notes in Computer Science*, pages 449–458. Springer, 2009.

[144] P. Neal. The generalised coupon collector problem. Journal of Applied Probability, 45(3), 621-629, 2008.

[145] Pierre-Simon Laplace. *Théorie analytique des probabilités*. pp. 194–195, 1812.

[146] Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei, 6: 215–220, MR 0150807, 1961.

[147] Farzaneh Abed, Christian Forler, and Stefan Lucks. General classification of the authenticated encryption schemes for the CAESAR competition. *Computer Science Review*, 22:13–26, 2016.

[148] Dhiman Saha and Dipanwita Roy Chowdhury. Internal differential fault analysis of parallelizable ciphers in the counter-mode. *Journal of Cryptographic Engineering*, Nov 2017.

[149] Alex Biryukov and Dmitry Khovratovich. PAEQ: parallelizable permutation-based authenticated encryption. In Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings*, volume 8783 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2014.

[150] J. Aumasson, P. Jovanovic, and S. Neves. NORX V3.0. https://competitions.cr.yp.to/round3/norxv30.pdf, 2016.

[151] J. Aumasson, P. Jovanovic, and S. Neves. NORX V1. http://competitions.cr.yp.to/round1/norxv1.pdf, 2014.

[152] J. Aumasson, P. Jovanovic, and S. Neves. NORX V2.0. http://competitions.cr.yp.to/round2/norxv20.pdf, 2015.

[153] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: authenticated encryption for low-end systems. *IACR Cryptology ePrint Archive*, 2015:1154, 2015.

[154] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: investigating differential and rotational properties. In *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014*, pages 306–324, 2014.

[155] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[156] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A new dedicated 256-bit hash function: FORK-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2006.

[157] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v2. https://competitions.cr.yp.to/round3/ketjev2.pdf.

[158] Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A new dedicated 256-bit hash function: FORK-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2006.

[159] Sourav Das, Subhamoy Maitra, and Willi Meier. Higher order differential analysis of NORX. *IACR Cryptology ePrint Archive*, 2015:186, 2015.

[160] J. Aumasson and W. Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009. http://aumasson.jp/data/papers/AM09.pdf.

[161] Nasour Bagheri, Tao Huang, Keting Jia, Florian Mendel, and Yu Sasaki. Cryptanalysis of reduced NORX. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016*, pages 554–574, 2016.

[162] Ashutosh Dhar Dwivedi, Milos Kloucek, Pawel Morawiecki, Ivica Nikolic, Josef Pieprzyk, and Sebastian Wójtowicz. Sat-based cryptanalysis of authenticated ciphers from the CAESAR competition. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, pages 237–246, 2017.

[163] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of NORX v2.0. *IACR Trans. Symmetric Cryptol.*, 2017(1):156–174, 2017.

[164] Alex Biryukov, Aleksei Udovenko, and Vesselin Velichkov. Analysis of the NORX core permutation. *IACR Cryptology ePrint Archive*, 2017:34, 2017.

[165] Sayandeep Saha, Arnab Bag, Debapriya Basu Roy, Sikhar Patranabis, and Debdeep Mukhopadhyay. Fault template attacks on block ciphers exploiting fault propagation. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 612–643. Springer, 2020.

[166] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*, pages 1–10. IEEE, 2019.

[167] J. Aumasson, P. Jovanovic, and S. Neves. C-source code of NORX v3.0, 2016.

[168] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm sram-cells. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2015.

[169] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Anshul, and Debdeep Mukhopadhyay. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 577–596. Springer, 2015.

[170] Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, Assia Tria, and Thierry Vaschalde. Fault round modification analysis of the advanced encryption standard. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*, pages 140–145. IEEE Computer Society, 2012.

[171] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*, pages 235–239. IEEE Computer Society, 2010.

[172] Sheldon Ross. *A first course in probability*. Prentice Hall, New York,7th edition, 2005.

[173] Aurélien Vasselle, Hugues Thiebeauld, Quentin Maouhoub, Adèle Morisset, and Sébastien Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017*, pages 41–48. IEEE Computer Society, 2017.

[174] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 369–395, 2016.

[175] Christoph Dobraunig, Stefan Mangard, Florian Mendel, and Robert Primas. Fault attacks on nonce-based authenticated encryption: Application to keyak and ketje. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018*, pages 257–277, 2018.

[176] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*,

volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[177] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[178] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference - DAC 2015 pp. 175:1–175:6 ACM (2015).*

[179] Arthur Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia*, 8(1):22–42, 1984.

[180] Bruce Schneier and John Kelsey. Unbalanced feistel networks and block cipher design. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996.

[181] Stefan Lucks. Faster luby-rackoff ciphers. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1996.

[182] National Security Agency. Skipjack and KEA algorithm specifications".

[183] Ronald L. Rivest, Matthew J. B. Robshaw, and Yiqun Lisa Yin. RC6 as the AES. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 337–342. National Institute of Standards and Technology„ 2000.

[184] IBM Corporation. MARS - a candidate cipher for AES.

[185] Andrey Bogdanov and Kyoji Shibutani. Generalized feistel networks revisited. *Des. Codes Cryptogr.*, 66(1-3):75–97, 2013.

[186] Viet Tung Hoang and Phillip Rogaway. On generalized feistel networks. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 613–630, 2010.

[187] Kaisa Nyberg. Generalized feistel networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1996.

[188] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized feistel. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised*

Selected Papers, volume 6147 of *Lecture Notes in Computer Science*, pages 19–39. Springer, 2010.

[189] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[190] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[191] Debapriya Basu Roy, Avik Chakraborti, Donghoon Chang, S. V. Dilip Kumar, Debdeep Mukhopadhyay, and Mridul Nandi. Two efficient fault-based attacks on CLOC and SILC. *J. Hardw. Syst. Secur.*, 1(3):252–268, 2017.

[192] Christoph Dobraunig, Stefan Mangard, Florian Mendel, and Robert Primas. Fault attacks on nonce-based authenticated encryption: Application to keyak and ketje. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 257–277. Springer, 2018.

[193] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. PHOTON-Beetle: Authenticated encryption and hash family, 2021. Submission to the NIST Lightweight Competition, May 17, 2021.

[194] Bishwajit Chakraborty and Mridul Nandi. ORANGE, 2019. Submission to the NIST Lightweight Competition, September 20, 2019.

[195] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 299–320. Springer, 2017.

[196] Muhammad Reza Z'aba, Norziana Jamil, Mohd Saufy Rohmad, Hazlin Abdul Rani, and Solahuddin Shamsuddin. The CiliPadi family of lightweight authenticated encryption. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/cilipadi-spec.pdf.

[197] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponges. http://sponge.noekeon.org.

[198] Mark Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, and Raghvendra Rohit. ACE: An authenticated encryption and hash algorithm. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ace-spec-round2.pdf.

[199] Arghya Bhattacharjee, Cuauhtemoc Mancillas López, Eik List, and Mridul Nandi. The oribatida v1.3 family of lightweight authenticated encryption schemes. *J. Math. Cryptol.*, 15(1):305–344, 2021.

[200] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Schwaemm and Esch: Lightweight authenticated encryption and hashing using the Sparkle permutation family". https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/sparkle-spec-round2.pdf.

[201] Riham AlTawy, Guang Gong, Morgan He, Kalikinkar Mandal, and Raghvendra Rohit. Spix: An authenticated cipher.

[202] Riham AlTawy, Guang Gong, Morgan He, Ashwin Jha, Kalikinkar Mandal, Mridul Nandi, and Raghvendra Rohit. SpoC: An authenticated cipher.

[203] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. https://keccak.team/files/KeccakDIAC2012.pdf.

[204] Internet of things global standards initiative. https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx, 2015.

[205] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In *Advances in Cryptology - ASIACRYPT 2012, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658, pages 208–225. Springer, 2012.

[206] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815, pages 123–153. Springer, 2016.

[207] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[208] George Hatzivasilis, Konstantinos Fysarakis, Ioannis Papaefstathiou, and Charalampos Manifavas. A review of lightweight block ciphers. *J. Cryptogr. Eng.*, 8(2):141–184, 2018.

[209] Alex Biryukov and Léo Perrin. State of the art in lightweight symmetric cryptography. *IACR Cryptol. ePrint Arch.*, page 511, 2017.

[210] Kerry A. McKay, Lawrence E. Bassham, Meltem Sonmez Turan, and Nicky W. Mouha. Report on lightweight cryptography. *NIST Interagency/Internal Report (NISTIR),National Institute of Standards and Technology, Gaithersburg, MD, [online]*, 2017.

[211] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013.

[212] Michael Gruber, Matthias Probst, and Michael Tempelmeier. Statistical ineffective fault analysis of GIMLI. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*, pages 252–261. IEEE, 2020.

[213] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 513–525.

[214] Amit Jana. Differential fault attack on feistel-based sponge AE schemes. *J. Hardw. Syst. Secur.*, 6(1), 2022.

[215] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 222–239. Springer, 2011.

[216] Tetsu Iwata, Ling Song, Zhenzhen Bao, and Jian Guo. SIV-TEM-PHOTON authenticated encryption and hash family, 2019. Submission to the NIST Lightweight Competition, March 28, 2019.

[217] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. ESTATE, 2019. Submission to the NIST Lightweight Competition, March 29, 2019.

[218] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[219] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: small universal deterministic authenticated encryption for the internet of things. *IACR Trans. Symmetric Cryptol.*, 2018(3):1–35, 2018.

[220] Amit Jana. Unoptimized C-implementation of PHOTON-BEETLE state recovery under random fault model. https://github.com/janaamit001/PhotonBeetle_state_recovery_under_RfaultModel.git, 2023.

[221] Amit Jana. Unoptimized C-implementation of PHOTON-BEETLE state recovery under different fault models. https://github.com/janaamit001/PHOTON-BEETLE.git, 2023.

[222] Amit Jana. Unoptimized C-implementation of ORANGE state recovery under different fault models. https://github.com/janaamit001/ORANGE.git, 2023.

[223] Amit Jana. Unoptimized C-implementation of SIV-TEM-PHOTON state recovery under different fault models. https://github.com/janaamit001/SIV-TEM-PHOTON.git, 2023.

[224] Amit Jana. Unoptimized C-implementation of ESTATE state recovery under different fault models. https://github.com/janaamit001/ESTATE.git, 2023.

[225] Amit Jana. Unoptimized C-implementation of faulty forgery simulation of PHOTON-BEETLE and ESTATE. https://github.com/janaamit001/Faulty_Forgery_Simulation.git, 2023.

[226] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptol.*, 4(1):3–72, 1991.

[227] Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.

[228] National Institue of Standards and Technology. Lightweight Cryptography. Technical report, Aug 27 2018.

[229] Hongjun Wu and Tao Huang. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/TinyJAMBU-spec.pdf. NIST LWC Round1 Candidate, 2019.

[230] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. Katan and ktantan — a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 272–288, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[231] Hongjun Wu and Tao Huang. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2). https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf. NIST LWC Finalist, 2021.

[232] Hongjun Wu and Tao Huang. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/TinyJAMBU-spec-round2.pdf. NIST LWC Round2 Candidate, 2019.

[233] Ferdinand Sibleyras, Yu Sasaki, Yosuke Todo, Akinori Hosoyamada, and Kan Yasuda. Birthday-bound slide attacks on tinyjambu's keyed-permutations for

all key sizes. In Chen-Mou Cheng and Mitsuaki Akiyama, editors, *Advances in Information and Computer Security - 17th International Workshop on Security, IWSEC 2022, Tokyo, Japan, August 31 - September 2, 2022, Proceedings*, volume 13504 of *Lecture Notes in Computer Science*, pages 107–127. Springer, 2022.

[234] Takanori Isobe, Yu Sasaki, and Jiageng Chen. Related-key boomerang attacks on KATAN32/48/64. In Colin Boyd and Leonie Simpson, editors, *Information Security and Privacy - 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings*, volume 7959 of *Lecture Notes in Computer Science*, pages 268–285. Springer, 2013.