Design of Provably Secure Lightweight Authenticated Encryption Schemes based on Tweakable Primitives

By

Munawar Hasan
PHD19001

Supervisor : Prof. Donghoon Chang (IIIT Delhi, NIST)

Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi
New Delhi– 110020

June, 2025

Design of Provably Secure Lightweight Authenticated Encryption Schemes based on Tweakable Primitives

By

Munawar Hasan
PHD19001

A Thesis

submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi
New Delhi– 110020

June, 2025

Certificate

This is to certify that the thesis titled *Design of Provably Secure Lightweight Authenticated Encryption Schemes based on Tweakable Primitives* being submitted by *Munawar Hasan* to the Indraprastha Institute of Information Technology, Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Prof. Donghoon Chang

June, 2025

Indraprastha Institute of Information Technology, Delhi

New Delhi 110020

Candidate's Declaration

The author hereby declares that the work presented in the thesis titled *Design of Provably Secure Lightweight Authenticated Encryption Schemes based on Tweakable Primitives*, submitted as partial fulfillment for the award of the degree of Doctor of Philosophy to the Indraprastha Institute of Information Technology, Delhi, is an original research work carried out under the supervision of Prof. Donghoon Chang (Indraprastha Institute of Information Technology, Delhi, India).

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Munawar Hasan

PhD19001

June, 2025

Indraprastha Institute of Information Technology, Delhi

New Delhi 110020

Abstract

Tweakable primitives are advanced cryptographic constructions, primarily used to enhance the flexibility, security, and performance of cryptographic systems. In case of the tweakable primitives, in addition to the key and the input, there is an additional or extra parameter (often referred to as the third parameter). This third parameter is called the tweak. Further, the tweak is a public value and is known to the adversary. Due to the availability of this third parameter, tweakable primitives find their use in creating modes of operation and in several applications in the area spanning across the design of cryptographic systems. As an example, consider the application of disk encryption. The index of a disk sector can be used as a tweak for disk encryption. This will create mutually exclusive encryptions of the sectors of the disk. These independent disk sectors can be accessed, edited and updated independently, thereby facilitating usability and increasing the end user experience.

Historically, right from the inception of the tweakable primitives, they have been mostly overlooked and hardly found their inclusion in cryptographic designs. This has changed in the last decade, the interest of research community has shown a growth and the tweakable primitives have started to receive traction. One of the area that has been explored is the design of authenticated encryption schemes with associated data (or AEAD). The tweak provides flexibility when designing an AEAD. Nonce, counter etc., are some of the values that may be known to an adversary in different attack models. Hence, these values become an ideal candidate for the tweak when designing AEAD using tweakable primitives. The design so proposed must also prove its resistance against adversaries under various attack models. This leads to the foundation of provable security proofs. A provable security proof is a formal method used to show that a cryptographic designs like AEAD, signature scheme etc., are secure under well-defined mathematical assumptions. It involves stating a formal argument and then showing that the design resists certain types of claimed attacks, assuming that a related hard problem (e.g., factoring large numbers or solving the discrete logarithm problem) is computationally infeasible.

Thus, in this thesis, we analyzed, **"how can we design authenticated encryption schemes with associated data using tweakable primitives with provable security proofs?"**. Specifically, we tried to answer questions like: How to use tweakable primitives to design AEAD? What designs can be provided with provable security proofs? What properties can be incorporated into the design that shows the true potential of tweakable primitives? Can we investigate for providing additional features .i.e., committing security?

We choose tweakable blockcipher as our tweakable primitive and begin the design of authenticated encryption scheme with associated data. We explore all the ways to create an AEAD based of *xor* operation and using tweakable blockcipher, were the tweak size is twice the block size. During the design, we found that usage of tweak provides several additional properties that

facilitates the area of lightweight cryptography. Keeping this in mind, we adapted the strategy of our design to fulfil the requirements of lightweight cryptosystem outlined by NIST (National Institute of Standards and Technology, USA). We proposed lynx, a family (with 14 members) of lightweight 1-pass and rate-1 AEAD based on a tweakable blockcipher. We further provided the provable security proof for each of the member of the lynx family. Further, the implementation of lynx highlights its potential for the *resource constraint* devices.

We introduce the notion of tweakable stream cipher (tS in short) with the property of partial collision resistance, and use it to create four new tweakable wide block cipher schemes: HBtSH, HtS, tS-double-decker and tS-docked-double-decker. These four proposed schemes can be used to create a CMT-4 secure authenticated encryption scheme with the property of partial collision under encode-then-encipher paradigm. Further, we provide provable security proof with partial collision resistance for the four proposed schemes against a CMT-4 adversary. Traditional AEAD schemes may be vulnerable to misuse by adversaries capable of adapting their attacks based on observed ciphertexts, leading to potential breaches in security. CMT-4 secure AEAD schemes aim to address this limitation by providing an additional security property known as commitment. Committing security of authenticated encryption schemes is an emerging area and an active field of research and is highly motivated by real-world scenarios. The proposal draft by NIST for the requirements of the Accordion Mode lists the committing security as one of the desirable properties.

To my Father

Acknowledgements

"What I cannot create, I do not understand." – Richard Feynman

First and foremost, I express my heartfelt gratitude to my esteemed Ph.D. supervisor, Prof. Donghoon Chang, for his invaluable guidance and persistently inspiring me to bring out the best me. He encouraged me to learn and gain knowledge throughout my Ph.D. journey. I would also like to express my gratitude to my esteemed Ph.D. care taker advisor Dr. Ravi Anand for his guidance and feedback. I am much indebted to them for the amazing support and constant motivation throughout my Ph.D. tenure. Prof. Donghoon not only guided me in my research work, but also influenced me to be a better human being. I am incredibly fortunate to have a friend, a philosopher, and a guide in the form of my Ph.D. supervisor, who pushed me to have a positive outlook towards my work and life. Without his enlightenment, this journey would not have been the same.

I appreciate the efforts of my Ph.D. committee member Dr. Sambuddho Chakravarty who helped me improve my work by providing his valuable feedback and comments.

I would like to thank Dr. Apostol Vassilev from NIST (National Institute of Standards and Technology, USA) for his guidance and direction during my stay at NIST.

On a personal front, I owe my heartfelt thanks to my parents and sisters for their unconditional support, understanding, and encouragement, without which this dissertation would not have been completed.

Above all, I am grateful to the Almighty, who bestowed upon me the opportunities and His blessings for giving me the strength and courage to complete this dissertation.

Research Papers From This Work

1. Lynx: Family of Lightweight Authenticated Encryption Schemes Based on Tweakable Blockcipher.
   Munawar Hasan, Donghoon Chang.
   IEEE Internet of Things Journal 2023 [Link].
   Full version: [Cryptology ePrint].
   Source Code: [Reference Implementation of Lynx].
   Page No.: 37-90.

2. Context-Committing Authenticated Encryptions using Tweakable Stream Cipher.
   Donghoon Chang, **Munawar Hasan**[1].
   IEEE Access 2024 [Link].
   Page No.: 91-128.

[1]Authors are listed in alphabetical order, Corresponding Author: Munawar Hasan

Contents

List of Tables

List of Figures

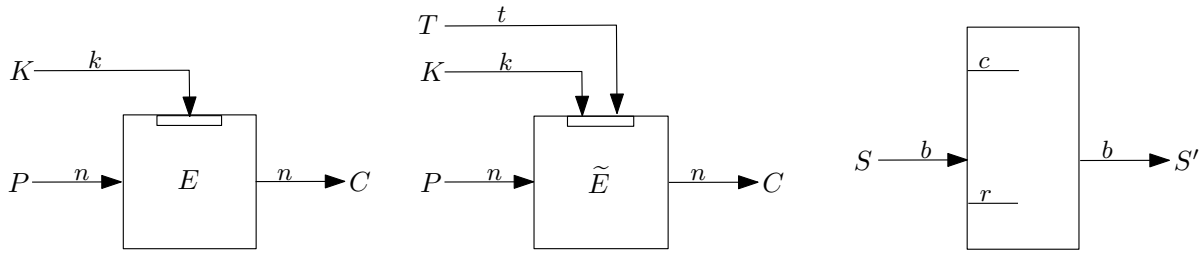
Chapter 1

Introduction

In the area of cryptography, one of the major concerns is to design a cryptographic system that is usable, provably secure, efficient and easy to analyze and verify. In the setup spanning to classical symmetric-key security notions, conventionally a blockcipher is often considered as the first choice in designing any cryptographic system. For example, given a secure blockcipher, one can design several varieties of provably secure systems, like authenticated encryption schemes with associated (AEAD) etc. These blockcipher based systems may include complex designs that are often difficult to analyze and present complicated security proofs that may be difficult to understand or even the research community may find it difficult to verify the security proof. On the contrary, tweakable primitives can offer some of the compelling properties that were described above, for example efficiency, high security bounds, and a simplistic approach to construct various operating modes; which can further lead to straightforward analysis and verification of the mode.

A blockcipher is a family of permutations parameterized by a secret parameter referred to as secret key, denoted by K. A tweakable blockcipher is a family of permutations parameterized by a secret key K and a public parameter referred to as tweak and denoted by T. Permutation is another type of primitive which is defined on b-bits. For example, a sponge function based permutation is defined on b-bits, where $b = c + r$ bits, such that, c is the capacity of the sponge function and r is the rate of the sponge function. From figure 1.1, we see that, in case of a blockcipher we have, $C = E_K(P)$ where $|K| = k$ and $|P| = |C| = n$, where n is the size of a

Figure 1.1: (Adapted from [1]) Structure of a blockcipher, a tweakable blockcipher and a sponge function based permutation.



block for the a given blockcipher. In case of tweakable blockcipher, we have an extra parameter .i.e., $C = \widetilde{E}(K, T, P)$ where $|T| = t$, while in case of permutation (sponge framework), we have $S' = \Pi(S)$, where $|S'| = |S| = b = c + r$. As a use case for a tweakable blockcipher, consider a web application, where the server has to encrypt one of its user's session ID and his profile. The server can utilize two separate tweaks for these two encryption tasks, thereby providing an interoperability between the two encryptions.

Historically, Mercy [4] appeared as the first evidence of a tweakable blockcipher. It defined a 128 bit parameter called Spice, that was known to the adversary. Mercy targeted disk encryption, where index of a disk sector can used as Spice. Mercy was later broken using a differential attack [5]. In a seminal paper [6], tweakable blockcipher was formally presented. The proposal listed three basic properties of tweakable blockcipher: (1) easy to design, (2) the cost of converting a blockcipher to a tweakable blockcipher is small, and (3) easy to design and prove modes of operation based on a tweakable blockcipher. Since, its inception, tweakable blockcipher and in general tweakable primitives were mostly overlooked. Only, during the last decade, the designs based on a tweakable blockciphers have gained some traction. Nevertheless, there are only a few available designs based on these primitives. For example, in the lightweight cryptography competition [7] organized by NIST [8], out of the ten finalists, Romulus [9] was the only finalist based on a tweakable blockcipher. Hence, there is considerable gap in the research community when it comes to the designs based on tweakable primitives. We leverage this opportunity and present our research, we propose several cryptographic designs based on the tweakable primitives. We are further motivated by the fact that NIST has recently shown interest for standardizing Accordion Mode [10], a tweakable enciphering mode (tweakable wide block cipher) that takes message input with variable input lengths (VIL). The term tweakable

enciphering mode indicates that the enciphering mode would take an additional input .i.e., a tweak.

In our research, we focus on creating authenticated encryption schemes with associated data or AEAD in short, (refer section 1.1), using tweakable primitives. Further, for our research, we do not limit ourselves only to a single design, but alternatively, we explore the entire class of AEAD designs that are based on *xor* operation and utilize a tweakable blockcipher as an underlying primitive, with the property that the tweak size of the tweakable blockcipher is twice as compared to the size of the block. Out of this entire class of AEAD designs, we analyze each of these designs and propose a family of AEAD with 14 members. We call this family of AEAD designs as lynx [11]. We provide provable security proof for each member of the lynx family. The members of the lynx family have several desirable properties like integrity assurance in *nonce misuse* and RUP scenario, *online*, free from *field multiplications* etc. Such properties are often desirable when the operating environment is very resource constraint and low-powered. This *resource constraint* environment is considered the design space of lightweight cryptography. We discuss this more in detail in the section 1.1.3.

We do not limit our research only on the primary features of an AEAD (.i.e., integrity and confidentiality), but we extend our work to the notion of committing security (refer section 1.3). Committing security is considered as secondary feature or additional feature of an AEAD. Notably, the specification requirement of the Accordion Mode also lists committing security as one of the desirable properties. Such initiatives have lead to the kicked-off of a new and active domain of research .i.e., the committing security of AEAD. Unfortunately, there is a lack of literature in this area of context commitment security analysis and, further, there is also lack of designs that provide context commitment. We fill this gap providing the security analysis of AEAD, by constructing an AEAD design using a tweakable wide blockcipher scheme under encode-then-cipher paradigm. We present successful CMT-4 attack with time complexity as $O(1)$ against four tweakable wide blockcipher schemes. Further, to thwart CMT-4 attack, we propose a new tweakable primitive called the tweakable stream cipher (or tS in short) with the property of partial collision resistance. Using tweakable stream cipher, we present four new constructions of tweakable wide blockcipher schemes, that are CMT-4 resistant under encode-then-encipher paradigm. We now discuss authenticated encryption scheme with associated data and provide its

properties that are relevant to our research and this thesis.

1.1 Authenticated Encryption with Associated Data Scheme (AEAD)

An authenticated encryption scheme with associated data (or AEAD)[1] provides both confidentiality and integrity simultaneously on the data. AEAD defined over two sub-routines, an encryption sub-routine (denoted by $\mathcal{E}$) and decryption sub-routine (denoted by $\mathcal{D}$):

1. Encryption sub-routine $(\mathcal{E})$:
   (a) Input: The encryption sub-routine $\mathcal{E}$ takes input as a key, a nonce, an optional associated data and a message.
   (b) Output: $\mathcal{E}$ outputs the ciphertext (corresponding to the message) and an authentication tag (message authentication code or MAC).
2. Decryption sub-routine $(\mathcal{D})$:
   (a) Input: The decryption sub-routine $\mathcal{D}$ takes input as a key, a nonce, an optional associated data, a ciphertext and a tag. $\mathcal{D}$ must use the same underlying primitive as $\mathcal{E}$.
   (b) Output: $\mathcal{D}$ outputs message if the message was used to create the supplied ciphertext and if the supplied tag was the one that was generated during the encryption sub-routine. If these two conditions doesn't hold, $\mathcal{D}$ outputs an error (usually denoted by $\perp$).

Usually, the associated data of an AEAD contains information related to the network communication between the two parties .i.e., packet header (example: network protocol information, packet headers, size etc., transaction IDs, size of the message in bytes, storage metadata etc.). It does not contain any sensitive information and hence, confidentiality is not necessary for associated data, but integrity is required for the associated data to ensure the recipient can verify

[1] We define AEAD formally in chapter 2.

the authenticity of the transferred data. In case of message, both confidentiality and integrity assurance is required.

1.1.1 Generic Composition Approach of AEAD

Initially, the protocol designers have addressed the design of AEAD using a composition approach [12], where one combines together two separate approaches .i.e., (1) an encryption scheme and, (2) a message authentication code (MAC). As an example, for a given message M and a header (or associated data) A, (1) encrypt M and then prepend A, (2) calculate the MAC of the resultant. Such solutions seemed inferential and obvious but had several cryptographic problems [13]. The cryptographic community later, started to recognize the idea of AEAD as cryptographic problem rather than merely a design problem. Hence, the design now would need to fulfill several cryptographic properties and must have security proofs to back the claim. This lead to the development of techniques that provided confidentiality and integrity as the primary feature of any proposed AEAD design. Some of the early efforts in this direction included the work of Jutla [14], Gligor *et al.* [15], and Rogaway *et al.* [16]. Mode of operation providing the dual property .i.e., confidentiality and integrity, and based on the blockcipher started to become very popular. Such *integrated* authenticated-encryption (AEAD) schemes promised improved efficiency compared to the generic composition of conventional mechanisms [13].

1.1.2 Types of AEAD

In this section, we discuss, three approaches for AEAD.

1.1.2.1 Encrypt-then-MAC (EtM)

In this approach, the plaintext is first encrypted using some encryption algorithm (for example AES [17]). This step ensures the confidentiality of the plaintext. After the encryption, a MAC is computed over the ciphertext itself using a MAC function like HMAC (keyed-hash message authentication code or hash-based message authentication code), which ensures integrity. This

MAC is generated using a separate key (in practice this key is usually derived from the same key material using some key derivation function). The ciphertext and the MAC are appended together and transmitted. Encrypt-then-MAC mechanism is generally considered the most secure of the methods for combining confidentiality and integrity, and the only one that ensures provable security. EtM can provide strong integrity guarantee since, the integrity of the ciphertext is verified before the decryption, reducing the risk of attacks on the decryption process. EtM is resistant to padding oracle attack and ciphertext modification attack; which can occur if the MAC is not applied to the ciphertext. EtM is often recommended and used in modern cryptographic protocols like IPsec and TLS (from version 1.2 onward)

The Galois/Counter mode (or GCM in short) [18] is based on EtM approach. It uses counter mode of operation, while the MAC is calculated using Wegman-Carter paradigm [19]. Due to counter mode of operation, GCM uses an initialization vector (or IV), which must be unique for each invocation to ensure confidentiality. Further, GCM uses the same key for the message encryption, and to derive a key for calculating GHASH.

1.1.2.2 Encrypt-and-MAC (E&M)

The plaintext is first encrypted using some encryption algorithm (for example AES [17]). This step ensures the confidentiality of the plaintext. After the encryption, a MAC is computed over the original plaintext (not the ciphertext). The MAC ensures the integrity of the original plaintext. The MAC is generated using a MAC algorithm like HMAC (though often derived from the same key material used for encryption). The ciphertext and the MAC are appended together and transmitted. While E&M provides both confidentiality and integrity, its primary weakness lies in applying the MAC to the plaintext rather than the ciphertext, making it vulnerable. Hence, it is generally considered less secure than EtM.

1.1.2.3 MAC-then-Encrypt (MtE)

The MAC is computed over the plaintext using some function like HMAC with a MAC key. The MAC ensures the integrity of the plaintext message. Now both the plaintext and the generated

MAC are appended together and then encrypted using a symmetric encryption algorithm (such as AES [17]). This ensures the confidentiality of both the plaintext and the MAC. Though this approach provides both confidentiality and integrity, its main disadvantage: decryption is invoked before the MAC verification, which can expose it to some potential attacks on the encryption scheme.

1.1.3 Design Goals of AEAD

Any proposed AEAD algorithm should have several desirable properties depending on the area and platform of application. Some of these properties may or may not have practical significances in general, for example: efficient encryption and decryption, at least birthday bound security for both authenticity and confidentiality, hardware friendly, nonce misuse resistant under various scenarios etc. We list down some design goals, that may be desired (but not all necessary simultaneously):

1. Inverse-free([20]): An AEAD is called inverse-free if both the encryption sub-routine $\mathcal{E}$ as well as the decryption sub-routine $\mathcal{D}$ invokes only the encryption algorithm of the underlying primitive. This property is desirable when considering the area of lightweight category. The inverse-free property eludes the necessity of implementing the decryption algorithm of the underlying primitive on the device, hence thereby making the AEAD more memory efficient both in the storage as well during the execution. This property also facilitates the idea of compact design. The designers often want to minimize the use of temporary variables. This can result into faster implementations which benefits the IoT devices in providing real-time updates. For example, Romulus [9] and GIFT-COFB [21] makes only the encryption calls to the underlying primitive for both $\mathcal{E}$ and $\mathcal{D}$.

2. Online ([20]): The property of online is among the most desirable property of any AEAD, since, it facilitates real-time communication. We refer this property as *stream processing*, since it emits of data stream continuously out of the AEAD sub-routines. In this property, during the invocation of encryption sub-routine $\mathcal{E}$, the generation of ciphertext from the plaintext depends only on the current block of the plaintext and all the previous blocks

of plaintext. This basically means that the current block processing does not need to know about future blocks and their properties like total length of message etc. The same argument applies for the decryption sub-routine $\mathcal{D}$. The property of online, also facilitates low memory usage. Due to the fact the ciphertext are released even before completion of whole message processing; a lot of memory is saved for particularly long messages. In resource constraint devices this property becomes even more relevant and useful.
Real-Time Messaging Protocol (RTMP), WebRTC (Web Real-Time Communication), Real-Time Transport Protocol (RTP) etc., are some of the examples that can make use of this property and hence, the message/data can be released on the fly, making low latency, high user experience.

3. Single-pass (or one-pass): A single pass AEAD is one that makes a single pass through the entire data .i.e., nonce, associated data and the message, for the generation of ciphertext and the tag. In contrast, a two-pass or in general a multi-pass AEAD needs to process data more than once before it can generate the output .i.e., ciphertext and tag. Hence, a single-pass AEAD is efficient due to the computational cost incurred in processing data as compared to a multi-pass AEAD. The rate of an AEAD is the number of blocks of plaintext processed per non-linear (block-cipher, field multiplication) operation. AEAD designs with high rate have a shorter latency and hence achieve higher speed, and is therefore more desirable. As an example, rate of SIV [22] is 0.5, while rate of OCB [16] is 1.

4. Parallelizable: In parallelizable AEAD, some of the components of encryption and decryption sub-routine can be done in parallel. GCM [18] utilizes counter mode of operation and hence encryption of the counters can be done in parallel. Similarly, GCM-SIV [23] can also obtain parallelism. The property is highly desired as it can increase the throughput of the communication many folds, especially when dealing with larger messages. On the other hand, any parallel computation needs the support from the hardware. A single core single thread CPU cannot provide parallel computation. Hence, a multi-core with hyperthreading architecture hardware is required to be able to use full parallelism potential of the underlying AEAD algorithm.

5. Security Goals: Integrity and confidentiality are two conventional security goals of AEAD.

They are defined under various security models as described below:

- UNAE - Unique nonce AEAD: These schemes provide security guarantees only when the nonce is used once with a given key. For example: the security GCM [18] is defined under UNAE.
- MRAE - Misuse Resistant AEAD: These schemes provide security in scenarios where nonce may be reused, on the condition that combination of nonce, associated data and message are not repeated. For example: GCM-SIV [24] and AEZ [25] provides MRAE. Providing MRAE may lead to sufficiently larger amount of block cipher calls and also the AEAD may loose the property of 'online' [26]. The design goals for nonce misuse resistant AEAD are as follows:
  - Even if the same nonce is used multiple times, the AEAD scheme must not break down or leak the key material.
  - If an adversary $\mathcal{A}$ attempts to create attack query using the repeated nonce, then the integrity security of the AEAD under scrutiny must not be compromised.

  The property of MRAE may be desirable in scenarios where generating nonce is a costly process. In the case of very low-powered IoT devices, both computation power and the randomness is limited, and generating a cryptographically secure nonce each time during communication is very costly process and even infeasible. SIV [22] is MRAE. Further, these IoT devices often have very low memory and would try to minimize the storage of processed data. INT-RUP (integrity security assurance under release of unverified plaintext scenario) assures integrity security when plaintext is released to the adversary even before the verification process is completed. This is often a desirable property from an AEAD targeting the area of lightweight cryptography. Further, the designs that provide provable security proofs are more desirable as compared to the design that are not provably secure. Provable security proofs provide the bounds on integrity security and confidential security under various security models.
- RAE - Robust AEAD: Under a robust authenticated encryption scheme, a user can choose an arbituray value of a parameter denoted by λ for ciphertext expansion, where

$\lambda \in \mathbb{N}$. Hence, for a given plaintext, the ciphertext generated is λ-bits long, e.g., AEZ [25].

- RUP - Release of Unverified Plaintext: Under the RUP scenario [27], the plaintext is released before even it is verified .i.e., the adversary $\mathcal{A}$ has the access of the plaintext even in case of a validation failure. Under the notion of RUP setting, the decryption sub-routine $\mathcal{D}$ of an AEAD is split into two separate sub-routines, (1) $\mathcal{D}$: a decryption sub-routine responsible for only decryption, and (2) $\mathcal{V}$: a verification sub-routine responsible for verification. The integrity security of an AEAD under the RUP scenario is denoted by INT-RUP. Some of the example of AEAD under this model are: TinyJAMBU [28], `LOCUS` and `LOTUS` [29] etc.
- IND - Indistinguishability: For a given AEAD scheme, indistinguishability is defined as the ability of an adversary $\mathcal{A}$ to distinguish the output produced by the encryption sub-routine $\mathcal{E}$ from an output produced by a random function (often denoted by $\$$). This notion of AEAD security is denoted by IND-CPA or in some cases by CONF (to denote confidentiality security of an AEAD). Further, to executed this attack, $\mathcal{A}$ must be nonce respecting.

We present the formal definitions of INT-RUP security and CONF security of an AEAD in section 2.1.

Committing security is not a conventional AEAD security goal but very much desired in real-world problems. The notion of committing security binds an AEAD to a context .i.e., its inputs to encryption sub-routine $\mathcal{E}$. This field is an active area of research and several AEAD schemes are analyzed under this security notion. In the area of lightweight category, this notion becomes paramount, since the devices might want to re-use some of the inputs to save computation. We discuss more about committing security of an AEAD in section 1.3.

6. Field Multiplication: It provides non-linearity to the design but can increase the overall computational cost of the design. Desktop grade hardware can easily handle such an increase in computational, but for an IoT device, field multiplication may not be a good choice. It is also worth mentioning that some designs utilize field multiplication in a

very niche way, thereby providing fast field computation. One such design example is Romulus [9].

7. Multi-User Security: In real-world scenario, there are often multiple users simultaneously communicating using multiple keys, e.g., TLS-1.3. In this multi-user setup, the adversary $\mathcal{A}$ may have access to multiple keys with encryption and verification oracle with respect to each of the key [30]. In this setup, some pre-computation may also be performed by $\mathcal{A}$. Further, $\mathcal{A}$ can also use massively parallel systems to deploy this attack.

Equipped with above design goals of an AEAD, we can now move our head towards the lightweight alternative.

1.2 Lightweight Cryptography

Lightweight cryptography is commonly defined as cryptography for resource constrained devices i.e., devices that have some physical limitations like menial computational power, extremely low memory, limited power source, low area of fabrication etc. Example of such devices could be RFID tags, IoT sensors etc. Lightweight cryptography covers wide range of application spanning across various diverse industries. Smart wearable like smart watches, smart glasses etc., intelligent vehicle monitoring systems like parking sensors, traffic monitoring, real time vehicle tracking etc., life critical medical devices like patient vital monitoring sensors, and much more. A lot of these applications of modern day computation falls into the area of the Internet of Things (or IoT for short). The IoT opens up immense number of areas for the applications of lightweight cryptography [31, 32, 33].

What sets lightweight cryptography apart from "conventional cryptography?" [34]. Upon browsing the above applications, it may seem that the design of lightweight cryptographic primitives are straight forward but ironically, it is extremely difficult to design a primitive or related application in this category. One of the fundamental problem of this difficulty is the lack of standard for the criteria to be in the lightweight category. There is a lack of consensus among the cryptographic community to list down the exact criteria, and hence, the line between the

conventional cryptography and the lightweight cryptography becomes blurry. Instead there has been several competitions [35, 7] going round the world that evaluate several candidates for lightweight cryptography.

It is very important to understand that lightweight cryptography should not be considered or compared with a weak cryptography. By no means, they are weak. Hence, their design requires great knowledge and expertise. The idea of using a small key size or blocksize or having a lower number of rounds of encryption algorithm (as compared to the conventional one) is flawed. The lightweight proposal must still stand under various security notions as the conventional counterparts. It is possible that under different practical applications, different properties from the lightweight proposal may be needed. In [34], the author describes the design space of lightweight cryptography and presents rationale behind its design and analysis. Some of the examples of the devices that have limited computational and memory capabilities and need optimized cryptographic operations are listed below:

1. IoT Devices: Where sensors and controllers need secure communication with minimal energy consumption.
2. Embedded Systems: Such as medical devices, where memory and power are highly constrained.
3. Mobile and Wearable Devices: Where power efficiency is key to extending battery life.
4. RFID and NFC Systems: Where low-latency encryption is required, and power consumption is critical.

1.2.1 Lightweight AEAD

Lightweight AEAD is a sub-area of lightweight cryptography. In section 1.1.3, we saw the design goals of conventional AEAD. The design of lightweight AEAD is similar to that of the conventional AEAD. The differentiating criteria is the area of application. By lightweight, we inherently consider that the designs would be used in resource constraint devices as described in section 1.2. Usually, *inverse free*, *compact state*, *online* and *single-pass* are some of the areas

that the designers concentrate when proposing an AEAD in the lightweight category. Last couple of years have seen several proposals [36, 37, 38, 9, 39, 40, 28, 41, 42, 43, 21, 11] in this category that utilizes various properties of the underlying primitives and provides resilience or security bounds under various security notions.

CAESAR competition [35] for authenticated encryption scheme with associated data had a lightweight category. Several candidates were submitted in this lightweight category. The CAESAR competition defines the requirement of the lightweight category for the candidates as the algorithms that are suited for *resource constrained environment*. ACORN [37] and ASCON [38] were selected as the winners in the lightweight category of the competition. National Institute of Standards and Technology (NIST) [8], initiated a process in 2017 to solicit, evaluate, and standardize lightweight cryptographic algorithms that are suitable for use in constrained environments where the performance of current NIST cryptographic standards was not acceptable and, in 2019, NIST started the lightweight cryptography competition, called LWC [7]. The standardization process at NIST relies on the efforts of researchers from the cryptographic community across the globe that provide security, implementation, and performance analysis of the candidate algorithms. Further, NIST strongly encourages public evaluation and publication of the results throughout the process. Hence, the winners of the NIST standardization competitions are recognized all around the globe and available for use in most of the commercial and the non-commercial cryptographic libraries [44, 45, 46, 47]. The LWC competition called for submissions for authenticated encryption with associated data (AEAD) and optional hashing functionalities. There were 57 submissions to this competition, out of which 56 were selected as round one candidates. Out of these 56 candidates, 10 were selected as the finalists. ASCON [38] was selected as the winner of the lightweight cryptography competition, and currently NIST is working on standardizing ASCON.

These competitions, their lightweightedness criteria and standardization process have led a huge traction in the area of design and analysis of lightweight authenticated encryption schemes with associated data. Further, the researches are eager to achieve a better security bound under various attack models and achieve a better implementation (low memory usage and computationally menial). Release of unverified plaintext or RUP [48] scenario, is one of the active area of research when designing any AEAD, lightweight or conventional one. The idea of releasing plaintext

before authenticating tag is very practical when underlying device has a low memory and hence, RUP is very desirable property for lightweight AEAD. One may favor a birthday bound security under RUP scenario for the integrity security rather than a beyond birthday bound (BBB) security without the RUP setting for the integrity security.

1.3 Committing Security of AEAD

Over the last decade or so, the notion of security realted to an AEAD have gone under several revisions [49]. We saw in section 1.1, that one of the inputs to the encryption sub-routine $\mathcal{E}$ and well as the decryption sub-routine $\mathcal{D}$ is the nonce. The idea was initially proposed in [50], referring them as the *IV*, later the term nonce was universally accepted. An AEAD might impose various security settings when it come to the use of nonce:

Instead of recent advancements and active research community using competitions for standardization (example: CAESAR [35] and NIST [8, 7]), the primary security goal of an AEAD design has always been integrity and confidentiality. These two are considered as the conventional security notions of AEAD schemes. However, these two notions do not necessarily capture the security requirements expected in real-world applications. Recently, the attack on Facebook's franking protocol [51], allowed an adversary to send objectionable contents to a recipient without being detected. Subscribe with Google (or SwG in short) is subscription service provided by Google. In an attack [52] on SwG, it was shown that a malicious publisher can display different contents to different authorizers. In yet another attack [53], it was shown that an adversary can build a practical partitioning oracle attack that it able to extract password from the server. In all the above attacks, the goal of the adversary was not to break the conventional security. In fact, the attack were possible only due to the fact that the ciphertexts were able to decrypt correctly (cipheretext and tag verification) into two different plaintexts using multiple keys. These recent progress have highlighted the vulnerabilities that exists in the design of an AEAD even though integrity and confidentiality is assured; and led to the realization of the importance of commitment security. Broadly, the question it raises: whether an authenticated encryption with associated data scheme will decrypt the adversarially-chosen ciphertext under

two different, adversarially chosen contexts (secret key, nonce, associated data and message) into two different plaintext? Despite of recent attacks, many open questions remain around the context commitment for several AEADs like [54, 55, 56]. Some of the recent work [57] have tried to answer the committing security of various AEADs, but a lot still remain to be analyzed, and hence opens up an active area of research.

Since, the introduction of the committing security [58] for an AEAD, the area has seen considerable amount of interest. The of notion context committing security can be formalizes using a security game: Let $\mathcal{E}$ denotes the encryption sub-routine of an AEAD for which the committing security needs to be proven. Now, the adversary $\mathcal{A}$ chooses two tuples (K, N, A, M) and (K', N', A', M'), denoted by τ .i.e., where $\tau = \{(K, N, A, M), (K', N', A', M')\}$, then $\mathcal{A}$ wins if $\mathcal{E}(K, N, A, M) = \mathcal{E}(K', N', A', M')$. Context committing security is also often denoted by notation CMT-4 security. The numeral 4, denotes the commitment of $\mathcal{E}$ to all the four inputs. Hence, CMT-4 security is considered as the strongest form of committing security. As outlined in [49], there are two approaches to provide committing security, (1) generic AEAD transform, and (2) specific AEAD transform. A generic transform is a methodology which can be combined with any existing scheme, resulting in a new AEAD scheme with committing security. While, a specific methodology is strictly tied to a specific AEAD that is under scrutiny. In this thesis, we focus on the later kind of methodology .i.e., specific transform (refer section 1.3.2).

1.3.1 Context Commitment in Lightweight Cryptography

Take lightweight cryptography under consideration, the notion of committing security becomes even more paramount. In section 1.1.3, we mentioned the design goals for a device in the lightweight category. We saw that, these devices (like IoT devices, RFC devices etc.) often struggle with the trade-off between security and computational power. It is often difficult for these devices to generate new parameters or even generate cryptographically secure parameters for each and every AEAD communication in the same session. For example, it is simply not possible to generate unique nonce for all the communincation within a session, and hence, these devices tend to reuse the parameters as much as possible. This leads to the issue of committing security for an AEAD. In figure 1.2, we see that in the design space of lightweight AEAD,

Figure 1.2: Committing Security as a secondary feature of AEAD

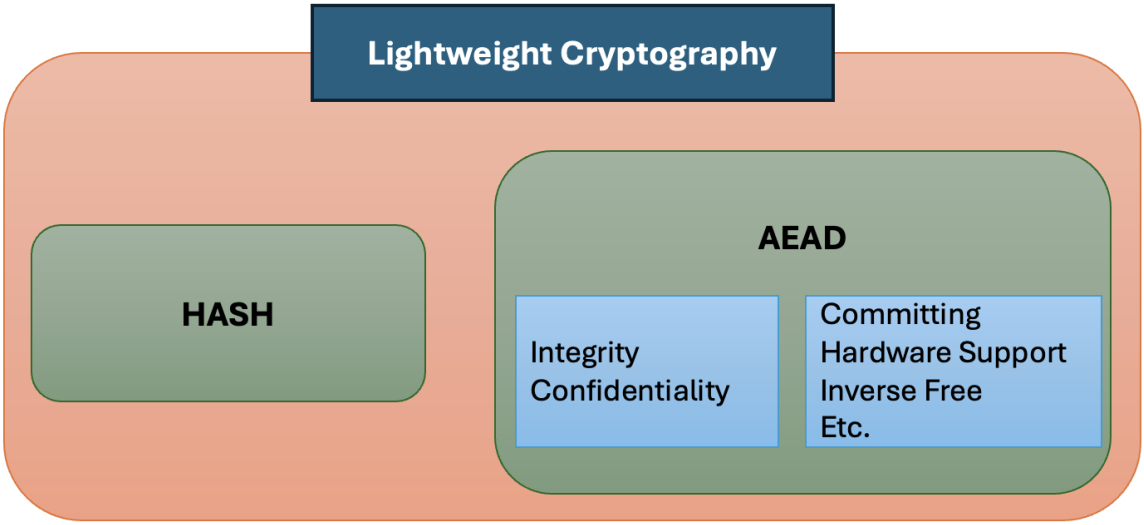
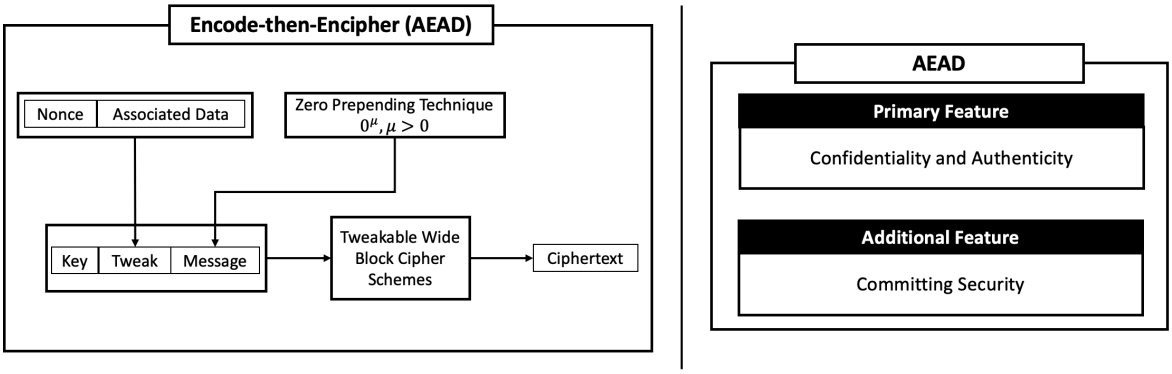


Figure 1.3: (Adapted from [2]) Committing Security as a secondary feature of AEAD



committing security is present as one of the desirable features, and hence, it is imperative to analyze the effect of reusing some of the parameters of AEAD during the communication and whether this reusage leads committing security compromise as described in section 1.3.

1.3.2 Encode-then-Encipher Paradigm

Encode-then-Encipher [59] (EtE in short) is a cryptographic design paradigm that combines two distinct operations to ensure both data integrity and confidentiality:

1. Encode: The most common technique is prepending or postpending zeros to the input message. Hence, for a given message M, the encode sub-routine may output $M' \leftarrow (0^{\mu} \parallel$

M) (prepending) or $M' \leftarrow (M \parallel 0^{\mu})$ (postpending), where $\mu > 0$.

2. Encipher: The encoded message M' is then encrypted.

Message encoding serves as a technique for providing integrity while, message enciphering leads to confidentiality.

A tweakable wide block cipher scheme [60] takes three inputs: a key, a tweak and a plaintext, and produces a ciphertext as the output, where size of the ciphertext is equal to size of the plaintext. The scheme is particularly useful in applications like disk encryption. Further, they also find application in the lightweight category where the resources are limited. Encode-then-encipher paradigm can be used to construct an authenticated encryption scheme from a tweakable wide block cipher scheme by inserting zeros at a specified position. In figure 1.3, we see that, we can create and AEAD using a tweakable wide block cipher scheme under EtE paradigm by prepending zeros. The scheme so created must also provide additional feature of committing security. We now present an overview of provable security.

1.4 Provable Security

Provable security is a formal approach in cryptography that provides mathematical guarantees about the security of cryptographic protocols or algorithms. Instead of relying solely on heuristic arguments or empirical testing, provable security aims to demonstrate that breaking a scheme is at least as difficult as solving a well-known hard problem (such as factoring large integers or solving discrete logarithms). Hence, it is often referred to as *reductionist proof* in the research community, due to the fact that the security of a cryptographic construction is reduced to some assumed hardness of an underlying problem or primitive. The notion of provable security was formally introduced by Goldwasser and Micali [61] in context of asymmetric encryption. It is worth noting that the provable security does ensure that a scheme is secure, but the security is precisely based under several assumptions like security model, attack models and the leverage that an adversary possesses. When providing a provable security proof for a scheme, informally, our task is to show either of the following two conditions (adapted from [62]):

- Logical contradiction, for example, security against an unbounded adversary.
- Breaking the underlying atomic primitive efficiently.

Having now equipped with the notion of provable security, we move forward to the related work.

1.5 Related Work

As discussed earlier, there has been limited work in the area of tweakable primitives. We first outline some of the available tweakable blockciphers. Then we move towards authenticated encryption scheme with associated data with a focus on the lightweight category. We then present the literature survey for tweakable wide blockcipher and encode-then-encipher paradigm with focus on committing security.

1.5.1 Tweakable Primitives

One of the early work of tweakable blockcipher emerged in [4, 63, 64]. Goldenberg *et al.* [65] showed mechanism to transform a Feistel scheme into a tweakable Feistel scheme. This tweakable Feistel scheme achieved birthday bound security. Later in around 2014, a TWEAKEY framework was proposed in [66], which amalgamates the functionality of the key and the tweak inputs of a tweakable block cipher. Some of the recent work on tweakable blockcipher include [67, 68]. `SKINNY` emerged as one of the most popular tweakable blockcipher. A low latency version of `SKINNY` called `MANTIS` is specially crafted for low-powered devices. `Deoxys-TBC` [69] a recent addition to tweakable blockcipher, is based on AES [17]. A Lightweight leakage-resilient PRNG (pseudo-random number generator) was proposed recently [70], the proposed PRNG is based on a tweakable blockcipher.

1.5.2 AEAD

The inception of authenticated encryption with associated data [71] simulated research in the area of authenticated encryption modes. In the recent years, there has been a lot of traction in

the area of lightweight cryptographic algorithms. Such algorithms are tailored for IoT devices. [72, 34, 73, 74] discussed a general overview of design strategy of lightweight cryptographic primitives. [75, 76] presents a comparative overview of several cryptographic primitives. We primarily try to focus on lightweight authenticated encryption schemes.

1.5.2.1 Lightweight Cryptography

Several authenticated encryption schemes that rely on sponge function [77, 78], utilize duplex mode of operation. The LWC competition at NIST [8] has seen wide range of submissions based on sponge functions that are in lightweight category. Sparkle [40] and Xoodyak [39] both NIST LWC competition finalists are based on sponge function. ASCON [38], the winner of NIST LWC and the winner of CAESAR [35] competition uses the duplex mode of authenticated encryption. PHOTON-Beetle [36] based on Beetle [79] which is a sponge based mode and ISAP [42], also based on sponge based function are also the NIST LWC finalists.

Next we discuss blockcipher based authenticated encryption schemes that fall into the lightweight category. GIFT-COFB [21], a blockcipher based authenticated encryption scheme with associated data uses GIFT blockcipher [80] as the underlying primitive. GIFT-COFB is also one of the finalists of NIST lightweight cryptography competition. Another finalist, Grain-128AEAD [81], uses stream cipher from the Grain [82] family of stream ciphers as the underlying primitive. In recent years, tweakable blockciphers has gained lot of interest among the cryptographic community because of the tweak material, which can be effectively used for designing authenticated encryption schemes with associated data. Romulus [9], another NIST lightweight cryptography competition finalist uses SKINNY [67] tweakable blockcipher for their authenticated encryption design. AET-LR [43] is also based on a tweakable blockcipher and uses associated data or message as both part of input as well as the part of tweak to the underlying tweakable blockcipher. TinyJAMBU [28] is based on keyed permutation while elephant [41] is based on cryptographic permutation masked using LFSRs. Both TinyJAMBU and elephant are the finalists of NIST lightweight cryptography competition.

1.5.3 Committing Security under Encode-then-Encipher Paradigm

Bellare and Rogaway presented a method for combining confidentiality and authenticity, called encode-then-encipher (EtE) [59]. The authors introduced various encoding schemes for the message, and analyzed the enciphering of the encoded message. The EtE paradigm was initially overlooked, but with the advent of CAESAR competition [35], it found traction. A tweakable wide block cipher scheme provides only confidentiality and not authenticity. Using encode-then-encipher paradigm, a tweakable wide block cipher scheme can be converted into an authenticated encryption scheme by inserting 0^λ at a specified position, where $\lambda > 0$. The common methodology is to prepend or append 0^λ. A *robust* (key-committing) authenticated encryption scheme (RAE in short) [25], must provide confidentiality and authenticity for the chosen value of λ. AEZ [25] based on AES [17] round function is an encode-then-encipher construction. AEZ appends 0^λ to the message M and uses a tweak T comprising of a nonce N, associated data A and λ .i.e., $T = (N \parallel A \parallel \lambda)$. Authenticator [25] .i.e., 0^λ, is used to provide integrity under EtE paradigm. We focus on creating authenticated encryption scheme using a tweakable wide block cipher scheme [60]. HBSH [83], HCTR2 [84] and two deck function based schemes .i.e., double-decker and docked-double-decker [85] are popular tweakable wide block cipher schemes that target file encryption and related applications. HBSH, double-decker and docked-double-decker target devices that lack cryptographic hardware instructions like AES. CBC [86] and XTS-AES [87] became quite popular in disk encryption applications like BitLocker [88], yet they were not very popular in the design space of IoT devices, since AES may be slow when hardware support is absent. CBC encrypted binary files was shown to be vulnerable to malleability, leading to arbitrary code execution [89]. XTS is based on ECB mode and hence is also prone to malleability .i.e., changing a bit of ciphertext influences its corresponding plaintext block only. The design space of IoT devices is called lightweight cryptography [35, 7]. There is an associated trade-off between the security and the efficiency in the design space of lightweight cryptography. This field has gained a lot of interest in the resent years, leading to proposal of several authenticated encryption schemes in this category [38, 21, 9, 11] under various security notions. In this paper, we concentrate on creating a CMT-4 secure authenticated encryption scheme from a tweakable wide block cipher under encode-then-encipher paradigm.

Key commitment security analysis on AEGIS [90], one of the winners of CAESAR competition, was presented in [91] with $O(1)$ time complexity. In [92], the authors analyze the committing security of Ascon [38], winner of NIST lightweight competition. KIVR [93] is a new mode that transforms existing authenticated encryption schemes to have CMT-4 security without increasing the ciphertext size. In [94], the authors provide committing security analysis for all the NIST LWC competition finalists (except Grain-128AEAD). In [95], the author describes methodologies to add key commitment property to an authenticated encryption scheme.

1.6 Thesis Contribution

Tweakable primitives can be used to create flexible, secure, and efficient cryptographic systems. Further, the designs must also provide provable security. In the recent years, there has been a lot of traction in the design of authenticated encryption schemes with associated data targeting the space of lightweight cryptography. With initiative of standardization by using public competitions, such as CAESAR and NIST LWC, the design of lightweight AEAD has becomes more and more relevant and important research field. Unfortunately, due the several constraint imposed by *limited resources*, all the designs in the lightweight category have some trade-offs. Researchers often try to address these trade-offs in their proposed AEAD designs. Tweakable primitives offer interesting design strategy that can be utilized for creating AEAD for these resource constraint devices. The primary requirements mandate that the property of integrity and confidentiality for any design. Further, provable security proofs is also desirable for the proposed design. In addition to primary requirements some additional features are desirable as well. Thus, it is natural to ask– how to ensure that a given authenticated encryption scheme with associated data is suitable to be used in a resource constraint environment? What kind of additional features are necessary and how to achieve them? In these context, the thesis broadly looks into the following research questions.

1. **Can we utilize a tweakable primitive (tweakable blockcipher) to design an authenticated encryption scheme with associated data with a provable security proof, such that the design suitable for operating in resource constraint environment?**

In other words, what are all the possible ways to create an AEAD using a tweakable blockcipher (with tweak size twice the block size), that provides birthday bound security for confidentiality in nonce respecting scenario and birthday bound security for integrity in nonce misuse and RUP scenario. The proposed AEAD must be suitable for operating in resource constraint environment and must have features like *stream processing*, *1-pass rate-1*, *no field multiplications* etc. We can find answers to these questions in Chapter 3.

2. **Can we design a CMT-4 secure AEAD using tweakabale wide blockcipher schemes under encode-then-encipher paradigm and, utilizing the notion of tweakable stream cipher providing partial collision resistance against a CMT-4 adversary?**

   To be precise, is it feasible to design an AEAD scheme using tweakable wide blockcipher schemes under EtE paradigm that ensures CMT-4 security? What are the security bounds for such schemes? We address these concerns in Chapter 4. Further, to answer this question, in this chapter, we present a new tweakable primitive called the tweakable stream cipher (or tS in short), with the property of partial collision resistance.

Chapter 2

Preliminaries

We use several concepts and functions to propose our constructions in chapter 3 and chapter 4. In this chapter, we discuss these concepts, notation, security notion and definitions. We begin by providing the notation of AEAD security in section 2.1. In this section we also provide the definition of AEAD and then followed by security notion of an AEAD in section 2.2. In section 2.2, we also provide security notions that are need for chapter 4.

2.1 Notations and Definitions

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string ϵ while $\{0,1\}^+$ denotes the set of all finite bit strings excluding ϵ. For $X \in \{0,1\}^*$, we denote the bit length of X as $|X|$. Hence $|\epsilon| = 0$. For $X \in \{0,1\}^*$ and $Y \in \{0,1\}^*$, $X||Y$ denotes concatenation of X and Y in respective order. We also denote XY as the concatenation, if it is clear from the context. $bin_x(y)$ is used to denote x bit wide binary representation of y. Let 0^i (or 1^i) be the string of i zero (or one) bits, for example: $1 \parallel 0^i$ or 10^i refers to bit 1 followed by i zero bits. $X \oplus Y$ denotes the bitwse xor between X and Y where $|X| = |Y|$. For a given $X \in \{0,1\}^+$, a function $\text{Trunc}_i(X)$ returns i least significant bits of X. If $i > |X|$, then $\text{Trunc}_i(X)$ returns a null character (a sentinel) denoting the error., while if $i = |X|$, then $\text{Trunc}_i(X)$ will return an empty string or ϵ. We define $\nu \in \{0,1\}^*$ as suffix of $X \in \{0,1\}^*$ if ν is an end part of X, hence ϵ and X are

always suffixes of X. A function $\text{Extract}_\nu(X)$ eliminates the suffix ν from X and then returns all the rest bits of X, if ν is not a suffix or $\nu > |X|$, then $\text{Extract}_\nu(X)$ returns a null character (a sentinel) denoting the error. For example $\text{Extract}_{10}(1010010)$ returns 10100 i.e., extract 10 from the end part of 1010010. For $X \in \{0,1\}^+$, we define X in x blocks (or block length of X is x) in the following way: $(X[1] \parallel X[2] \parallel ... \parallel X[x]) = X)$ where $x = \lceil \frac{|X|}{b} \rceil$ and $|X[x]| \leq b$. The block representation of X can also be called as the parsing of X into b bit blocks. Now, we define a padding function: for a given $X \in \{0,1\}^{<b}$:

$$pad_b(X) = \begin{cases} \epsilon & \text{if } |X| = 0 \\ X \parallel 10^{b-|X|-1} & \text{if } |X| < b \end{cases} \tag{2.1}$$

Next we define a zero padding function denoted by pad-$0_l(X) = X \parallel 0^v$, where $v \geq 0$ and $(|X| + v) \pmod{l} = 0$. Now, we define another padding function: for a given $X \in \{0,1\}^{<\eta}$:

$$\text{pad}_\eta(X) = \begin{cases} X \parallel 10^{\eta-|X|-1} & \text{if } |X| < \eta \\ \perp & \text{otherwise.} \end{cases} \tag{2.2}$$

The padding function defined by equation (2.1) is used in chapter 3, while the padding function defined by equation (2.2) is used in chapter 4.

We use several concepts definitions and functions in our construction. They are as discussed below.

- **Authenticated Encryption (AEAD):** An authenticated encryption with associated data (in short AEAD) is a family of algorithms denoted by AEAD=$(\mathcal{E}, \mathcal{D})$ that consists of an encryption sub-routine $\mathcal{E}$ and a decryption sub-routine $\mathcal{D}$ defined over a key $K \in \{0,1\}^k$, a nonce $N \in \{0,1\}^n$, an associated data $A \in \{0,1\}^*$, a message $M \in \{0,1\}^*$, a ciphertext $C \in \{0,1\}^*$ and a tag $T \in \{0,1\}^{tag}$ such that:

$$\begin{aligned} \mathcal{E} &: (K, N, A, M) \mapsto (C, T) \\ \mathcal{D} &: (K, N, A, C, T) \mapsto M / \perp \end{aligned} \tag{2.3}$$

where:

$$\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) \mapsto M$$

Alternatively using the notation from [48], we can define RUP security. In the RUP setting, the real-world constitutes both the encryption $\mathcal{E}_K$ sub-routine and the decryption $\mathcal{D}_K$ sub-routine over a random key K. Further, the decryption sub-routine $\mathcal{D}_K$ provides the decrypted ciphertext (message) without any verification.

Hence, alternatively, an authenticated encryption with associated data can be defined in a non-conventional way as AEAD=$(\mathcal{E}, \mathcal{D}, \mathcal{V})$ as follows:

$$\begin{aligned} \mathcal{E} &: (K, N, A, M) \mapsto (C, T) \\ \mathcal{D} &: (K, N, A, C, T) \mapsto M \\ \mathcal{V} &: (K, N, A, C, T) \mapsto \top / \perp \end{aligned} \tag{2.4}$$

where:

$$\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) \mapsto M$$

$$\mathcal{V}(K, N, A, \mathcal{E}(K, N, A, M)) \mapsto \top$$

From the equation (2.4), it is clear that the decryption sub-routine of non-conventional AEAD unlike the the decryption sub-routine of conventional AEAD returns the message without verification. In both conventional and non-conventional definition of AEAD, $\perp$ symbol denotes always a false outcome while a $\top$ symbol denotes always a true outcome. In an alternative notation, we denote an authenticated encryption scheme by following sub-routines: $(\mathcal{E}, \mathcal{D}, \mathcal{V})$, where $\mathcal{V}$ is the verification sub-routine.

Further, in yet another notation, we can define an AEAD where the tag T is a part of the ciphertext. This notation is used for encode-then-encipher paradigm and is defined below:

Authenticated Encryption (AEAD): An authenticated encryption with associated data (in short AEAD) is a family of algorithms denoted by AEAD=$(\mathcal{E}, \mathcal{D})$ that consists of an encryption sub-routine $\mathcal{E}$ and a decryption sub-routine $\mathcal{D}$. The encryption sub-routine takes

input as a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an associated data $A \in \mathcal{M}$, a message $M \in \mathcal{M}$ and produces output as a ciphertext $C \in \mathcal{C}$ such that:

$$\begin{aligned} \mathcal{E} &: (K, N, A, M) \rightarrow C \\ \mathcal{D} &: (K, N, A, C) \rightarrow M/\perp \end{aligned} \tag{2.5}$$

where:

$$\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) \rightarrow M.$$

We use the above definition of AEAD .i.e., equation (2.5) in chapter 4.

- **Tweakable Blockcipher:** A tweakable blockcipher $(\widetilde{E}, \widetilde{D})$ is defined over a key $K \in \{0,1\}^k$, a tweak $\mathcal{T} \in \{0,1\}^t$, a message $M \in \{0,1\}^b$ and a ciphertext $C \in \{0,1\}^b$ such that:

$$\begin{aligned} \widetilde{E} &: (K, \mathcal{T}, M) \mapsto C \\ \widetilde{D} &: (K, \mathcal{T}, C) \mapsto M \end{aligned} \tag{2.6}$$

where:

$$\widetilde{D}(K, \mathcal{T}, \widetilde{E}(K, \mathcal{T}, M)) \mapsto M$$

A tweakable blockcipher should be efficient i.e., both encryption $\widetilde{E}$ and $\widetilde{D}$ should be easy to compute [6]. Occasionally we also denote a tweakable blockcipher by following notation: $(\widetilde{E}_K, \widetilde{D}_K)$

2.2 Security Notions

There are two security notions of an authenticated encryption scheme with associated data AEAD: confidentiality and integrity. We provide the formal definition of the confidentiality and the integrity of AEAD followed by the combined notion of AEAD. The definitions of AEAD is taken from [96]. Let $K \xleftarrow{r} \{0,1\}^k$ be a randomly generated key out of the set $\{0,1\}^k$. We define $\mathrm{F}(n, *, *)$ informally as the set of all functions that takes an n bit nonce, associated data of any size and message of any size as input and produces a ciphertext and a tag as the output where we

assume that the size of the ciphertext depends on the size of message while the size of tag is fixed. We use the non-conventional definition of the authenticated scheme to present formal definition of confidentiality and integrity of an AEAD scheme. From definition 1: the confidentiality advantage is the measure of the adversarial power to distinguish the encryption sub-routine $\mathcal{E}_K$ of AEAD against a random function \$ i.e., given AEAD=$(\mathcal{E}, \mathcal{D})$, we define \$ $\xleftarrow{r}$ F$(n, *, *)$ as a random function that depends on the encryption sub-routine of the AEAD.

Definition 1. *Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme with associated data. Let $\mathcal{A}$ be an adversary under nonce respecting scenario. Then for a randomly chosen key K, the CONF advantage of an adversary $\mathcal{A}$ against Π is given by:*

$$\text{CONF}_{\Pi}(\mathcal{A}) = [Pr(\mathcal{A}^{\mathcal{E}_K} \rightarrow 1) - Pr(\mathcal{A}^{\$} \rightarrow 1)] \tag{2.7}$$

In definition 2, we present the case of integrity advantage where the adversary is given access to the verification sub-routine in addition to the encryption sub-routine.

Before moving to the next definition, we define an event called *forges* in the following way: The event *forges* occurs if the verification sub-routine $\mathcal{V}_K$ returns true (or $\top$) for an input (K, N, A, C, T) for some randomly chosen key K where (C, T) was not the output from the encryption sub-routine $\mathcal{E}_K$ i.e., (K, N, A, M) did not returned (C, T). The idea of *forges* is to take into account only the event that is new to $\mathcal{V}_K$.

Definition 2. *Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme with associated data. Let $\mathcal{A}$ be an adversary under nonce misuse scenario that makes $q_{\mathcal{E}}$ encryption queries and $q_{\mathcal{V}}$ verification queries such that $q = q_{\mathcal{E}} + q_{\mathcal{V}}$. Then for a randomly chosen key K, the INT advantage of an adversary $\mathcal{A}$ against Π is given by:*

$$\text{INT}_{\Pi}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} forges) \tag{2.8}$$

The INT advantage measures the ability of an adversary $\mathcal{A}$ to generate a valid tag for a new input i.e., for an input, $\mathcal{A}$ has not seen before.

It is often desirable to combine the equations (2.7) and (2.8) in a single security notion as defined

below:

Definition 3. *Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme with associated data. Let $\mathcal{A}$ be an adversary that makes $q_{\mathcal{E}}$ encryption queries and $q_{\mathcal{V}}$ verification queries such that $q = q_{\mathcal{E}} + q_{\mathcal{V}}$. Then for a randomly chosen key K, the Π advantage (or combined advantage) of an adversary $\mathcal{A}$ against Π is given by:*

$$\text{AE}_{\Pi}(\mathcal{A}) = [Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} \rightarrow 1) - Pr(\mathcal{A}^{\$, \perp} \rightarrow 1)] \tag{2.9}$$

Formal definition of RUP (releasing unverified plaintext) security is provided by [48]. In definition 4, we take the notion of RUP setting from [48, 96]

Definition 4. *Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme with associated data. Let $\mathcal{A}$ be an adversary under nonce misuse scenario that makes $q_{\mathcal{E}}$ encryption queries, $q_{\mathcal{D}}$ decryption queries and $q_{\mathcal{V}}$ verification queries such that $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$. Then for a randomly chosen key K, the INT-RUP advantage of an adversary $\mathcal{A}$ against Π is given by:*

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) = Pr(\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} forges) \tag{2.10}$$

Next we define the security notion of tweakable blockcipher $(\widetilde{E}, \widetilde{D})$. Let $K \xleftarrow{r} \{0,1\}^k$ be a randomly generated key. We define $\widetilde{\text{F}}(t, b)$ informally as the set of all the functions that takes a t bit tweak and b bit message and produces b bit output. Let $\widetilde{\$} \xleftarrow{r} \widetilde{\text{F}}(t, b)$ be a random function chosen from the set of all possible tweakable blockcipher with t bit tweak and b bit block. We call $\widetilde{\$}$ as random tweakable permutation. Further, let $\widetilde{\$}^{-1}$ be the inverse of the random tweakable permutation $\widetilde{\$}$.

Definition 5. *Given a tweakable blockcipher $(\widetilde{E}, \widetilde{D})$, the indistinguishability advantage against an adversary $\mathcal{A}$ for a randomly chosen key K is given by:*

$$
\begin{aligned}
IND_{\widetilde{E}_K}(\mathcal{A}) &= [Pr(\mathcal{A}^{\widetilde{E}_K} \rightarrow 1) - Pr(\mathcal{A}^{\widetilde{\$}} \rightarrow 1)] \\
IND_{\widetilde{E}_K,\widetilde{D}_K}(\mathcal{A}) &= [Pr(\mathcal{A}^{\widetilde{E}_K,\widetilde{D}_K} \rightarrow 1) \\
&\quad - Pr(\mathcal{A}^{\widetilde{\$},\widetilde{\$}^{-1}} \rightarrow 1)]
\end{aligned} \tag{2.11}
$$

Using the results from Rogaway and Shrimpton [22], we can write the co-relation between (2.7), (2.8) and (2.9) as:

Lemma 1. *[Adapted from [96]] Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme with associated data. Let $\mathcal{A}_1$ be an adversary with query complexity as q_1 (encryption oracle), $\mathcal{A}_2$ be an adversary with query complexity as q_2 (encryption +verification oracle) and $\mathcal{A}_3$ be an adversary with query complexity as q_3 (encryption + verification oracle):*

$$
\begin{aligned}
\text{CONF}_{\Pi}(\mathcal{A}_1) &\leq \text{AE}_{\Pi}(\mathcal{B}_1) \\
\text{INT}_{\Pi}(\mathcal{A}_2) &\leq \text{AE}_{\Pi}(\mathcal{B}_2) \\
\text{AE}_{\Pi}(\mathcal{A}_3) &\leq \text{CONF}_{\Pi}(\mathcal{B}_3) + \text{INT}_{\Pi}(\mathcal{B}_4)
\end{aligned} \tag{2.12}
$$

where $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$ are adversaries with query complexities as q_1, q_2 and q_3 respectively.

We now present definitions and security notions that are used in chapter 4. We begin first by presenting the list of abbreviations in table 2.1. These abbreviations are used for providing CMT-4 security proofs. We require these abbreviations in this chapter, as well as in chapter 4. Further, we also present notions in table 2.2, these notation are used in this chapter as well as in chapter 4.

We now proceed to describe several concepts and definitions in this section, that are used in chapter 4. Let $\mathcal{K}$ denote the key space, $\mathcal{N}$ denote the nonce space, the message space is denoted by $\mathcal{M}$, and the tweak space is denoted by $\mathcal{T}$. $K \xleftarrow{\$} \mathcal{K}$ denotes a randomly selected key K from the key space $\mathcal{K}$. We follow the security notions of [60]. Let $\mathcal{A}$ be a computationally bounded adversary. We denote t as the run time taken by $\mathcal{A}$, t includes two things; memory occupied by $\mathcal{A}$ and the time to answer all the queries that $\mathcal{A}$ makes to the oracle. We know that the security

Table 2.1: Abbreviations used in the thesis.

Abbreviation	**Definition**
HBSH	Hash, Block cipher, Stream cipher, Hash.
HCTR2	Updated version of HCTR [97].
tS	Tweakable Stream cipher.
HBtSH	Hash, Block cipher, Tweakable Stream cipher, Hash.
HtS	Hash, Tweakable Stream cipher.
tS-double-decker	Tweakable Stream cipher double-decker.
tS-docked-double-decker	Tweakable Stream cipher docked-double-decker.
EtE	Encode-then-Encipher.
XOF	eXtendable-Output Function.
Deck	Doubly-extendable cryptographic keyed.
bhk	Blinded keyed hash.

of a scheme under various notions is outlined by the advantage that an adversary has on that scheme. We denote advantage as 'max' function over all the adversaries that has some restricted access to the oracle. We first start with the definition of ϵ-*almost*-Δ-*universal* family of hash functions. Then we define the primitives with the advantages that the adversary has over these primitives.

Definition 6. ϵ-***Almost***-Δ-***Universal*** $(\epsilon$-$\Delta U)$ ***Family of Hash Functions:*** *Let H be a family of hash functions with domain D and range R. Further, let R be an Abelian group where '$-$' denotes the group subtraction operation. Let ϵ be a constant such that $\frac{1}{|R|} \leq \epsilon < 1$. Then, for two distinct inputs $x \in D$ and $y \in D$, and for any $b \in R$, H is called ϵ-almost-Δ-universal (ϵ-AΔU or in short ϵ-ΔU) family of hash functions [98], if we have:*

$$Pr[h(x) - h(y) = b : h \xleftarrow{\$} H] \leq \epsilon. \tag{2.13}$$

Definition 7. ***Blinded Keyed Hash (bhk):*** *[85] Let H be a keyed hash, such that $H = \{H_K : \{0,1\}^* \rightarrow \{0,1\}^n | K \xleftarrow{\$} \mathcal{K}\}$. For two inputs (X, Δ), let $\mathcal{R} : \mathcal{O}_1(H_K(X) \oplus \Delta)$ be the real world oracle, and $\mathcal{I} : \mathcal{O}_1(X, \Delta)$ be the ideal world oracle, where $\mathcal{O}_1$ and $\mathcal{O}_2$ are two secret random oracles, then the advantage for an adversary A for the blinded keyed hash or bhk is given by:*

Table 2.2: Notations used in this chapter and chapter 4.

$\{0,1\}^*$	Set of all finite bit strings including the empty string ϵ.
$\{0,1\}^+$	Set of all finite bit strings excluding ϵ.
$\|X\|$	Bit length of X, where $X \in \{0,1\}^*$. $\|\epsilon\| = 0$.
$X\|\|Y$ (or XY)	Concatenation of X and Y in respective order.
int(X)	Denotes integer representation of X.
bin$_l(y) = X$	Denotes a unique l bit sequence such that int$(X) \equiv y \pmod{2^l}$.
0^i (or 1^i)	String of i zero (or one) bits, for example: $1 \parallel 0^i$ or 10^i refers to bit 1 followed by i zero bits.
$X[a;l]$	Subsequence of X of length l and starting at index a (one-based indexing), where $X \in \{0,1\}^*$.
$X \oplus Y$	Bitwse xor between X and Y where $\|X\| = \|Y\|$.
$\mathbb{Z}/2^{128}\mathbb{Z}$	Group generated by $\pmod{2^{128}}$.
$\boxplus, \boxminus$	Addition and subtraction inside group.
$\mathbb{N}$	The set of all natural numbers.

$$\begin{aligned} \text{Adv}_H^{\text{bhk}}(A) &= |Pr[A^{\mathcal{I}} \mapsto 1] - Pr[A^{\mathcal{R}} \mapsto 1]| \\ \text{Adv}_H^{\text{bhk}}(q,\sigma) &= \max_{A \in \mathcal{A}(q,\sigma)} \text{Adv}_H^{\text{bhk}}(A), \end{aligned} \tag{2.14}$$

where $\mathcal{A}(q,\sigma)$ is an adversary that makes at most q with data complexity $\sigma = \sum_i^q |X_i| + |\Delta_i|$.

Definition 8. ***Tweakable Stream Cipher:*** *We introduce the notion of tweakable stream cipher (tS in short). A Tweakable Stream Cipher is defined as $S : \mathcal{K} \times \mathcal{N} \times \mathcal{T} \to \{0,1\}^{l_S}$, where $\mathcal{K}$ is the key space, $\mathcal{N}$ is the nonce space, $\mathcal{T}$ is the tweak space and l_S denotes the maximum output length. S generates a key stream for encryption of the message (or decryption of the respective ciphertext). Let q_i be the i^{th} query that consists of a tuple $(N_i, T_i, l_i) \in (\mathcal{N} \times \mathcal{T} \times \mathbb{N})$, where $0 < l_i \leq l_S$, and the adversary A receives $S_K(N_i, T_i)[1; l_i]$ as the response of the i^{th} query, for a randomly chosen key K. Then we define:*

$$
\begin{aligned}
\mathrm{Adv}^{\mathrm{tS}}_{S}(A) = & \\
& Pr[A^{S_K(\cdot,\cdot)[1;\cdot]} \mapsto 1 : K \xleftarrow{\$} \mathcal{K}] \\
& - Pr[A^{\mathcal{F}(\cdot,\cdot)[1;\cdot]} \mapsto 1 : \\
& \quad \mathcal{F} \xleftarrow{\$} ((\mathcal{N} \times \mathcal{T}) \to \{0,1\}^{l_{\mathrm{tS}}})] \\
\mathrm{Adv}^{\mathrm{tS}}_{S}(q,l,l',t) = & \max_{A \in \mathcal{A}(q,l,l',t)} \mathrm{Adv}^{\mathrm{tS}}_{S}(A),
\end{aligned}
\tag{2.15}
$$

where $(\mathcal{N} \times \mathcal{T}) \to \{0,1\}^{l_S}$ denotes the set of all functions from $(\mathcal{N} \times \mathcal{T})$ to $\{0,1\}^{l_S}$. $\mathcal{A}(q,l,l',t)$ is the set of all adversaries that makes at most q queries and takes at most t time to execute, such that, l is the upper bound on total input length .i.e., $\sum_i^q(|N_i| + |T_i|)$ and l' is the upper bound on total output length .i.e., $\sum_i^q l_i$.

Definition 9. ***eXtendable-Output Function (XOF):*** *An eXtendable-Output function takes input a message M from the message space $\mathcal{M}$ and an output length, hence XOF : $\mathcal{M} \times \mathbb{N} \to \{0,1\}^{\leq l_S}$., where l_S is the maximum output length.*

We propose XOF-based tweakable stream cipher. For a given key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$ $\wedge\ |N| < \eta$, a tweak $T \in \mathcal{T}$, a maximum output length of tweakable stream cipher l_{tS} and a reversible encoding denoted by ENC, a tweakable stream cipher can be constructed using XOF as follows: $S(K,N,T) = \mathrm{XOF}(K \parallel \mathrm{ENC}(N,T), l_S)$. ENC can be written as $(\mathrm{pad}_\eta(N) \parallel T)$ for a nonce N with $|N| < \eta$, a tweak T and the padding function defined in equation (2.2).

Definition 10. ***Block Cipher:*** *Let E be an n bit block cipher that takes a key $K \xleftarrow{\$} \mathcal{K}$ and a message $M \in \mathcal{M}$ as input and produces a ciphertext $C \in \mathcal{C}$ as the output .i.e., $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and $E^{-1} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$, where $\mathcal{C}$ is the ciphertext space, $|K| = k$, $|M| = n$ and $|C| = n$. Let Perm(n) denote the set of all permutations on $\{0,1\}^n$. Then for a randomly chosen key K, we define the $\pm$PRP advantage that an adversary A has over the block cipher E in the following way:*

$$\begin{aligned}\text{Adv}_{E}^{\pm\text{PRP}}(A) &= Pr[A^{E_K, E_K^{-1}} \mapsto 1 : K \xleftarrow{\$} \mathcal{K}] \\ &\quad - Pr[A^{\pi, \pi^{-1}} \mapsto 1 : \pi \xleftarrow{\$} \text{Perm}(n)] \\ \text{Adv}_{E}^{\pm\text{PRP}}(q, t) &= \max_{A \in \mathcal{A}(q,t)} \text{Adv}_{E}^{\pm\text{PRP}}(A),\end{aligned} \tag{2.16}$$

where $\mathcal{A}(q, t)$ is an adversary that makes at most q queries and takes at most t time to execute.

Definition 11. ***Tweakable Enciphering Scheme:*** *Let $\boldsymbol{E}$ be a tweakable enciphering scheme defined as $\boldsymbol{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ for a key space $\mathcal{K}$, a tweak space $\mathcal{T}$ and a message space $\mathcal{M}$, such that, for every key $K \in \mathcal{K}$ and tweak $T \in \mathcal{T}$, $\boldsymbol{E}(K, T, \cdot)$ is a length preserving permutation .i.e., $|\boldsymbol{E}(K, T, M)| = |M|$. Further, $\boldsymbol{E}^{-1}(K, T, \boldsymbol{E}(K, T, M)) = M$. Let Perm$^{\mathcal{T}}(\mathcal{M})$ denote the set of all functions $\boldsymbol{\pi} : \mathcal{T} \times \mathcal{M} \to \mathcal{M}$, such that for $T \in \mathcal{T}$, $M \in \mathcal{M}$, $|\boldsymbol{\pi}(T, M)| = |M|$, and $\boldsymbol{\pi}(T, \cdot)$ is bijective. Then the $\pm\widetilde{PRP}$ advantage that an adversary A has is given by the following expression:*

$$\begin{aligned}\text{Adv}_{\boldsymbol{E}}^{\pm\widetilde{\text{PRP}}}(A) &= Pr[A^{\boldsymbol{E}_K, \boldsymbol{E}_K^{-1}} \mapsto 1 : K \xleftarrow{\$} \mathcal{K}] \\ &\quad - Pr[A^{\boldsymbol{\pi}, \boldsymbol{\pi}^{-1}} \mapsto 1 : \\ &\qquad \boldsymbol{\pi} \xleftarrow{\$} \text{Perm}^{\mathcal{T}}(\mathcal{M})] \\ \text{Adv}_{\boldsymbol{E}}^{\pm\widetilde{\text{PRP}}}(q, l_T, l_M, t) &= \max_{A \in \mathcal{A}(q, l_T, l_M, t)} \text{Adv}_{\boldsymbol{E}}^{\pm\widetilde{\text{PRP}}}(A),\end{aligned} \tag{2.17}$$

where $\mathcal{A}(q, l_T, l_M, t)$ is the set of all adversaries that make at most q queries, total tweak length is at most l_T, and total message length is at most l_M, and take at most t time.

Definition 12. ***Partial Collision Resistance:*** *Let $H : \mathcal{X} \to \mathcal{Y}$ be a function with $\mathcal{X}$ as domain space and $\mathcal{Y}$ as range space, where $\mathcal{X}$ is the set of tuples of inputs and $\mathcal{Y}$ is the set of outputs. For an adversary $\mathcal{A}$, we define the advantage of breaking the μ bit partial collision resistance of the function H using Algorithm 1, in the following way:*

Algorithm 1 : Game Partial Collision Resistance (PCR): $\mathbf{G}_{H}^{\mu\text{-PartialColl}}(\mathcal{A})$

$\{X_1, X_2\} \xleftarrow{\$} \mathcal{A}$ ▷ $\{X_1, X_2\} \in \mathcal{X}$
$Y_1 \leftarrow H(X_1)$ ▷ $Y_1 \in \mathcal{Y}$
$Y_2 \leftarrow H(X_2)$ ▷ $Y_2 \in \mathcal{Y}$
if $Y_1[1;\mu] = Y_2[1;\mu]$ **then** ▷ Condition for Partial Collision
    return $True$
else
    return $False$
end if

Algorithm 2 : Game CMT-4: $\mathbf{G}_{\text{AEAD}}^{\text{CMT-4}}(\mathcal{A})$, where AEAD=$(\mathcal{E}, \mathcal{D})$

$\{(K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2)\} \xleftarrow{\$} \mathcal{A}$
$C_1 \leftarrow \mathcal{E}(K_1, N_1, A_1, M_1)$
$C_2 \leftarrow \mathcal{E}(K_2, N_2, A_2, M_2)$
if $(K_1, N_1, A_1, M_1) = (K_2, N_2, A_2, M_2)$ or $C_1 \neq C_2$ **then**
    return $False$
else
    return $True$
end if

$$\text{Adv}_{H}^{\mu\text{-PartialColl}}(\mathcal{A}) = Pr[\mathbf{G}_{H}^{\mu\text{-PartialColl}}(\mathcal{A}) \mapsto 1]. \tag{2.18}$$

We proceed now to define the CMT-4 security of an AEAD in definition 13 followed by the definition of encode-then-encipher paradigm in definition 14. As discussed in chapter 1 (section 1.3.2), there are multiple ways to create and AEAD under EtE paradigm. In this thesis, we focus on prepending technique for creating the AEAD.

Definition 13. ***CMT-4 Security of AEAD:*** *Let AEAD be an authenticated encryption scheme with associated data defined as AEAD=$(\mathcal{E}, \mathcal{D})$. Then CMT-4 security of AEAD is given by the game in algorithm 2. For an adversary $\mathcal{A}$, the CMT-4 advantage on AEAD is given by the following equation:*

$$\text{Adv}_{\text{AEAD}}^{\text{CMT-4}}(\mathcal{A}) = Pr[\mathbf{G}_{\text{AEAD}}^{\text{CMT-4}}(\mathcal{A}) \mapsto 1]. \tag{2.19}$$

Definition 14. ***Encode-then-Encipher (EtE)*** *[59]:* *An AEAD takes four inputs .i.e., a key K, a nonce N, an associated data A and a message M. AEAD can be constructed under EtE*

paradigm using a two step approach. The first step is to use message encoding, followed by message enciphering. We use a tweakable enciphering scheme (definition 11) for message enciphering. Let μ be the number of zero bits that are used for the prepending technique, then we define message encoding and message enciphering as follows:

Message Encoding: We borrow the notation of encoding the messages from [59]. Let $\mathcal{M}$ be the message space and let $\mathcal{M}^*$ be the encoding space, then we define two sub-routines, the Encode sub-routine and Decode sub-routine as follows:

- We present the Encode sub-routine in equation (2.20) below:

$$M' = \text{Encode}(M, \mu) = \begin{cases} (0^\mu \parallel M) & \text{if } \mu > 0 \\ \perp & \text{otherwise.} \end{cases} \tag{2.20}$$

- The Decode sub-routine in equation (2.21) below:

$$\text{Decode}(M', \mu) = \begin{cases} M & \text{if } \mu > 0 \wedge (M' = (0^\mu \parallel M)) \\ \perp & \text{otherwise.} \end{cases} \tag{2.21}$$

Message Enciphering: After the message is encoded using the Encode sub-routine of equation (2.20), a tweakable enciphering scheme $\boldsymbol{E}$ (definition 11) is invoked to encrypt M' and generate a ciphertext C, .i.e,

$$\boldsymbol{E}(K, T, M') \mapsto C,$$

where K is the key of the AEAD and nonce and associated of AEAD is used to create the tweak .i.e., $T \leftarrow (N \parallel A)$.
In case of decryption, $\boldsymbol{E}^{-1}$ is invoked to recover back M' from C .i.e.,

$$\boldsymbol{E}^{-1}(K, T, C) \mapsto M',$$

where K is the key of the AEAD and the tweak is given by: $T \leftarrow (N \parallel A)$, where N is the nonce and A is the associated data of the AEAD. After the decryption, the Decode sub-routine

of equation (2.21) is invoked and is then used to get back the original message M. If the first μ-bits of M' are non-zero, then we reject the message M and output $\perp$ (error). On the contrary, if first μ-bits are verified to be zero, then we accept the message.

Chapter 3

Lynx: Family of Lightweight Authenticated Encryption Schemes based on Tweakable Blockcipher

3.1 Introduction

As explained in chapter 1, our goal is use tweakable primitives to create cryptographic designs which are provably secure. Further, we also outlined the factors driving the current cryptographic research towards the lightweight alternatives. Moving in this direction, we utilized one of the tweakable primitives .i.e., tweakable blockcipher to create lightweight authentication encryption schemes. An AEAD (refer section 1.1 from chapter 1) provides both confidentiality and integrity based on various factors. Further, in chapter 1, we also saw that recent advancements and research is shifting towards the lightweight alternative that could be implemented efficiently on low-powered devices like IoT sensors and handheld devices. Such low-powered devices require several features from the design and implementation perspective of any authenticated encryption scheme, like acceptable security bounds in nonce respecting or nonce misuse scenarios, low memory usage, computationally menial, less power-hungry, etc. In real-world scenarios, it may not be possible to provide all such features simultaneously, and hence there is an associated

trade-off in the design and the implementation of the authenticated encryption schemes for these low-powered devices. Further, there is no established global standard yet that lists out requirements for lightweight cryptography, instead, there are several competitions that evaluate candidates for lightweightness.

One of the finalists of the NIST lightweight competition, Romulus [9], utilizes tweakable blockcipher as the underlying primitive, for their AEAD algorithm. Tweakable blockcipher was introduced by Liskov *et al.* at CRYPTO 2002 [6]. Unlike a blockcipher, which takes a key and a message as the input and produces a ciphertext as the output; a tweakable blockcipher takes in three inputs: a key, a message, and a tweak and produces a ciphertext as the output. Refer chapter 1 for details on tweakable blockcipher. Since the inception of these tweakable primitive, they have been widely studied under various security scenarios for creating authenticated encryption schemes, for example, Romulus [9], SKINNY-AEAD [99], ForkAE [100] etc.

3.1.1 Motivation

An AEAD construction that can ensure both integrity security and confidentiality security by making only a single sweep through the data is called a 1-pass or single-pass [20] AEAD. Since, such an AEAD constructions have lower computational costs as compared to a multi-pass AEAD construction; and hence they are often desirable in the world of lightweight cryptography. The number of blocks of data (message) processed per non-linear operation is called rate [20] of AEAD. A higher rate AEAD would ensure a lower latency and high speed. Romulus-N [9] which is 1-pass and rate-1 AEAD design, provides beyond birthday bound (or BBB) security in nonce respecting scenario, but is nonce misuse resilient [101], and not nonce misuse resistant, under nonce misuse scenario. Our motivation is to target the area of 1-pass and rate-1 nonce misuse resistant AEAD constructions which ensures integrity security, and integrity security in RUP [48] scenarios as well.

In the real world setting, it is possible, that some users may want at least birthday bound security or BBB security for integrity in both, the nonce respecting, as well as the nonce misuse and RUP scenarios together with efficient implementations. In fact, none of the NIST lightweight cryptography competition finalists which are 1-pass and rate-1, provide birthday bound security

or BBB security for integrity for both nonce respecting as well nonce misuse and RUP scenarios. In NIST lightweight cryptography workshop 2020 [102], an AEAD scheme called AET-LR [43], based on Romulus family was introduced. In this scheme, the authors used associated data or message as both, a part of the input (together with the field multiplication) as well as a part of tweak for the underlying primitive i.e., tweakable blockcipher. AET-LR provides birthday bound security for both nonce respecting scenarios as well as nonce misuse and RUP scenarios. The authors in AET-LR were able to present only one such design where the associated data or the message can be used both, as the part of input as well as the tweak.

Lynx[1] is 1-pass and rate-1 family of lightweight authenticated encryption schemes based on a tweakable blockcipher, were the tweak size as twice the size of the block. The target area of our proposal is to provide a balance between the security bounds of nonce respecting and nonce misuse and RUP scenarios for integrity as well as privacy but at the same time, the architecture or design is light enough so that it can be implemented efficiently on low powered devices. Further, unlike AET-LR, lynx supports stream processing or online processing i.e., a block of associated data or message can be encrypted or decrypted (decryption in case of message only) on the fly without having any information about the next block (like full or partial, last block or not, etc.). Such capabilities are very desirable in lightweight category, since, the low powered devices have limited memory. Also unlike AET-LR which is just one construction, lynx is a family of authenticated encryption schemes with 14 members. Internally, AET-LR uses field multiplication while lynx is based on simple *xor* operation which can be efficiently implemented even on an 8 bit micro-controllers.

The lynx family can be divided into two sub-families: lynx-A and lynx-B (refer section 3.2 for details). Both the families, lynx-A and lynx-B have confidentiality and integrity assurance in nonce respecting scenario and integrity assurance in nonce-misuse and RUP scenarios, i.e., we provide birthday bound security for both nonce respecting as well as nonce misuse and RUP scenarios and birthday bound security for privacy. Further, the members of lynx-A family are length preserving while the members in the lynx-B family are not length preserving. There are 10 members in lynx-A family and are referred to as: lynx-A1, lynx-A2, . . . , lynx-A9 and lynx-A10

[1]Lynx [pronounced: lɪ ks] is a species of lightweight and small but agile wild cats found in Eurasian and American belt and is known for its speed and adaptability.

Table 3.1: Summary of Lynx-A family. There are 10 members in lynx-A family with each member constructed using a function from $\mathcal{F}^A$ family. For more details see section 3.2.

Member of Lynx-A Family	**Case of $\mathcal{F}^A$ Family**	**Member of Lynx-A Family**	**Case of $\mathcal{F}^A$ Family**
Lynx-A1	$\mathcal{F}^A_{25}$ / $\overline{\mathcal{F}^A_{25}}$	Lynx-A6	$\mathcal{F}^A_{30}$ / $\overline{\mathcal{F}^A_{30}}$
Lynx-A2	$\mathcal{F}^A_{26}$ / $\overline{\mathcal{F}^A_{26}}$	Lynx-A7	$\mathcal{F}^A_{31}$ / $\overline{\mathcal{F}^A_{31}}$
Lynx-A3	$\mathcal{F}^A_{27}$ / $\overline{\mathcal{F}^A_{27}}$	Lynx-A8	$\mathcal{F}^A_{32}$ / $\overline{\mathcal{F}^A_{32}}$
Lynx-A4	$\mathcal{F}^A_{28}$ / $\overline{\mathcal{F}^A_{28}}$	Lynx-A9	$\mathcal{F}^A_{34}$ / $\overline{\mathcal{F}^A_{34}}$
Lynx-A5	$\mathcal{F}^A_{29}$ / $\overline{\mathcal{F}^A_{29}}$	Lynx-A10	$\mathcal{F}^A_{35}$ / $\overline{\mathcal{F}^A_{35}}$

Table 3.2: Summary of Lynx-B family. There are 4 members in lynx-B family with each member of lynx-B family constructed using a function from $\mathcal{F}^B$ family. For more details see section 3.2.

Member of Lynx-B Family	**Case of $\mathcal{F}^B$ Family**	**Member of Lynx-B Family**	**Case of $\mathcal{F}^B$ Family**
Lynx-B1	$\mathcal{F}^B_{18}$ / $\overline{\mathcal{F}^B_{18}}$	Lynx-B3	$\mathcal{F}^B_{30}$ / $\overline{\mathcal{F}^B_{30}}$
Lynx-B2	$\mathcal{F}^B_{20}$ / $\overline{\mathcal{F}^B_{20}}$	Lynx-B4	$\mathcal{F}^B_{32}$ / $\overline{\mathcal{F}^B_{32}}$

while there are 4 members in lynx-B family and are referred to as: lynx-B1, lynx-B2, lynx-B3 and lynx-B4.

To create such a family of authenticated encryption schemes, we present a family of functions, $\mathcal{F}$, which is used to create the lynx family. For the lynx-A family, the function family is referred to as $\mathcal{F}^A$ and for the lynx-B family, the function family is referred to as $\mathcal{F}^B$ respectively. There are 36 cases of $\mathcal{F}^A$ (refer table 3.5) and 36 cases of $\mathcal{F}^B$ (refer table 3.6) enumerated as $\mathcal{F}^A_{1...36}$ and $\mathcal{F}^B_{1...36}$ respectively, thereby making a total of 72 cases. Internally, both the function families utilizes simple *xor* operations (refer section 3.2.3 and section 3.2.4). We perform analysis of each of these 72 cases in the section 3.3 and sort them into the correct and the incorrect groups. Out of the 36 cases of $\mathcal{F}^A$, 10 of them fall into the correct group while out of the 36 cases of $\mathcal{F}^B$, 4 of them fall into the correct one, making a total of 14 out of 72 cases. The lynx family is created using only the cases of $\mathcal{F}^A$ and $\mathcal{F}^B$ that fall into the correct group. Hence, there are a

total of 14 members in the lynx family. When the decryption sub-routine of lynx-A and lynx-B families are invoked, we use the notation $\overline{\mathcal{F}^A}$ and $\overline{\mathcal{F}^B}$ for the respective function families.
In table 3.1, we present a mapping of all the members of the lynx-A family with the cases of the function family $\mathcal{F}^A$. Similarly, in table 3.2, we present a mapping of all the members of the lynx-B family with the cases of the function family $\mathcal{F}^B$.

3.1.2 Contributions

- We propose family of 1-pass and rate-1 AEAD schemes called lynx (table 3.1 and table 3.2) based on a tweakable blockcipher where tweak size is twice the block size. We divide the lynx family into two sub-families; lynx-A with 10 members and lynx-B with 4 members, making a total of 14 members in the lynx family. For each family, we propose an AEAD construction and an AEAD algorithm: lynx-A (refer figure 3.1 and algorithm 3) and lynx-B (refer figure 3.2 and algorithm 4).
- We propose a family of functions, called $\mathcal{F}$ (sub-family $\mathcal{F}^A$ and $\mathcal{F}^B$) and present 72 cases of $\mathcal{F}$ (refer table 3.5 and table 3.6 i.e., 36 cases of $\mathcal{F}^A$ and 36 cases of $\mathcal{F}^B$).
- We present correctness criteria of $\mathcal{F}$ and then we analyze each case of $\mathcal{F}$ (in light of constructing an authenticated encryption scheme) and group them into correct and incorrect cases. For the incorrect ones, we further group them into implausible, non-confidential and non-integrity cases (refer table 3.7), based on the issues in the internal construction of $\mathcal{F}$.
- We present formal provable security proofs of lynx-A and lynx-B in the information-theoretic model (refer section 3.6) for integrity security in both nonce respecting and nonce misuse and RUP scenarios, and the confidentiality security in nonce respecting scenario.
- Finally, we present implementation of one of the member of the lynx family on six different platforms and compare them with the available implementations of Romulus-N1 (refer section 3.7). Due to the simplistic design structure, lynx member outperforms Romulus-N1 on all the six platforms.

We want to point out that lynx provides birthday bound security for both nonce respecting as well

as nonce misuse and RUP scenarios, but if someone requires security for only nonce respecting setup, then, Romulus-N1 may be a better choice due to its beyond birthday bound security in nonce respecting scenario.

3.1.3 Organization of the chapter:

The rest of the chapter is organized as follows. Section 3.2 presents the authenticated encryption scheme proposed in the thesis. This section also gives the internal construction of the family of functions $\mathcal{F}^A$, $\mathcal{F}^B$, $\overline{\mathcal{F}^A}$ and $\overline{\mathcal{F}^B}$, with its 72 cases. In section 3.3, we present the analysis of function family $\mathcal{F}^A$ and $\mathcal{F}^B$, and show the cases of $\mathcal{F}^A$ and $\mathcal{F}^B$ families that can be used to construct lynx. We present design rationale of lynx in section 3.5 followed by provable security proofs in the section 3.6. Section 3.7 shows the implementation of one of the member of lynx family and its comparison to Romulus-N1 followed by conclusion in section 3.8.

3.2 Lynx

We propose family of 1-pass rate-1 authenticated encryption schemes with associated data called lynx-A and lynx-B. We use $\mathcal{E}$ to denote the encryption sub-routine for the members of lynx-A and lynx-B families, and $\mathcal{D}$ to denote the decryption sub-routine for the members of lynx-A and lynx-B families respectively. Both the families have confidentiality and integrity assurance in nonce respecting scenario and integrity assurance in nonce-misuse and RUP scenario i.e. birthday bound security for both nonce respecting as well as nonce misuse and RUP setting for integrity security, and birthday bound security for privacy in nonce respecting scenario.

There are 10 members in lynx-A family and are referred to as: lynx-A1, lynx-A2, . . . , lynx-A10. Each member of the lynx-A family is based on a case from $\mathcal{F}^A$ family that fall into the correct group. Figure 3.1 shows the construction of lynx-A family and algorithm 3 presents the encryption and decryption sub-routine of the lynx-A family.

In lynx-B family, we have 4 members and they are referred to as: lynx-B1, lynx-B2, lynx-B3 and lynx-B4. Each member of the lynx-B family is based on a case from $\mathcal{F}^B$ family that fall into

the correct group. In figure 3.2, we show the construction of Lynx-B family and the algorithm 4 presents the encryption and decryption sub-routine of the Lynx-B family.

In table 3.3, we provide the specification of both the families. All the members of Lynx-A and Lynx-B family takes in a 128 bit nonce, a 128 bit block (associated data or message), and a 256 bit tweak. The tag size produced is 128 bits. The specification of both Lynx-A and Lynx-B is in line with [103].

Next, we describe the Lynx-A and Lynx-B in detail followed by the internal constructions of $\mathcal{F}^A$ and $\mathcal{F}^B$.

Figure 3.1: Encryption and Decryption sub-routine of Lynx-A family. $\mathcal{F}^A$ in below figure is used from the cases that fall into correct group (refer table 3.1 for list)

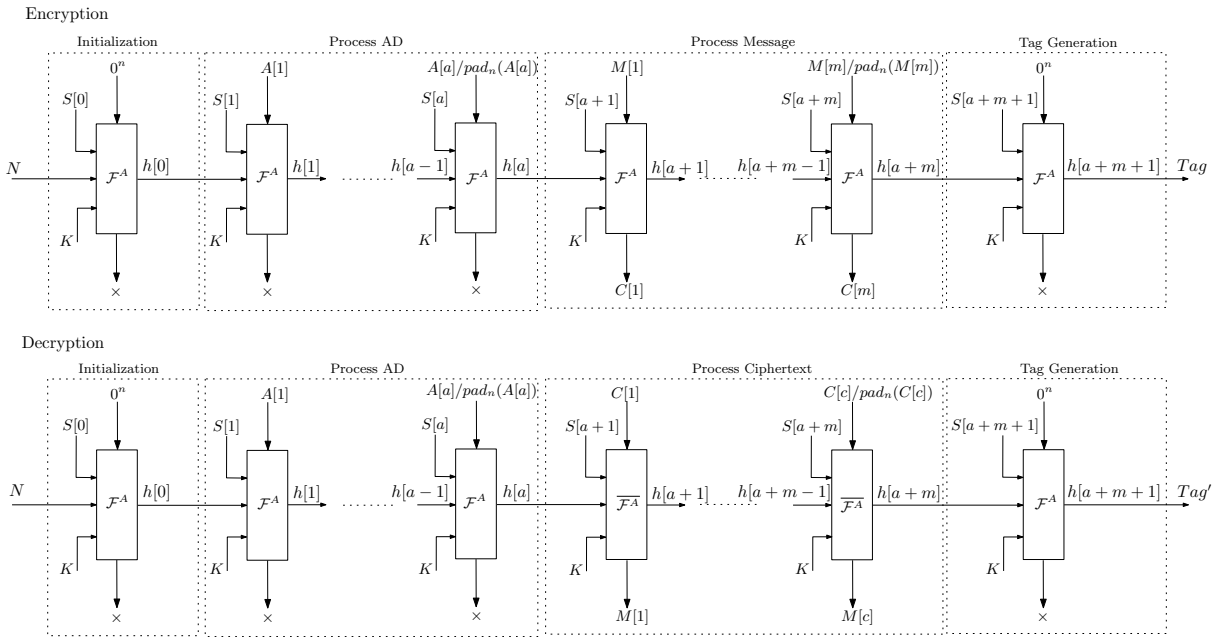
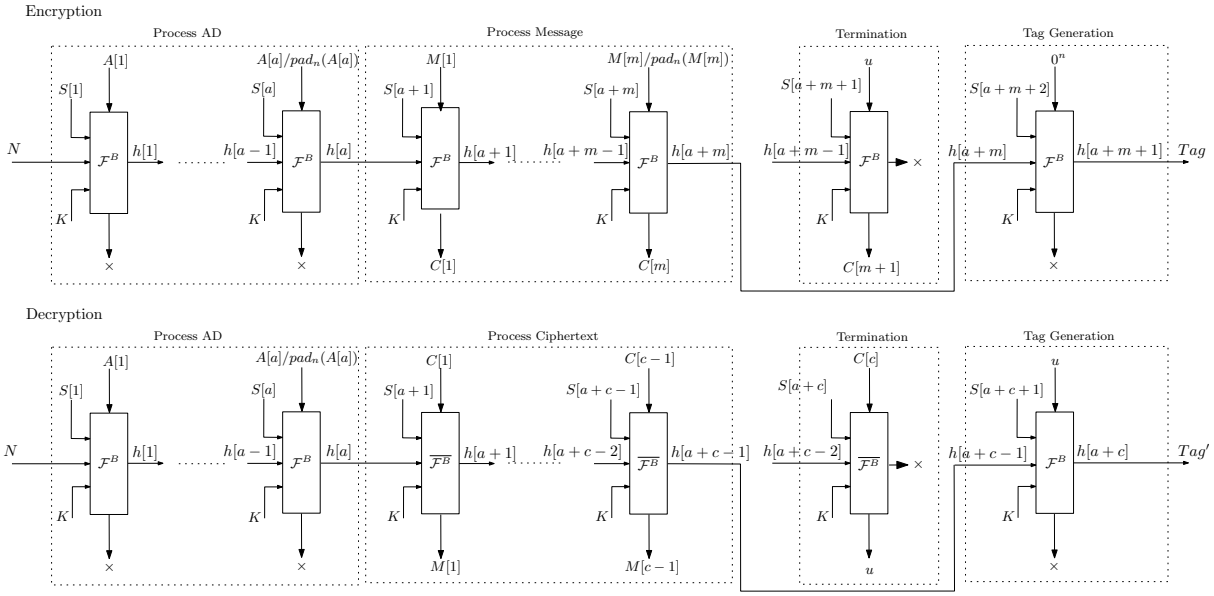


Table 3.3: Lynx specification. Lynx takes input as nonce with size as 128 bits, block size as 128 bits, tweak size as 256 bits and return a tag of size 128 bits

Lynx Family	**Nonce Size** (n)	**Block Size** (b)	**Tweak Size** $(t = 2 * b)$	**Tag Size** (tag)
Lynx-A Family	128	128	256	128
Lynx-B Family	128	128	256	128

Figure 3.2: Encryption and Decryption sub-routine of Lynx-B family. $\mathcal{F}^B$ in below figure is used from the cases that fall into correct group (refer table 3.2 for list)



3.2.1 Lynx-A

The construction of lynx-A family can be divided into three phases: an initialization phase, an associated data and a message processing phase, and a tag generation phase (refer figure 3.1). The initialization phase and the tag generation phase require an input with a zero state i.e., an input 0^b is fed during the initialization phase as well as during the tag generation phase. In addition to a zero state, the initialization phase also takes the a nonce as an input. The associated data and message are processed in blocks of size b, after the initialization phase and before the tag generation phase. Upon encountering a partial block, a padding function is used which is described in equation (2.1). The initialization phase in lynx-A family is necessary to ensure confidentiality in nonce respecting scenario. The tag generation phase is necessary to maintain integrity. Since the output generated by the initialization phase and the tag generation phase is not part of the ciphertext, thereby making the length of the ciphertext produced by lynx-A equal to the length of the message and hence, lynx-A is length preserving.

3.2.2 Lynx-B

The construction of lynx-B family can be divided into three phases: an associated data and message processing phase, a termination phase, and a tag generation phase (refer figure 3.2). The members of lynx-B family start with the associated data and the message processing. The first block processing also takes in nonce as the input. The associated data and message are processed in blocks of size b (or with padding) bits. Once the message processing is complete, the termination phase starts. The termination phase keeps the track of message length. Please note that if message is empty, there is no termination phase. During the termination phase an input u fed whose value depends on whether the last block of message was full or partial, $u = 10^{b-1}$ for a full last block while $u = 1010^{b-3}$ for partial last block. The termination phase is the part of the ciphertext and hence members of lynx-B family are not length preserving. In the tag generation phase, similar to lynx-A family, lynx-B family also uses a zero state i.e., an input 0^b during the tag generation phase. The tag generation phase is necessary to maintain integrity. If the associated data and the message, both are empty i.e., $A = \epsilon$ (or $|A| = 0$) and $M = \epsilon$ (or $|M| = 0$), then lynx-A executes the initialization phase as well as the tag generation phase (refer figure 3.1 and algorithm 3) while lynx-B executes only the tag generation phase (refer figure 3.2 and algorithm 4).

3.2.3 $\mathcal{F}^A$

The function family $\mathcal{F}^A$ takes in three inputs and returns two outputs (refer figure 3.3) as described by the following equation:

$$\mathcal{F}^A(K, h[i-1], S[i], V[i]) \mapsto (h[i], W[i]) \tag{3.1}$$

where $K \in \{0,1\}^k$, $h[i-1] \in \{0,1\}^b$, $S[i] \in \{0,1\}^s$, $V[i] \in \{0,1\}^b$, $h[i] \in \{0,1\}^b$ and $W[i] \in \{0,1\}^b$. Like $\mathcal{F}^A$, $\overline{\mathcal{F}^A}$ also takes in three inputs and returns two outputs (refer figure 3.3):

$$\overline{\mathcal{F}^A}(K, h[i-1], S[i], W[i]) \mapsto (h[i], V[i]) \tag{3.2}$$

Algorithm 3 : Lynx-A encryption and decryption

$\mathcal{E}(K, N, A, M)$

Note: $\mathcal{F}^A$ is used from the cases that fall into correct group (refer table 3.1)

$(A[1] \parallel A[2] \parallel ... \parallel A[a]) = A$, $(M[1] \parallel M[2] \parallel ... \parallel M[m]) = M$, $l_{7...0} \leftarrow 0$, $K \xleftarrow{\$} \{0,1\}^k$, $S[0] \leftarrow bin_{120}(0) \parallel l_{7...0}$
$(h[0], \times) \leftarrow \mathcal{F}^A(K, N, S[0], 0^b)$ ▷ *Initialization*
for $i = 1$ to a **do** ▷ *Process Associated Data*
    if $|A[i]| \pmod b = 0$ **then**
        $l_0 \leftarrow 1$
    else
        $l_1 \leftarrow 1$, $A[i] \leftarrow pad_b(A[i])$
    end if
    $S[i] \leftarrow bin_{120}(i) \parallel l_{7...0}$
    $(h[i], \times) \leftarrow \mathcal{F}^A(K, h[i-1], S[i], A[i])$
end for
for $i = 1$ to m **do** ▷ *Process Message*
    if $|M[i]| \pmod b = 0$ **then**
        $l_0, l_1 \leftarrow 1$, $\nu \leftarrow ||M[i]||$
    else
        $l_0, l_1 \leftarrow 0$, $l_2 \leftarrow 1$, $\nu \leftarrow ||M[i]||$, $M[i] \leftarrow pad_b(M[i])$
    end if
    $S[a+i] \leftarrow bin_{120}(a+i) \parallel l_{7...0}$
    $(h[a+i], C[i]) \leftarrow \mathcal{F}^A(K, h[a+i-1], S[a+i], M[i])$
end for
$l_{3...0} \leftarrow 1$ ▷ *Tag Generation*
$S[a+m+1] \leftarrow bin_{120}(a+m+1) \parallel l_{7...0}$
$(h[a+m+1], \times) \leftarrow \mathcal{F}^A(K, h[a+m], S[a+m+1], 0^n)$
$Tag \leftarrow h[a+m+1]$, $C \leftarrow C[0] \parallel C[1] \parallel ... \parallel \text{Trunc}_\nu(C[m])$
Return (C, Tag)

$\mathcal{D}(K, N, A, C, Tag)$

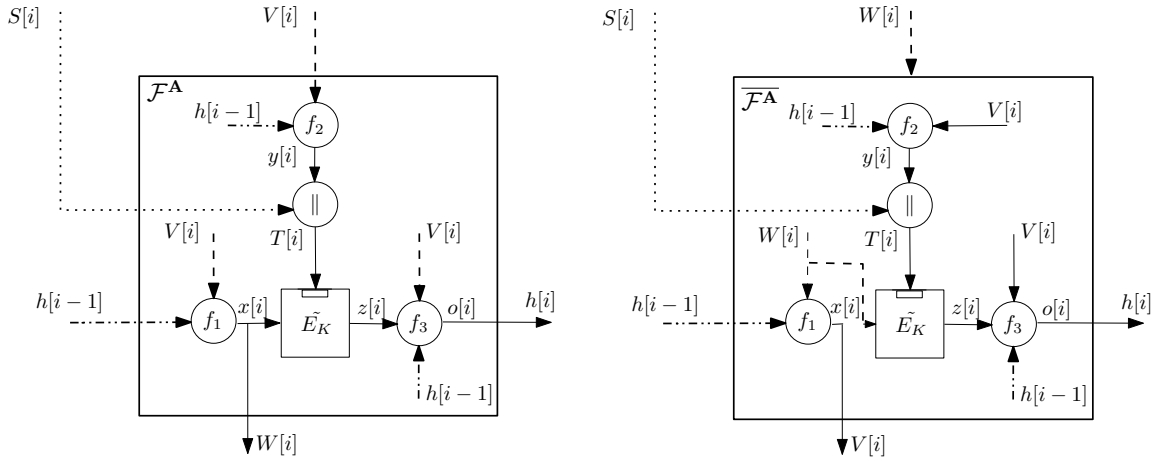
$(A[1] \parallel A[2] \parallel ... \parallel A[a]) = A$, $(C[1] \parallel C[2] \parallel ... \parallel C[c]) = C$, $l_{7...0} \leftarrow 0$, $S[0] \leftarrow bin_{120}(0) \parallel l_{7...0}$
$(h[0], \times) \leftarrow \mathcal{F}^A(K, N, S[0], 0^b)$ ▷ *Initialization*
for $i = 1$ to a **do** ▷ *Process Associated Data*
    if $|A[i]| \pmod b = 0$ **then**
        $l_0 \leftarrow 1$
    else
        $l_1 \leftarrow 1$, $A[i] \leftarrow pad_b(A[i])$
    end if
    $S[i] \leftarrow bin_{120}(i) \parallel l_{0...7}$
    $(h[i], \times) \leftarrow \mathcal{F}^A(K, h[i-1], S[i], A[i])$
end for
for $i = 1$ to c **do** ▷ *Process Ciphertext*
    if $|C[i]| \pmod b = 0$ **then**
        $l_0, l_1 \leftarrow 1$, $\nu \leftarrow |C[i]|$

else
    $l_0, l_1 \leftarrow 0, l_2 \leftarrow 1, \nu \leftarrow ||C[i]||, C[i] \leftarrow pad_b(C[i])$
end if
$S[a+i] \leftarrow bin_{120}(a+i) \parallel l_{7...0}$
$(h[a+i], M[i]) \leftarrow \overline{\mathcal{F}^A}(K, h[a+i-1], S[a+i], C[i])$
end for
$l_{3...0} \leftarrow 1$ ▷ *Tag Generation*
$S[a+m+1] \leftarrow bin_{120}(a+m+1) \parallel l_{7...0}$
$(h[a+m+1], \times) \leftarrow \mathcal{F}^A(K, h[a+m], S[a+m+1], 0^n)$
$Tag' \leftarrow h[a+m+1], M \leftarrow M[0] \parallel M[1] \parallel ... \parallel \text{Trunc}_\nu(M[c])$
if $Tag = Tag'$ **then**
    Return M
else
    Return $\perp$
end if

We present the internal construction of $\mathcal{F}^A$ and $\overline{\mathcal{F}^A}$ in figure 3.3 and present their pseudocode in algorithm 5.

All the 36 cases of $\mathcal{F}^A$ (and $\overline{\mathcal{F}^A}$) are shown in table 3.5, out of which only 10 cases that are marked in bold red font can be used to create an authenticated encryption scheme (check section 3.3 for details). Hence, the members of lynx-A family are created only using these 10 cases of $\mathcal{F}^A$ (and $\overline{\mathcal{F}^A}$). As stated earlier, $\mathcal{F}^A$ is used during the invocation of encryption sub-routine ($\mathcal{E}$) while $\overline{\mathcal{F}^A}$ is used during the invocation of decryption sub-routine ($\mathcal{D}$) of *lynx*-A respectively. Further, in equation (3.1), $V[i]$ is either associated data $A[i]$ or message $M[i]$, and $W[i]$ is the ciphertext $C[i]$ corresponding to the message $M[i]$ and in equation (3.2), $V[i]$ is the original message corresponding to the ciphertext $W[i]$.

Figure 3.3: Internal construction of the function family $\mathcal{F}^A$ / $\overline{\mathcal{F}^A}$. Internally it is composition of three sub-functions: f_1 and f_2 takes in two inputs and returns one output while f_3 takes in three input and returns one output



Algorithm 4 : Lynx-B encryption and decryption

$\mathcal{E}(K, N, A, M)$

Note: $\mathcal{F}^B$ is used from the cases that fall into correct group (refer table 3.2)

$(A[1] \parallel A[2] \parallel ... \parallel A[a]) = A$, $(M[1] \parallel M[2] \parallel ... \parallel M[m]) = M$, $l_{7...0} \leftarrow 0$, $l_3 \leftarrow 1$, $h[0] \leftarrow N$
for $i = 1$ to a **do** ▷ *Process Associated Data*
  if $|A[i]| \pmod n = 0$ **then**
    $l_0 \leftarrow 1$, $S[i] \leftarrow bin_{120}(i) \parallel l_{7...0}$
  else
    $l_1 \leftarrow 1$, $A[i] \leftarrow pad_n(A[i])$, $S[i] \leftarrow bin_{120}(i) \parallel l_{7...0}$
  end if
  $(h[i], \times) \leftarrow \mathcal{F}^B(K, h[i-1], S[i], A[i])$
end for
for $i = 1$ to $m - 1$ **do** ▷ *Process Message*
  $l_0, l_1 \leftarrow 1$, $S[a+i] \leftarrow bin_{120}(a+i) \parallel l_{7...0}$
  $(h[a+i], C[i]) \leftarrow \mathcal{F}^B(K, h[a+i-1], S[a+i], M[i])$
end for
if $M \neq \epsilon$ **then**
  if $|M[m]| \pmod b = 0$ **then** ▷ *Termination*
    $l_0, l_1 \leftarrow 0$, $u \leftarrow 10^{b-1}$
    $S[a+m+1] \leftarrow bin_{120}(a+m+1) \parallel l_{7...0}$
    $(\times, C[m+1]) \leftarrow \mathcal{F}^B(K, h[a+m-1], S[a+m+1], u)$
    $l_0, l_1 \leftarrow 1$ ▷ *Process Last Message Block*
    $S[a+m] \leftarrow bin_{120}(a+m) \parallel l_{7...0}$
    $(h[a+m], C[m]) \leftarrow \mathcal{F}^B(K, h[a+m-1], S[a+m], M[m])$
  else
    $l_2 \leftarrow 1$, $u \leftarrow 1010^{b-3}$, $M[m] \leftarrow pad_b(M[m])$
    $S[a+m+1] \leftarrow bin_{120}(a+m+1) \parallel l_{7...0}$
    $(\times, C[m+1]) \leftarrow \mathcal{F}^B(K, h[a+m-1], S[a+m+1], u)$
    $l_0, l_1 \leftarrow 0$, $l_2 \leftarrow 1$ ▷ *Process Last Message Block*
    $S[a+m] \leftarrow bin_{120}(a+m) \parallel l_{7...0}$
    $(h[a+m], C[m]) \leftarrow \mathcal{F}^B(K, h[a+m-1], S[a+m], M[m])$
  end if
end if
$l_{3...0} \leftarrow 1$ ▷ *Tag Generation*
$S[a+m+2] \leftarrow bin_{120}(a+m+2) \parallel l_{7...0}$
$(h[a+m+1], \times) \leftarrow \mathcal{F}^B(K, h[a+m], S[a+m+2], 0^n)$
$Tag \leftarrow h[a+m+1]$, $C \leftarrow C[0] \parallel C[1] \parallel ... \parallel C[m+1]$
Return (C, Tag)

$\mathcal{D}(K, N, A, C, Tag)$

$(A[1] \parallel A[2] \parallel ... \parallel A[a]) = A$, $(C[1] \parallel C[2] \parallel ... \parallel C[c]) = C$, $l_{7...0} \leftarrow 0$, $l_3 \leftarrow 1$, $h[0] \leftarrow N$

for $i = 1$ to a **do** ▷ *Process Associated Data*
  if $|A[i]| \pmod b = 0$ **then**
    $l_0 \leftarrow 1, S[i] \leftarrow bin_{120}(i) \parallel l_{7...0}$
  else
    $l_1 \leftarrow 1, A[i] \leftarrow pad_n(A[i]), S[i] \leftarrow bin_{120}(i) \parallel l_{7...0}$
  end if
  $(h[i], \times) \leftarrow \mathcal{F}^B(K, h[i-1], S[i], A[i])$
end for
for $i = 1$ to $c - 2$ **do** ▷ *Process Ciphertext*
  $l_0, l_1 \leftarrow 1, S[a+i] \leftarrow bin_{120}(a+i) \parallel l_{7...0}$
  $(h[a+i], M[i]) \leftarrow \overline{\mathcal{F}}_B(K, h[a+i-1], S[a+i], C[i])$
end for
if $C \neq \epsilon$ **then**
  $l_0, l_1 \leftarrow 0, S[a+c] \leftarrow bin_{120}(a+c) \parallel l_{7...0}$ ▷ *Termination*
  $(\times, M[c]) \leftarrow \overline{\mathcal{F}}_B(K, h[a+c-2], S[a+c], C[c])$
  if $M[c] = 10^{b-1}$ **then** ▷ *Process Last Ciphertext Block*
    $l_0, l_1 \leftarrow 1, S[a+c-1] \leftarrow bin_{120}(a+c-1) \parallel l_{7...0}$
    $(h[a+c-1], M[c-1]) \leftarrow \overline{\mathcal{F}}_B(K, h[a+c-2], S[a+c-1], C[c-1])$
  else
    $l_2 \leftarrow 1, S[a+c-1] \leftarrow bin_{120}(a+c-1) \parallel l_{7...0}$
    $\nu \leftarrow 10^*$ ▷ *Padded according to Eq. 2.1*
    $(h[a+c-1], M[c-1]) \leftarrow \overline{\mathcal{F}}_B(K, h[a+c-2], S[a+c-1], C[c-1])$
    $M[c-1] \leftarrow \text{Extract}_\nu(M[c-1]))$
  end if
end if
$l_{3...0} \leftarrow 1, S[a+c+1] \leftarrow bin_{120}(a+c+1) \parallel l_{7...0}$ ▷ *Tag Generation*
$(h[a+c], \times) \leftarrow \mathcal{F}^B(K, h[a+c-1], S[a+c+1], 0^n)$
$Tag' \leftarrow h[a+c], M \leftarrow M[0] \parallel M[1] \parallel ... \parallel M[c-1]$
if $Tag = Tag'$ **then**
  Return M
else
  Return $\perp$
end if

Algorithm 5 : $\mathcal{F}^A$ and $\overline{\mathcal{F}^A}$

$\mathcal{F}^A(K, h[i-1], S[i], V[i])$

1: $x[i] \leftarrow f_1(h[i-1], V[i]) \mid x[i] \in \{h[i-1], V[i], h[i-1] \oplus V[i]\}$
2: $W[i] \leftarrow x[i]$
3: $y[i] \leftarrow f_2(h[i-1], V[i]) \mid y[i] \in \{h[i-1], V[i], h[i-1] \oplus V[i]\}$
4: $\mathcal{T}[i] \leftarrow S[i] \parallel y[i]$
5: $z[i] \leftarrow \widetilde{E}(K, \mathcal{T}[i], x[i])$
6: $o[i] \leftarrow f_3(z[i], h[i-1], V[i]) \mid o[i] \in \{z[i], z[i] \oplus h[i-1], z[i] \oplus V[i], z[i] \oplus h[i-1] \oplus V[i]\}$
7: $h[i] \leftarrow o[i]$
8: **Return** $(h[i], W[i])$

$\overline{\mathcal{F}^A}(K, h[i-1], S[i], W[i])$

1: $x[i] \leftarrow f_1(h[i-1], W[i]) \mid x[i] \in \{h[i-1], W[i], h[i-1] \oplus W[i]\}$
2: $V[i] \leftarrow x[i]$
3: $y[i] \leftarrow f_2(h[i-1], W[i]) \mid y[i] \in \{h[i-1], W[i], h[i-1] \oplus W[i]\}$
4: $\mathcal{T}[i] \leftarrow S[i] \parallel y[i]$
5: $z[i] \leftarrow \widetilde{E}(K, \mathcal{T}[i], x[i])$
6: $o[i] \leftarrow f_3(z[i], h[i-1], W[i]) \mid o[i] \in \{z[i], z[i] \oplus h[i-1], z[i] \oplus W[i], z[i] \oplus h[i-1] \oplus W[i]\}$
7: $h[i] \leftarrow o[i]$
8: **Return** $(h[i], V[i])$

3.2.4 $\mathcal{F}^B$

Similar to the $\mathcal{F}^A$, $\mathcal{F}^B$ takes in three inputs and returns two outputs (refer figure 3.4) as described by the following equation:

$$\mathcal{F}^B(K, h[i-1], S[i], V[i]) \mapsto (h[i], W[i]) \tag{3.3}$$

where $K \in \{0,1\}^k$, $h[i-1] \in \{0,1\}^b$, $S[i] \in \{0,1\}^s$, $V[i] \in \{0,1\}^b$, $h[i] \in \{0,1\}^b$ and $W[i] \in \{0,1\}^b$. $\overline{\mathcal{F}^B}$ takes in three inputs and returns two outputs (refer figure 3.4):

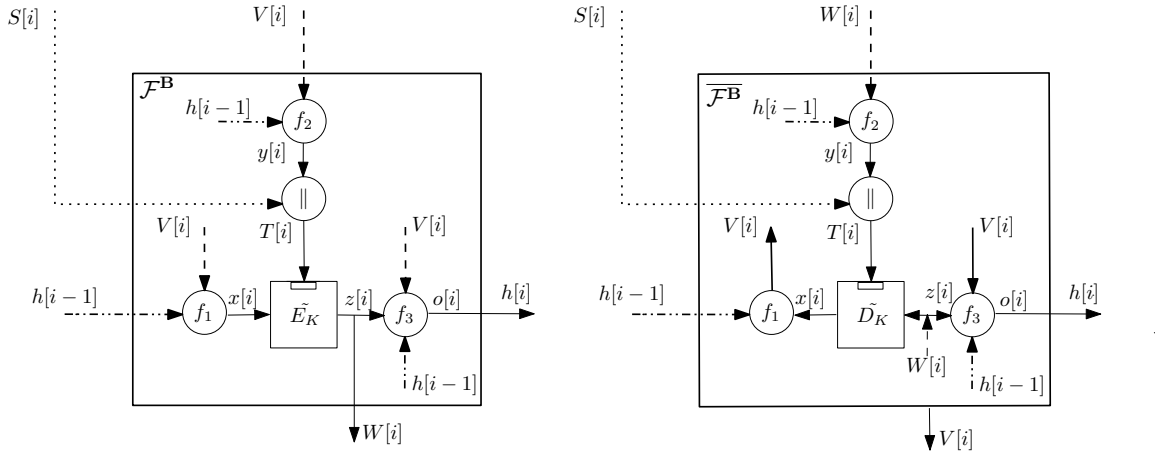
$$\overline{\mathcal{F}^B}(K, h[i-1], S[i], W[i]) \mapsto (h[i], V[i]) \tag{3.4}$$

We present the internal construction of $\mathcal{F}^B$ and $\overline{\mathcal{F}^B}$ in figure 3.4 and present their pseudocode in algorithm 6.

Table 3.6 gives the 36 cases of $\mathcal{F}^B$ (and $\overline{\mathcal{F}^B}$). The 4 cases marked in bold red font can only be used to create an authenticated encryption scheme (check section 3.3 for details). These 4 cases are used to create *lynx*-B family.

The fundamental difference between $\mathcal{F}^A$ and $\mathcal{F}^B$ family is based on the idea of generating the ciphertext. In $\mathcal{F}^A$, the ciphertext is generated after the *xor* operation i.e., $C[i] = x[i]$ (refer $W[i]$ in figure 3.3) while in case of $\mathcal{F}^B$, the ciphertext is produced from the output of the tweakable blockcipher i.e., $C[i] = z[i]$ (refer $W[i]$ in figure 3.4). Hence, in case of $\mathcal{F}^A$, a single bit difference causes one bit difference in the ciphertext, while, a single bit difference in case of $\mathcal{F}^B$, affects all the bits of the ciphertext (given a secure tweakable blockcipher).

Figure 3.4: Internal construction of the function family $\mathcal{F}^B$ / $\overline{\mathcal{F}^B}$. Internally it is composition of three sub-functions: f_1 and f_2 takes in two inputs and returns one output while f_3 takes in three input and returns one output



Algorithm 6 : $\mathcal{F}^B$ and $\overline{\mathcal{F}^B}$

$\mathcal{F}^B(K, h[i-1], S[i], V[i])$

1: $x[i] \leftarrow f_1(h[i-1], V[i]) \mid x[i] \in \{h[i-1], V[i], h[i-1] \oplus V[i]\}$
2: $y[i] \leftarrow f_2(h[i-1], V[i]) \mid y[i] \in \{h[i-1], V[i], h[i-1] \oplus V[i]\}$
3: $\mathcal{T}[i] \leftarrow S[i] \parallel y[i]$
4: $z[i] \leftarrow \widetilde{E}(K, \mathcal{T}[i], x[i])$
5: $W[i] \leftarrow z[i]$
6: $o[i] \leftarrow f_3(z[i], h[i-1], V[i]) \mid o[i] \in \{z[i], z[i] \oplus h[i-1], z[i] \oplus V[i], z[i] \oplus h[i-1] \oplus V[i]\}$
7: $h[i] \leftarrow o[i]$
8: **Return** $(h[i], W[i])$

$\overline{\mathcal{F}^B}(K, h[i-1], S[i], W[i])$

1: $x[i] \leftarrow f_1(h[i-1], W[i]) \mid x[i] \in \{h[i-1], W[i], h[i-1] \oplus W[i]\}$
2: $y[i] \leftarrow f_2(h[i-1], W[i]) \mid y[i] \in \{h[i-1], W[i], h[i-1] \oplus W[i]\}$
3: $\mathcal{T}[i] \leftarrow S[i] \parallel y[i]$
4: $z[i] \leftarrow \widetilde{D}(K, \mathcal{T}[i], x[i])$
5: $V[i] \leftarrow z[i]$
6: $o[i] \leftarrow f_3(z[i], h[i-1], W[i]) \mid o[i] \in \{z[i], z[i] \oplus h[i-1], z[i] \oplus W[i], z[i] \oplus h[i-1] \oplus W[i]\}$
7: $h[i] \leftarrow o[i]$
8: **Return** $(h[i], V[i])$

Table 3.4: Flag for lynx family. There are 8 bits in the flag with four most significant bits as always zero. The rest four bits denote processing of associated data, message, partial or full block, initialization phase and termination phase

Family of Lynx	**Operation**	**Flag Value**
Lynx-A1, Lynx-A2, Lynx-A3, Lynx-A4, Lynx-A5, Lynx-A6, Lynx-A7, Lynx-A8, Lynx-A9, Lynx-A10	Initialization	$0^4 0000$
	Process AD	$0^4 0001$ **if** $\lvert A[i]\rvert \mod b = 0$
		$0^4 0010$ **if** $\lvert A[i]\rvert \mod b \neq 0$
	Process Message	$0^4 0011$ **if** $\lvert M[i]\rvert \mod b = 0$
		$0^4 0100$ **if** $\lvert M[i]\rvert \mod b \neq 0$
	Tag Generation	$0^4 1111$
Lynx-B1, Lynx-B2, Lynx-B3, Lynx-B4	Process AD	$0^4 1001$ **if** $\lvert A[i]\rvert \mod b = 0$
		$0^4 1010$ **if** $\lvert A[i]\rvert \mod b \neq 0$
	Process Message	$0^4 1011$ **if** $\lvert M[i]\rvert \mod b = 0$
		$0^4 1100$ **if** $\lvert M[i]\rvert \mod b \neq 0$
	Termination	$0^4 1000$
	Tag Generation	$0^4 1111$

To establish the correctness of $\mathcal{F}^A/\overline{\mathcal{F}^A}$ and $\mathcal{F}^B/\overline{\mathcal{F}^B}$, we present following criteria:

Correctness: For a given K, $h[i-1]$, $V[i]$ and $S[i]$, a valid construction is defined as:

$$\begin{aligned} &\mathcal{F}^A(K, h[i-1], S[i], V[i]) \mapsto (h[i], W[i]) \\ &\overline{\mathcal{F}^A}(K, h[i-1], S[i], W[i]) \mapsto (h'[i], V'[i]) \\ &\qquad \text{where } (h'[i] = h[i]) \wedge (V'[i] = V[i]) \end{aligned} \tag{3.5}$$

and

$$\begin{aligned} &\mathcal{F}^B(K, h[i-1], S[i], V[i]) \mapsto (h[i], W[i]) \\ &\overline{\mathcal{F}^B}(K, h[i-1], S[i], W[i]) \mapsto (h'[i], V'[i]) \\ &\qquad \text{where } (h'[i] = h[i]) \wedge (V'[i] = V[i]) \end{aligned} \tag{3.6}$$

Table 3.5: $\mathcal{F}^{\mathcal{A}}$ and $\overline{\mathcal{F}^{\mathcal{A}}}$: Enumerated from 1 to 36. The bold letters in red color in the table denote the cases that fall into correct group.

$\mathcal{F}^{\mathbf{A}}$: $W[i] = x[i]$ And $\overline{\mathcal{F}^{\mathbf{A}}}$: $V[i] = x[i]$					
$x[i]$	$y[i]$	$o[i]$			
		$z[i]$	$z[i] \oplus V[i]$	$z[i] \oplus h[i-1]$	$z[i] \oplus V[i] \oplus h[i-1]$
$h[i-1]$ / $h[i-1]$	$V[i]$ / $W[i]$	$\mathcal{F}_1^A$ / $\overline{\mathcal{F}_1^A}$	$\mathcal{F}_2^A$ / $\overline{\mathcal{F}_2^A}$	$\mathcal{F}_3^A$ / $\overline{\mathcal{F}_3^A}$	$\mathcal{F}_4^A$ / $\overline{\mathcal{F}_4^A}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_5^A$ / $\overline{\mathcal{F}_5^A}$	$\mathcal{F}_6^A$ / $\overline{\mathcal{F}_6^A}$	$\mathcal{F}_7^A$ / $\overline{\mathcal{F}_7^A}$	$\mathcal{F}_8^A$ / $\overline{\mathcal{F}_8^A}$
	$V[i] \oplus h[i-1]$ / $W[i] \oplus h[i-1]$	$\mathcal{F}_9^A$ / $\overline{\mathcal{F}_9^A}$	$\mathcal{F}_{10}^A$ / $\overline{\mathcal{F}_{10}^A}$	$\mathcal{F}_{11}^A$ / $\overline{\mathcal{F}_{11}^A}$	$\mathcal{F}_{12}^A$ / $\overline{\mathcal{F}_{12}^A}$
$V[i]$ / $W[i]$	$V[i]$ / $W[i]$	$\mathcal{F}_{13}^A$ / $\overline{\mathcal{F}_{13}^A}$	$\mathcal{F}_{14}^A$ / $\overline{\mathcal{F}_{14}^A}$	$\mathcal{F}_{15}^A$ / $\overline{\mathcal{F}_{15}^A}$	$\mathcal{F}_{16}^A$ / $\overline{\mathcal{F}_{16}^A}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_{17}^A$ / $\overline{\mathcal{F}_{17}^A}$	$\mathcal{F}_{18}^A$ / $\overline{\mathcal{F}_{18}^A}$	$\mathcal{F}_{19}^A$ / $\overline{\mathcal{F}_{19}^A}$	$\mathcal{F}_{20}^A$ / $\overline{\mathcal{F}_{20}^A}$
	$V[i] \oplus h[i-1]$ / $W[i] \oplus h[i-1]$	$\mathcal{F}_{21}^A$ / $\overline{\mathcal{F}_{21}^A}$	$\mathcal{F}_{22}^A$ / $\overline{\mathcal{F}_{22}^A}$	$\mathcal{F}_{23}^A$ / $\overline{\mathcal{F}_{23}^A}$	$\mathcal{F}_{24}^A$ / $\overline{\mathcal{F}_{24}^A}$
$h[i-1] \oplus V[i]$ / $h[i-1] \oplus W[i]$	$V[i]$ / $V[i]$	$\mathcal{F}_{\mathbf{25}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{25}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{26}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{26}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{27}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{27}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{28}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{28}}^{\mathbf{A}}}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_{\mathbf{29}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{29}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{30}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{30}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{31}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{31}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{32}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{32}}^{\mathbf{A}}}$
	$V[i] \oplus h[i-1]$ / $V[i] \oplus h[i-1]$	$\mathcal{F}_{33}^A$ / $\overline{\mathcal{F}_{33}^A}$	$\mathcal{F}_{\mathbf{34}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{34}}^{\mathbf{A}}}$	$\mathcal{F}_{\mathbf{35}}^{\mathbf{A}}$ / $\overline{\mathcal{F}_{\mathbf{35}}^{\mathbf{A}}}$	$\mathcal{F}_{36}^A$ / $\overline{\mathcal{F}_{36}^A}$

Table 3.6: $\mathcal{F}^{\mathcal{B}}$ and $\overline{\mathcal{F}^{\mathcal{B}}}$: Enumerated from 1 to 36. The bold letters in red color in the table denote the cases that fall into correct group.

$\mathcal{F}^{\mathbf{B}}, W[i] = z[i]$ And $\overline{\mathcal{F}^{\mathbf{B}}}$: $V[i] = x[i]$					
$x[i]$	$y[i]$	$o[i]$			
		$z[i]$	$z[i] \oplus V[i]$	$z[i] \oplus h[i-1]$	$z[i] \oplus V[i] \oplus h[i-1]$
$h[i-1]$ / $h[i-1]$	$V[i]$ / $W[i]$	$\mathcal{F}_1^B$ / $\overline{\mathcal{F}_1^B}$	$\mathcal{F}_2^B$ / $\overline{\mathcal{F}_2^B}$	$\mathcal{F}_3^B$ / $\overline{\mathcal{F}_3^B}$	$\mathcal{F}_4^B$ / $\overline{\mathcal{F}_4^B}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_5^B$ / $\overline{\mathcal{F}_5^B}$	$\mathcal{F}_6^B$ / $\overline{\mathcal{F}_6^B}$	$\mathcal{F}_7^B$ / $\overline{\mathcal{F}_7^B}$	$\mathcal{F}_8^B$ / $\overline{\mathcal{F}_8^B}$
	$V[i] \oplus h[i-1]$ / $W[i] \oplus h[i-1]$	$\mathcal{F}_9^B$ / $\overline{\mathcal{F}_9^B}$	$\mathcal{F}_{10}^B$ / $\overline{\mathcal{F}_{10}^B}$	$\mathcal{F}_{11}^B$ / $\overline{\mathcal{F}_{11}^B}$	$\mathcal{F}_{12}^B$ / $\overline{\mathcal{F}_{12}^B}$
$V[i]$ / $V[i]$	$V[i]$ / $W[i]$	$\mathcal{F}_{13}^B$ / $\overline{\mathcal{F}_{13}^B}$	$\mathcal{F}_{14}^B$ / $\overline{\mathcal{F}_{14}^B}$	$\mathcal{F}_{15}^B$ / $\overline{\mathcal{F}_{15}^B}$	$\mathcal{F}_{16}^B$ / $\overline{\mathcal{F}_{16}^B}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_{17}^B$ / $\overline{\mathcal{F}_{17}^B}$	$\mathcal{F}_{\mathbf{18}}^{\mathbf{B}}$ / $\overline{\mathcal{F}_{\mathbf{18}}^{\mathbf{B}}}$	$\mathcal{F}_{19}^B$ / $\overline{\mathcal{F}_{19}^B}$	$\mathcal{F}_{\mathbf{20}}^{\mathbf{B}}$ / $\overline{\mathcal{F}_{\mathbf{20}}^{\mathbf{B}}}$
	$V[i] \oplus h[i-1]$ / $W[i] \oplus h[i-1]$	$\mathcal{F}_{21}^B$ / $\overline{\mathcal{F}_{21}^B}$	$\mathcal{F}_{22}^B$ / $\overline{\mathcal{F}_{22}^B}$	$\mathcal{F}_{23}^B$ / $\overline{\mathcal{F}_{23}^B}$	$\mathcal{F}_{24}^B$ / $\overline{\mathcal{F}_{24}^B}$
$h[i-1] \oplus V[i]$ / $h[i-1] \oplus V[i]$	$V[i]$ / $W[i]$	$\mathcal{F}_{25}^B$ / $\overline{\mathcal{F}_{25}^B}$	$\mathcal{F}_{26}^B$ / $\overline{\mathcal{F}_{26}^B}$	$\mathcal{F}_{27}^B$ / $\overline{\mathcal{F}_{27}^B}$	$\mathcal{F}_{28}^B$ / $\overline{\mathcal{F}_{28}^B}$
	$h[i-1]$ / $h[i-1]$	$\mathcal{F}_{29}^B$ / $\overline{\mathcal{F}_{29}^B}$	$\mathcal{F}_{\mathbf{30}}^{\mathbf{B}}$ / $\overline{\mathcal{F}_{\mathbf{30}}^{\mathbf{B}}}$	$\mathcal{F}_{31}^B$ / $\overline{\mathcal{F}_{31}^B}$	$\mathcal{F}_{\mathbf{32}}^{\mathbf{B}}$ / $\overline{\mathcal{F}_{\mathbf{32}}^{\mathbf{B}}}$
	$V[i] \oplus h[i-1]$ / $W[i] \oplus h[i-1]$	$\mathcal{F}_{33}^B$ / $\overline{\mathcal{F}_{33}^B}$	$\mathcal{F}_{34}^B$ / $\overline{\mathcal{F}_{34}^B}$	$\mathcal{F}_{35}^B$ / $\overline{\mathcal{F}_{35}^B}$	$\mathcal{F}_{36}^B$ / $\overline{\mathcal{F}_{36}^B}$

3.2.5 Domain Separation

We use one byte flag to provide domain separation in various scenarios. The flag we use is represented by following notation: $l_{7...0}$. The four most significant bits of flag is always 0 i.e., $l_{7...4} \leftarrow 0$. Rest of the four bits i.e., the four least significant bits provide domain separation for

Table 3.7: Incorrect cases of $\mathcal{F}^A$ and $\mathcal{F}^B$ that fall into implausible case or non-confidential case or non-integrity case

Implausible Cases	$\mathcal{F}^A_1, \mathcal{F}^A_2, \mathcal{F}^A_3, \mathcal{F}^A_4, \mathcal{F}^A_5, \mathcal{F}^A_6, \mathcal{F}^A_7, \mathcal{F}^A_8$ $\mathcal{F}^A_9, \mathcal{F}^A_{10}, \mathcal{F}^A_{11}, \mathcal{F}^A_{12}, \mathcal{F}^B_1, \mathcal{F}^B_2, \mathcal{F}^B_3, \mathcal{F}^B_4$ $\mathcal{F}^B_5, \mathcal{F}^B_6, \mathcal{F}^B_7, \mathcal{F}^B_8, \mathcal{F}^B_9, \mathcal{F}^B_{10}, \mathcal{F}^B_{11}, \mathcal{F}^B_{12}$ $\mathcal{F}^B_{13}, \mathcal{F}^B_{14}, \mathcal{F}^B_{15}, \mathcal{F}^B_{16}, \mathcal{F}^B_{21}, \mathcal{F}^B_{22}, \mathcal{F}^B_{23}, \mathcal{F}^B_{24}$ $\mathcal{F}^B_{25}, \mathcal{F}^B_{26}, \mathcal{F}^B_{27}, \mathcal{F}^B_{28}, \mathcal{F}^B_{33}, \mathcal{F}^B_{34}, \mathcal{F}^B_{35}, \mathcal{F}^B_{36}$
Non-Confidential Cases	$\mathcal{F}^A_{13}, \mathcal{F}^A_{14}, \mathcal{F}^A_{15}, \mathcal{F}^A_{16}, \mathcal{F}^A_{17}, \mathcal{F}^A_{18}, \mathcal{F}^A_{19}, \mathcal{F}^A_{20}$ $\mathcal{F}^A_{21}, \mathcal{F}^A_{22}, \mathcal{F}^A_{23}, \mathcal{F}^A_{24}$
Non-Integrity Cases	$\mathcal{F}^A_{33}, \mathcal{F}^A_{36}, \mathcal{F}^B_{17}, \mathcal{F}^B_{19}, \mathcal{F}^B_{29}, \mathcal{F}^B_{31}$

various scenarios: distinction between underlying lynx family, initialization phase, associated data processing, message processing, termination phase (in case of lynx-B) and tag generation phase. Flag also keeps the track of blocks, weather the encountered block is full or partial. Further, the flag usage facilitates integrity and confidentiality. Table 3.4 shows the values taken by the flag bits. The bit l_3 serves as domain separator between the two families of the lynx.

3.3 Analysis of $\mathcal{F}^A$ and $\mathcal{F}^B$

As stated earlier, only 10 cases of $\mathcal{F}^A$ and 4 cases of $\mathcal{F}^B$ can be used for creating authenticated encryption scheme (marked in bold red font in table 3.5 and table 3.6). In this section, we analyze each case of $\mathcal{F}^A$ and $\mathcal{F}^B$ in light of creating an authenticated encryption scheme and show why 58 cases cannot be used for creating an authenticated encryption scheme. Hence, we divide the 72 cases into two groups: incorrect group (58 cases) and correct group (14 cases). Further there are three sub-cases in the incorrect group: implausible cases, non-confidential cases and non-integrity cases. We next list $\mathcal{F}^A$ and $\mathcal{F}^B$ into these sub-cases of the incorrect group. In this section, we use notation $\mathcal{F}^A_{i...k}$ to cover all the cases of $\mathcal{F}^A$ between $\mathcal{F}^A_i$ (inclusive) and $\mathcal{F}^A_k$ (inclusive), thereby making a total of $k - i + 1$ cases. We use similar notation for $\mathcal{F}^B$ as well.

- **Implausible Cases:** Such cases of $\mathcal{F}^A$ and $\mathcal{F}^B$ where the ciphertext is independent of the

input message i.e., there is no relation between ciphertext and the input message and hence it is impossible to recover back the original message given the ciphertext. We focus on $\mathcal{F}^A_1$. From table 3.5, it is clear that $x[i] = h[i-1] = C[i]$, this makes ciphertext independent of the input message. Hence, given $C[i]$, it not possible to recover back the input message $M[i]$. $\mathcal{F}^A_{2...12}$ fall into this sub-class as well.

In case of $\mathcal{F}^B$, one can see from the design that if the message is a part of the tweak then it is not possible to recover back the message given the ciphertext. Hence, $\mathcal{F}^B_{1...4}$, $\mathcal{F}^B_{9...12}$, $\mathcal{F}^B_{13...16}$, $\mathcal{F}^B_{21...24}$, $\mathcal{F}^B_{25...28}$ and $\mathcal{F}^B_{33...36}$ fall into this category as well.

Now consider $\mathcal{F}^B_5$, message is neither part of the input nor part of the tweak and hence this case as well, is independent of the input message. $\mathcal{F}^B_{6...8}$ also have similar pattern.

- **Non-Confidential Cases:** In such cases, the message itself becomes the ciphertext hence offering no privacy at all. In case of $\mathcal{F}^A_{13}$, one can see that the message block $M[i]$ becomes the ciphertext block, there by making input message public. $\mathcal{F}^A_{14...24}$ fall into this sub-class as well.

- **Non-Integrity Cases:** Consider $\mathcal{F}^A_{33}$, the input to the tweakable blockcipher, $x[i] = h[i-1] \oplus M[i] = C[i]$. Further, the tweak to the tweakable blockcipher, $\mathcal{T}[i] = S[i] \parallel h[i-1] \oplus M[i] = S[i] \parallel C[i]$. Now consider two ciphertext C and C' of lengths c and c'. Without loss of generality, if the last block of the two ciphertexts are equal i.e., $C[c] = C'[c']$, then both the ciphertext produces same tag even though other blocks of ciphertext may be different.

  Now consider $\mathcal{F}^A_{36}$ which has similar design to $\mathcal{F}^A_{33}$. Input to the tweakable blockcipher, $x[i] = h[i-1] \oplus M[i] = C[i]$ and tweak to the tweakable blockcipher, $\mathcal{T}[i] = S[i] \parallel h[i-1] \oplus M[i] = S[i] \parallel C[i]$. Further, in case of $\mathcal{F}^A_{36}$, $h[i] = z[i] \oplus h[i-1] \oplus M[i] = z[i] \oplus C[i]$. Hence, similar to the case of $\mathcal{F}^A_{33}$, the dependency is only on the ciphertext.

  In $\mathcal{F}^B_{17}$, one can see that the output is only dependent on the ciphertext i.e., $C[i]$ and hence like in the previous cases forgery is possible by tweaking the last block. $\mathcal{F}^B_{29}$ has similar problem as well.

  In $\mathcal{F}^B_{19}$, the output is dependent on both the ciphertext and the output from the previous block i.e., $h[i-1]$. One can see that it is possible to change the bits of two blocks of the

ciphertext to create forgery. $\mathcal{F}_{31}^{B}$ has similar problem as well.

Table 3.7 summarizes all the incorrect cases. The cases that do not fall into the incorrect group, fall into the correct group. The correct group cases have constructions that make them fundamentally distinct from implausible cases, non-confidential cases or non-integrity cases. We present formal security proofs for lynx-A and lynx-B, based on these 14 cases, in the security proof section (section 3.6).

3.4 Internal Design of Members of the Lynx Family

In this section, we provide the detailed internals of each of the members of the lynx family, under the assumption that there are two blocks of associated data and two blocks of message .i.e., $a = 2$ and $m = 2$. Further, we assume that both the associated data as well as the message are complete, hence no padding is required. Hence, following are the values for the flag .i.e., the domain separation:

1. Lynx-A$\{1 \ldots 10\}$
    - Initialization: $0^4 0000$.
    - Associated data processing: $0^4 0001$.
    - Message Processing: $0^4 0011$.
    - Tag Generation: $0^4 1111$.
2. Lynx-B$\{1 \ldots 4\}$
    - Associated data processing: $0^4 1001$.
    - Message Processing: $0^4 1011$.
    - Termination: $0^4 1000$
    - Tag Generation: $0^4 1111$.

Figure 3.5: Encryption and Decryption sub-routine of Lynx-A1 (Function: $\mathcal{F}_{25}^{A}$).

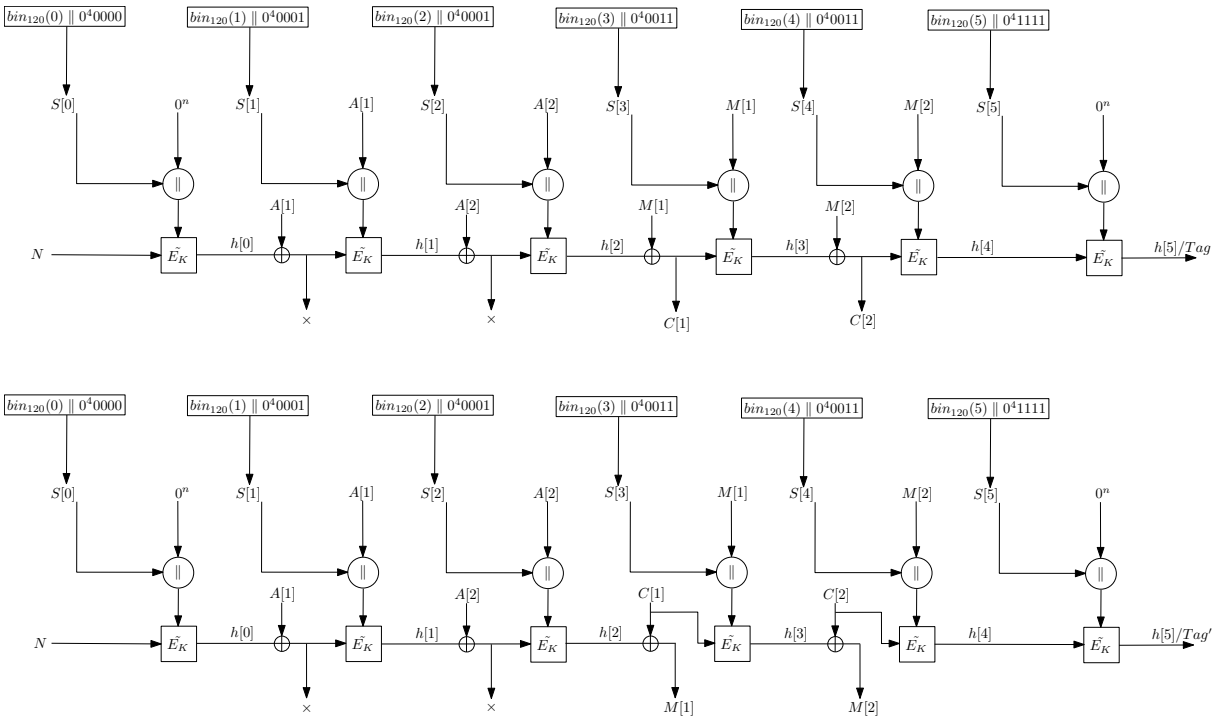


Figure 3.5 through figure 3.14 shows the internal states for lynx-A1 through lynx-A10 respectively, while figure 3.15 through figure 3.18 shows the internal states for lynx-B1 through lynx-B4 respectively.

3.5 Design Rationale

In this section, we discuss the significance of various components of lynx.

3.5.1 Initialization in Lynx-A

Lynx-A starts with the initialization phase (refer figure 3.1) which requires an extra block processing before the start of the processing of the associated data and the message. This initialization phase utilizes a public parameter called the nonce (N) and produces an output $h[0]$. This $h[0]$ is used later in the processing of associated data and the message. In the absence of the initialization phase and when $A = \epsilon$ or $|A| = 0$, nonce N is directly used to

Figure 3.6: Encryption and Decryption sub-routine of Lynx-A2 (Function: $\mathcal{F}_{26}^{A}$).

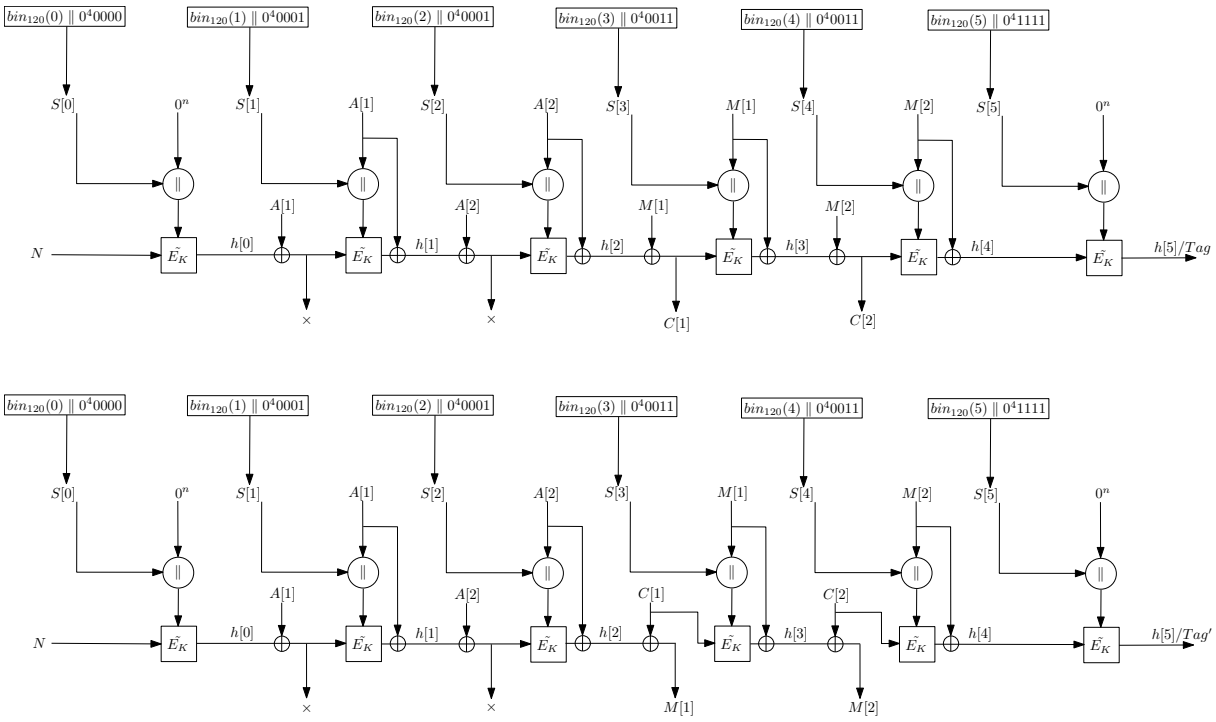


Figure 3.7: Encryption and Decryption sub-routine of Lynx-A3 (Function: $\mathcal{F}_{27}^{A}$).

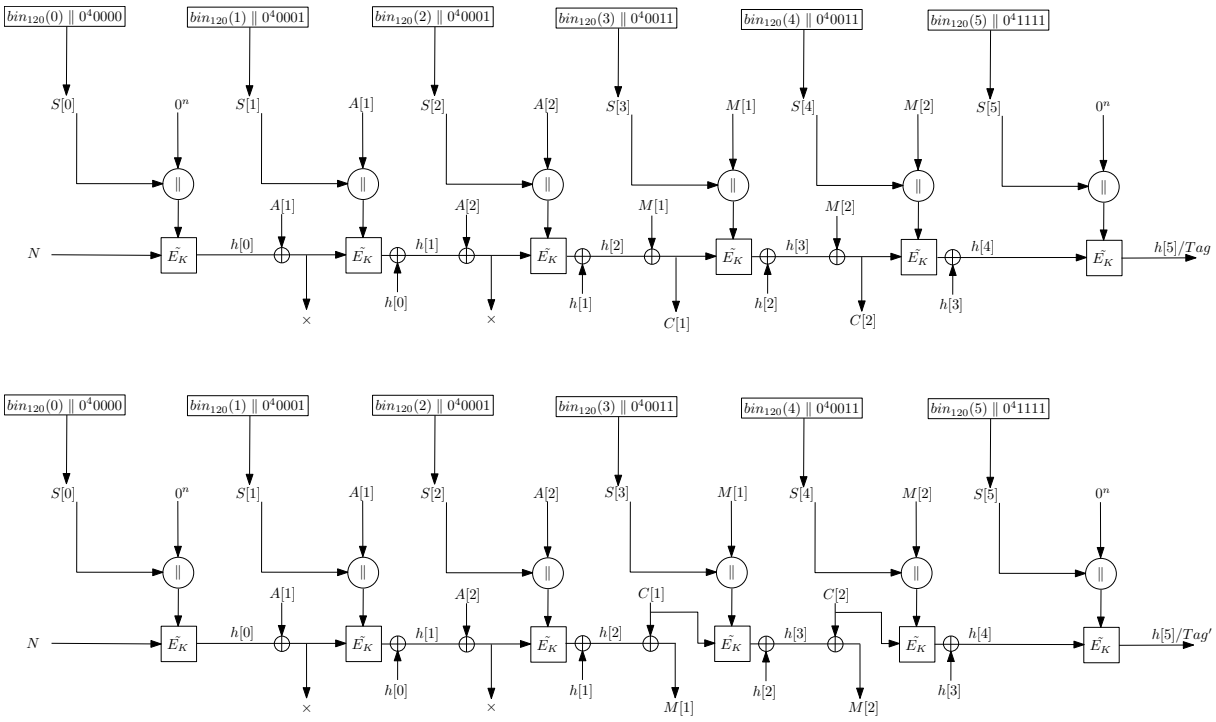


Figure 3.8: Encryption and Decryption sub-routine of Lynx-A4 (Function: $\mathcal{F}_{28}^{A}$).

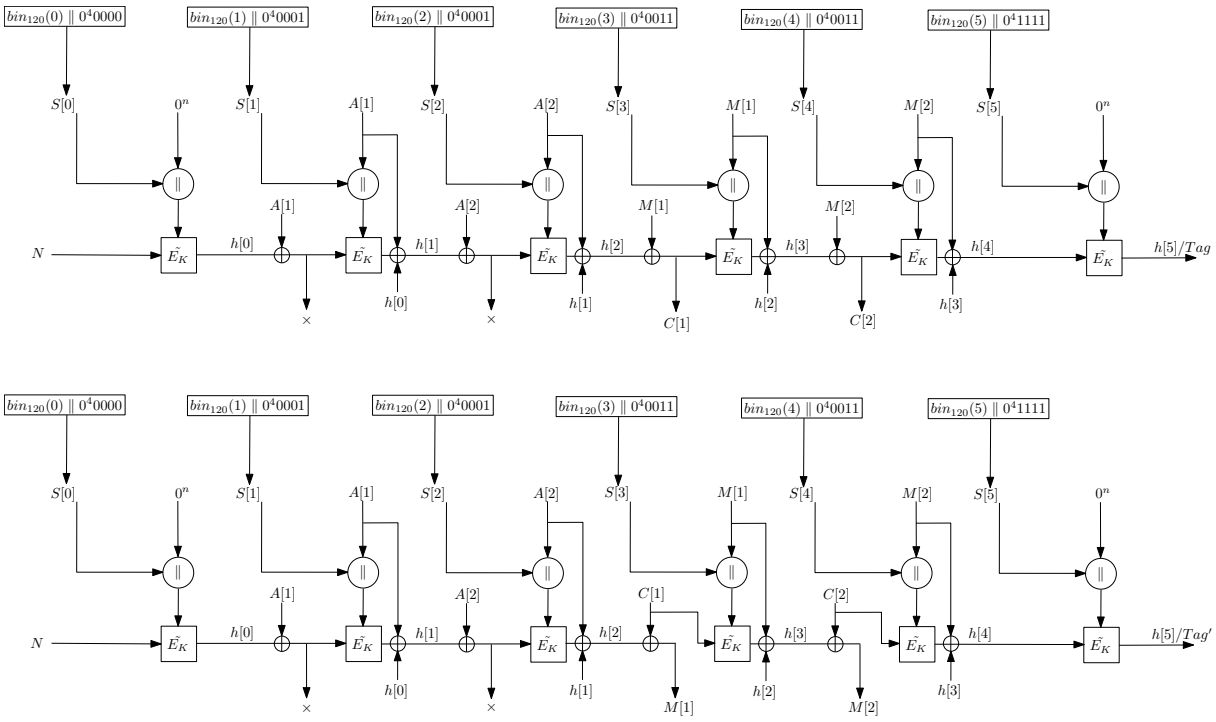


Figure 3.9: Encryption and Decryption sub-routine of Lynx-A5 (Function: $\mathcal{F}_{29}^{A}$).

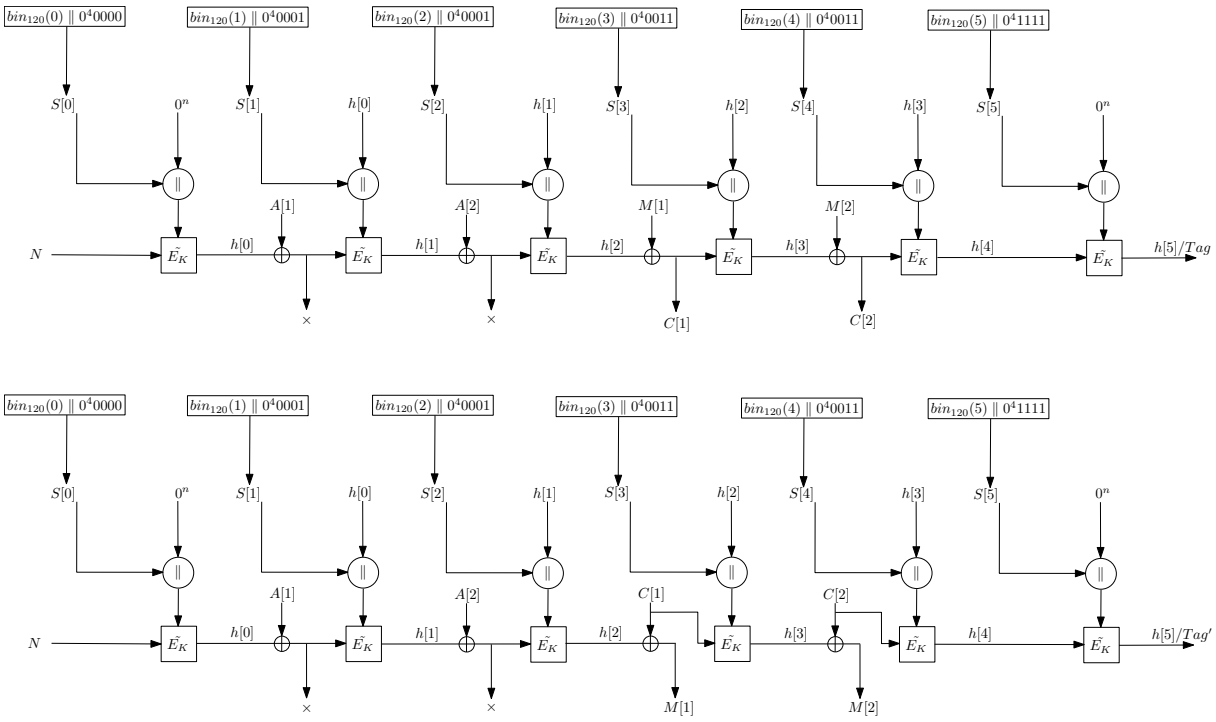


Figure 3.10: Encryption and Decryption sub-routine of Lynx-A6 (Function: $\mathcal{F}_{30}^{A}$).

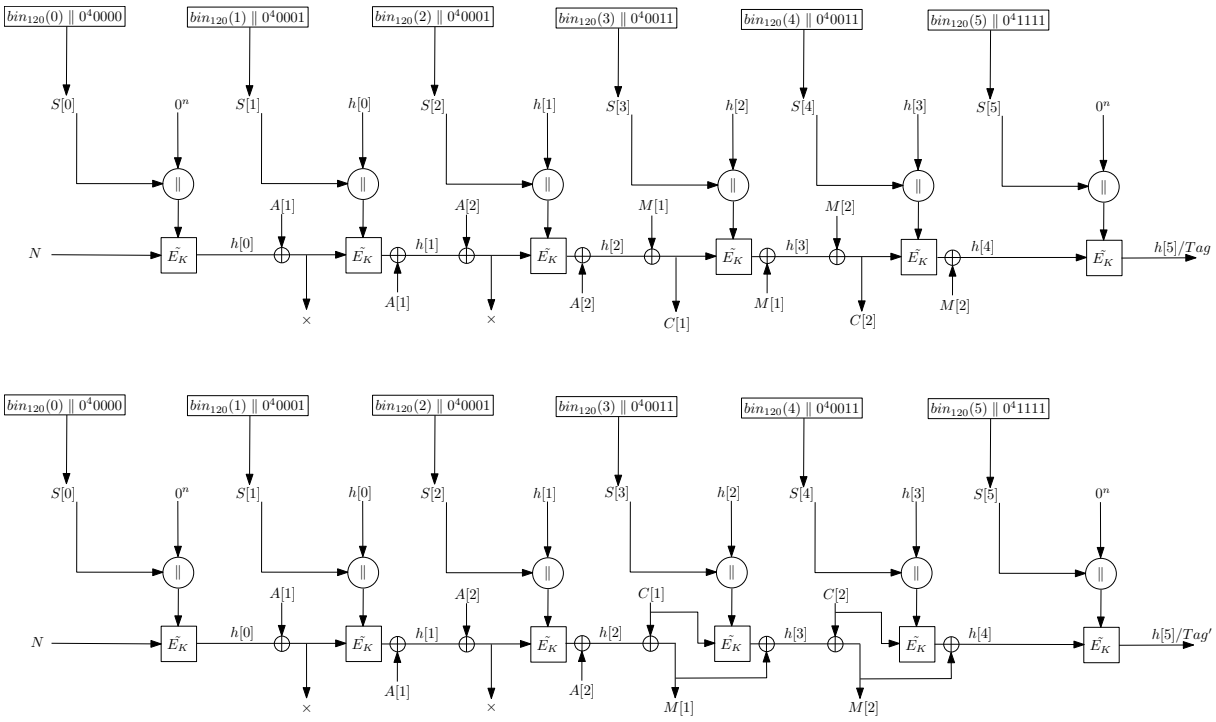


Figure 3.11: Encryption and Decryption sub-routine of Lynx-A7 (Function: $\mathcal{F}_{31}^{A}$).

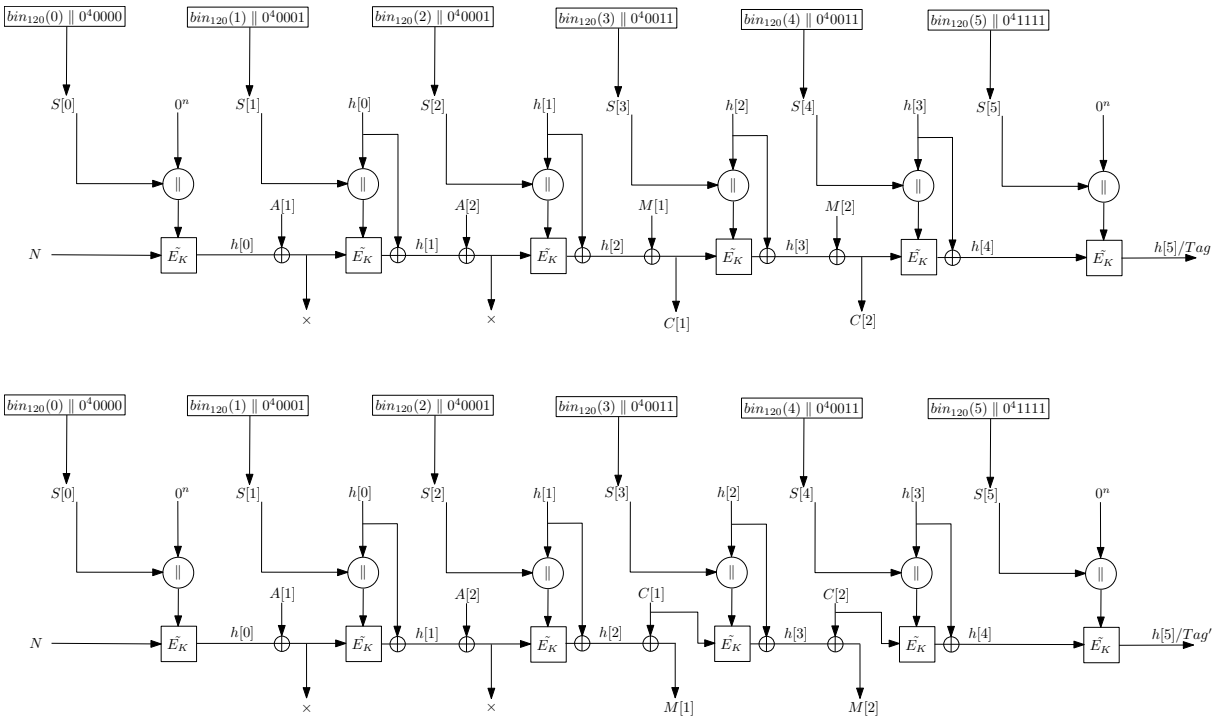


Figure 3.12: Encryption and Decryption sub-routine of Lynx-A8 (Function: $\mathcal{F}_{32}^{A}$).

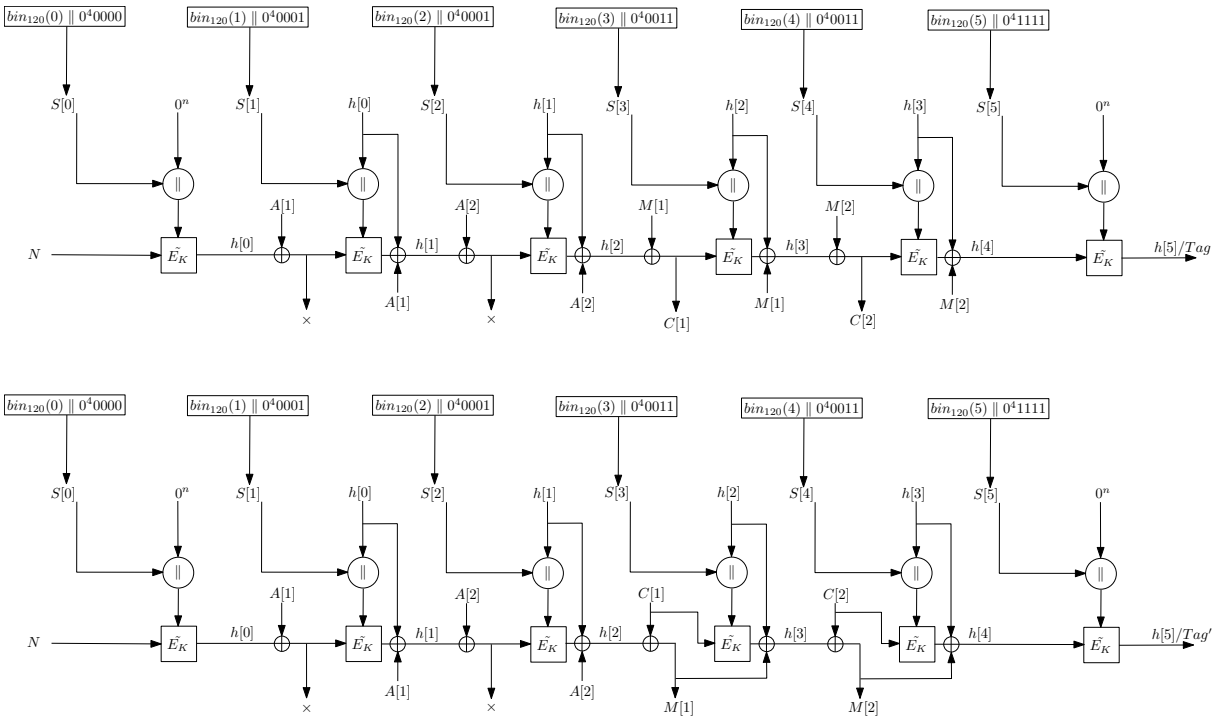


Figure 3.13: Encryption and Decryption sub-routine of Lynx-A9 (Function: $\mathcal{F}_{34}^{A}$).

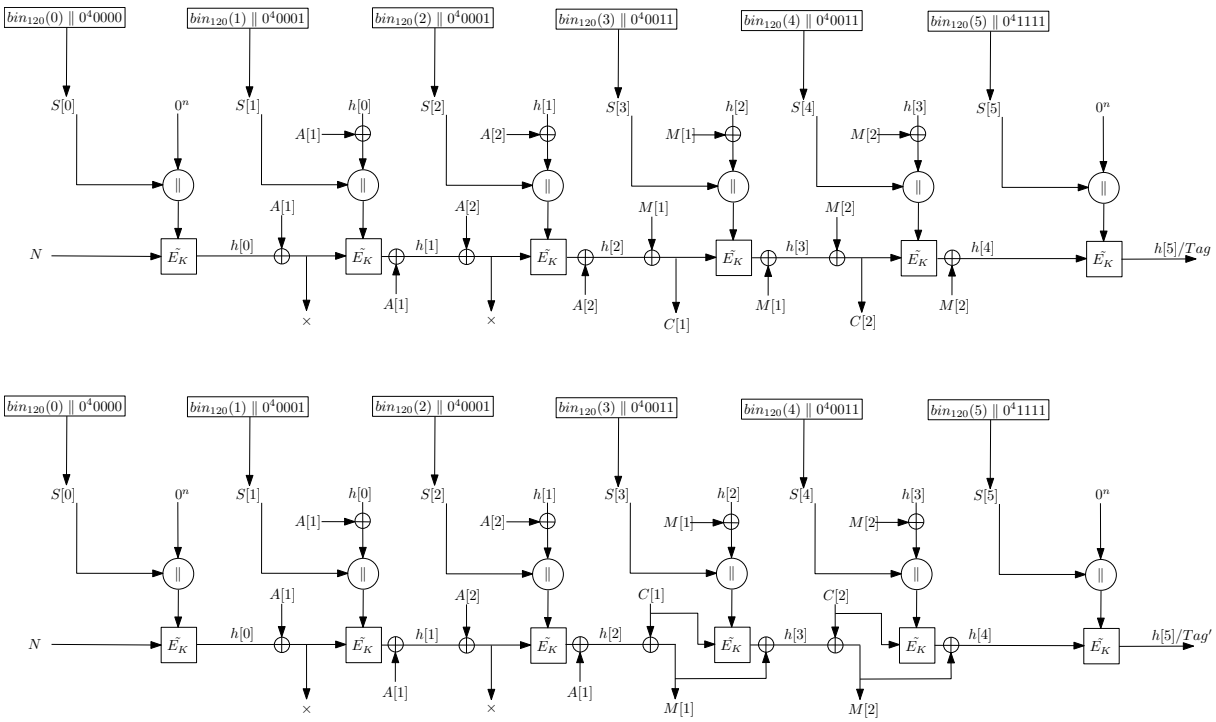


Figure 3.14: Encryption and Decryption sub-routine of Lynx-A10 (Function: $\mathcal{F}_{35}^{A}$).

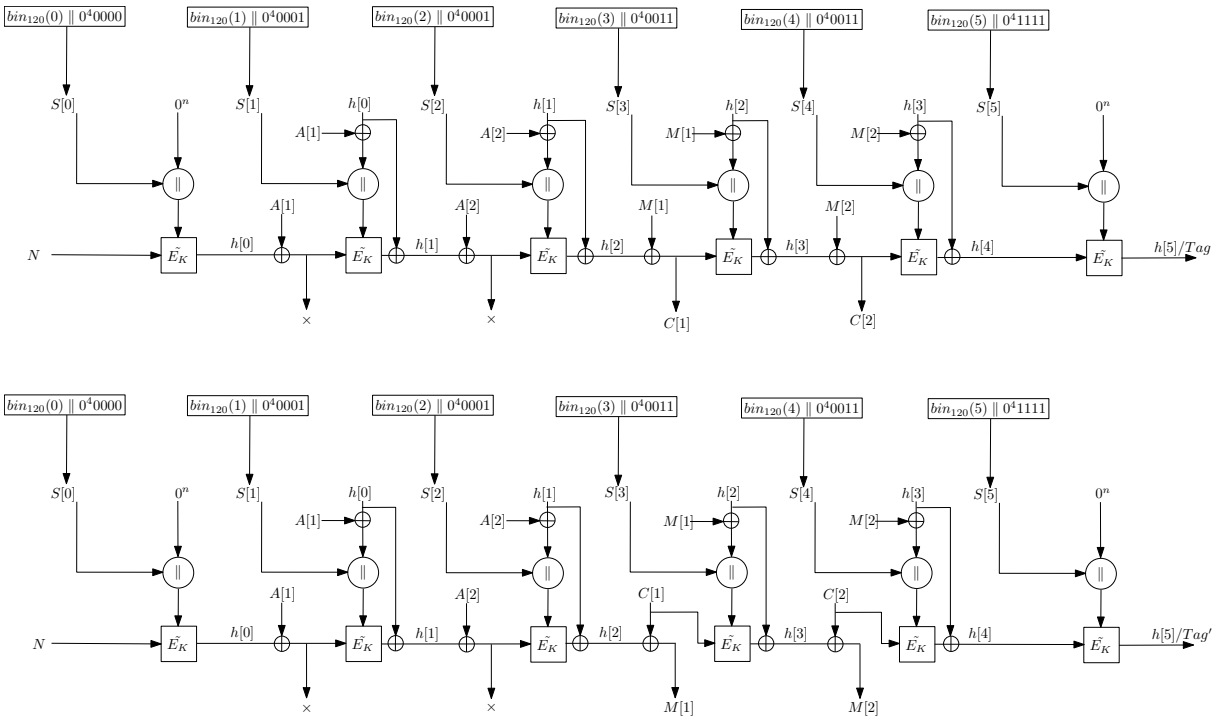


Figure 3.15: Encryption and Decryption sub-routine of Lynx-B1 (Function: $\mathcal{F}_{18}^{B}$).

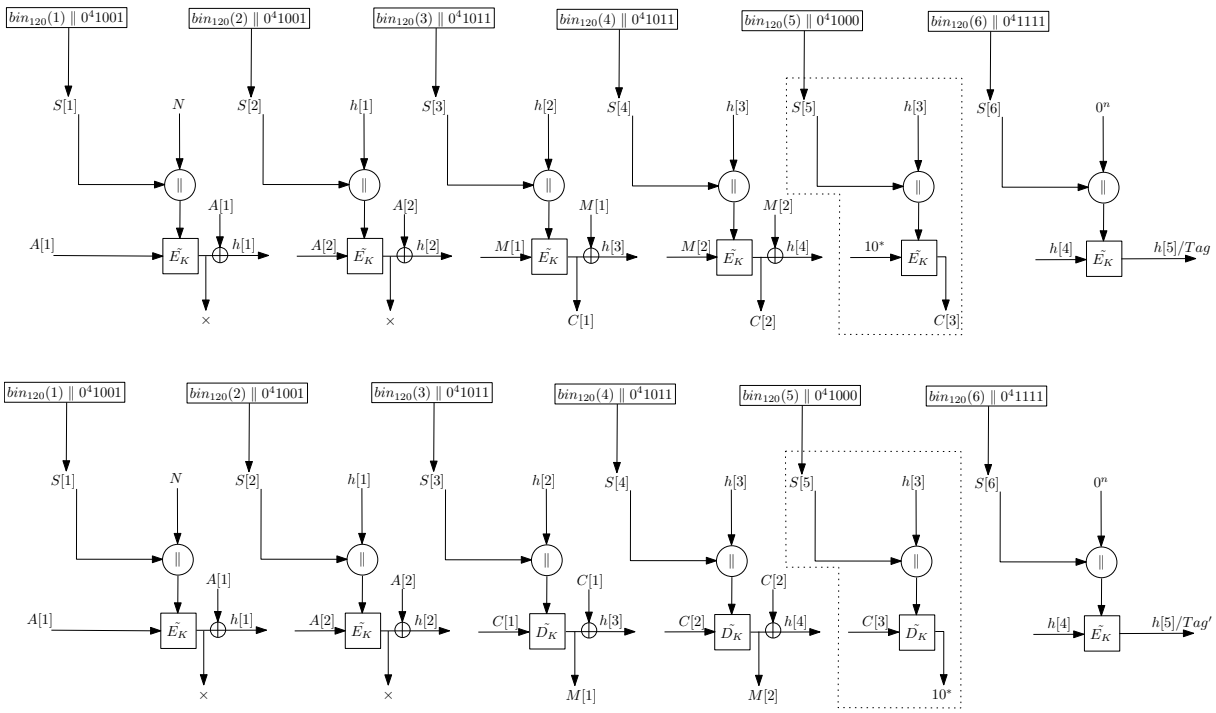


Figure 3.16: Encryption and Decryption sub-routine of Lynx-B2 (Function: $\mathcal{F}_{20}^{B}$).

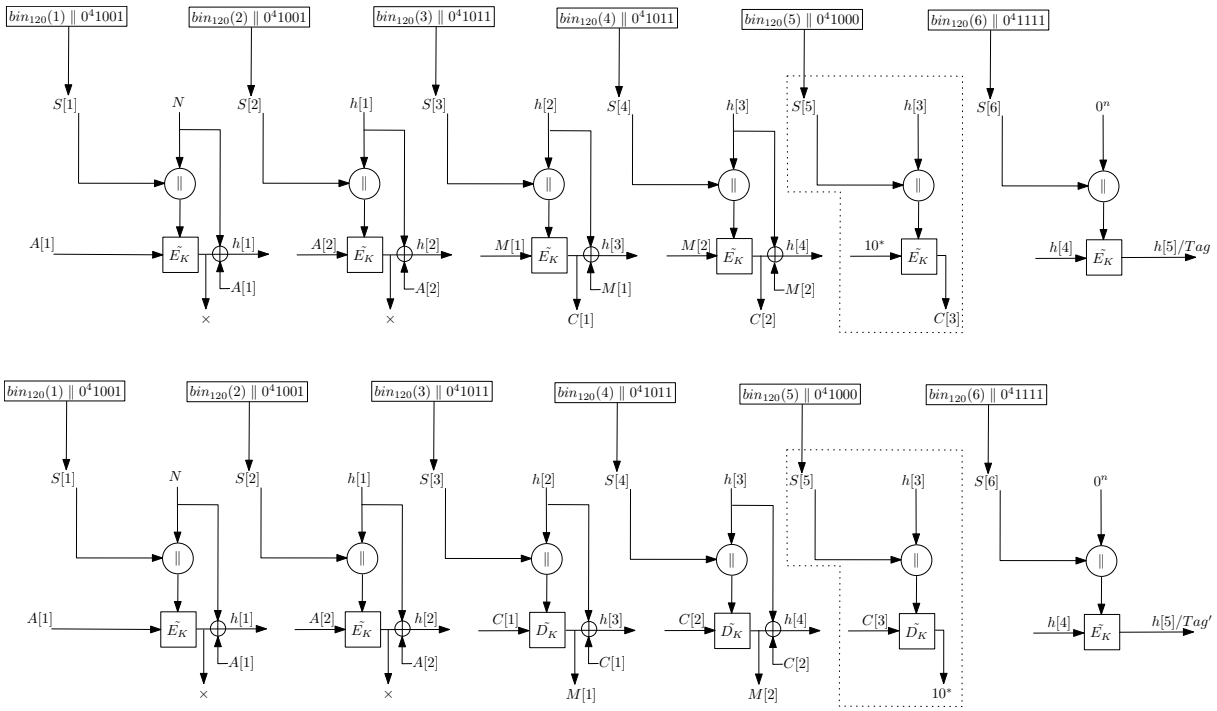


Figure 3.17: Encryption and Decryption sub-routine of Lynx-B3 (Function: $\mathcal{F}_{30}^{B}$).

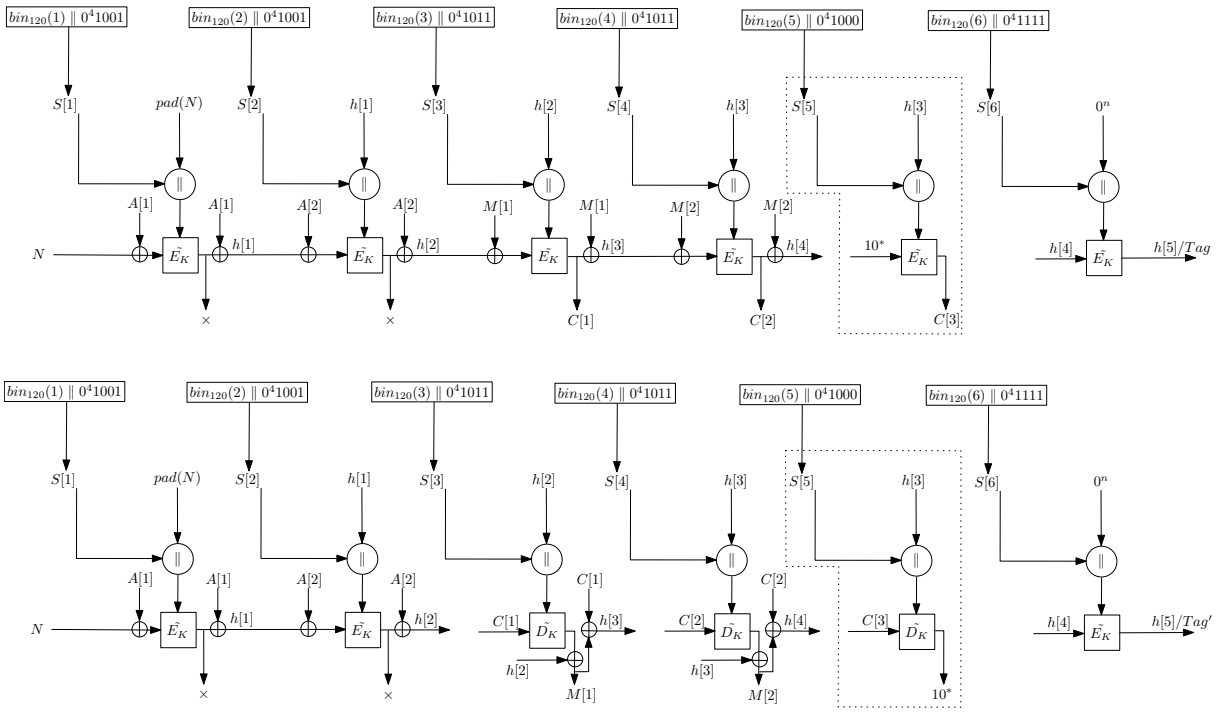
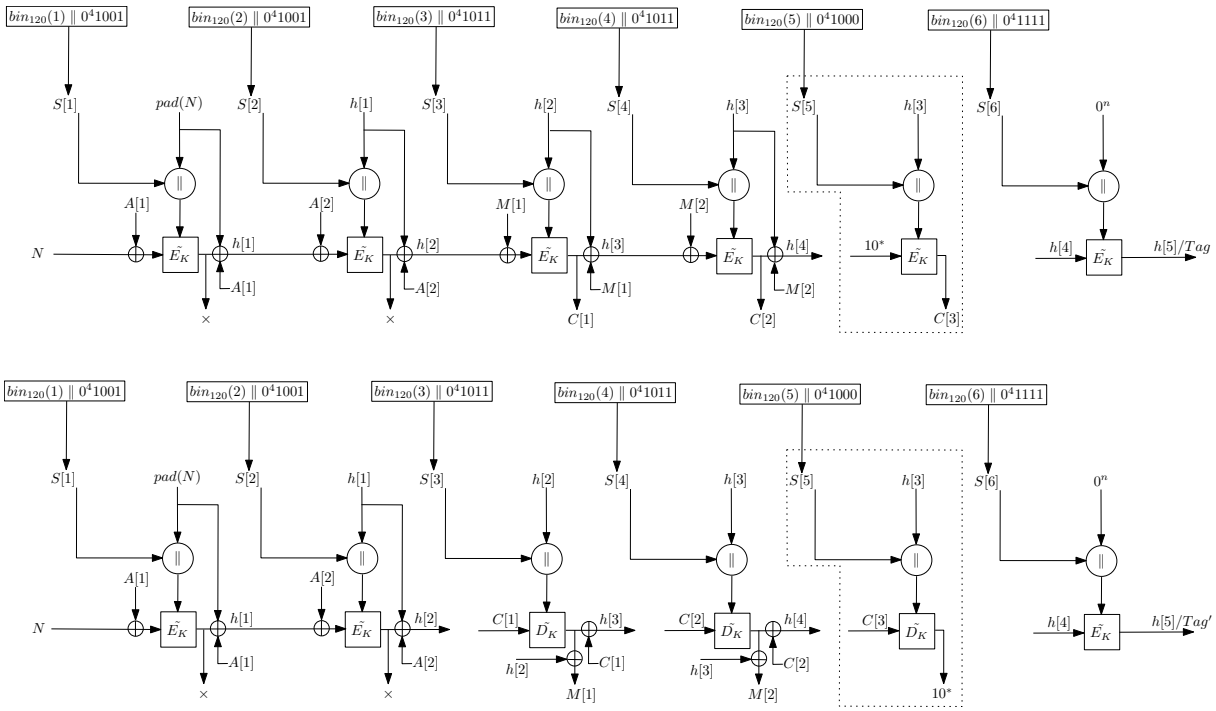


Figure 3.18: Encryption and Decryption sub-routine of Lynx-B4 (Function: $\mathcal{F}_{32}^{B}$).



produce the ciphertext. Since N is public, it is trivial to recover the first message block knowing the first ciphertext block even in nonce respecting scenario. To thwart such a condition, the initialization phase is needed. In case of lynx-B, due to its internal construction, we do not need any initialization phase.

3.5.2 Termination in Lynx-B

In Lynx-B, we use termination phase (refer figure 3.2) which is an extra block processing after processing the associated data and the message and before the tag generation phase. The idea behind the termination phase is to recover complete message given the ciphertext. In lynx-B, if the last message block is incomplete or partial then we pad it using the equation (2.1). Lynx-B loses the track of message length after using the padding function and hence needs an extra block that keeps the track of the message length of the last message block. Further, due to this extra block, lynx-B is not length preserving. In case of lynx-A, due to its construction, we do not need any termination phase.

3.5.3 Tag Generation

The lynx family uses tag generation phase (refer figure 3.1 for lynx-A and figure 3.2 for lynx-B) which is an extra block processing phase after processing the associated data and the message. This extra block processing marks the termination of lynx. For lynx-A, addition of this tag generation phase eases the security proof. For lynx-B, in the absence of tag generation phase, it is easy to perform forgery due to the internal construction of functions $\mathcal{F}_{18}^{B}$, $\mathcal{F}_{20}^{B}$, $\mathcal{F}_{30}^{B}$ and $\mathcal{F}_{32}^{B}$ that are used for constructing lynx-B1, lynx-B2, lynx-B3 and lynx-B4 respectively. In case of $\mathcal{F}_{18}^{B}$ and $\mathcal{F}_{30}^{B}$, if there is no tag generation phase then from table 3.6 we know that the output from $\mathcal{F}_{18}^{B}$ and $\mathcal{F}_{30}^{B}$ is dependent only on the message and ciphertext, hence by simply replacing the last block of the ciphertext, one can easily perform forgery. In case of $\mathcal{F}_{20}^{B}$ and $\mathcal{F}_{32}^{B}$, when there is no tag generation phase then performing forgery attack is a two step process; firstly from message and ciphertext, the attacker can get the previous h values (refer table 3.6), then in the second step attacker can simply replacing the last block of the ciphertext to perform tag forgery. In the presence of tag generation phase, no such tag forgery attack is possible.

3.5.4 Stream Processing

The design of lynx ensures the current block processing doesn't need information of the block coming ahead. For example: if the next block is partial or full, or if next next block is a message block while processing associated data. Such scenarios are practical and of significance in low powered devices where it may not be possible to hold up ciphertext or messages during encryption or decryption due to the memory constraints.

3.5.5 Design of Flag

The choice of one byte flag was done to facilitate stream processing (see section 3.5.4). The bits provide domain separation but at the same time doesn't require extra knowledge about the upcoming blocks.

3.5.6 Counter

The counter serves as the part of the tweak of the tweakable blockcipher. Hence, for any two queries, if the query length is not same, then due to the usage of counter as part of the tweak, the tweaks of the tweakable blockcipher of the respective queries becomes different.

3.5.7 Simple Operation

Lynx is completely based on *xor* operations and does not have any matrix or field computations. This makes lynx computationally efficient and cost effective due to low area requirement for hardware implementations.

The design of lynx ensures the three essential components of lightweight cryptographic primitives (security, performance and cost) [104]. We next present provable security proofs.

3.6 Provable Security Proofs

The lynx family consists of 14 members divided into two sub-families lynx-A and lynx-B. These two families are created based on function family $\mathcal{F}^A$ and $\mathcal{F}^B$ respectively. There are 10 members of lynx-A namely lynx-A1, lynx-A2, $\ldots$, lynx-A10 while there are 4 members of lynx-B namely lynx-B1, lynx-B2, lynx-B3 and lynx-A10. In this section for simplicity and conciseness, we refer the lynx-A family as lynx-A$\{1 \ldots 10\}$ and lynx-B family as lynx-B$\{1 \ldots 4\}$. Figure 3.1 and figure 3.2 shows the internal construction of lynx-A and lynx-B while the internal construction of the function $\mathcal{F}^A$ and $\mathcal{F}^B$ is shown in figure 3.3 and figure 3.4. Table 3.1 and table 3.2 lists down $\mathcal{F}^A$ and $\mathcal{F}^B$ that are used to create respective lynx-A and lynx-B constructions.
We provide the security proofs of lynx in the information-theoretic model, where each tweakable blockcipher is replaced by random tweakable permutation i.e., for each tweak a permutation is randomly selected. Further, lemmas, definitions and security proofs are provided for each of the respective member of the lynx family and we do not mix these amongst the members.
In this section, we start by providing **lemma 2**, **lemma 3**. Then we present two types of collision

event **h-Coll** and **input-Coll** in **definition 15** and **definition 17**. On the basis of these lemmas and definitions we give the bound of the two collision events (h-Coll and input-Coll) in **lemma 4**, **lemma 5** and **lemma 7**. These bounds are used to provide the security proofs of lynx i.e., INT-RUP, INT and CONF security as presented in **theorem 1**, **theorem 2**, **theorem 3** and **theorem 4**. First, we start by presenting these lemmas related to lynx-A and lynx-B. We then move forward to present the security proofs of the lynx family. At this point, it is also important to highlight that the adversary for CONF security proof is a nonce respecting adversary while in case of INT-RUP and INT security proof, the adversary maybe a nonce misuse adversary.

Adversary $\mathcal{A}$ executes the encryption queries, decryption queries and verification queries and builds tuples with each tuple containing values (N, A, M, C, T), where N is nonce, A is associated data, M is message, C is ciphertext and T is the tag generated. $h[i-1]$ denotes the input while $h[i]$ denotes the output for the i^{th} invocation of $\mathcal{F}^A$ (for lynx-A) or $\mathcal{F}^B$ (for lynx-B) respectively. $x[i]$ and $\mathcal{T}[i]$ denotes the two inputs (message and tweak) to the tweakable blockcipher of $\mathcal{F}^A$ (for lynx-A) or $\mathcal{F}^B$ (for lynx-B) for the i^{th} invocation of $\mathcal{F}^A$ and $\mathcal{F}^B$ respectively.

Lemma 2. *In lynx-A$\{1 \dots 8\}$ and lynx-B$\{1 \dots 4\}$, given the two tuples (N, A, M, C, T) and (N', A', M', C', T'), and for any i if $h[i-1] \neq h'[i-1]$ then $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$, where next invocation is defined for i.*

Proof. We want to show that if $h[i-1] \neq h'[i-1]$, then the respective inputs to the tweakable blockcipher ($x[i]$ and $\mathcal{T}[i]$) is also different. Hence, it is suffice to show that for an i either $x[i] \neq x'[i]$ or $\mathcal{T}[i] \neq \mathcal{T}'[i]$ or both $x[i] \neq x'[i]$ and $\mathcal{T}[i] \neq \mathcal{T}'[i]$. Equation (3.7) describes the inputs to the tweakable blockcipher

$$\begin{aligned} z[i] &= \widetilde{E}(K, \mathcal{T}[i], x[i]) \\ x[i] &= \widetilde{D}(K, \mathcal{T}[i], z[i]) \\ \mathcal{T}[i] &= bin(i) \parallel l_{7\dots0} \parallel y[i] \\ y[i] &= \{h[i-1], M[i], h[i-1] \oplus M[i]\} \end{aligned} \tag{3.7}$$

There are two inputs $x[i]$ and $\mathcal{T}[i]$ to the tweakable blockcipher. Based on these two inputs and the construction of function $\mathcal{F}^A$ and $\mathcal{F}^B$, the proof can be divided into four separate sections.

We use two cases $C[i] = C'[i]$ and $C[i] \neq C'[i]$ to prove the given lemma.

Lynx-A$\{1..4\}$: The first four members lynx-A$\{1..4\}$ have same input to the tweakable blockcipher and hence their analysis is similar. The internal construction of the four members are given below. It is clear from the construction that $x[i]$ and $y[i]$ is same for lynx-A1, lynx-A2, lynx-A3 and lynx-A4.

Lynx-A1

- $\mathcal{F}_{25}^{A} : x[i] = h[i-1] \oplus M[i], y[i] = M[i], o[i] = z[i]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{25}^{A}} : M[i] = h[i-1] \oplus x[i], y[i] = M[i], o[i] = z[i]$

Lynx-A2

- $\mathcal{F}_{26}^{A} : x[i] = h[i-1] \oplus M[i], y[i] = M[i], o[i] = z[i] \oplus M[i]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{26}^{A}} : M[i] = h[i-1] \oplus x[i], y[i] = M[i], o[i] = z[i] \oplus M[i]$

Lynx-A3

- $\mathcal{F}_{27}^{A} : x[i] = h[i-1] \oplus M[i], y[i] = M[i], o[i] = z[i] \oplus h[i-1]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{27}^{A}} : M[i] = h[i-1] \oplus x[i], y[i] = M[i], o[i] = z[i] \oplus h[i-1]$

Lynx-A4

- $\mathcal{F}_{28}^{A} : x[i] = h[i-1] \oplus M[i], y[i] = M[i], o[i] = z[i] \oplus M[i] \oplus h[i-1]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{28}^{A}} : M[i] = h[i-1] \oplus x[i], y[i] = M[i], o[i] = z[i] \oplus M[i] \oplus h[i-1]$

- **Case a:** $C[i] = C'[i]$

  Since $x[i] = C[i], \Rightarrow (x[i] = x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$. Hence, $M[i] \neq M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel M[i]) \wedge (\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel M'[i])$, hence $\mathcal{T}[i] \neq \mathcal{T}'[i]$

  Hence, $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

- **Case b:** $C[i] \neq C'[i]$
  - $C[i] \oplus h[i-1] = C'[i] \oplus h'[i-1]$

    Since, $(x[i] = C[i]) \Rightarrow (x[i] \neq x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$, Hence, $M[i] = M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel M[i])$ and $(\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel M'[i])$, hence $\mathcal{T}[i] = \mathcal{T}'[i]$.

    $\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$
  - $C[i] \oplus h[i-1] \neq C'[i] \oplus h'[i-1]$

    Since, $(x[i] = C[i]) \Rightarrow (x[i] \neq x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$, Hence, $M[i] \neq M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel M[i])$ and $(\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel M'[i])$, hence $\mathcal{T}[i] \neq \mathcal{T}'[i]$.

    $\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

Lynx-A$\{5..8\}$: Lynx-A$\{5..8\}$ have same input to the tweakable blockcipher and hence their analysis is similar. The internal construction of these members are given below where it is clear that $x[i]$ and $y[i]$ is same for lynx-A5, lynx-A6, lynx-A7 and lynx-A8.

Lynx-A5

- $\mathcal{F}_{29}^{A} : x[i] = h[i-1] \oplus M[i]$, $y[i] = h[i-1]$, $o[i] = z[i]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{29}^{A}} : M[i] = h[i-1] \oplus x[i]$, $y[i] = h[i-1]$, $o[i] = z[i]$

Lynx-A6

- $\mathcal{F}_{30}^{A} : x[i] = h[i-1] \oplus M[i]$, $y[i] = h[i-1]$, $o[i] = z[i] \oplus M[i]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{30}^{A}} : M[i] = h[i-1] \oplus x[i]$, $y[i] = h[i-1]$, $o[i] = z[i] \oplus M[i]$

Lynx-A7

- $\mathcal{F}_{31}^{A} : x[i] = h[i-1] \oplus M[i]$, $y[i] = h[i-1]$, $o[i] = z[i] \oplus h[i-1]$, where $x[i] = C[i]$
- $\overline{\mathcal{F}_{31}^{A}} : M[i] = h[i-1] \oplus x[i]$, $y[i] = h[i-1]$, $o[i] = z[i] \oplus h[i-1]$

Lynx-A8

➥ $\mathcal{F}_{32}^{A} : x[i] = h[i-1] \oplus M[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$, where $x[i] = C[i]$

➥ $\overline{\mathcal{F}_{32}^{A}} : M[i] = h[i-1] \oplus x[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$

- **Case a:** $C[i] = C'[i]$

  Since $x[i] = C[i]$, hence $(x[i] = x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$

  Hence, $M[i] \neq M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel h[i-1])$ and $(\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel h'[i-1])$, hence $\mathcal{T}[i] \neq \mathcal{T}'[i]$.

  Hence, $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

- **Case b:** $C[i] \neq C'[i]$

  – $C[i] \oplus h[i-1] = C'[i] \oplus h'[i-1]$

  Since, $(x[i] = C[i]) \Rightarrow (x[i] \neq x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$, Hence, $M[i] = M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel h[i-1])$ and $(\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel h'[i-1])$, hence $\mathcal{T}[i] \neq \mathcal{T}'[i]$.

  $\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

  – $C[i] \oplus h[i-1] \neq C'[i] \oplus h'[i-1]$

  Since, $(x[i] = C[i]) \Rightarrow (x[i] \neq x'[i])$. In case of lynx-A: $M[i] = C[i] \oplus h[i-1]$ and $M'[i] = C'[i] \oplus h'[i-1]$, Hence, $M[i] \neq M'[i]$. The tweaks are given by: $(\mathcal{T}[i] = bin(i) \parallel l_{7...0} \parallel h[i-1])$ and $(\mathcal{T}'[i] = bin(i) \parallel l_{7...0} \parallel h'[i-1])$, hence $\mathcal{T}[i] \neq \mathcal{T}'[i]$.

  $\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

Lynx-B$\{1...4\}$: The lynx-B member uses the decryption function of tweakable blockcipher during the decryption sub-routine. Hence, in this section we try to show the diference in the

input for the decryption function. Lynx-B$\{1, 3\}$ have same input to the tweakable blockcipher and hence their analysis is similar. The internal construction of these members are given below showing that $z[i]$ and $y[i]$ is same for lynx-B1, lynx-B2, lynx-B3 and lynx-B4.

Lynx-B1

- $\mathcal{F}_{18}^{B} : x[i] = M[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i]$, where $z[i] = C[i]$
- $\overline{\mathcal{F}_{18}^{B}} : M[i] = x[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i]$

Lynx-B2

- $\mathcal{F}_{20}^{B} : x[i] = M[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$, where $z[i] = C[i]$
- $\overline{\mathcal{F}_{20}^{B}} : M[i] = x[i], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$

Lynx-B3

- $\mathcal{F}_{30}^{B} : x[i] = M[i] \oplus h[i-1], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$, where $z[i] = C[i]$
- $\overline{\mathcal{F}_{30}^{B}} : M[i] = x[i] \oplus h[i-1], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$

Lynx-B4

- $\mathcal{F}_{32}^{B} : x[i] = M[i] \oplus h[i-1], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$, where $z[i] = C[i]$
- $\overline{\mathcal{F}_{32}^{B}} : M[i] = x[i] \oplus h[i-1], y[i] = h[i-1], o[i] = z[i] \oplus M[i] \oplus h[i-1]$

Since, $h[i-1] \neq h'[i-1] \Rightarrow \mathcal{T}[i] \neq \mathcal{T}'[i]$.
$\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$

In case of lynx-B1 and lynx-B2, $x[i] = M[i]$ while in case of lynx-B3 and lynx-B4, $x[i] = h[i-1] \oplus M[i]$. Hence, for lynx-B as well $x[i] \neq x'[i]$.
$\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$.

□

Lemma 3. *We present below two lemmas for* Lynx-A9 *and* Lynx-A10.

3.1 In lynx-A9 *and* lynx-A10*, given any two tuples* (N, A, M, C, T) *and* (N', A', M', C', T') *and for any* i*, if* $(h[i-1] = h'[i-1]) \wedge (M[i] \neq M'[i])$ *then* $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$.

Proof. Similar to **lemma 2** (refer **lemma** 2), we want to show that the inputs to the tweakable blockcipher is different if $(h[i-1] = h'[i-1]) \wedge (M[i] \neq M'[1])$. In case of lynx-A, one can check that $x[i] = h[i-1] \oplus M[i]$, but from the lemma we know that $h[i-1] \neq h'[i-1]$ and $M[i] = M'[i]$. Hence, for lynx-A $x[i] \neq x'[i]$.
$\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. □

3.2 In lynx-A9 *and* lynx-A10*, given any two tuples* (N, A, M, C, T) *and* (N', A', M', C', T') *and for any* i*, if* $(h[i-1] \neq h'[i-1]) \wedge (M[i] = M'[i])$ *then* $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$.

Proof. The proof is similar to **lemma 3.1** (refer **lemma** 3.1). In case of lynx-A, $x[i] = h[i-1] \oplus M[i]$. Since $h[i-1] \neq h'[i-1]$ then $x[i] \neq x'[i]$. In case of lynx-B, $y[i] = h[i-1]$ and $\mathcal{T}[i] = bin(i) \parallel l_{7\ldots0} \parallel y[i]$ but since $h[i-1] \neq h'[i-1]$ then $\mathcal{T}[i] \neq \mathcal{T}'[i]$
$\Rightarrow (x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. □

Based on **lemma** 2 and **lemma** 3, we now define three collision events **h-Coll**, **h*-Coll** and **input-Coll**. Let (N, A, M, C, T) and (N', A', M', C', T') be any two tuples obtained after executing encryption, decryption and verification queries. Using these two tuples, we have following definitions:

Definition 15. ***[h-Coll]*** *If there exists an* i*, such that* $h[i] = h'[i]$ *when* $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$*, then the h-Coll event has occurred for the two tuples* (N, A, M, C, T) *and* (N', A', M', C', T')

Definition 16. ***[h*-Coll]*** *If there exists an* i*, such that* $h[i-1] \neq h'[i-1]$ *but* $h[i] = h'[i]$*, then the h*-Coll event has occurred for the two tuples* (N, A, M, C, T) *and* (N', A', M', C', T')

Definition 17. ***[input-Coll]*** *If there exists an* i*, such that* $h[i-1] \neq h'[i-1]$ *but* $(x[i], \mathcal{T}[i]) = (x'[i], \mathcal{T}'[i])$*, then the the input-Coll event has occurred for the two tuples* (N, A, M, C, T) *and* (N', A', M', C', T')

Before moving forward to the probability bounds of h-Coll, h*-Coll and input-Coll, we first need to describe how the encryption query, decryption query and verification query looks like. Let $q_{\mathcal{E}}$, $q_{\mathcal{D}}$ and $q_{\mathcal{V}}$ denote the number of encryption queries, decryption queries and verification queries respectively. Let the tuple (N_i, A_i, M_i) denote the i^{th} encryption query, the tuple (N_i, A_i, C_i, T_i) denote the i^{th} decryption query and (N_i, A_i, C_i, T_i) denote the i^{th} verification query. The block length of a query for lynx-A is defined as the total number of invocations of $\mathcal{F}^A$ and the block length of a query for lynx-B is defined as the total number of invocations of $\mathcal{F}^B$ respectively. We can now present **lemma 4** and **lemma 5** that gives the probability bound of **h-Coll** for lynx-A$\{1 \ldots 8\}$ and lynx-B$\{1 \ldots 4\}$, and **lemma 5** that gives the probability bound of **h-Coll** for lynx-A9 and lynx-A10.

Lemma 4. *Let $\mathcal{A}$ be an adversary with $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number of decryption queries and $q_{\mathcal{V}}$ number of verification queries where maximum block length of any query is l, $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$ and $q \leq 2^{b-1}$ then for each lynx-A$\{1 \ldots 8\}$ and for each lynx-B$\{1 \ldots 4\}$ the $Pr[\text{h-Coll}^{\mathcal{A}}] \leq l \bullet \frac{q^2}{2^b}$, where h-Coll$^{\mathcal{A}}$ is the occurrence of h-Coll event when the adversary $\mathcal{A}$ is interacting with the lynx.*

Proof. We assume that the output from queries are compared up to same length (i.e., the counter must be same). We can have at most $\binom{q}{2}$ pairs of tuples $(N, A, M, C, \mathcal{T})$ and $(N', A', M', C', \mathcal{T}')$ from q possible queries. From **definition 15**, we know that for the two pairs $(h[i], x[i], \mathcal{T}[i])$ and $(h'[i], x'[i], \mathcal{T}'[i])$, if $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$, then $h[i] = h'[i]$ will occur, for a block with size b bits, with the probability $\frac{1}{2^b - q} \leq \frac{1}{2^{b-1}}$, since $q \leq 2^{b-1}$. Further, this probability must hold for all the blocks, hence we multiply this probability of each block by the maximum block length l, there by: $Pr[\text{h-Coll}^{\mathcal{A}}] \leq l \bullet \frac{1}{2^{b-1}} \bullet \binom{q}{2} \leq l \bullet \frac{q^2}{2^b}$. □

Lemma 5. *Let $\mathcal{A}$ be an adversary with $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number of decryption queries and $q_{\mathcal{V}}$ number of verification queries where maximum block length of any query is l, $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$ and $q \leq 2^{b-1}$ then for lynx-A9 and lynx-A10 the $Pr[\text{h-Coll}^{\mathcal{A}}] \leq l \bullet \frac{q^2}{2^b}$, where h-Coll$^{\mathcal{A}}$ is the occurrence of h-Coll event when the adversary $\mathcal{A}$ is interacting with the lynx.*

Proof. The lemma is closely related to the construction of lynx-A9 and lynx-A10. For the proof, we choose lynx-A9 since the proof of lynx-A10 is similar. Further, we fix V as message M and

provide h-Coll$^{\mathcal{A}}$ bound using the function $\mathcal{F}^A$. Further let K be a randomly chosen key. The proof has been adapted from [105]

For some i, we know $\mathcal{F}^A_{34}(K, h[i-1], S[i], M[i]) \mapsto (h[i], C[i])$. For the sake of proof, we omit $C[i]$. The adversary asks q queries to the function $\mathcal{F}^A$ and constructs a tuple set of following form: $\{(K, h_1[i-1], S_1[i], M_1[i], h_1[i]) \dots (K, h_q[i-1], S_q[i], M_q[i], h_q[i])\}$. The adversary $\mathcal{A}$ is successful if it can output two tuple $(K, h[i-1], S[i], M[i], h[i])$ and $(K, h'[i-1], S'[i], M'[i], h'[i])$ such that following condition holds:

- $((K, h[i-1], S[i], M[i], h[i]) \neq$
  $(K, h'[i-1], S'[i], M'[i], h'[i])) \wedge (h[i] = h'[i])$
  where $h[i] = \widetilde{E}(K, \mathcal{T}[i], M[i])$ and $h'[i] = \widetilde{E}(K, \mathcal{T}'[i], M'[i])$, $\mathcal{T}[i] = S[i] \parallel (M[i] \oplus h[i-1])$, $\mathcal{T}'[i] = S[i] \parallel (M'[i] \oplus h'[i-1])$

We show that the above condition is unlikely to occur.

Let h-Coll$^{\mathcal{A}}_j$ be the event such that $\widetilde{E}(K, \mathcal{T}_j[i], M_j[i]) \oplus M_j[i] = \widetilde{E}(K, \mathcal{T}_k[i], M_k[i]) \oplus M_k[i]$ or equivalently $h_j[i] = h_k[i]$ such that $j \neq k$. It can be seen that h-Coll$^{\mathcal{A}}_j$ to happen, $h_j[i]$ must selected from a set of size at least $2^b - (j-1)$, hence $Pr[\text{h-Coll}^{\mathcal{A}}_j] \leq \frac{j}{2^b - j}$. But $Pr[\text{h-Coll}^{\mathcal{A}}]$ $\leq Pr[\text{h-Coll}^{\mathcal{A}}_1 \vee \dots \vee \text{h-Coll}^{\mathcal{A}}_q] \leq \sum_{i=j}^{q} Pr[\text{h-Coll}^{\mathcal{A}}_j] \leq \sum_{j=1}^{q} \frac{j}{2^b - j} \leq \frac{q^2}{2^b}$ since $b \leq 2^{b-1}$. Since, the maximum block length is l, we multiply whole expression by l. Hence, $Pr[\text{h-Coll}^{\mathcal{A}}] \leq l \cdot \frac{q^2}{2^b}$. This completes our proof. □

Lemma 6. *Let $\mathcal{A}$ be an adversary with $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number of decryption queries and $q_{\mathcal{V}}$ number of verification queries where maximum block length of any query is l, $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$ and $q \leq 2^{b-1}$ then for* lynx*-A9 and* lynx*-A10 the $Pr[h^*\text{-}Coll^{\mathcal{A}}] \leq l \cdot \frac{q^2}{2^b}$, where h^*-Coll$^{\mathcal{A}}$ is the occurrence of h^*-Coll event when the adversary $\mathcal{A}$ is interacting with the* lynx.

Proof. There are two cases:

- $(h[i-1], h'[i-1])$ difference and $(M[i], M'[i])$ difference are equal. Clearly, in this case $h[i] \neq h'[i]$. Hence, $Pr[\text{h*-Coll}^{\mathcal{A}}] \leq l \cdot \frac{q^2}{2^b}$
- $(h[i-1], h'[i-1])$ difference and $(M[i], M'[i])$ difference are not equal. In this case, the proof is same as the **lemma 5** (refer **lemma** 5) i.e., $Pr[\text{h*-Coll}^{\mathcal{A}}] \leq l \cdot \frac{q^2}{2^b}$

□

Lemma 7. *Let $\mathcal{A}$ be a nonce respecting adversary with $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number of decryption queries and $q_{\mathcal{V}}$ number of verification queries where maximum block length of any query is l, $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$ and $q \leq 2^{b-1}$ then for* lynx*-A9 and* lynx*-A10 the $Pr[\textit{input-Coll}^{\mathcal{A}}] \leq l \bullet \frac{q^2}{2^b}$, where input-Coll$^{\mathcal{A}}$ is the occurrence of input-Coll event when the adversary $\mathcal{A}$ is interacting with the* lynx*.*

Proof. The proof for confidentiality is provided in nonce respecting scenario. Hence, the adversary $\mathcal{A}$ will get the h values only after all the ciphertext is generated. Clearly, the adversary $\mathcal{A}$ cannot change any message in between. We take the notation from **lemma 5** (refer **lemma** 5) and add a *xor* with the next message block to obtain the input collision: $\widetilde{E}(K, \mathcal{T}_j[i], M_j[i]) \oplus M_j[i] \oplus M_j[i+1] = \widetilde{E}(K, \mathcal{T}_k[i], M_k[i]) \oplus M_k[i] \oplus M_k[i+1]$. Hence, the proof now is exactly same as **lemma 5** (refer **lemma** 5) and hence we omit it for brevity. □

Now we present INT-RUP, INT and CONF security proofs for lynx.

3.6.1 Integrity in RUP

We start with the definition of INT-RUP from **definition** 4. Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. Let $\mathcal{A}$ be an adversary under nonce misuse scenario that makes $q_{\mathcal{E}}$ encryption queries, $q_{\mathcal{D}}$ decryption queries and $q_{\mathcal{V}}$ verification queries such that $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$. Then for a randomly chosen key K, the INT-RUP advantage of an adversary $\mathcal{A}$ against Π is given by:

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) = Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} forges]$$

Now present the INT-RUP bound for lynx-A$\{1 \ldots 8\}$ and lynx-B$\{1 \ldots 4\}$.

Theorem 1. *[**INT-RUP** in **Lynx-A**$\{1 \ldots 8\}$ and **Lynx-B**$\{1 \ldots 4\}$] Let Π represent any one of the twelve authenticated encryption schemes from* lynx*-A$\{1 \ldots 8\}$ and* lynx*-B$\{1 \ldots 4\}$ where underlying tweakable blockcipher is replaced by random tweakable permutation. Let $\mathcal{A}$ be an adversary under nonce misuse scenario that makes $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number*

of decryption queries and $q_{\mathcal{V}}$ number of verification queries to Π such that $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$. The number of blocks for each query is at most l. Then

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) \leq l \cdot \frac{q^2}{2^b} + \frac{q_{\mathcal{V}}}{2^{b-1}} \tag{3.8}$$

Proof. Let h-Coll denote the event as described in **definition 15**. Let $Pr[\text{h-Coll}]$ be the probability of the occurrence of h-Coll event while $Pr[\overline{\text{h-Coll}}]$ be the probability of no h-coll event. Then the INT-RUP advantage of an adversary $\mathcal{A}$ against Π can be described using h-Coll by the following generic expression of INT-RUP:

$$\begin{aligned} \text{INT-RUP}_{\Pi}(\mathcal{A}) &= Pr[(\mathcal{A}\ forges)] \\ &= Pr[\text{h-Coll} \wedge (\mathcal{A}\ forges)] \\ &\quad + Pr[\overline{\text{h-Coll}} \wedge (\mathcal{A}\ forges)] \end{aligned} \tag{3.9}$$

We know that $Pr[\text{h-Coll} \wedge (\mathcal{A}\ forges)] \leq Pr[\text{h-Coll}]$. From **lemma 4**, we know the that $Pr[\text{h-Coll}] \leq l \cdot \frac{q^2}{2^b}$ (please refer appendix for proof). Hence, the expression $Pr[\overline{\text{h-Coll}} \wedge (\mathcal{A}\ forges)]$ is now only left to evaluate from the equation (3.9), we next try to find its bound. Using the laws of probability, we can write $Pr[\overline{\text{h-Coll}} \wedge (\mathcal{A}\ forges)] = Pr[(\mathcal{A}\ forges) \mid \overline{\text{h-Coll}}] \cdot Pr[\overline{\text{h-Coll}}]$. But we know that $Pr[(\mathcal{A}\ forges) \mid \overline{\text{h-Coll}}] \cdot Pr[\overline{\text{h-Coll}}] \leq Pr[(\mathcal{A}\ forges) \mid \overline{\text{h-Coll}}]$. Hence, we now consider only no h-Coll scenario. Let (N, A, M, C, T) and (N', A', M', C', T') be two tuples obtained after executing encryption, decryption and verification queries respectively. Since, there is no h-Coll event, we have two cases. In the first case, one of the tuple is simply the prefix of the other. Since, we are using counter as the part of the tweak, the input to the tweakable blockcipher in the tag generation block is always different. In the second case, when the tuples are non-prefix, we assume that every new query has at least one bit difference. Also since we assumed that there is no h-Coll, hence, we know from **definition 15**, there exists i such that when the inputs to the tweakable blockcipher is different i.e., when $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$ then $h[i] \neq h'[i]$. Further, from **lemma 2**, we know that if $h[i] \neq h'[i]$ then if there exists next input, then the next input to the tweakable blockcipher is different .i.e, $(x[i+1], \mathcal{T}[i+1]) \neq (x'[i+1], \mathcal{T}'[i+1])$. In this way, we use **definition 15** and **lemma 2** as

a chain there by concluding that the inputs to the tweakable blockcipher in the tag generation phase is also different. This completes our two cases that if there is no h-Coll event, the inputs to the tag generation phase for both the cases are new. Based on above observation, we now calculate the probability bound of $Pr[(\mathcal{A}\ forges) \mid \overline{\text{h-Coll}}]$.

We begin with the assumption that out of the two tuples (N, A, M, C, T) and (N', A', M', C', T'), there is at least one bit difference between (N, A, M, C) and (N', A', M', C') since a difference only in the tags i.e., $T \neq T'$ when $(N, A, M, C) = (N', A', M', C')$ will always be rejected. Hence, as shown in the above paragraph, the inputs to the tag generation phase is always different. This implies that one output will be selected out of $\frac{1}{2^b - q}$ values, which means forgery can happen with probability $\leq \frac{1}{2^b - q}$ which can be written as $\frac{1}{2^{b-1}}$, since $q \leq 2^{b-1}$. Continuing in this way, $\mathcal{A}$ can perform $q_{\mathcal{V}}$ such trials. Hence, $Pr[(\mathcal{A}\ forges) \mid \overline{\text{h-Coll}}] \leq \frac{q_{\mathcal{V}}}{2^{b-1}}$. Using this result we can write the equation (3.9) in the following way:

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) \leq l \cdot \frac{q^2}{2^b} + \frac{q_{\mathcal{V}}}{2^{b-1}}$$

This completes the INT-RUP proof of lynx-A$\{1 \ldots 8\}$ and lynx-B$\{1 \ldots 4\}$. □

We now present the INT-RUP bounds of lynx-A9 and lynx-A10. Let Π represent any one of the two authenticated encryption schemes from lynx-A9 and lynx-A10.

Theorem 2. [***INT-RUP*** *in* ***Lynx-A***9 *and* ***Lynx-A***10] *Let* Π *represent any one of the two authenticated encryption schemes from* lynx-A9 *and* lynx-B10 *where underlying tweakable blockcipher is replaced by random tweakable permutation. Let* $\mathcal{A}$ *be an adversary that makes* $q_{\mathcal{E}}$ *number of encryption queries,* $q_{\mathcal{D}}$ *number of decryption queries and* $q_{\mathcal{V}}$ *number of verification queries to* Π *such that* $q = q_{\mathcal{E}} + q_{\mathcal{D}} + q_{\mathcal{V}}$. *The number of blocks for each query is at most* l. *Then*

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) \leq l \cdot \frac{q^2}{2^b} + \frac{q_{\mathcal{V}}}{2^{b-1}} \tag{3.10}$$

Proof. We know from **lemma** 5 that for lynx-A9 and lynx-A10, the $Pr[\text{h-Coll}] \leq l \cdot \frac{q^2}{2^b}$. Now, in case of no h-Coll event, we consider the two tuples (N, A, M, C, T) and (N', A', M', C', T') obtained after executing encryption, decryption and verification queries respectively. Similar

to the **theorem** 1, we show that the inputs to the tweakable blockcipher in the tag generation phase is different. Further, as in **theorem** 1, we know that, if one of the tuple is a prefix of the other, then due to the usage of counter as the part of the tweak, the inputs to the tweakable blockcipher in the tag generation phase is different. Now, we handle the cases when the tuples (N, A, M, C, T) and (N', A', M', C', T') are not prefixes.

In the first case, from **lemma** 3 (specifically 3.1), we know that for the two tuples if $(h[i-1] = h'[i-1]) \wedge (M[i] \neq M'[i])$, then the inputs to the tweakable blockcipher is different i.e., $(x[i], \mathcal{T}[i]) \neq (x[i]', \mathcal{T}'[i])$. In case of no h-Coll event, we know from **definition 15**, that when inputs to the tweakable blockcipher is different i.e., when $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$ then $h[i] \neq h'[i]$. In the second case, if $(h[i-1] \neq h'[i-1]) \wedge (M[i] = M'[i])$, then using **lemma** 3 (specifically 3.2), we know that the inputs to the tweakable blockcipher i.e., $(x[i], \mathcal{T}[i])$ and $(x[i]', \mathcal{T}'[i])$ are different and hence we can use the **definition** 15 in the similar way as in the previous case to infer that $h[i] \neq h'[i]$. In the third and the last case, for the two tuples (N, A, M, C, T) and (N', A', M', C', T'), when $(h[i-1] \neq h'[i-1]) \wedge (M[i] \neq M'[i])$, then due to the construction of $\mathcal{F}_A^{34}$ and $\mathcal{F}_A^{35}$, it is possible that the difference of $(h[i-1], h'[i-1])$ and $(M[i], M'[i])$ can cancel out each other and hence the inputs to the tweakable blockcipher becomes the same i.e., $(x[i], \mathcal{T}[i]) = (x'[i], \mathcal{T}'[i])$. But we know from **definition** 16, that if $h[i-1] \neq h'[i-1]$ then $h[i] \neq h'[i]$, hence, this case is bounded by h^*-Coll of **lemma** 6 i.e. $Pr[\text{h*-Coll}] \leq l \cdot \frac{q^2}{2^b}$. Proceeding in this way, using the above three cases, we conclude that the inpus to the tweakable blockcipher in the tag generation phase is different i.e., $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. We begin with the first trial or the first verification query by the adversary $\mathcal{A}$, and then continue as in **theorem** 1. The rest of the proof is very similar to **theorem 1** and omitted for brevity. Using it we can write the INT-RUP proof of lynx-A9 and lynx-A10 in the following way:

$$\text{INT-RUP}_{\Pi}(\mathcal{A}) \leq l \cdot \frac{q^2}{2^b} + \frac{q\nu}{2^{b-1}}$$

□

3.6.2 Integrity

We start with the definition of INT-RUP from **definition 2**. Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. Let $\mathcal{A}$ be an adversary under nonce misuse scenario that makes $q_{\mathcal{E}}$ encryption queries and $q_{\mathcal{V}}$ verification queries such that $q = q_{\mathcal{E}} + q_{\mathcal{V}}$. Then for a randomly chosen key K, the INT advantage of the adversary $\mathcal{A}$ against Π is given by:

$$\text{INT}_{\Pi}(\mathcal{A}) = Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} forges]$$

Let Π represent any one of the fourteen authenticated encryption schemes from lynx-A$\{1 \ldots 10\}$ and lynx-B$\{1 \ldots 4\}$. We know that $\text{INT}_{\Pi}(\mathcal{A}) \leq \text{INT-RUP}_{\Pi}(\mathcal{A})$. Since the bound are similar to the previous proof, we omit the proof for brevity.

3.6.3 Confidentiality

We start with the definition of CONF from **definition 1**. Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. Let $\mathcal{A}$ be an adversary under nonce respecting scenario. Then for a randomly chosen key K, the CONF advantage of an adversary $\mathcal{A}$ against Π is given by:

$$\text{CONF}_{\Pi}(\mathcal{A}) = Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1] - Pr[\mathcal{A}^{\$} \rightarrow 1]$$

Let Π represent any one of the twelve authenticated encryption schemes from lynx-A$\{1 \ldots 8\}$ and lynx-B$\{1 \ldots 4\}$, then we can show the CONF security of lynx-A$\{1 \ldots 8\}$ and lynx-B$\{1 \ldots 4\}$ in **theorem 3**.

Theorem 3. [***Confidentiality*** *in* ***Lynx-A***$\{1 \ldots 8\}$ *and* ***Lynx-B***$\{1 \ldots 4\}$] *Let* Π *represent any one of the twelve authenticated encryption schemes from lynx-A*$\{1 \ldots 8\}$ *and lynx-B*$\{1 \ldots 4\}$ *where underlying tweakable blockcipher is replaced by random tweakable permutation. Let* $\mathcal{A}$ *be a nonce respecting adversary that makes a total of* q *queries to the encryption sub-routine of* Π. *Then*

$$\text{CONF}_{\Pi}(\mathcal{A}) \leq 2 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}} \tag{3.11}$$

Proof. For the CONF proof of Π, we replace tweakable blockcipher by a tweakable random permutation i.e., a random permutation with different key and tweak. In case of real world, let h-Coll$_{\mathcal{R}eal}$ denote the h-Coll event (from **definition 15**) induced by the encryption sub-routine of Π i.e., for any i and pairs $(h[i], x[i], \mathcal{T}[i])$ and $(h'[i], x'[i], \mathcal{T}'[i])$, h-Coll$_{\mathcal{R}eal}$ occurs if $h[i] = h'[i]$ when $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. From **lemma 4**, we know that: $Pr[\text{h-Coll}_{\mathcal{R}eal}] \leq l \cdot \frac{q^2}{2^{b+1}}$
In case of ideal world, from the encryption sub-routine of Π we get the ciphertext and tag as the output. Using the ciphertext and the input message one can get the h values in the ideal world setup. All of these h values are random (since $\mathcal{A}$ is nonce respecting). Hence, the probability of h-Coll$_{\mathcal{I}deal}$ i.e., for any i and pairs $(h[i], x[i], \mathcal{T}[i])$ and $(h'[i], x'[i], \mathcal{T}'[i])$, h-Coll$_{\mathcal{I}deal}$ occurs if $h[i] = h'[i]$ when $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. Since, the h values are always random, $h[i]$ and $h'[i]$ would be equal with a probability $\frac{1}{2^b}$ and thereby taking the maximum block length into account, the $Pr[\text{h-Coll}_{\mathcal{I}deal}] \leq l \cdot \frac{q^2}{2^b}$.
Then we have: $Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1] = Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{R}eal}] + Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{R}eal}]$ and $Pr[\mathcal{A}^{\$} \rightarrow 1] = Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{I}deal}] + Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{I}deal}]$. Hence, we can write the CONF advantage of a nonce respecting adversary $\mathcal{A}$ for Π in the following way:

$$\begin{aligned}
&Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1] - Pr[\mathcal{A}^{\$} \rightarrow 1] \leq \\
&\quad (Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{R}eal}] \\
&\qquad + Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{R}eal}]) \\
&\quad - (Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{I}deal}] \\
&\qquad + Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{I}deal}]) \\
&\quad + l \cdot \frac{\binom{q}{2}}{2^b}
\end{aligned} \tag{3.12}$$

where we replace $\mathcal{E}_K$ with F_K and hence, $l \cdot \frac{\binom{q}{2}}{2^b}$ comes from random permutation-random function (RP-RF) switching ([106]) i.e., we replace tweakable random permutation with tweakable random function. In case of lynx-B, we require decryption function of the tweakable blockcipher for

the decryption sub-routine of lynx-B, however, for the confidentiality proof, we require only the encryption sub-routine, and hence, we can safely use RP-RF switching for lynx-B as well. Further, we multiply $\frac{\binom{q}{2}}{2^b}$ by the maximum block length i.e., l, since due to the construction of Π, if the block number is different, then, due to the use of counter, the tweak also becomes different for each block.

Due to the construction of Π, we make following claim:

- $Pr[\mathcal{A}^{F_K} \rightarrow 1 \mid \overline{\text{h-Coll}}_{\mathcal{R}eal}] = Pr[\mathcal{A}^{\$} \rightarrow 1 \mid \overline{\text{h-Coll}}_{\mathcal{I}deal}]$

Proof. In case of confidentiality security, we know that the adversary $\mathcal{A}$ is nonce respecting. Since, for given two tuples (N, A, M, C, T) and (N', A', M', C', T'), we have $N \neq N'$, then for lynx-A, $h[0] \neq h'[0]$ and for lynx-B $N \neq N'$. From **lemma** 2, we know that if $h[i-1] \neq h'[i-1]$ then the inputs to the tweakable blockcipher are different i.e., $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$. But from **definition** 15, we know that if $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$, then the output from $\mathcal{F}^A$ (for lynx-A) and $\mathcal{F}^B$ (for lynx-B) are different .i.e., $h[i] \neq h'[i]$. Hence, using **lemma** 2 and **definition** 15 as a chain, we can conclude that all the h values are different which leads to the scenario that the input to the tag generation phase is also different, thereby making the output (tag) of the tag generation phase also different. This leads to the fact that the output distribution from Π is as same the output distribution by $\$$ (section 2.2). This completes the proof of our claim. □

We apply the above results in equation (3.12) and also use the results from *Chang et.al.* [107] to solve the CONF advantage of the adversary.

$$\begin{aligned}
&(Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{R}eal}] \\
&\qquad + Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{R}eal}]) \\
&- (Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{I}deal}] \\
&\qquad + Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{I}deal}]) \\
&\leq 2 \cdot max\{Pr[\text{h-Coll}_{\mathcal{R}eal}], Pr[\text{h-Coll}_{\mathcal{I}deal}]\}
\end{aligned} \tag{3.13}$$

We know from **lemma 4** that $Pr[\text{h-Coll}_{\mathcal{R}eal}] = l \cdot \frac{q^2}{2^b}$ while $Pr[\text{h-Coll}_{\mathcal{I}deal}] = l \cdot \frac{q^2}{2^{b+1}}$. Hence,

$$\begin{aligned}\text{CONF}_{\Pi}(\mathcal{A}) &\leq 2 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{\binom{q}{2}}{2^b} \\ &\leq 2 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}}\end{aligned}$$

□

We now present the CONF bound of lynx-A9 and lynx-A10.

Theorem 4. [***Confidentiality*** *in* ***Lynx-A***9 *and* ***Lynx-A***10] *Let* Π *represent any one of the two authenticated encryption schemes from lynx-A*9 *and lynx-A*10 *where underlying tweakable block-cipher is replaced by random tweakable permutation. Let* $\mathcal{A}$ *be a nonce respecting adversary that makes a total of* q *queries to the encryption sub-routine of* Π*. Then*

$$\text{CONF}_{\Pi}(\mathcal{A}) \leq 4 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}} \tag{3.14}$$

Proof. In case of lynx-A9 and lynx-A10, the CONF security is influenced by both the collision events, **h-Coll** (**definition** 15) as well as **input-Coll** (**definition** 17). As in **theorem** 3, we consider nonce respecting adversary, hence if the one bit difference in inputs to the tweakable blockcipher i.e., $(x[i], \mathcal{T}[i]) \neq (x'[i], \mathcal{T}'[i])$, then from **definition** 15, we know that $h[i] \neq h'[i]$. But from **definition** 17, we know that if $h[i] \neq h'[i]$ then $(x[i+1], \mathcal{T}[i+1]) \neq (x'[i+1], \mathcal{T}'[i+1])$. Hence, we can use these two definitions to show that the input to the tweakable blockcipher in the tag generation phase is different. Hence, using these observations, we can write equation (3.12)

using h-Coll and input-Coll as follows:

$$
\begin{aligned}
Pr[\mathcal{A}^{\mathcal{E}_K} \rightarrow 1] - Pr[\mathcal{A}^{\$} \rightarrow 1] = \\
(Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{R}eal}] \\
+ Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{R}eal}]) \\
- (Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \text{h-Coll}_{\mathcal{I}deal}] \\
+ Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \overline{\text{h-Coll}}_{\mathcal{I}deal}]) \\
+ (Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \text{input-Coll}_{\mathcal{R}eal}] \\
+ Pr[\mathcal{A}^{F_K} \rightarrow 1 \wedge \overline{\text{input-Coll}}_{\mathcal{R}eal}]) \\
- (Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \text{input-Coll}_{\mathcal{I}deal}] \\
+ Pr[\mathcal{A}^{\$} \rightarrow 1 \wedge \overline{\text{input-Coll}}_{\mathcal{I}deal}]) \\
+ l \cdot \frac{\binom{q}{2}}{2^b}
\end{aligned}
\tag{3.15}
$$

The solution of equation (3.15) is similar to the results of **theorem 3** and hence

$$
\begin{aligned}
\text{CONF}_{\Pi}(\mathcal{A}) &\leq 4 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{\binom{q}{2}}{2^b} \\
&\leq 4 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}}
\end{aligned}
$$

□

3.7 Experiments and Performance Analysis

In this section, we present the implementation of lynx[2], specifically lynx-A1, since all other members of the lynx family have similar experimental results. We perform the experiments on wide range of hardware, spanning from desktop grade machine to low powered IoT devices. In all, we have six different implementations of lynx-A1. We compare each of these six implementations of lynx-A1, with the best available implementations of Romulus-N1 [9] for each of these six platforms. We want to point out that the underlying primitive of lynx-A1 is same as that of

[2] https://github.com/mhasan08/lynx

Table 3.8: Software performance comparison between Lynx-A1 and Romulus-N1 in Python language

Message Size (Bytes)	**Lynx-A1** (secs)		**Romulus-N1** (secs)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
0	0.000 917	0.001 001	0.001 574	0.001 453
64	0.001 301	0.001 104	0.003 692	0.003 828
512	0.011 231	0.013 132	0.024 530	0.023 575
1024	0.031 817	0.029 991	0.046 911	0.045 18
4096	0.099 909	0.101 907	0.178 141	0.176 999
8192	0.291 951	0.299 438	0.356 414	0.348 989
16384	0.691 117	0.700 001	0.707 041	0.702 977

Romulus-N1 i.e., SKINNY tweakable blockcipher [67] with 128 bit key and 256 bit tweak, hence we chose Romulus-N1 for the comparision.

The implementation results and comparisons are presented in table 3.8 through table 3.13. For table 3.8 and table 3.9, we use Macbook Pro with 16 GB RAM and running a 2.6GHz i7 intel skylake processor. We use python and C language for comparison on this platform. In table 3.10, we present the comparison on arduino uno platfrom (8 bit microcontroller). Table 3.11 shows the comparison on arduino due i.e., 32 bit arm microcontroller. In case of table 3.12, we use a raspberry pi 3 model B that uses a 1.2GHz broadcom SoC. Though, this broadcom SoC is 64 bit, the Raspberry Pi OS [108] runs in 32 bit mode on the hardware. In table 3.13, we present the comparison on arm64 architecture. The arm64 architecture is very diverse, different OEM manufactures have varied fabrication of cores and varied number and size of transistors. We use Samsung Galaxy smartphone with Samsung's exynos 9820 [109] SoC. For profiling on this exynos SoC, we implemented an android app and created a JNI (Java Native Interface) module to build an interface between the Java code and the C implementation of lynx-A1 and Romulus-N1. The numbers reported in table 3.13 are only for the C code.

All the implementations were run 100 times, and an average was taken of these 100 runs and reported in the table 3.8 through table 3.13. It can be clearly seen from the tables (table 3.8 - table 3.13) that lynx-A1 outperforms Romulus-N1 in all the implementations. There is a gain of $1\% - 18\%$ approximately between the implementations of lynx-A1 and Romulus-N1. One of the reason for such an outcome is the straightforward structure of lynx. The design of

Table 3.9: Software performance comparison between Lynx-A1 and Romulus-N1 in C language

Message Size (Bytes)	**Lynx-A1** (C/B)		**Romulus-N1** (C/B)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
0	150	157	250	244
64	140	144	220	217
512	120	125	200	195
1024	110	114	170	165
4096	100	102	160	153
8192	96	101	140	133
16384	80	83	130	120

Table 3.10: AVR performance (Arduino Uno) comparison between Lynx-A1 and Romulus-N1

Message Size (Bytes)	**Lynx-A1** (millisecs)		**Romulus-N1** (millisecs)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
8	1.801	1.799	2.608	2.648
32	2.702	2.788	3.896	3.948
64	6.001	6.129	6.448	6.528
128	10.6	10.991	11.556	11.676
256	19.001	20.1	21.752	21.940
512	36.12	36.893	42.156	42.460

Table 3.11: 32 bit ARM performance (Arduino Due) comparison between Lynx-A1 and Romulus-N1

Message Size (Bytes)	**Lynx-A1** (millisecs)		**Romulus-N1** (millisecs)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
8	0.159	0.163	0.178	0.181
32	0.201	0.219	0.229	0.341
64	0.329	0.338	0.345	0.348
128	0.551	0.556	0.579	0.581
256	0.761	0.782	0.987	0.989
512	1.56	1.618	1.866	1.889

lynx doesn't have any matrix computations or field multiplications and only uses simple *xor* operations, while Romulus-N1 uses field multiplication. Due to such design choice of lynx, one can see in table 3.10, for an 8 bit microcontroller, the performance difference is apparent as the message size increases.

Note that, subject to evaluation, there are always claims of improved implementations of au-

Table 3.12: Raspberry Pi v3 (running in 32 bit mode) comparison between Lynx-A1 and Romulus-N1

Message Size (Bytes)	**Lynx-A1** (millisecs)		**Romulus-N1** (millisecs)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
0	0.144	0.165	0.178	0.181
64	0.199	0.209	0.229	0.341
512	0.291	0.3	0.345	0.348
1024	0.531	0.542	0.579	0.581
4096	0.786	0.799	0.987	0.989
8192	1.31	1.401	1.866	1.889
16384	3.302	3.396	3.506	3.619

Table 3.13: Arm 64 bit (Samsung Exynos 9820) comparison between Lynx-A1 and Romulus-N1

Message Size (Bytes)	**Lynx-A1** (millisecs)		**Romulus-N1** (millisecs)	
	Encrypt	**Decrypt**	**Encrypt**	**Decrypt**
0	0.011	0.12	0.012	0.012
64	0.021	0.023	0.028	0.029
512	0.141	0.143	0.158	0.161
1024	0.291	0.296	0.307	0.311
4096	0.441	0.497 1	0.520 1	0.520 11
8192	0.911	0.921 08	0.940 31	0.940 55
16384	1.463 05	1.491 03	1.782 11	1.882 301

thenticated encryption schemes and their underlying primitives. An improved implementation of underlying primitive i.e., the tweakable blockcipher (`SKINNY` in this case) will improve performance of both lynx-A1 as well as Romulus-N1 proportionally.

3.7.1 Comparative Performance of Lynx-A1

In this section, we present a comparative analysis of the encryption sub-routine and the decryption sub-routine of lynx-A1 on different platforms. For the purpose of comparison, we choose two pairs of hardware. The first pair: Arduino Uno (an 8-bit microcontroller) with Arduino Due (a 32-bit ARM SoC), while the second pair: raspberry pi v3 with the Samsung exynos 9820 SoC (a 64-bit ARM SoC). In figure 3.19, the performance of encryption sub-routine of lynx-A1 is compared on Arduino Uno and Arduino Due, while in figure 3.20, the performance of decryption

Figure 3.19: Performance of encryption sub-routine of lynx-A1 on 8-bit microcontroller (Arduino Uno) vs 32-bit ARM SoC (Arduino Due).

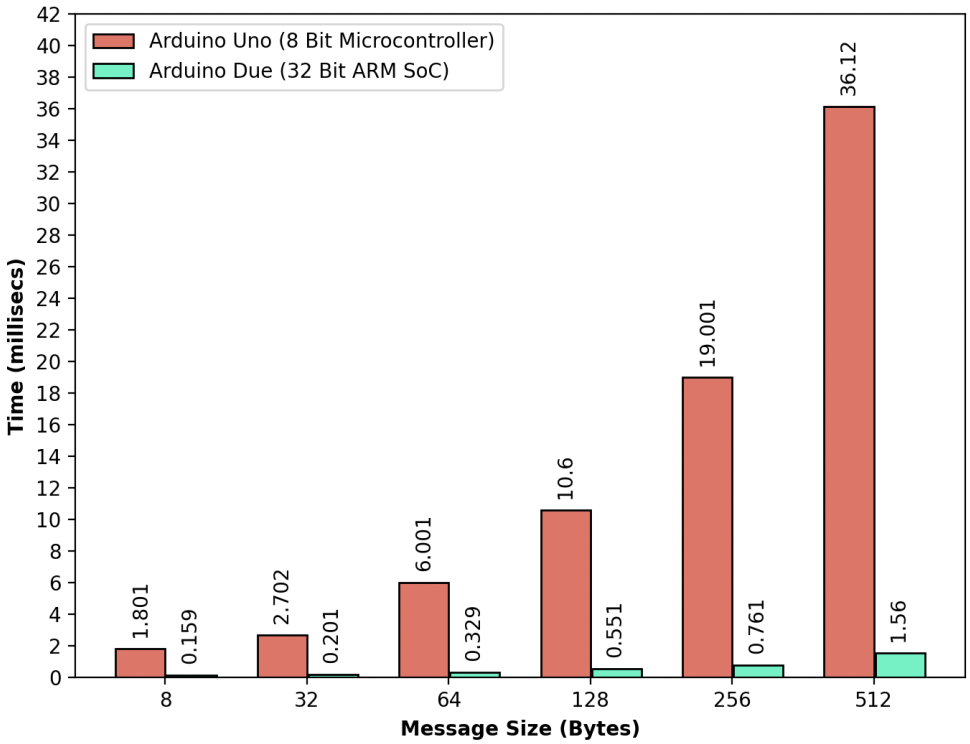


Figure 3.20: Performance of decryption sub-routine of lynx-A1 on 8-bit microcontroller (Arduino Uno) vs 32-bit ARM SoC (Arduino Due).

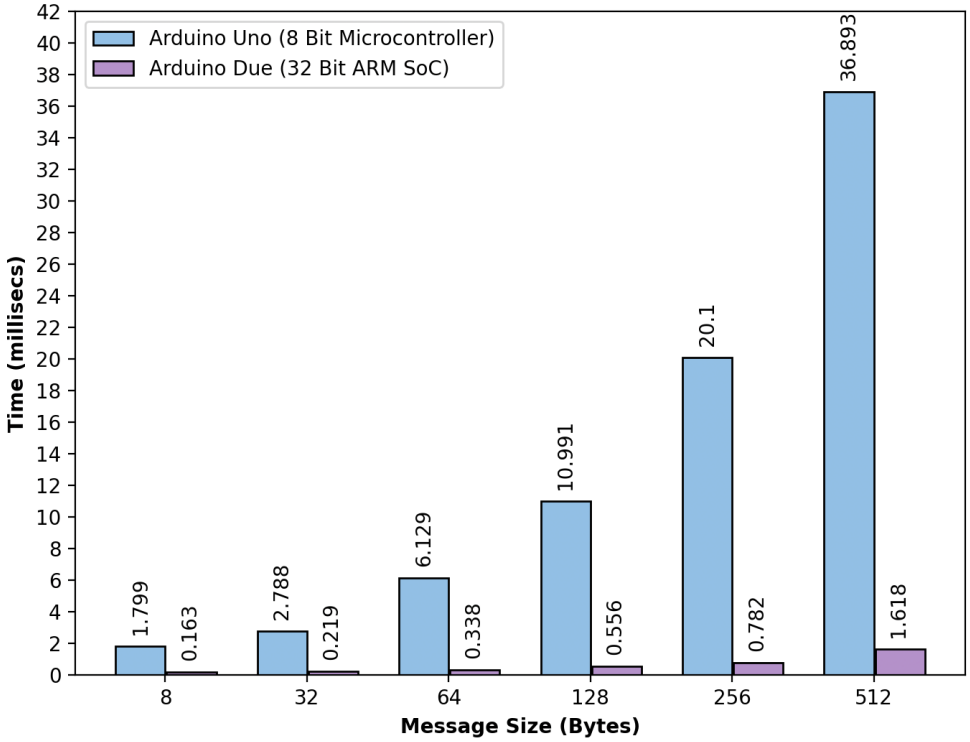


Figure 3.21: Performance of encryption sub-routine of lynx-A1 on Raspberry Pi v3 vs 64-bit ARM SoC (Samsung Exynos SoC).

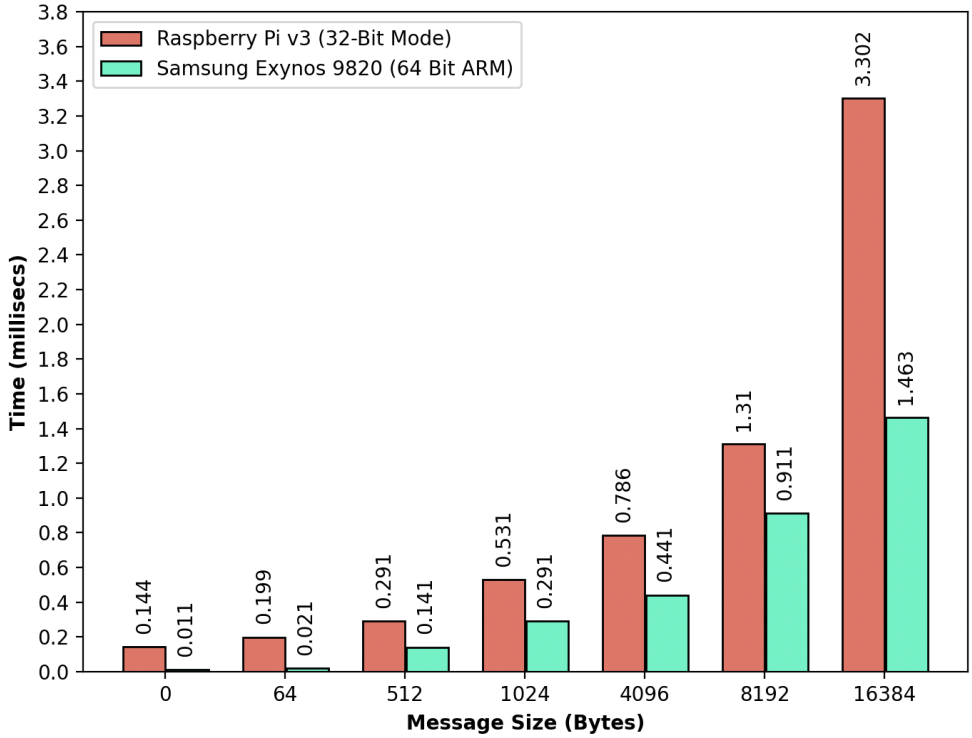
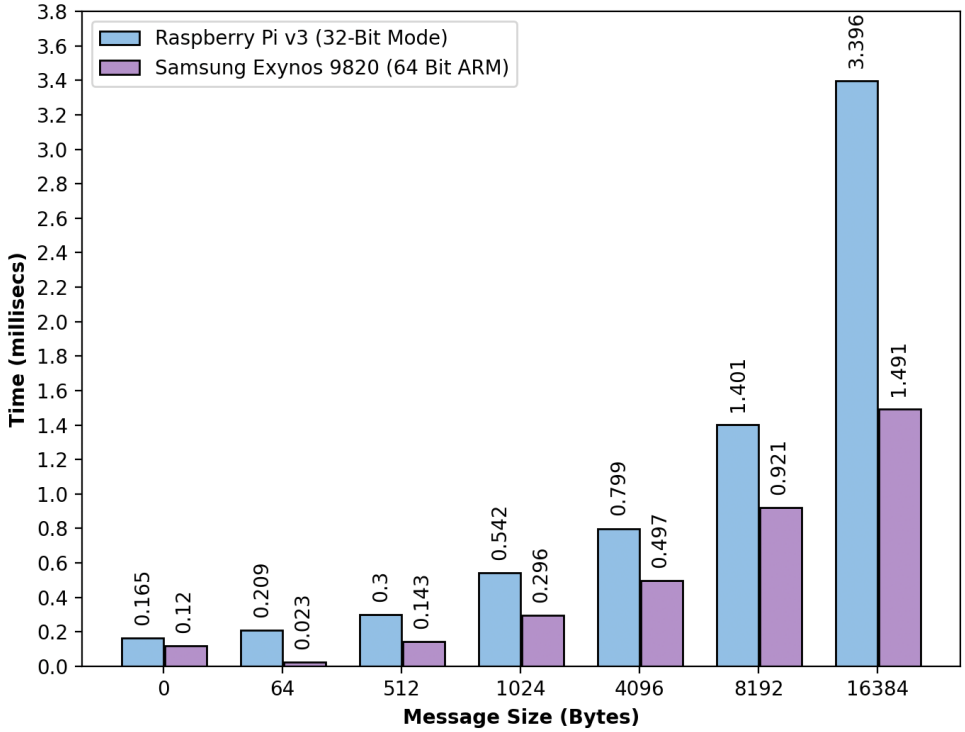


Figure 3.22: Performance of decryption sub-routine of lynx-A1 on Raspberry Pi v3 vs 64-bit ARM SoC (Samsung Exynos SoC).



sub-routine of lynx-A1 is compared for the two platforms. For both the sub-routines, we see that the implementation of Arduino Due comfortably outperforms the the on Arduino Uno. This is expected, since running the same algorithm on a 32-bit platform will be much faster than on an 8-bit platform. In figure 3.21 and figure 3.22, we present the comparison of the performance of the encryption sub-routine and the decryption sub-routine of lynx-A1 on raspberry pi v3 and Samsung exynos SoC. We see that the exynos SoC outperforms the raspberry pi v3 significantly. Since, raspberry pi v3 is shipped with a 64-bit quadcore broadcom SoC [110], one might think that why there so much difference between the two 64-bit implementations? The answer to this question has already been mentioned in the section 3.7, .i.e., raspberry pi v3 is running the operating system in a 32-bit mode. Therefore, though a 64-bit SoC, the installed kernel on raspberry pi v3 is not able to utilize that full power of the hardware. Hence, the implementation comparison shows a much better performance on a 64-bit exynos SoC.

3.8 Conclusion

In this chapter, we utilized one of the tweakable primitives .i.e., a tweakable blockcipher and proposed family of 1-pass and rate-1 lightweight authenticated encryption scheme, called lynx. The lynx family has two sub-families, namely, lynx-A and lynx-B. Lynx has several advantages in the lightweight category like *stream processing*, *computationally menial*, *simplistic operations* (*xor* operations) etc. Further, to create such an authenticated encryption scheme, we propose a family of functions denoted by $\mathcal{F}$. We present 72 cases of $\mathcal{F}$ and divide each of these $\mathcal{F}$ into incorrect and correct groups. For each $\mathcal{F}$ in the incorrect group, in light of creating an AEAD, we further categorize them into *implausible cases*, *non-confidential cases* and *non-integrity* cases. We show that only 14 cases of $\mathcal{F}$ can be used to create an authenticated encryption scheme (a.k.a lynx) using a tweakable blockcipher. We denote the cases of $\mathcal{F}$ that are used to create lynx-A as $\mathcal{F}^A$, and the cases that are used to create lynx-B as $\mathcal{F}^B$. We provided provable security proof for all the members of the lynx family. The integrity security of lynx under nonce respecting as well as under nonce misuse and RUP scenario is $\leq l \cdot \frac{q^2}{2^b} + \frac{q\nu}{2^{b-1}}$. In case of confidentiality (nonce respecting), the security bound of lynx-$A1 \ldots A8$ and lynx-$B1 \ldots B4$ is $\leq 2 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}}$ while

the security bound of lynx-A9 and lynx-A10 is $\leq 4 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}}$. From the implementation perspective, we see that due to the simplistic structure of lynx and lack of field multiplication, lynx-A1 outperforms Romulus-N1 on all the six platforms as shown in the experiments section.

Chapter 4

Context-Committing Authenticated Encryptions using Tweakable Stream Cipher

4.1 Introduction

A tweakable wide block cipher scheme [60] takes three inputs: a key, a tweak and a plaintext, and produces a ciphertext as the output, where size of the ciphertext is equal to size of the plaintext. The scheme is particularly useful in applications like disk encryption. The disk is partitioned into sectors, typically of size 512 bytes; in some newer hard disks and solid state devices using advance format, sector size of 4096 bytes are used. Each of these sectors have a permanent index that identifies them uniquely. Encrypting a disk requires encryption of each of these sectors. To enhance user experience and performance, preference might be given to an algorithm that considers encryption of a sector mutually exclusive with the encryption of other sectors of the disk. Further, when content of a sector changes, then the sector is encrypted again. HBSH [83], a tweakable wide block cipher scheme, emerged as one of the major work in this area. Adiantum (HBSH specification) is available in Android 9.0 and higher for the encryption [111]. HCTR2 [84] is another tweakable wide block cipher scheme, that is available

in Android 14.0 and higher for file encryption applications [112]. Double-decker and docked-double-decker [85] are two deck function (doubly-extendable cryptographic keyed function) based tweakable wide block cipher schemes targeting file encryption applications. In [113], the authors present two instances of docked-double-decker based on AES [17], and hence, it can make use of existing cryptographic hardware accelerators. Another target area of the tweakable wide block cipher scheme is the design space of IoT devices, such devices are computationally menial when compared to desktop or smartphone grade chips, and often lack cryptographic hardware instructions. Schemes like HBSH, double-decker and docked-double-decker are useful in such scenarios.

Encode-then-encipher [59] (EtE in short) paradigm can be used to construct an authenticated encryption scheme from a tweakable wide block cipher scheme by inserting zeros at a specified position. An authenticated encryption scheme provides both privacy (or confidentiality) and authenticity (or integrity) of data simultaneously. This dual functionality prevents unauthorized access to the encrypted data while also detecting any attempts at tampering or modification of the data. Due to this streamline and efficient process, authenticated encryption schemes are widely used for numerous security critical applications like including secure communication protocols, data storage, file encryption, blockchain and cryptocurrencies, cloud services etc. We target the notion of committing security of authenticated encryption scheme, which is inspired by real-world attacks and is an active area of research. Further, NIST proposal on the requirements for an accordion mode mentions key commitment and context commitment as the desirable properties [10]. AEZ [25], a CAESAR [35] submission, is an authenticated encryption scheme that uses zero appending technique together with nonce, associated data and length of the append (stretch) as the tweak, resulting into a ciphertext that is longer by the length of the append. During the third NIST workshop on block cipher modes of operation [114], key commitment security with time complexity $O(1)$ was presented on AEZ [115], and an overview of CMT-4 attack was also presented on HBSH and HCTR2 under encode-then-encipher paradigm. Facebook's message franking technique in their end-to-end encrypted messenger for generating cryptographically verifiable report for shared images and video, based on AES-GCM [18], was shown to be vulnerable to key commitment attack, due to non-key committing authenticated

encryption scheme [51, 116]. Facebook uses a hash of the AES-GCM ciphertext, along with a randomly generated value, as an identifier for the attachment. The attack in [51], finds two different keys and a ciphertext efficiently for a message, such that one of the key decrypts the ciphertext to an abusive attachment while the other key successfully decrypts the same ciphertext, but to another harmless attachment. The adversary, then sends two messages with different keys but the same attachment ciphertext, making Facebook's deduplicate algorithm to report only non-abusive attachment. This vulnerability was patched by making deduplicate algorithm more vigilant. In [52][1], the authors showed that a key commitment lacking AES-GCM based ciphertext can be decrypted into two plaintexts of different file formats such as PDF, Windows executable, DICOM etc.

In this chapter, we continue our work towards the tweakable primitives, and explore the idea of creating an authenticated encryption scheme using HBSH, HCTR2, double-decker and docked-double-decker under encode-then-encipher paradigm. For message encoding, we use zero prepending technique, and then for enciphering, we use the four tweakable wide block cipher schemes. Under this EtE setup, we evaluate CMT-4 security of each of these authenticated encryption schemes, and provide detailed CMT-4 attack with time complexity $O(1)$ for all the four schemes. In this work, we introduce a new tweakable primitive, called the tweakable stream cipher (or tS in short). We use tS to create four new tweakable wide block cipher schemes that are CMT-4 secure .i.e., all the four of them provide partial collision resistance against a CMT-4 adversary under encode-then-encipher paradigm.

4.1.1 Motivation

The idea of context commitment security starts with the sub-routine $\mathcal{E}$ committing to a key K. A CMT-1 adversary generates two distinct tuples (K_1, N_1, A_1, M_1) and (K_2, N_2, A_2, M_2) .i.e., $\tau = \{(K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2)\}$, such that $\mathcal{E}(K_1, N_1, A_1, M_1) = \mathcal{E}(K_2, N_2, A_2, M_2)$ but $K_1 \neq K_2$ .i.e., that two keys K_1 and K_2 are distinct. Since, $\mathcal{E}$ is committing to only one of the input parameters .i.e., the key, hence we use the notation as CMT-1. In a more generic sense, we can use the notation CMT-ℓ, where ℓ denotes the number of inputs of $\mathcal{E}$, to show

[1] https://github.com/kste/keycommitment

the commitment of the encryption sub-routine on the number of inputs it takes respectively. In case of CMT-4, the task of adversary is to generate $\tau = \{(K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2)\}$, such that $\mathcal{E}(K_1, N_1, A_1, M_1) = \mathcal{E}(K_2, N_2, A_2, M_2)$ but $(K_1, N_1, A_1, M_1) \neq (K_2, N_2, A_2, M_2)$. Intuitively, a CMT-4 secure $\mathcal{E}$ requires commitment to all of its inputs. Further, we can also see that CMT-4 $\implies$ CMT-1 [58]. CMT-4 security is a strong goal, since commitment is required for all the four inputs of the encryption sub-routine.

HBSH [83], HCTR2 [84] and the two deck based schemes .i.e., double-decker and docked-double-decker [85] target applications like file encryption. Further, HBSH, double-decker and docked-double-decker target the area of low powered devices where cryptographic hardware instructions are absent. Using EtE paradigm, one can convert these tweakable wide block cipher scheme to an authenticated encryption scheme. The authenticated encryption schemes so created must provide security assurance under various notions like committing security. An overview of CMT-4 attack was presented [115] for HBSH and HCTR2 under encode-then-encipher paradigm. We take inspiration from this overview of CMT-4 attack and present detailed CMT-4 analysis with time complexity $O(1)$, for all the four tweakable wide block cipher schemes under EtE paradigm. We further, try to investigate the design of the four tweakable wide block cipher schemes, and introduce the notion of tweakable stream cipher (tS). We use tweakable stream cipher to create four new constructions of tweakable wide block cipher schemes that are CMT-4 secure, when used for creating authenticated encryption schemes by prepending zeros.

4.1.2 Contribution

- We analyze CMT-4 security of four tweakable wide block cipher schemes under the encode-then-encipher (EtE) paradigm by prepending zeros. An overview of CMT-4 attack of EtE-HBSH and EtE-HCTR2 was presented at the third NIST workshop on the block cipher modes of operation [114]. We present the algorithm for CMT-4 attack on EtE-HBSH and EtE-HCTR2 with time complexity $O(1)$. Further, we present two new results of CMT-4 attack under EtE paradigm of the two deck function based schemes .i.e., EtE-double-decker and EtE-docked-double-decker, with time complexity $O(1)$.

- We introduce the notion of a tweakable stream cipher (tS) with the property of partial collision resistance. Tweakable stream cipher takes three inputs: a key, a nonce and a tweak and produces a key stream as the output. The key stream can be used for message encryption (or decryption in case of ciphertext). Further, we demonstrate the procedure to create a tweakable stream cipher using eXtendable-Output Function (XOF).
- We present four new tweakable wide block cipher schemes based on a tweakable stream cipher, namely HBtSH, HtS, tS-double-decker and tS-docked-double-decker. These proposed schemes provide CMT-4 secure constructions of authenticated encryption schemes under encode-then-encipher paradigm. We provide CMT-4 security proof to prove our claim.

4.1.3 Organization of this Chapter

The rest of the chapter is organized as follows. In Section 4.2 presents the four tweakable wide block cipher schemes and their corresponding EtE versions. In section 4.3.4, we present collision attack on Farfalle [3]. In section 4.3, we present CMT-4 attack on the four tweakable wide block cipher schemes under EtE paradigm. In section 4.4, we present the four new tweakable wide block cipher schemes based on a tweakable stream cipher. We present design rationale of the four new proposed schemes in section 4.5, followed by the security proof of the proposed schemes in section 4.6. We conclude the chapter in section 4.7.

4.2 Tweakable Wide Block Cipher Schemes and EtE Paradigm

In this section, we discuss four tweakable wide block cipher schemes and present their respective EtE versions. We describe HBSH in section 4.2.1 and present EtE-HBSH in section 4.2.1.1. HCTR2 is describe in section 4.2.2 while EtE-HCTR2 is presented in section 4.2.2.1. The two deck function based schemes .i.e., double-decker and docked-double-decker are described section 4.2.3, EtE-double-decker and and EtE-docked-double-decker are presented in section 4.2.3.3.

4.2.1 HBSH

Hash Block cipher Stream cipher Hash (or HBSH) [83] is shown in figure 4.1. HBSH is a tweakable wide cipher scheme [60] and resembles an unbalanced Feistel structure [117]. It takes in three inputs: a key K from the key space $\mathcal{K} \in \{0,1\}^k$, a tweak T from the tweak space $\mathcal{T} \in \bigcup_{i=0}^{l_S}\{0,1\}^i$ and a message (or plaintext) P from the message space $\mathcal{M} \in \bigcup_{i=n}^{i=l_S}\{0,1\}^i$. During the invocation of encryption sub-routine, the plaintext P is first split into two parts, P_L and P_R respectively, where $(P_L \parallel P_R) \leftarrow P$ and $|P_R| = n$. Hence, $|P| \geq n$. The ciphertext C, is also generated by the encryption sub-routine in two parts (refer figure 4.1), .i.e., $(C_L \parallel C_R) \leftarrow C$. HBSH derives three keys K_H, K_E and K_S from K; K_H is used for the hash function H, K_E is used for the block cipher and K_S is used for the stream cipher. Each of these three cryptographic primitives that HBSH uses to generate a ciphertext C from the given plaintext P (or recover P from C) is explained below:

- **Hash:** H is an ϵ-almost-Δ-universal (ϵ-ΔU) function (definition 6) that represents a group element in an n bit string. The group operations $\boxplus$ and $\boxminus$ is computed using $\mathbb{Z}/2^n\mathbb{Z}$. The hash function is invoked two times.
  - For the first invocation, H takes in a tweak T and the left part of the plaintext .i.e., P_L, and uses key K_H to produce a fixed size output. This output is added with P_R .i.e., the right part of the plaintext P, to obtain P_M. Mathematically, $P_M \leftarrow H_{K_H}(T, P_L) \boxplus P_R$ (refer figure 4.1).
  - During the second invocation, H produces the right part of the ciphertext C (C_R) .i.e., $C_R \leftarrow H_{K_H}(T, C_L) \boxminus C_M$ (refer figure 4.1), where C_M is the output of the block cipher E and C_L is the left part of the ciphertext.
- **Block cipher:** A single invocation of block cipher (denoted by E) is used with block size n and key K_E (refer figure 4.1). E produces C_M .i.e., $C_M \leftarrow E_{K_E}(P_M)$. C_M is used as a nonce for the stream cipher.
- **Stream cipher:** In figure 4.1, S denotes a stream cipher that takes in a key K_S and a nonce C_M and produces a long random stream. This long random stream is *xored* with the left

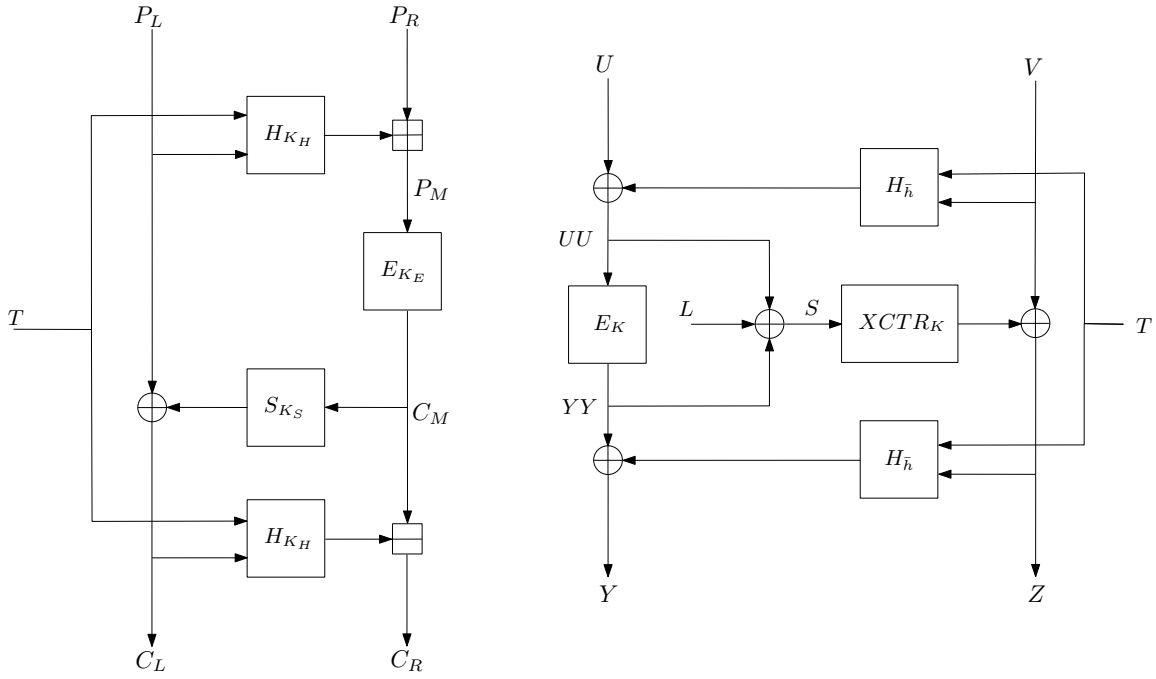
part of the plaintext P_L, producing the left part of the ciphertext C_L respectively.

As mentioned earlier, HBSH derives three keys K_H, K_E and K_S from K using a key derivation function. The stream cipher S is used as the key derivation function by instantiating it with a zero-length nonce .i.e., $K_E \parallel K_H = S_{K_S}(\epsilon)$, where $K_S = K$ and $|\epsilon| = 0$. We now describe Adiantum, a specification of HBSH.

Adiantum: We obtain Adiantum from HBSH when we fix $n = 128$ and $l_S = 2^{73}$. Hence, for the three inputs: a key K, a tweak T and a message P: $n = 128$ and $l_S = 2^{73}$. For the three cryptographic primitives, Adiantum uses XChaCha12 [118] as the stream cipher with key $K_S(= K)$ and nonce C_M from the nonce space $\mathcal{N} \in \bigcup_{i=0}^{191}$; such that, $S_{K_S} = \text{XChaCha12}_{K_S}(\text{pad}_{192}(C_M \parallel 1))$. For the block cipher, Adiantum uses AES256 [17, 119] with key K_E. C_M is generated from the block cipher E_{K_E}, and is used as the nonce after padding it to 192 bits. For the hashing, Adiantum uses combination of NH [120] and Poly1305 [121] with key K_H. Poly1305 is invoked two times, once with NH and once for hashing message length and tweak. Hence, we need a total of three keys for hashing .i.e., $K_T \leftarrow K_H[0:128]$ for Poly1305 which hashes message length and tweak, $K_N \leftarrow K_H[256:8576]$ for NH and $K_L \leftarrow K_H[128:256]$ for Poly1305, there by making a total of $128 + 8576 + 128 = 8832$ bit hash key or K_H. Let $H_T \leftarrow \text{Poly1305}_{K_T}(\text{bin}_{128}(|L|) \parallel T)$ and $H_L \leftarrow \text{Poly1305}_{K_L}(\text{NH}_{K_N}(\text{pad-0}_{128}(L)))$, then the final hash is given by $H_T \boxplus H_L$. Now, using the definition 14, we define EtE-HBSH.

4.2.1.1 EtE-HBSH

In this section, we present an authenticated encryption scheme called EtE-HBSH, based on the tweakable block cipher scheme HBSH (refer section 4.2.1), using encode-then-encrypt paradigm (or EtE in short) [59]. As described in definition 14, constructing such an authenticating encryption scheme is a two step approach, message encoding followed by enciphering. We prepend μ zero bits to the message for message encoding. For enciphering, we use HSBH. We now describe the EtE-HBSH. Let $\mathcal{E}$ be the encryption sub-routine and $\mathcal{D}$ be the decryption sub-routine of EtE-HBSH. $\mathcal{E}$ takes in three inputs, a key K, a tweak T and a message P, where



(a) HBSH (b) HCTR2

Figure 4.1: HBSH (Hash Block cipher Stream cipher Hash) and HCTR2 (extension of HCTR)

$T \leftarrow (N \parallel A)$ for a fixed size nonce N and associated data A. The message encoding is done using equation (4.1), where the plaintext P is first split into two parts P_L and P_R; then we create P'_L from P_L by prepending μ zero bits to P_L .i.e., $P'_L \leftarrow (0^\mu \parallel P_L)$, where $\mu > 0$. The tweakable block cipher scheme, HBSH, generates the ciphertext using key, tweak and the encoded message.

$$\text{Encode}(P, \mu) = \begin{cases} P' & \text{if } \mu > 0 \\ & \quad \text{where } P' \leftarrow (P'_L \parallel P_R) \\ & \quad \wedge\ P'_L \leftarrow (0^\mu \parallel P_L) \\ & \quad \wedge\ (P_L \parallel P_R) \leftarrow P \\ \perp & \text{otherwise.} \end{cases} \tag{4.1}$$

$$\text{Decode}(P', \mu) = \begin{cases} P & \text{if } \mu > 0 \wedge (P'_L = (0^\mu \parallel P_L)) \\ & \quad \text{where } (P'_L \parallel P_R) \leftarrow P' \\ & \quad \wedge\ P \leftarrow (P_L \parallel P_R) \\ \perp & \text{otherwise.} \end{cases}$$

In equation (4.1), P' is the input of HBSH and P is the input of AEAD .i.e, EtE-HBSH. The decryption sub-routine $\mathcal{D}$, takes in key K, tweak T and the ciphertext C. After the ciphertext is decrypted using HBSH (refer section 4.2.1), then equation (4.1) is used to retrieve the message or plaintext. The decryption sub-routine $\mathcal{D}$ of EtE-HBSH returns message if the first μ bits of P'_L are zero, else it returns $\perp$ (error).

4.2.2 HCTR2

HCTR2 [84] is an extension of HCTR [97]. From the likes of the Adiantum, HCTR2's primary focus is also on low powered devices and disk encryption scenarios. The encryption sub-routine of HRCR2 takes three inputs: a key K, a tweak T and a message or plaintext P. The key K is used for AES encryption, where $K \in \{\{0,1\}^{128}, \{0,1\}^{192}, \{0,1\}^{256}\}$. $\bar{h} \leftarrow E_K(\text{bin}(0))$, where $\bar{h}$ denotes the key used by the hash function and $|\bar{h}| = 128$. HCTR2 uses polyval for hashing. For a hash key $\bar{h} \in \{0,1\}^n$, a tweak T and a message M, the hash function is defined as follows:

$$H_{\bar{h}}(T, M) = \begin{cases} \text{POLYVAL}\big(\bar{h}, \text{bin}(2|T|+2) \parallel \text{pad}(T) \\ \quad \parallel M\big) \qquad\qquad \text{if } n \text{ divides } |M| \\ \text{POLYVAL}\big(\bar{h}, \text{bin}(2|T|+3) \parallel \text{pad}(T) \\ \quad \parallel \text{pad}(M \parallel 1)\big) \quad \text{otherwise.} \end{cases}$$

Polyval is calculated in the folowing way: $\text{POLYVAL}(\bar{h}, \lambda) = 0^n$ and $\text{POLYVAL}(\bar{h}, A \parallel B) = (\text{POLYVAL}(\bar{h}, A) \oplus B) \otimes \bar{h} \otimes x^{-n}$, where $|\bar{h}| = |B| = n = 128$ and $\otimes$ is multiplication over finite field. The polynomial used for reduction is $x^{128} + x^{127} + x^{126} + x^{121} + 1$ and $x^{-n} = x^{127} + x^{124} + x^{121} + x^{114} + 1$. Let $(U \parallel V) \leftarrow P$ (refer figure 4.1), where $|U| = n = 128$. Let $L \leftarrow E_k(\text{bin}(1))$, $UU \leftarrow U \oplus H_{\bar{h}}(T, N)$, $YY \leftarrow E_K(UU)$, and $S \leftarrow (UU \oplus YY \oplus L)$. HCTR2 uses *XCTR* as the stream encryption: $\text{XCTR}_K(S) = E_K(S \oplus \text{bin}(1)) \parallel E_K(S \oplus \text{bin}(2)) \parallel E_K(S \oplus \text{bin}(3)) \parallel \ldots$ and so on. The ciphertext C can be generated as follows: $Z \leftarrow (V \oplus \text{XCTR}_K(S)[1; |V|])$, $Y \leftarrow (YY \oplus H_{\bar{h}}(T, Z))$, and $C \leftarrow (Y \parallel Z)$.

4.2.2.1 EtE-HCTR2

In case of HCTR2, we know that the plaintext P is divided into two parts .i.e., $(U \parallel V) \leftarrow P$ (refer figure 4.1), where $|M| = n$. Using definition 14, we now present the EtE-HCTR2. Let $\mathcal{E}$ and $\mathcal{D}$ be the encryption and decryption sub-routine of EtE-HCTR2 respectively, under the encode-then-encipher paradigm. We use Encode sub-routine (refer equation 4.2) for V during the invocation of $\mathcal{E}$, and Decode sub-routine (refer equation 4.2) during the invocation of $\mathcal{D}$.

$$\text{Encode}(P, \mu) = \begin{cases} P' & \text{if } \mu > 0 \\ & \quad \text{where } P' \leftarrow (U \parallel V') \\ & \quad \wedge\ V' \leftarrow (0^{\mu} \parallel V) \\ & \quad \wedge\ (U \parallel V) \leftarrow P \\ \perp & \text{otherwise.} \end{cases} \tag{4.2}$$

$$\text{Decode}(P', \mu) = \begin{cases} P & \text{if } \mu > 0 \wedge (V' = (0^{\mu} \parallel V)) \\ & \quad \text{where } (U \parallel V') \leftarrow P' \\ & \quad \wedge\ P \leftarrow (U \parallel V) \\ \perp & \text{otherwise.} \end{cases}$$

In equation (4.2), P' is the input of HCTR2 and P is the input of AEAD .i.e, EtE-HCTR2. Similar to EtE-HBSH, the plaintext from the decryption sub-routine is considered valid if and only if, the first μ bits of V' are zero.

4.2.3 Deck-Based Wide Block Cipher Schemes

In this section we first describe two deck (doubly-extendable cryptographic keyed) function based tweakable wide block cipher schemes [85], namely, double-decker (section 4.2.3.1) and docked-double-decker (section 4.2.3.2). Double-decker (figure 4.2) is based on Farfalle [3], while docked-double-decker (figure 4.2) is a slight modification of the double-decker scheme. In

section 4.2.3.3, we present the EtE based authenticated encryption schemes of double-decker and docked-double-decker.

4.2.3.1 Double-decker

The construction of double-decker (figure 4.2) can be seen as a generalized four-round Feistel structure, with F_{K_1} and F_{K_2} as the two deck functions and two invocations of H_K. Double-decker is defined over a key space $\mathcal{K}$, a tweak space $\mathcal{W}$, a message space $\mathcal{M}$. The encryption sub-routine $\mathcal{E}$, takes in three keys $(K, K_1, K_2) \in \mathcal{K}$, a tweak $W \in \mathcal{W}$ and a message $P \in \mathcal{M}$, and produces a ciphertext $C \in \mathcal{C}$, where $|P| = |C|$ and $\mathcal{C}$ is the ciphertext space. The decryption sub-routine $\mathcal{D}$, takes input $(K, K_1, K_2) \in \mathcal{K}$, tweak $W \in \mathcal{W}$ and the ciphertext $C \in \mathcal{C}$ and returns back the output P as the plaintext, if C was generated using (K, K_1, K_2), W and P. From the construction of double-decker in figure 4.2, we know that the plaintext is divided into four separate parts .i.e., $(U_L \parallel U_R \parallel V_L \parallel V_R) \leftarrow P$, where $|U_L| = |V_R| = n$ and U_R and V_L can be of arbitrary length, while the ciphertext is the combination of four separate parts .i.e. $C \leftarrow (X_L \parallel X_R \parallel Y_L \parallel Y_R)$, where $|X_L| = |Y_R| = n$ and $|X_R| = |U_R|$, and $|Y_L| = |V_L|$. We define EtE-double-decker in section 4.2.3.3.

4.2.3.2 Docked-double-decker

Similar to the construction double-decker (section 4.2.3.1), docked-double-decker (figure 4.2) can also be seen as a generalized four-round Feistel structure. In this case, the plaintext is split into three separate parts .i.e., $(T \parallel U \parallel V) \leftarrow P$, with $|T| = |V| = n$ and U can be of arbitrary length. The ciphertext is the combination of three separate parts .i.e. $C \leftarrow (X \parallel Y \parallel Z)$, where $|X| = |Z| = n$ and $|Y| = |U|$. From figure 4.2, we can see that the input length to the two deck functions F_{K_1} and F_{K_2} are fixed, hence, for a fixed tweak length one can conceptualize docked-double-decker as a stream cipher [85]. We now define EtE-double-decker and EtE-docked-double-decker.

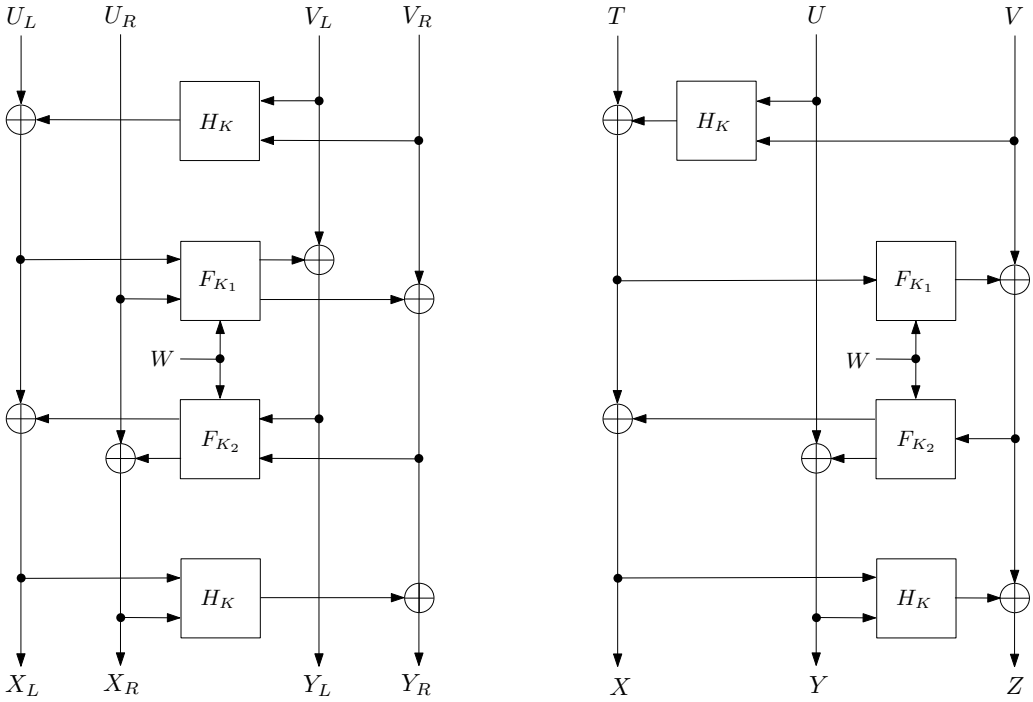


Figure 4.2: Double-decker and Docked-double-decker

4.2.3.3 EtE-Double-Decker and EtE-Docked-Double-Decker

Using definition 14, we present the encode-then-encipher version of double-decker and docked-double-decker.

- **EtE-Double-Decker:** Let $\mathcal{E}$ and $\mathcal{D}$ be the encryption and decryption sub-routines of EtE-double-decker respectively. From section 4.2.3.1, we know that the plaintext is split into four separate parts .i.e., $(U_L \parallel U_R \parallel V_L \parallel V_R) \leftarrow P$ respectively (refer figure 4.2). To create encoding for EtE-double-decker from double-decker, we utilize the second split of the plaintext P .i.e., U_R. Invocation of $\mathcal{E}$, starts with the encoding of U_R, by using the Encode sub-routine of equation (4.3). In case of the decryption sub-routine $\mathcal{D}$, we have an

addition step .i.e., Decode sub-routine of equation (4.3).

$$\text{Encode}(P,\mu) = \begin{cases} P' & \text{if } \mu > 0, \text{ where} \\ & P' \leftarrow (U_L \parallel U'_R \parallel V_L \parallel V_R) \\ & \wedge\ U'_R \leftarrow (0^{\mu} \parallel U_R) \\ & \wedge\ (U_L \parallel U_R \parallel V_L \parallel V_R) \leftarrow P \\ \perp & \text{otherwise.} \end{cases}$$

$$\text{Decode}(P',\mu) = \begin{cases} P & \text{if } \mu > 0 \wedge (U'_R = (0^{\mu} \parallel U_R)) \\ & \text{where,} \\ & (U_L \parallel U'_R \parallel V_L \parallel V_R) \leftarrow P' \\ & \wedge \\ & P \leftarrow (U_L \parallel U_R \parallel V_L \parallel V_R) \\ \perp & \text{otherwise.} \end{cases} \tag{4.3}$$

In equation (4.3), P' is the input of double-decker and P is the input of AEAD .i.e, EtE-double-decker.

- **EtE-Docked-Double-Decker:** In case of docked-double-decker, we know that the plain-text is split into three parts .i.e., $(T \parallel U \parallel V) \leftarrow P$. We use U, to create EtE-docked-double-decker. Let $\mathcal{E}$ and $\mathcal{D}$ be the encryption and decryption sub-routines of EtE-docked-double-decker, then we use equation (4.4), for creating encoding and decoding rules of $\mathcal{E}$

and $\mathcal{D}$ respectively.

$$\text{Encode}(P,\mu) = \begin{cases} P' & \text{if } \mu > 0 \text{ where,} \\ & P' \leftarrow (T \parallel U' \parallel V) \\ & \wedge\ U' \leftarrow (0^{\mu} \parallel U) \\ & \wedge\ P \leftarrow (T \parallel U \parallel V) \\ \perp & \text{otherwise.} \end{cases} \tag{4.4}$$

$$\text{Decode}(P',\mu) = \begin{cases} P & \text{if } \mu > 0 \wedge (U' = (0^{\mu} \parallel U)) \\ & \text{where,} \\ & (T \parallel U' \parallel V) \leftarrow P' \\ & \wedge\ P \leftarrow (T \parallel U \parallel V) \\ \perp & \text{otherwise.} \end{cases}$$

In equation (4.4), P' is the input of double-decker and P is the input of AEAD .i.e, EtE-double-decker.

In case of EtE-double-decker, the plaintext is accepted if and only if, the first μ bits of U'_R are zero, while for EtE-docked-double-decker, the plaintext is accepted if and only if, the first μ bits of U' are zero.

4.3 CMT-4 Attack on Encode-then-Encipher Schemes

In this section, we present CMT-4 attack under EtE paradigm for four tweakable wide block cipher schemes, namely EtE-HBSH (section 4.2.1.1, algorithm 7), EtE-HCTR2 (section 4.2.2.1, algorithm 8), EtE-double decker and EtE-docked-double-decker (section 4.2.3.3).

4.3.1 CMT-4 Attack on EtE-HBSH

Let $\mathcal{A}$ be a computationally bounded CMT-4 adversary that has access to EtE-HBSH construction, defined in section 4.2.1.1. $\mathcal{A}$ interacts with the encryption sub-routine $\mathcal{E}$ of EtE-HBSH. $\mathcal{E}$ takes in four inputs: a key K, a nonce N, an associated data A and a message P and returns a ciphertext C. The task of $\mathcal{A}$ is to create two distinct tuples (K_1, N_1, A_1, P_1) and (K_2, N_2, A_2, P_2) .i.e., $\tau = \{(K_1, N_1, A_1, P_1), (K_2, N_2, A_2, P_2)\}$ such that $(K_1, N_1, A_1, P_1) \neq (K_2, N_2, A_2, P_2)$ but $\mathcal{E}(K_1, N_1, A_1, P_1) = \mathcal{E}(K_2, N_2, A_2, P_2)$ .i.e., the ciphertext C generated by the encryption sub-routine for the two distinct tuples are the same. For EtE-HBSH, $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$. CMT-4 attack on EtE-HBSH is presented in Algorithm 7. We explain the attack step by step as described in the algorithm. We start by deriving the two keys K_E and K_H from K_S. We then obtain P_L and P_R from P, then we create EtE-HBSH .i.e., we apply Encode sub-routine of equation (4.1) on P_L to obtain P'_L .i.e., $P'_L \leftarrow (0^\mu \parallel P_L)$, where $\mu > 0$ and $(P_L \parallel P_R) \leftarrow P$. Let $P_M \leftarrow 0^n$ .i.e., we fix P_M to a constant value. Note that, in case of Adiantum, $n = 128$. Next, we fix the key to a constant value .i.e., $K_1 = K_2 = K$. We know that $K_S = K$ and K_E and K_H is obtained from the stream cipher using key K_S and instantiation with ϵ. Let $C_M \leftarrow E_{K_E}(P_M)$. We now compute the left part of the ciphertext .i.e., $C_L \leftarrow P'_L \oplus S_{K_S}(C_M)[1; |P'_L|]$. Now, we choose and fix a value for T_1, and compute $H_{K_H}(T_1, C_L)$. Then, it is trivial to find T_2 using the linear equation: $H_{K_H}(T_1, C_L) = H_{K_H}(T_2, C_L)$, such that $T_1 \neq T_2$. Hence, $C_R \leftarrow H_{K_H}(T_1, C_L) \boxminus C_M$. We can now, simply append C_L and C_R to get the ciphertext, $C \leftarrow (C_L \parallel C_R)$. We can derive P_{R_1} and P_{R_2} as follows, $P_{R_1} = H_{K_H}(T_1, P'_L) \boxminus P_M$ and $P_{R_2} = H_{K_H}(T_2, P'_L) \boxminus P_M$. Hence, $P_1 \leftarrow (P'_L \parallel P_{R_1})$, $P_2 \leftarrow (P'_L \parallel P_{R_2})$. The adversary $\mathcal{A}$ has successfully generated two tuples (K, N_1, A_1, P_1) and (K, N_2, A_2, P_2) such that $\forall i\ (N_i \parallel A_i) \leftarrow T_i$, enforcing generation of same ciphertext C from EtE-HBSH using these two distinct tuples. Hence, $\tau = \{(K, N_1, A_1, P_1), (K, N_2, A_2, P_2)\}$, where $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$.

4.3.2 CMT-4 Attack on EtE-HCTR2

Let $\mathcal{A}$ be a computationally bounded CMT-4 adversary that has access to the EtE-HCTR2 algorithm as described in section 4.2.2.1. $\mathcal{A}$ must create two distinct tuples (K_1, N_1, A_1, P_1) and

Algorithm 7 : CMT-4 Attack on EtE-HBSH (EtE-Adiantum when $n = 128$)

1: $K_S \xleftarrow{\$} \mathcal{K}$
2: $K_E \parallel K_H = S_{K_S}(\epsilon)$ ▷ $|\epsilon| = 0$, *Derive* K_H *and* K_E *from* K_S
3: $P \xleftarrow{\$} \mathcal{M}$, where $|P| > n$
4: $P_L \parallel P_R \leftarrow P, |P_L| = n$
5: $P'_L \leftarrow (0^\mu \parallel P_L), \mu > 0$ ▷ *Construct* P'_L *from* P_L
6: $P_M \leftarrow 0^n$
7: $C_M \leftarrow E_{K_E}(P_M)$
8: $C_L \leftarrow P'_L \oplus S_{K_S}(C_M)[1; |P'_L|]$
9: $\exists\ (T_1, T_2) \mid H_{K_H}(T_1, C_L) = H_{K_H}(T_2, C_L)$
10: $P_{R_1} = H_{K_H}(T_1, P'_L) \boxminus P_M$ and $P_{R_2} = H_{K_H}(T_2, P'_L) \boxminus P_M$ ▷ *Derive* P_{R_1} *and* P_{R_2}
11: $P_1 \leftarrow (P'_L \parallel P_{R_1}), P_2 \leftarrow (P'_L \parallel P_{R_2})$
12: $\tau = \{(K, N_1, A_1, P_1), (K, N_2, A_2, P_2)\}$, where $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$
13: **Return** τ

(K_2, N_2, A_2, P_2) such that the ciphertext C produced by the encryption sub-routine $\mathcal{E}$ of EtE-HCTR2 is the same. CMT-4 attack on EtE-HCTR2 is presented as a pseudocode in Algorithm 8. We now describe each step of the attack. First, we start by fixing the key .i.e., $K_1 = K_2 = K$. Now, we proceed by calculating $\bar{h}$ and L as follows: $\bar{h} \leftarrow E_K(\text{bin}(0))$ and $L \leftarrow E_K(\text{bin}(1))$ respectively. We know that $(U \parallel V) \leftarrow P$ (refer figure 4.1). We construct V' from V using equation (4.2) .i.e., $V' \leftarrow (0^\mu \parallel V), \mu > 0$. Let $UU \leftarrow 0^n$, then we have $YY \leftarrow E_K(UU)$ and $S \leftarrow (UU \oplus YY \oplus L)$. We next calculate Z, $Z \leftarrow V' \oplus XCTR_K(S)[1; |V'|]$. We now fix T_1 and calculate $H_{\bar{h}}(T_1, V)$. Then we can obtain T_2 using equation $H_{\bar{h}}(T_1, V) = H_{\bar{h}}(T_2, V)$, such that $T_1 \neq T_2$. We can now calculate Y .i.e., $Y \leftarrow H_{\bar{h}}(T_1, V)$. Further, we can now find U_1 and U_2 as follows: $U_1 = H_{\bar{h}}(T_1, V') \oplus UU$ and $U_2 = H_{\bar{h}}(T_1, V') \oplus UU$. The ciphertext can be calculated as $C \leftarrow (Y \parallel Z)$. Further, P_1 and P_2 can be calculated as follows: $P_1 \leftarrow (U_1 \parallel V')$, $P_2 \leftarrow (U_2 \parallel V')$. Hence, $\tau = \{(K, N_1, A_1, P_1), (K, N_2, A_2, P_2)\}$, where $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$.

4.3.3 CMT-4 Attack on EtE-Double-Decker and EtE-Docked-Double-Decker

In this section, we present two new results of CMT-4 attack. In section 4.3.3.1, we present CMT-4 attack on EtE-double-decker and in section 4.3.3.2, we present the CMT-4 attack on EtE-docked-double-decker. Note that, we assume the deck function is based on Farfalle [3], for both double-decker and docked-double-decker. We present an overview of Farfalle with CMT-4

Algorithm 8 : CMT-4 Attack on EtE-HCTR2

1: $K \xleftarrow{\$} \mathcal{K}$
2: $\bar{h} \leftarrow E_K(\text{bin}(0))$
3: $L \leftarrow E_K(\text{bin}(1))$
4: $P \xleftarrow{\$} \mathcal{M}$, where $|P| > n$
5: $(U \parallel V) \leftarrow P, |U| = n$
6: $V' \leftarrow 0^{\mu} \parallel V, \mu > 0$ ▷ *Construct V' from V*
7: $UU \leftarrow 0^n$
8: $YY \leftarrow E_K(UU)$
9: $S \leftarrow (UU \oplus YY \oplus L)$
10: $Z \leftarrow V' \oplus XCTR_K(S)[1; |V'|]$
11: $\exists\ (T_1, T_2) \mid H_{\bar{h}}(T_1, Z) = H_{\bar{h}}(T_2, Z)$
12: $M_1 = H_{\bar{h}}(T_1, V') \oplus UU$ and $M_2 = H_{\bar{h}}(T_1, V') \oplus UU$ ▷ *Derive U_1 and U_2*
13: $P_1 \leftarrow (U_1 \parallel V'), P_2 \leftarrow (U_2 \parallel V')$
14: $\tau = \{(K, N_1, A_1, P_1), (K, N_2, A_2, P_2)\}$, where $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$
15: **Return** τ

attack in appendix 4.3.4.

4.3.3.1 CMT-4 Attack on EtE-Double-Decker

Let $\mathcal{A}$ be a computationally bounded CMT-4 adversary. The task of $\mathcal{A}$ is to create two distinct tuples (K_1, N_1, A_1, P_1) and (K_2, N_2, A_2, P_2), where $W_1 \leftarrow (N_1 \parallel A_1)$ and $W_2 \leftarrow (N_2 \parallel A_2)$, such that the output or the ciphertext C generated by the encryption sub-routine of EtE-double-decker for the two tuples are the same. Note that, in case of double-decker and docked-double-decker, each key K_1 and K_2 is actually a tuple of three keys .i.e., (K_1, K_{1_1}, K_{1_2}) and (K_2, K_{2_1}, K_{2_2}). We start by fixing the keys of the two tuples, and hence, enumerating the tuple as $\bar{K}$ .i.e., $\bar{K} = (K_1, K_{1_1}, K_{1_2}) = (K_2, K_{2_1}, K_{2_2})$. Further, we know that, the plaintext is split into four parts .i.e., $(U_L \parallel U_R \parallel V_L \parallel V_R) \leftarrow P$, where $|U_L| = |V_R| = n$. W denotes the tweak and the ciphertext C is given by $C \leftarrow (X_L \parallel X_R \parallel Y_L \parallel Y_R)$ where $|X_L| = |Y_R| = n$. In our attack, we assume $U_{R_1} = U_{R_2} = 0^{\mu}$ where $\mu > 0$. Hence, we can construct U'_{R_1} and U'_{R_2} using equation (4.3), in the following way: $U'_{R_1} = (0^{\mu} \parallel U_{R_1})$ and $U'_{R_2} = (0^{\mu} \parallel U_{R_2})$, thereby making the size of of U'_{R_1} and U'_{R_2} as $2 \bullet \mu$ bits. This setup of U_{R_1} and U_{R_1} is one of the example, our CMT-4 attack will work on any arbitrary size. We advise the reader to augment the explanation of the attack with figure 4.3.

1. We select F_{K_2} for starting the CMT-4 attack. F_{K_2} is defined over three input parameters and a key (refer figure 4.3).We already fixed key .i.e.,

$$\bar{K} = (K_1, K_{1_1}, K_{1_2}) = (K_2, K_{2_1}, K_{2_2}).$$

2. Fix two input parameters of F_{K_2} as g and h. Note that the third parameter is the tweak. Now, we construct the inputs of Farfalle[2]. A padding function is required to ensure that inputs are multiple of $\bar{b}$ bits .i.e., $\text{pad10}^*(M) = M \parallel 10^{|M| \pmod{\bar{b}}-1}$ for $M \in \{0,1\}^*$. Let ENC be an encoding, then for two tweaks W_1 and W_2 we can construct input of Farfalle as follows:

$$\text{ENC}(W_1, g, h) = (\text{pad10}^*(W_1), \text{pad10}^*(g \parallel h)),$$
$$\text{ENC}(W_2, g, h) = (\text{pad10}^*(W_2), \text{pad10}^*(g \parallel h)).$$

3. Using the collision attack on Farfalle (refer appendix 4.3.4.3), we can say that, there exists two tweaks, such that output of F_{K_2} is deterministically produced .i.e., $\exists\ (W_1, W_2) \mid F_{K_2}(\text{ENC}(W_1, g, h)) = F_{K_2}(\text{ENC}(W_2, g, h)) = (a, b)$. This implies $X_{R_1} = X_{R_2} = b$, since $U_{R_1} = U_{R_2} = 0^\mu$ and $X_{R_1} = U_{R_1} \oplus b$ and $X_{R_2} = U_{R_2} \oplus b$ respectively.

4. We now fix another value c, one of the parameters to the deck function F_{K_1} (please refer figure 4.3). Hence, $X_{L_1} = X_{L_2} = c \oplus a$. The input to H_K is deterministic .i.e., $c \oplus a$ (or X_L) and 0^μ (or X_R) are completely determined, there by fixing the output of H_K as f, we have $Y_{R_1} = Y_{R_2} = h \oplus f$.

5. Since, g is fixed, hence $Y_{L_1} = Y_{L_2} = g$.

6. Since, $W_1 \neq W_2$, hence, $F_{K_1}(\text{ENC}(W_1, c, U_{R_1})) \neq F_{K_1}(\text{ENC}(W_2, c, U_{R_2}))$, or, we have, $F_{K_1}(\text{ENC}(W_1, c, U_{R_1})) = (d_1, e_1)$ and $F_{K_1}(\text{ENC}(W_2, c, U_{R_2})) = (d_2, e_2)$. Further, since, we already fixed g and h, hence $V_{R_1} = h \oplus e_1$ and $V_{R_2} = h \oplus e_2$. Also, $V_{L_1} = g \oplus d_1$ and $V_{L_2} = g \oplus d_2$.

7. Since $V_{L_1} \neq V_{L_2}$, this makes the input to H_K as distinct .i.e., $H_K(g \oplus d_1, h \oplus e_1) = i_1$ and $H_K(g \oplus d_2, h \oplus e_2) = i_2$; making $U_{L_1} = i_1 \oplus c$ and $U_{L_2} = i_2 \oplus c$.

[2]Farfalle takes input as bits of string and a key, and processes the input in multiples of size $\bar{b}$-bits (refer section 4.3.4.2)

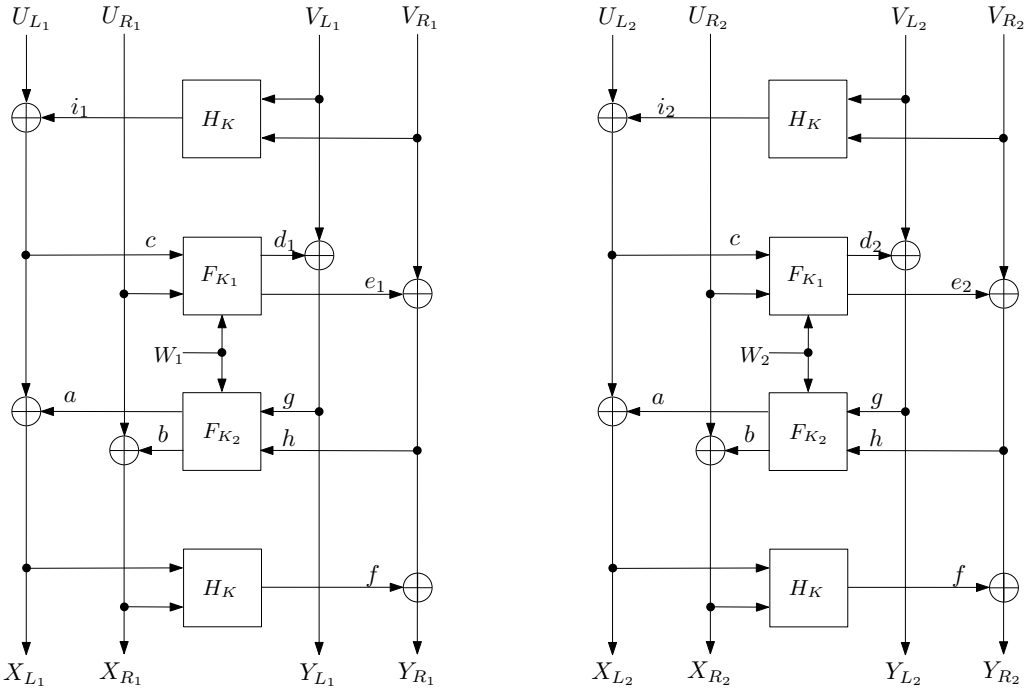


Figure 4.3: CMT-4 Attack on EtE-Double-Decker

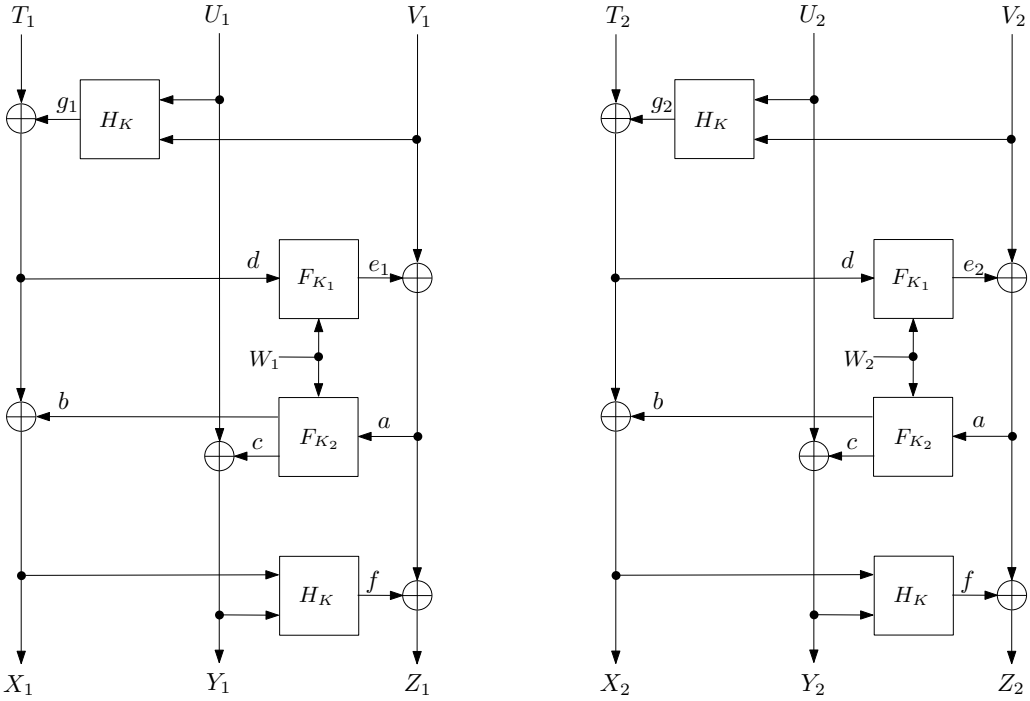


Figure 4.4: CMT-4 Attack on EtE-Docked-Double-Decker

We can now construct the message and the ciphertext for the encryption sub-routine of

EtE-double-decker:

$$\begin{aligned} P_1 &= (i_1 \oplus c) \parallel 0^{\mu} \parallel (g \oplus d_1) \parallel (h \oplus e_1), \quad \mu > 0 \\ C_1 &= (c \oplus a) \parallel b \parallel g \parallel (h \oplus f) \\ P_2 &= (i_2 \oplus c) \parallel 0^{\mu} \parallel (g \oplus d_2) \parallel (h \oplus e_2), \quad \mu > 0 \\ C_2 &= (c \oplus a) \parallel b \parallel g \parallel (h \oplus f). \end{aligned} \tag{4.5}$$

From equation (4.5), we see that $C_1 = C_2$, hence we can now construct two distinct tuples in the following way:

$$\begin{aligned} \tau = \{&(\bar{K}, N_1, A_1, P_1), (\bar{K}, N_2, A_2, P_2)\} \\ &\text{where } (N_1 \parallel A_1) \leftarrow W_1 \text{ and } (N_2 \parallel A_2) \leftarrow W_2, \\ &P_1 \leftarrow (i_1 \oplus c) \parallel 0^{\mu} \parallel (g \oplus d_1) \parallel (h \oplus e_1), \\ &P_2 \leftarrow (i_2 \oplus c) \parallel 0^{\mu} \parallel (g \oplus d_2) \parallel (h \oplus e_2). \end{aligned} \tag{4.6}$$

From equation (4.6), we have constructed two tuples, such that $P_1 \neq P_2$ and $W_1 \neq W_2$ but $C_1 = C_2$. This completes our CMT-4 attack on EtE-double-decker.

4.3.3.2 CMT-4 Attack on EtE-Docked-Double-Decker

The task of $\mathcal{A}$ is to create two distinct tuples (K_1, N_1, A_1, P_1) and (K_2, N_2, A_2, P_2), where $W_1 \leftarrow (N_1 \parallel A_1)$ and $W_2 \leftarrow (N_2 \parallel A_2)$, such that the output or the ciphertext C generated by the encryption sub-routine of EtE-docked-double-decker are the same. As mentioned earlier (section 4.3.3.1), each key K_1 and K_2 is actually a tuple of three keys .i.e., (K_1, K_{1_1}, K_{1_2}) and (K_2, K_{2_1}, K_{2_2}). We fix the keys by enumerating them as a tuple $\bar{K}$ .i.e., $\bar{K} = (K, K_1, K_2)$. The plaintext is divided into three parts .i.e., $(T \parallel U \parallel V) \leftarrow P$, where $|T| = |V| = n$. W denotes the tweak and the ciphertext C is given by $C \leftarrow (X \parallel Y \parallel Z)$ where $|X| = |Z| = n$. As in the case of EtE-double-decker (section 4.3.3.1), we proceed by making $U_1 = U_2 = 0^{\mu}$, where $\mu > 0$, and derieve U_1' and U_2' using equation (4.4) .i.e., $U_1' = (0^{\mu} \parallel U_1)$ and $U_2' = (0^{\mu} \parallel U_2)$, thereby $|U_1'| = |U_2'| = 2 \bullet \mu$. Please refer figure 4.4 for the explanation of the attack.

1. We select F_{K_2} for the CMT-4 attack. The key is already fixed .i.e.,

$$\bar{K} = (K_1, K_{1_1}, K_{1_2}) = (K_2, K_{2_1}, K_{2_2}).$$

2. We fix a .i.e., one of the parameters of F_{K_2}. Let ENC be an encoding, then for two tweaks W_1 and W_2 we can construct input of Farfalle as follows:

$$\text{ENC}(W_1, a) = (\text{pad10}^*(W_1), \text{pad10}^*(a)),$$
$$\text{ENC}(W_2, a) = (\text{pad10}^*(W_2), \text{pad10}^*(a)).$$

3. Using the collision attack of Farfalle (refer appendix 4.3.4.3), we can say that, there exists two tweaks, W_1 and W_2 such that the output from F_{K_2} is completely determined .i.e., $\exists$ $(W_1, W_2) \mid F_{K_2}(\text{ENC}(W_1, a)) = F_{K_2}(\text{ENC}(W_2, a)) = (b, c)$. Hence, $Y = 0^\mu \oplus c = c$.

4. We now fix d, then $X = b \oplus d$. Further, we have $H_K(b \oplus d, c) = f$, this implies the value of Z is also completely determined .i.e., $Z = f \oplus a$.

5. Since, $W_1 \neq W_2$, hence, $F_{K_1}(\text{ENC}(W_1, d)) \neq F_{K_1}(\text{ENC}(W_2, d))$. Let $F_{K_1}(\text{ENC}(W_1, d)) = e_1$ and $F_{K_1}(\text{ENC}(W_2, d)) = e_2$. Proceeding this way, we have $V_1 = e_1 \oplus a$ and $V_2 = e_2 \oplus a$.

6. Since, $g_1 = H_K(0^\mu, e_1 \oplus a)$ and $g_2 = H_K(0^\mu, e_2 \oplus a)$, hence $T_1 = d \oplus g_1$ and $T_2 = d \oplus g_2$.

We can now construct the message and the ciphertext for the encryption sub-routine of EtE-docked-double-decker:

$$\begin{aligned} P_1 &= (d \oplus g_1) \parallel 0^\mu \parallel (e_1 \oplus a), \quad \mu > 0 \\ C_1 &= (b \oplus d) \parallel c \parallel (f \oplus a). \\ P_2 &= (d \oplus g_2) \parallel 0^\mu \parallel (e_2 \oplus a), \quad \mu > 0 \\ C_2 &= (b \oplus d) \parallel c \parallel (f \oplus a). \end{aligned} \tag{4.7}$$

Table 4.1: **Summary of CMT-4 attack.** EtE-HBSH, EtE-HCTR2, EtE-Double-Decker and EtE-Docked-Double-Decker. Two new results of CMT-4 attack are shown in red color. In case of EtE-HBSH and EtE-HCTR2, we present detailed analysis and algorithm of CMT-4 attack.

AEAD under EtE Paradigm	**Message Encoding** ($\mu > 0$)	**Tweakable Wide Block Cipher Scheme (Message Enciphering)**	**CMT-4 Attack**
EtE-HBSH	$((0^{\mu} \parallel P_L) \parallel P_R) \leftarrow P$	HBSH	Section 4.3.1 [115, 122]
			Time Complexity $= O(1)$
EtE-HCTR2	$(M \parallel (0^{\mu} \parallel N)) \leftarrow P$	HCTR2	Section 4.3.2 [115, 122]
			Time Complexity $= O(1)$
EtE-Double-Decker	$(U_L \parallel (0^{\mu} \parallel U_R) \parallel V_L \parallel V_R) \leftarrow P$	Double-decker	Section 4.3.3.1
			Time Complexity $= O(1)$
EtE-Docked-Double-Decker	$(T \parallel (0^{\mu} \parallel U) \parallel V) \leftarrow P$	Docked-double-decker	Section 4.3.3.2
			Time Complexity $= O(1)$

From equation (4.7), we see that $C_1 = C_2$, hence we can now construct two distinct tuples in the following way:

$$\begin{aligned} \tau = \{&(\bar{K}, N_1, A_1, P_1), (\bar{K}, N_2, A_2, P_2)\} \\ &\text{where } (N_1 \parallel A_1) \leftarrow W_1 \text{ and } (N_2 \parallel A_2) \leftarrow W_2, \\ &P_1 \leftarrow (d \oplus g_1) \parallel 0^{\mu} \parallel (e_1 \oplus a) \\ &P_2 \leftarrow (d \oplus g_2) \parallel 0^{\mu} \parallel (e_2 \oplus a). \end{aligned} \tag{4.8}$$

From equation (4.8), we have constructed two tuples such that $P_1 \neq P_2$ and $W_1 \neq W_2$ but $C_1 = C_2$. This completes our CMT-4 attack on EtE-docked-double-decker.

In table 4.1, we present the summary of CMT-4 attack on encode-then-encipher versions of HBSH, HCTR2, double-decker and docked-double-decker. New results of CMT-4 attack is shown in red color, while in case of EtE-HBSH and EtE-HCTR2, we provide detailed analysis and algorithm for CMT-4 attack.

4.3.4 Farfalle

4.3.4.1 Overview

Farfalle [3] can be used for building a deck (doubly-extendable keyed) function. It takes bits of string and a key K as the input and returns an arbitrary-length output. From the architecture point of view, a Farfalle can be divided into two layers, a compression layer, followed by an expansion layer (refer figure 4.5). It can be instantiated using following notation:

$$\text{Farfalle}[p_b, p_c, p_d, p_e, \text{roll}_c, \text{roll}_e],$$

where p_b, p_c, p_d and p_e are permutations and, roll_c and roll_e are rolling functions, such that:

- p_b: used for deriving the initial mask from the key K
- p_c: used in the compression layer
- p_d: used between the compression and expansion layer
- p_e: used in the expansion layer
- roll_c: used in the compression layer for generating masks that are added to the input blocks
- roll_e: used in the expansion layer to update the internal state

The rolling function, roll_c is a lightweight linear function with huge order [3], analogous to LFSR. As an example, XOOFFF [123] is based on Farfalle, with XOODOO [123] as the underlying permutation with six rounds for p_b, p_c, p_d and p_e and LFSR like rolling functions for roll_c and roll_e.

4.3.4.2 Specification

Inside Farfalle, bits of string are processed in blocks of $\bar{b}$-bits or in multiple of $\bar{b}$ bits, where $\bar{b}$ denotes the width of underlying permutation. Hence, a padding rule is needed to convert the input bits of string into multiple of $\bar{b}$-bits. For a given message $M \in \{0,1\}^*$, padding rule is defined

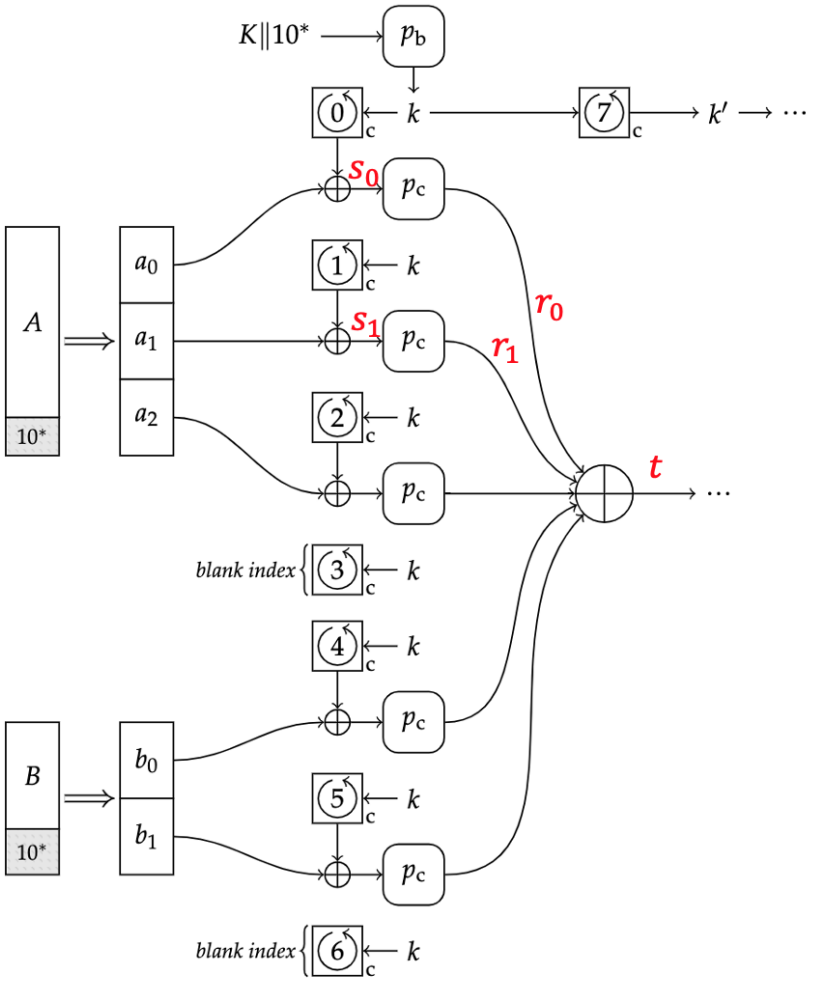


Figure 4.5: Compression layer of Farfalle (Adapted from [3])

as follows: $P = \text{pad10}^*(M)$, where $P \leftarrow (p_0 \parallel p_2 \parallel \cdots \parallel p_{n-1})$ and $\forall\, i \in \{0, n-1\}\ |p_i| = b$. Similarly, the key K is padded to $(K \parallel 10^*)$ before applying p_b. Two keys k and k' is derived from the input key K; k is used during the compression layer while k' is used during the expansion layer.

4.3.4.3 Collision Attack

For performing collision attack on Farfalle, the task the adversary $\mathcal{A}$ is to create two distinct inputs of Farfalle, such that the output generated by Farfalle for the two inputs are the same. From the construction, we know that if the output of the expansion layer, .i.e., 't' in figure 4.5, is same for the two inputs then the output produced by the expnasion layer (or Farfalle) is also same.

Hence, we choose compression layer of Farfalle for demonstrating our collision attack. Our task is to construct two inputs: $\text{ENC}(\text{pad10}^*(A_1), \text{pad10}^*(B))$ and $\text{ENC}(\text{pad10}^*(A_2), \text{pad10}^*(B))$, where $A_1 \neq A_2$, such that the output of the expansion layer is t. We require that both A_1 and A_2 contains at least two full blocks .i.e., $|A_1| \geq 2 \bullet \bar{b}$ and $|A_2| \geq 2 \bullet \bar{b}$. We start our attack by fixing the key K, hence we obtain k and k' deterministically. Since, 't' is obtained by *xor* operation, it is trivial to find r'_0 and r'_1, that produces the same output 't'. Now, since p_c is a permutation, we can find values 's_0', 's_1', 's'_0' and 's'_1' such that $s_0 = p_c^{-1}(r_0)$, $s_1 = p_c^{-1}(r_1)$, $s'_0 = p_c^{-1}(r'_0)$ and $s'_1 = p_c^{-1}(r'_1)$. We know that the rolling function roll_c is linear, hence it is trivial to obtain two distinct $\bar{b}$-bit blocks a_{1_0} and a_{2_0}, such that a_{1_0} and a_{2_0} denotes the two $\bar{b}$-bit blocks of A_1; a'_{1_0} and a'_{2_0}, such that a'_{1_0} and a'_{2_0} denotes two $\bar{b}$-bit blocks of A_2. This leads to the fact that we have constructed two distinct inputs producing the same output. This concludes the collision attack on Farfalle.

4.4 HBtSH, HtS, tS-Double-Decker and tS-Docked-Double-Decker

In this section, we present four new designs of tweakable wide block cipher schemes that utilizes tweakable stream cipher (tS). In section 4.4.1, we introduce HBtSH, which is based on HBSH. In section 4.4.2, we present HtS based on HCTR2. We present tS-double-decker based on double-decker, and tS-docked-double-decker based on docked-double-decker in section 4.4.3. In this chapter, we use eXtendable-Output Function (or XOF in short) [124] for instantiating the tweakable stream cipher.

4.4.1 HBtSH

We present a new tweakable wide block cipher scheme called HBtSH (**H**ash **B**lock cipher **t**weakable **S**tream cipher **H**ash). The architecture of the scheme is shown in figure 4.6a, and we present the pseudocode of the scheme in algorithm 9. HBtSH based on a tweakable stream cipher (refer definition 8), and uses the structural design of HBSH (refer section 4.2.1 for HBSH), but replaces its one of the underlying primitive .i.e., instead of using a stream cipher, HBtSH uses a

tweakable stream cipher. We now describe the construction of HBtSH.

Let $\mathcal{E}$ and $\mathcal{D}$ be the encryption sub-routine and the decryption sub-routine of HBtSH. $\mathcal{E}$ takes in three inputs, a key K, a tweak T and a plaintext P and produces a ciphertext C as the output. $\mathcal{D}$ takes input as a key K, a tweak T and a ciphertext C and outputs the corresponding plaintext P. The tweakable stream cipher is invoked two times, once for generating the ciphertext C_L, where $(C_L \parallel C_R) \leftarrow C$, and once for the key derivation function.

- **Generating Ciphertext:** S is used to generate the left part of the ciphertext .i.e., C_L. S takes in three inputs: a key K, a fixed size nonce C_M and a tweak T, and a maximum output length denoted by l_S, such that $|P_L| \leq l_S$. Hence, the invocation of S can be represented by following expression:

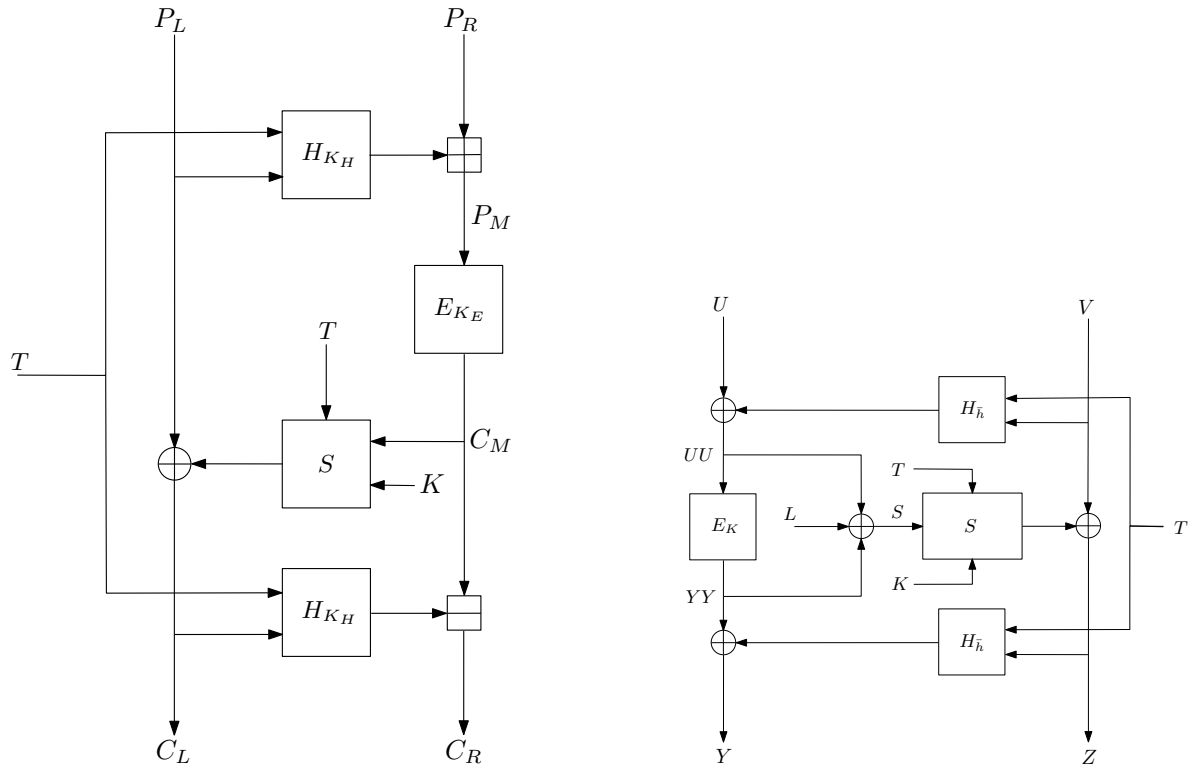
$$S(K, C_M, T)[1; |P_L|]. \tag{4.9}$$

  The key stream generated by equation (4.9) is *xored* with P_L for generating C_L. When a XOF-based S is used, a reversible encoding denoted by ENC is utilized for combining C_M and T .i.e., ENC(C_M, T). For a nonce with size $< \eta$, C_M in this case, ENC can be written as $(\text{pad}_\eta(C_M) \parallel T)$, where padding function is defined in equation (2.2).

- **Key Derivation Function:** The key K_E (used for the block cipher) and the key K_H (used for the hash function) is derived using the tweakable stream cipher with input as the key K and empty nonce and empty tweak, with a maximum output length given by l_S such that $|K_E| + |K_H| \leq l_S$. The invocation of S can be represented by following expression:

$$S(K, \epsilon, \epsilon)[1; |K_E| + |K_H|]. \tag{4.10}$$

The key stream generated by equation (4.10) is used as the two keys K_E and K_H. We enforce the condition that when using XOF, equation (4.9) and equation (4.10) must be unrelated. This will lead to an unrelated output stream. HBtSH from algorithm 9 can be used to describe the Adiantum specification. We present CMT-4 security proof of HBtSH in section 4.6.1 under encode-then-encipher paradigm.



(a) HBtSH (b) HtS

Figure 4.6: HBtSH and HtS: Tweakable wide block cipher scheme based on tweakable stream cipher. HBtSH is based on HBSH and HtS is based on HCTR2. The construction of HtS is similar to HBtSH and hence, we omit the CMT-4 security proof of HtS.

4.4.2 HtS

HtS is based on HCTR2. The tweakable stream cipher S is used instead of the stream encryption $XCTR_K$ (refer figure 4.1), where K is the key. The invocation of S is given by the following expression:

$$S(K, S, T)[1; |V|]. \tag{4.11}$$

The key stream generated by equation (4.11) is *xored* with V to obtain Z. We present the construction of HtS in figure 4.6b and the pseudocode of HtS in presented the algorithm 10.

At this stage, we want to point out that the construction of HtS eliminates the use of stream encryption .i.e., $XCTR_K(S) = E_K(S \oplus \text{bin}(1)) \parallel E_K(S \oplus \text{bin}(2)) \parallel \ldots$, which makes the construction similar to HBtSH, and hence we omit the CMT-4 security proof of HtS under encode-then-encipher paradigm.

Algorithm 9 : HBtSH

$\mathcal{E}(K, T, P)$

1: $(K_E \parallel K_H) \leftarrow S(K, \epsilon, \epsilon)[1; |K_H| + |K_E|]$ ▷ *Derive* K_H *and* K_E *from* K
2: $(P_R \parallel P_L) \leftarrow P, |P_L| = n$
3: $P_M \leftarrow P_R \boxplus H_{K_H}(T, P_L)$
4: $C_M \leftarrow E_{K_E}(P_M)$
5: $C_L \leftarrow P_L \oplus S(K_S, C_M, T)[1; |P_L|]$
6: $C_R \leftarrow C_M \boxminus H_{K_H}(T, C_L)$
7: $C \leftarrow (C_L \parallel C_R)$
8: **Return** C

$\mathcal{D}(K, T, C)$

1: $(K_E \parallel K_H) \leftarrow S(K, \epsilon, \epsilon)[1; |K_H| + |K_E|]$ ▷ *Derive* K_H *and* K_E *from* K
2: $(C_R \parallel C_L) \leftarrow C, |C_L| = n$
3: $C_M \leftarrow C_R \boxplus H_{K_H}(T, C_L)$
4: $P_L \leftarrow C_L \oplus S(K_S, C_M, T)[1; |C_L|]$
5: $P_M \leftarrow E^{-1}_{K_E}(C_M)$
6: $P_R \leftarrow P_M \boxminus H_{K_H}(T, P_L)$
7: $P \leftarrow (P_L \parallel P_R)$
8: **Return** P

4.4.3 tS-Double-Decker and tS-Docked-Double-Decker

In this section, we present two new constructions of tweakable wide block cipher scheme, tS-double-decker based on double-decker, and tS-docked-double-decker based on the docked-double-decker. Similar to HBtSH, both of these constructions are based on a tweakable stream cipher S. We describe each of the schemes in section 4.4.3.1 and section 4.4.3.2. We start with the description of the key derivation function for tS-double-decker and tS-docked-double-decker. We use the key K_2 for the second instance of the tweakable stream cipher and also use the key K_2 with the empty instantiation of tweakable stream cipher for generating the other two keys .i.e., K and K_1:

$$S(K_2, \epsilon, \epsilon)[1; |K| + |K_1|]. \tag{4.12}$$

We want to point out that, it is necessary to generate the two keys K and K_1 using the key derivation function of equation (4.12) with K_2 as one of the input, due to the fact that, if the keys are unrelated then it will become trivial to find two keys (K', K'') producing the same output from the polynomial hash function.

Algorithm 10 : HtS

$\mathcal{E}(K, T, P)$

1: $\bar{h} \leftarrow E_K(\text{bin}(0))$
2: $L \leftarrow E_K(\text{bin}(1))$
3: $(U \parallel V) \leftarrow P$, where $|U| = n$
4: $UU \leftarrow U \oplus H_{\bar{h}}(T, V)$
5: $YY \leftarrow E_K(UU)$
6: $S \leftarrow (UU \oplus YY \oplus L)$
7: $Z \leftarrow V \oplus S(K, S, T)[1; |V|]$
8: $Y \leftarrow YY \oplus H_{\bar{h}}(T, Z)$
9: $C \leftarrow (Y \parallel Z)$
10: **Return** C

$\mathcal{D}(K, T, C)$

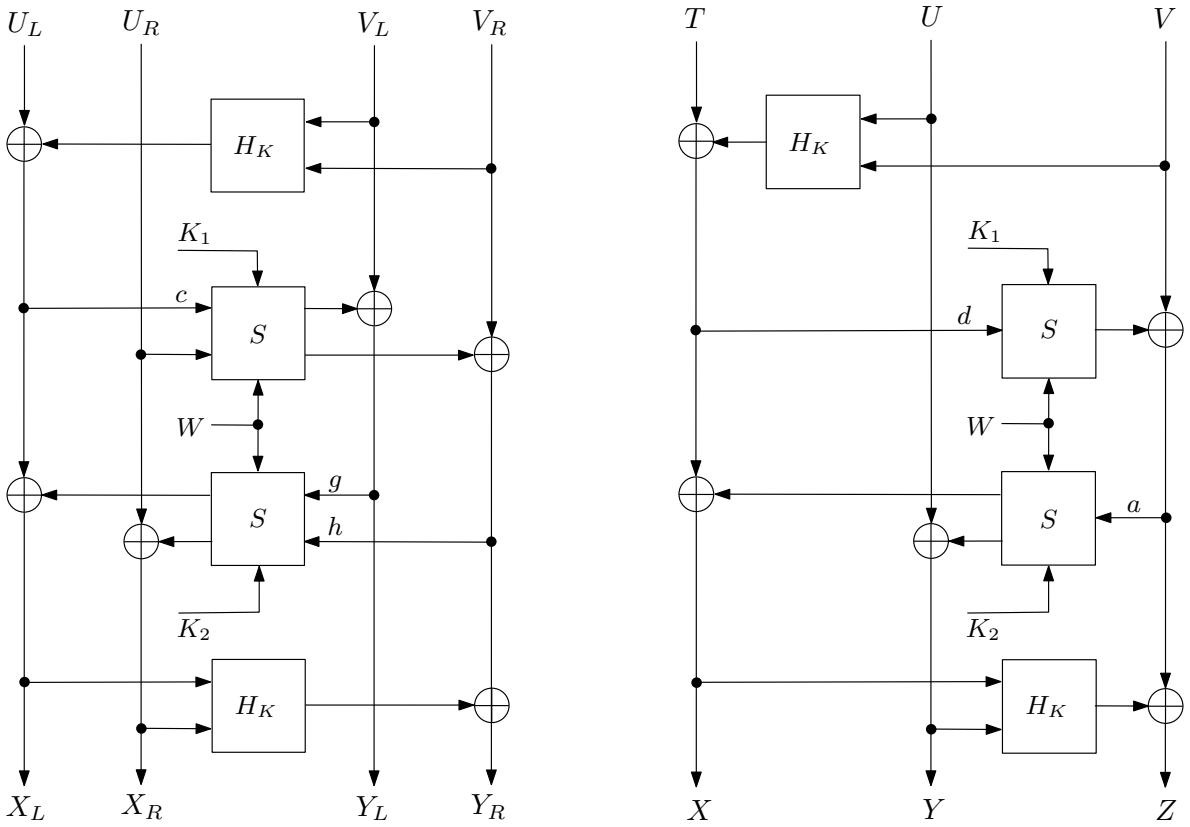
1: $\bar{h} \leftarrow E_K(\text{bin}(0))$
2: $L \leftarrow E_K(\text{bin}(1))$
3: $(Y \parallel Z) \leftarrow C$, where $|Y| = n$
4: $YY \leftarrow Y \oplus H_{\bar{h}}(T, Z)$
5: $UU \leftarrow E_K^{-1}(YY)$
6: $S \leftarrow (UU \oplus YY \oplus L)$
7: $V \leftarrow Z \oplus S(K, S, T)[1; |Z|]$
8: $U \leftarrow UU \oplus H_{\bar{h}}(T, V)$
9: $P \leftarrow (U \parallel V)$
10: **Return** P

4.4.3.1 tS-Double-Decker

In section 4.2.3.1, we saw that the construction of double-decker uses deck function two times .i.e., F_{K_1} with key K_1 and F_{K_2} with key K_2. We replace these two instances of the deck function with a tweakable stream cipher. Figure 4.7a shows the construction of ts-double-decker. The two instances of S for ts-double-decker is show below:

$$\begin{aligned} &S(K_1, c, \text{ENC}(U_R, W))[1; |V_L| + |V_R|] \\ &S(K_2, h, \text{ENC}(g, W))[1; |V_L| + |V_R|]. \end{aligned} \tag{4.13}$$

where $\text{ENC}(X, Y) = (\text{bin}_\eta(|X|) \parallel X \parallel Y)$ and $|X| < 2^\eta$. The key stream generated by the first instance of S in equation (4.13) is *xored* with V_L and V_R. The key stream generated by the second instance S is *xored* with U_L and U_R.



(a) tS-Double-Decker

(b) tS-Docked-Double-Decker

Figure 4.7: tS-Double-decker and tS-docked-double-decker: Tweakable wide block cipher scheme based on a tweakable stream cipher.

4.4.3.2 tS-Docked-Double-Decker

Similar to double-decker, docked-double-decker uses the deck function two times (refer section 4.2.3.2). We replace these two instances of the deck function by the tweakable stream cipher, hence resulting into ts-docked-double-decker (refer figure 4.7b). The two instances of S for ts-docked-double-decker is show below:

$$\begin{aligned} &S(K_1, d, W))[1; |V|] \\ &S(K_2, a, W))[1; |T| + |U|]. \end{aligned} \tag{4.14}$$

The key stream generated by S in equation (4.14) during the first instance is *xored* with V, while, in case of the second instance S, the key stream generated is *xored* with T and U.

In table 4.2, we present summary of all the four proposed tweakable wide block cipher schemes.

Table 4.2: Summary of proposed tweakable wide block cipher schemes.

Proposed Scheme	**CMT-4 Security Proof under EtE paradigm**
HBtSH (Section 4.4.1)	Theorem 5.
HtS (Section 4.4.2)	Omitted for brevity (design is similar to HBtSH).
tS-double-decker (Section 4.4.3)	Theorem 7.
tS-docked-double-decker (Section 4.4.3)	Theorem 7.

4.5 Design Rationale

In this section, we describe the design rationale behind the four proposed tweakable wide block cipher schemes.

- **Tweakable Stream Cipher (tS):** In section 4.3.1 and section 4.3.2, we saw that the CMT-4 attack on EtE-HBSH and EtE-HCTR2 was possible since the stream cipher in EtE-HBSH and the stream encryption in EtE-HCTR2 did not take the tweak as the input. Further, for ts-double-decker and ts-docked-double-decker, the two deck functions based on Farfalle [3] construction was unable to counter CMT-4 attack. The tweakable stream cipher ensures partial collision resistance against a CMT-4 adversary under encode-then-encipher paradigm. In this work, we use XOF [124] as an instantiation to the S.

  XOF-based Tweakable Stream Cipher:

  1. Security Claim: Let a sponge function be based on an ideal permutation and total bit length of data be at most σ, then the indifferentiability of the sponge function from a random oracle is given by the expression: $\leq \frac{\sigma^2}{2^{c+1}}$ [125], where c is the capacity of the sponge function. Now, for performing μ-bit partial collision on random oracle, its probability bound is given by the expression: $\leq \frac{q^2}{2^{\mu+1}}$, where q is the number of queries to the random oracle and μ is the number of bits for partial collision. Hence, the probability bound of the sponge function for performing μ-bit partial collision, where the underlying permutation is ideal, is given by the expression: $\leq \frac{\sigma^2}{2^{c+1}} + \frac{q^2}{2^{\mu+1}}$. For example, in case of SHAKE [126], the capacity c of SHAKE128 is 256, thereby making the probability bound for performing μ-bit partial collision on SHAKE128

to $\leq \frac{\sigma^2}{2^{257}} + \frac{q^2}{2^{\mu+1}}$, assuming underlying permutation of SHAKE128 is ideal, while in case of SHAKE256, $c = 512$, hence, the probability bound for performing μ-bit partial collision on SHAKE256 is $\leq \frac{\sigma^2}{2^{513}} + \frac{q^2}{2^{\mu+1}}$, where underlying permutation of SHAKE256 is ideal.

2. Implementation: TurboSHAKE [127] is a family of eXtendable Output Functions (XOF) with reduced round, that can be used for the applications requiring fast computation or hardware that have low computing power. Hence, TurboSHAKE may be suitable for application curated for IoT devices like disk encryption scenarios. Further, XOF is an active area of research and future advancements might result into more efficient designs.

- **Recycle Proofs:** The construction of HBtSH, HtS, tS-double-decker and tS-docked-double-decker ensures a minimal change in the original design .i.e., HBSH, HCTR2, double-decker and docked-double-decker respectively. The proposed designs are structurally similar to their original counter parts, hence verification of the proposed designs are intuitive. The security proofs of the original design can be recycled, giving us the leverage to provide only the CMT-4 security proof for the respective proposed designs under encode-then-encipher paradigm. The will further lead to easy verification by the research community.

- **Facilitate Existing Implementations:** Due to the structural similarity of all the four proposed designs, it fairly easy task to replace the existing implementations with the new proposed ones. This will further lead to easy benchmarking under standard metrics. Moreover, the property of structural similarity in the proposed designs will facilitate the replacement in the existing deployments.

4.6 Provable Security Proofs

4.6.1 Security Proof of HBtSH

We begin with some definitions. Let $\mathcal{K} \xleftarrow{\$} \{0,1\}^k$ denote the key space with key size equal to k, $\mathcal{M} \in \{0,1\}^*$ is the message space and $\mathcal{T} \in \{0,1\}^*$ denotes the space of the tweak.

HBtSH (algorithm 9) takes in three inputs: a key $K \in \mathcal{K}$, a tweak $T \in \mathcal{T}$ and a message (or a plaintext) $P \in \mathcal{M}$, and outputs a ciphertext $C \in \mathcal{C}$, where $\mathcal{C}$ is the ciphertext space and $|M| = |C|$. In case of CMT-4 security of an authenticated encryption scheme, the task of the adversary $\mathcal{A}$, is to construct two distinct tuples (K_1, N_1, A_1, P_1) and (K_2, N_2, A_2, P_2) for the encryption sub-routine of EtE-HBtSH, such that $\tau = \{(K_1, N_1, A_1, P_1), (K_2, N_2, A_2, P_2)\}$, where $(K_1, N_1, A_1, P_1) \neq (K_2, N_2, A_2, P_2)$ and $\mathcal{E}(K_1, N_1, A_1, P_1) = \mathcal{E}(K_2, N_2, A_2, P_2)$. In case of EtE-HBtSH, $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$. CMT-4 security is defined in definition 13 and EtE-HBtSH is defined in definition 14. We now present the proof of CMT4 security of EtE-HBtSH in **theorem** 5.

Theorem 5. *Let $\mathcal{A}$ be a CMT-4 adversary defined in algorithm 2 that has access to EtE-HBtSH with μ-bit zero prepending, and creates two distinct tuples, $\tau = \{(K_1, N_1, A_1, P_1), (K_2, N_2, A_2, P_2)\}$ s.t $C_1 \leftarrow \mathcal{E}(K_1, N_1, A_1, P_1)$, $C_2 \leftarrow \mathcal{E}(K_2, N_2, A_2, P_2)$, and $T_i = (N_i \parallel A_i)$, $i \in \{1, 2\}$, where $(K_1, N_1, A_1, P_1) \neq (K_2, N_2, A_2, P_2)$ and $C_1 = C_2$, then, we can construct a partial collision finding adversary $\mathcal{B}_\mathcal{A}$ on tS given by:*

$$\text{Adv}^{\text{CMT-4}}_{\text{EtE-HBtSH}}(\mathcal{A}) \leq \text{Adv}^{\mu\text{-PartialColl}}_{\text{tS}}(\mathcal{B}_\mathcal{A}). \tag{4.15}$$

Proof. We start by presenting a algorithm (refer Algorithm 11) that transforms the query executed by the adversary $\mathcal{A}$ as a halting problem. The adversary $\mathcal{B}_\mathcal{A}$ takes input from the adversary $\mathcal{A}$ and executes the partial collision attack.

Note that in case of HBtSH, we combine the nonce and the associated data to form the tweak .i.e., $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$. We now proceed to calculate the CMT-4 bound of EtE-HBtSH using following two cases:

- **Case 1:** $(K_1, T_1) = (K_2, T_2)$

  For the two tuples generated by $\mathcal{A}$ .i.e., $\{(K_1, N_1, A_1, P_1), (K_2, N_2, A_2, P_2)\} \xleftarrow{\$} \mathcal{A}$, we have $(K_1, T_1) = (K_2, T_2)$, where $(N_1 \parallel A_1) \leftarrow T_1$ and $(N_2 \parallel A_2) \leftarrow T_2$. Now from the construction of HBtSH in section 4.4.1 and figure 4.6a, we know that the tweakable stream cipher S is defined in the following way: $S(K_1, C_{M_1}, T_1)$ and $S(K_2, C_{M_2}, T_2)$, where C_{M_1}

Algorithm 11 : $\mathcal{B}_{\mathcal{A}}$

$\tau \leftarrow \phi$
Run $\mathcal{A}$**:**
  Generate $\{(K_1, N_1, A_1, P_1), (K_2, N_2, A_2, P_2)\} \xleftarrow{\$} \mathcal{A}$ ▷ $\forall i\ T_i \leftarrow (N_i \parallel A_i)$
  Execute: $C_1 \leftarrow \mathcal{E}(K_1, N_1, A_1, P_1)$, $C_2 \leftarrow \mathcal{E}(K_2, N_2, A_2, P_2)$
Halt: $(C_1 \stackrel{?}{=} C_2) \wedge (K_1, N_1, A_1, P_1) \stackrel{?}{=} (K_2, N_2, A_2, P_2)$
  if $(C_1 = C_2) \wedge (K_1, N_1, A_1, P_1) \neq (K_2, N_2, A_2, P_2)$
    then $(P_{L_1} \parallel P_{R_1}) \leftarrow P_1$, $(P_{L_2} \parallel P_{R_2}) \leftarrow P_2$ ▷ Generate inputs of tS
      $P'_{L_1} \leftarrow (0^{\mu} \parallel P_{L_1})$ and $P'_{L_2} \leftarrow (0^{\mu} \parallel P_{L_2})$
      $(K_{1_E}, K_{1_H}) \leftarrow S(K_1, \epsilon, \epsilon)[1; |K_{1_E}| + |K_{1_H}|]$
      $(K_{2_E}, K_{2_H}) \leftarrow S(K_2, \epsilon, \epsilon)[1; |K_{2_E}| + |K_{2_H}|]$
      $P_{M_1} \leftarrow P_{R_1} \boxplus H_{K_{1_H}}(T_1, P'_{L_1})$
      $P_{M_2} \leftarrow P_{R_2} \boxplus H_{K_{2_H}}(T_2, P'_{L_2})$
      $C_{M_1} \leftarrow E_{K_{1_E}}(P_{M_1})$, $C_{M_2} \leftarrow E_{K_{2_E}}(P_{M_2})$
      $\tau = \{(K_1, C_{M_1}, T_1), (K_2, C_{M_2}, T_2)\}$
return τ

and C_{M_2} are the two nonces. The key stream generated by the tweakable stream cipher S is used to generate the left part of the ciphertext .i.e., $C_{L_1} \leftarrow P'_{L_1} \oplus S(K_1, C_{M_1}, T_1)[1; |P'_{L_1}|]$ and $C_{L_2} \leftarrow P'_{L_2} \oplus S(K_2, C_{M_2}, T_2)[1; |P'_{L_2}|]$, where $P'_{L_1} \leftarrow (0^{\mu} \parallel P_{L_1})$, $P'_{L_2} \leftarrow (0^{\mu} \parallel P_{L_2})$, $(P_{L_1} \parallel P_{R_1}) \leftarrow P_1$ and $(P_{L_2} \parallel P_{R_2}) \leftarrow P_2$. Now, since S is a permutation, it is not possible to generate two different outputs .i.e., C_{L_1} and C_{L_2}, given the same inputs, since $(K_1, T_1) = (K_2, T_2)$. This leads to the fact that, in this case, it is not possible to generate two different outputs from the tweakable stream cipher, hence, in equation (4.15), we have $\text{Adv}^{\text{CMT-4}}_{\text{EtE-HBtSH}}(\mathcal{A}) = 0$.

- **Case 2:** $(K_1, T_1) \neq (K_2, T_2)$

  In this case, the tweakable stream cipher takes two distinct tuples .i.e., $S(K_1, C_{M_1}, T_1)$ and $S(K_2, C_{M_2}, T_2)$. Under encode-then-encipher paradigm, we know from definition 14 and equation (4.1), that in case of EtE-HBtSH, we have $P'_{L_1} \leftarrow (0^{\mu} \parallel P_{L_1})$ and $P'_{L_2} \leftarrow (0^{\mu} \parallel P_{L_2})$, where $(P_{L_1} \parallel P_{R_1}) \leftarrow P_1$, $(P_{L_2} \parallel P_{R_2}) \leftarrow P_2$ and $\mu > 0$. Hence, to generate the same ciphertext part $C_L = C_{L_1} = C_{L_2}$, the tweakable stream cipher must produce μ-bit partial collision in its output. This leads to the fact that the game in Algorithm 2 returns 1, making the game in Algorithm 1 to return 1 as well. Hence, using definition 13 and definition 12, we have $Pr[\mathbf{G}^{\text{CMT-4}}_{\text{EtE-HBtSH}}(\mathcal{A}) \mapsto 1] = Pr[\mathbf{G}^{\mu\text{-PartialColl}}_{\text{tS}}(\mathcal{B}_{\mathcal{A}}) \mapsto 1]$ .i.e., $\text{Adv}^{\text{CMT-4}}_{\text{EtE-HBtSH}}(\mathcal{A}) = \text{Adv}^{\mu\text{-PartialColl}}_{\text{tS}}(\mathcal{B}_{\mathcal{A}})$.

From case 1 and case 2, we conclude that $\text{Adv}^{\text{CMT-4}}_{\text{EtE-HBtSH}}(\mathcal{A}) \leq \text{Adv}^{\mu\text{-PartialColl}}_{\text{tS}}(\mathcal{B}_{\mathcal{A}})$, which proofs our claim in equation (4.15). Hence, we conclude our CMT-4 security proof of EtE-HBtSH. □

We now present the security proof of HBtSH in theorem 6 below.

Theorem 6. *Let HBtSH be the scheme defined in section 4.4.1, where H is ϵ-almost-Δ-universal for inputs, then we define:*

$$\begin{aligned}\text{Adv}^{\pm\widetilde{PRP}}_{\text{HBtSH}}(q, l_T, l_M, t) \leq (\epsilon + \frac{2}{2^n}) \bullet \binom{q}{2} \\ + \text{Adv}^{\text{tS}}_{S}(q+1, l_T, l_M + |K_E| + |K_H| - q \bullet n, t') \\ + \text{Adv}^{\pm\text{PRP}}_{E}(q, t'),\end{aligned}$$

where K_E is key for E, K_H is key for H, n is the output size of E and $t' = t + O(l_T + l_M)$.

Proof. The design of HBtSH is similar to HBSH with the exception of a tweakable stream cipher (definition 8), instead of a stream cipher, and therefore we leave the proof for brevity. We encourage the reader to refer theorem 1 in [83] for the proof. □

4.6.2 Security Proof of tS-Double-Decker and tS-Docked-Double-Decker

We present the CMT-4 proof of EtE-tS-double-decker and EtE-tS-docked-double-decker in theorem 7. In case of EtE-ts-double-decker, μ bit zero prepending is done on U_R, while for EtE-ts-docked-double-decker, we use U for μ bit zero prepending. Further, please note that, we use key K_2 for the second instantiation of the tweakable stream cipher and generate the other two keys K and K_1 using key K_2 (refer equation (4.12)).

Theorem 7. *Let Π be EtE-tS-Double-Decker or EtE-tS-Docked-Double-Decker created using tS-Double-Decker and tS-Docked-Double-Decker respectively, as defined in section 4.4.3.1 and section 4.4.3.2. Let $\mathcal{A}$ be a CMT-4 adversary defined in algorithm 2 that has access to*

Π *with* μ*-bit zero prepending to* U_R *or* U *respectively, and creates two distinct tuples* $\tau = \{(K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2)\}$ *s.t* $C_1 \leftarrow \mathcal{E}(K_1, N_1, A_1, M_1)$, $C_2 \leftarrow \mathcal{E}(K_2, N_2, A_2, M_2)$, *and* $W_i = (N_i \parallel A_i)$, $i \in \{1, 2\}$, *where* $(K_1, N_1, A_1, M_1) \neq (K_2, N_2, A_2, M_2)$ *and* $C_1 = C_2$, *then, we can construct a partial collision finding adversary* $\mathcal{B}_\mathcal{A}$ *on tS given by:*

$$\text{Adv}^{\text{CMT-4}}_{\Pi}(\mathcal{A}) \leq \text{Adv}^{\mu\text{-PartialColl}}_{\text{tS}}(\mathcal{B}_\mathcal{A}). \tag{4.16}$$

Proof. The proof is similar to theorem 5, hence we omit it for brevity. □

Theorem 8. *Let tS-Double-Decker be the scheme defined in section 4.4.3.1, where* H *is blinded keyed hash, then we define:*

$$\begin{aligned}
&\text{Adv}^{\pm\widetilde{PRP}}_{\text{tS-Double-Decker}}(q, l_W, l_M, t) \leq \\
&\quad \text{Adv}^{\text{tS}}_{S}(q, l_W + l_{U_R}, l_M - l_{U_R} - q \bullet n, t') \\
&\quad + \text{Adv}^{\text{tS}}_{S}(q + 1, l_W + l_{V_L}, l_M \\
&\qquad + |K| + |K_1| - l_{V_L} - q \bullet n, t') \\
&\quad + \sum_{W \in \mathcal{W}} \Bigg(\text{Adv}^{\text{bhk}}_{H}(q_W, \sigma_{H_1,W}) + \text{Adv}^{\text{bhk}}_{H}(q_W, \sigma_{H_2,W}) \\
&\qquad + \binom{q_W}{2} \bullet 2^{-2 \bullet n} \Bigg),
\end{aligned}$$

where l_{U_R} and l_{V_L} are the total lengths of U_R and V_L respectively (refer figure 4.7a), q_W is the total number of queries with tweak W and $\sigma_{H_1,W}$, $\sigma_{H_2,W}$ are the data complexities of the two instances of H .i.e., H_1 and H_2 respectively, K is key for H, K_1 is key for the first instance of S, n is the outside branch length .i.e., $|U_L| = |V_R| = n$ and $t' = t + O(l_T + l_M)$.

Proof. The design of tS-double-decker is similar to double-decker with the exception of a tweakable stream cipher (definition 8), instead of a deck function, and therefore we leave the proof for brevity. We encourage the reader to refer theorem 1 in [85] for the proof. □

Theorem 9. *Let tS-Docked-Double-Decker be the scheme defined in section 4.4.3.2, where H is blinded keyed hash, then we define:*

$$\begin{aligned}
&\mathrm{Adv}^{\pm\widetilde{PRP}}_{\text{tS-Docked-Double-Decker}}(q, l_W, l_M, t) \leq \\
&\quad \mathrm{Adv}^{\mathrm{tS}}_{S}(q, l_W, l_M - q \bullet n), t') \\
&\qquad + \mathrm{Adv}^{\mathrm{tS}}_{S}(q+1, l_W, l_M \\
&\qquad\qquad + |K| + |K_1| - q \bullet n, t') \\
&\qquad + \sum_{W \in \mathcal{W}} \Bigg(\mathrm{Adv}^{\mathrm{bhk}}_{H}(q_W, \sigma_{H_1,W}) + \mathrm{Adv}^{\mathrm{bhk}}_{H}(q_W, \sigma_{H_2,W}) \\
&\qquad\qquad + \binom{q_W}{2} \bullet 2^{-2 \bullet n} \Bigg),
\end{aligned}$$

where q_W is total number of queries with tweak W and $\sigma_{H_1,W}$, $\sigma_{H_2,W}$ are the data complexities of the two instances of H .i.e., H_1 and H_2 respectively, K is key for H, K_1 is key for the first instance of S, n is outside branch length .i.e., $|T| = |V| = n$ and $t' = t + O(l_T + l_M)$.

Proof. The design of tS-docked-double-decker is similar to docked-double-decker with the exception of a tweakable stream cipher (definition 8), instead of a deck function, and therefore we leave the proof for brevity. We encourage the reader to refer theorem 1 in [85] for the proof. □

4.7 Conclusion

In this paper, we provide detail analysis of CMT-4 security of four tweakable wide block cipher schemes under encode-then-encipher paradigm. We analyze the scenario of creating authentication encryption schemes by prepending zeros. We presented successful CMT-4 attack on EtE-HBSH in section 4.3.1, EtE-HCTR2 in section 4.3.2, and the two deck function based constructions in section 4.3.3 .i.e., EtE-double-decker in section 4.3.3.1 and EtE-docked-double in section 4.3.3.2 respectively. All the four attacks have time complexity $O(1)$ (refer table 4.1). We introduce the notion of tweakable stream cipher (or tS in short) in definition 8. We use tS

to design four new tweakable wide block cipher schemes namely, HBtSH (refer section 4.4.1), HtS (refer section 4.4.2), tS-double-decker (refer section 4.4.3.1) and tS-docked-double-decker (refer section 4.4.3.2). All the four proposed schemes provide partial collision resistance against a CMT-4 adversary under encode-then-encipher paradigm (refer theorem 5 and theorem 7). Further, we provide security proof of HBtSH in theorem 6, ts-double-decker in theorem 8 and ts-docked-double-decker in theorem 9.

The notion of tweakable stream cipher opens a new direction. Several constructions can be analyzed for CMT-4 security under encode-then-encipher paradigm, and tweakable stream cipher can be explored to see the resilience against a CMT-4 adversary for the analyzed constructions. Further, in this paper, we use eXtendable-Output Function (or XOF) (refer definition 9) for creating a tweakable stream cipher. It can be of interest to explore other primitives that can be used as a tweakable stream cipher. From the implementation perspective, a tS based scheme may seem to be slower than a traditional stream cipher based construction or a deck function based construction, but on a positive note, XOF is an active area, where attempts are made to make it faster by reducing the number of rounds [127]. This also opens up a future research work in this area. Further, since, the committing security is very practical in the real-world setting, it is always good know the resilience of authenticated encryption schemes against such attacks.

Chapter 5

Conclusion and Future Work

Motivated by the need to provide an authenticated encryption scheme with associated data in the lightweight category, we provided all the ways to create an AEAD using a tweakable blockcipher with tweak size as twice the size of input.

5.1 Summary

In this thesis, we focused to fill the gap of cryptographic designs based on tweakable primitives. We explore the area of lightweight cryptography and present following:

1. Lynx

   We analyzed all the ways of creating an authenticated encryption scheme with associated using *xor* operation and a tweakable blockcipher, where the size of the tweak is twice the size of the block. In this work we designed two functions $\mathcal{F}^A/\overline{\mathcal{F}^A}$ and $\mathcal{F}^B/\overline{\mathcal{F}^B}$ using the tweakable blockcipher and showed 72 ways to create an AEAD using these two functions. We analyzed the construction of each of these 72 functions and categorized the ones (58 in number) that cannot be used for creating an AEAD into: (1) implausible Cases, (2) non-confidential cases, and (2) non-integrity Cases.

   We proposed family of lightweight authenticated encryption scheme called with associated lynx having 14 members in the family, with lynx-A using $\mathcal{F}^A$ for creating AEAD and lynx-

B using and $\mathcal{F}^B$ for creating AEAD. Further, we provide provable security proof for each of these members of the lynx family. We show that lynx provides birthday bound security for confidentiality under nonce respecting scenario, and provides birthday bound security for integrity under nonce misuse and RUP scenario. The security bounds we achieve for lynx is in line with the NIST lightweight competition requirements [103]. From the implementation perspective, we profiled lynx on five different hardware architecture with size different implementations. Each of the implementations outperformed Romulus [9] on all the platforms. Lynx has several properties that makes it ideal to be used in the lightweight category for the resource constraint devices:

- *Inverse-Free*: Lynx-A is inverse-free .i.e., it does not need the decryption algorithm of the underlying primitive (tweakable blockcipher where the tweak size is twice the size of the input). This saves memory for the devices that target lightweight category.
- *Online* or *Stream processing*: The current block processing needs no information about the block ahead or the total length of the message. The property is useful for real time applications like video streaming etc., but the this property shines when applied to resource constraint devices. These devices are computationally menial and have limited memory, hence the property of *stream processing*, the device is relieved from already processed blocks. This property is supported by both lynx-A as well as lynx-B.
- *One-Pass*: The encryption sub-routine $\mathcal{E}$ lynx family requires only one scan to process through the entire data (associated data and message) to produce the ciphertext and the tag. Similarly, the decryption sub-routine $\mathcal{D}$ also does single scan to output corresponding message if tag verification is successful else $\perp$.
- *Rate*: The lynx family is rate 1 and hence, have shorter latency and hence achieves higher speed.
- *INT-RUP Security*: Lynx provides birthday bound security for confidentiality in nonce respecting scenario. For integrity security, lynx provides birthday bound security in nonce misuse and RUP scenario. This property further complements the property of *stream processing*.

- *Performance*: Lynx has a simplistic structure, and is free from any matrix computations or field multiplications. This structure makes the lynx family suitable for low-powered devices, since matrix or field operations are costly on these devices.

2. Context-Committing Authenticated Encryptions using Tweakable Stream Cipher

   We created AEAD using four tweakable wide blockciphers namely, HBSH, HCTR2, double-decker and docked-double-decker under encode-then-encipher (EtE in short) paradigm. We use the approach of prepending zeros for creating the AEAD. Let μ be the number of zeros that is prepended to the input message, where $\mu > 0$. Under this setup, we provide detailed analysis the CMT-4 security of the created AEADs under this paradigm (EtE-HBHS, EtE-HCTR2, EtE-double-decker and EtE-docked-double-decker) . We provide successful CMT-4 attack against all the four AEAD constructions with a time complexity of $O(1)$. This shows that none of these constructions are CMT-4 secure under EtE paradigm.

   We propose a new tweakable primitive called, tweakable stream cipher (or tS in short) which has the property of partial collision resistance. Tweakable stream cipher takes three inputs .i.e., a key, a nonce and a tweak and generates a key stream as the output. The key stream can be used for message encryption (or decryption in case of ciphertext). Further, we demonstrate the procedure for creating a tweakable stream cipher using eXtendable-Output Function (XOF). As an example, let us consider SHAKE [126], specifically SHAKE128. The capacity of c of SHAKE128 is 256, and hence the probability bound for performing μ-bit partial collision on SHAKE128 to $\leq \frac{\sigma^2}{2^{257}} + \frac{q^2}{2^{\mu+1}}$, where we assume that the underlying permutation of SHAKE128 is ideal.

   Using this proposed primitive .i.e., tweakable stream cipher, we introduced four new constructions of tweakable wide blockcipher schemes, namely (1) HBtSH (Hash tweakable Stream cipher Hash), (2) HtS (Hash tweakable stream cipher), (3) tS-double-decker (tweakable stream cipher-double-docker) and, (4) tS-docked-double-decker (tweakable stream cipher-docked-double-docker). We show that using these proposed tweakable wide blockcipher schemes to create an AEAD under encode-then-encipher paradigm provides partial collision resistance against CMT-4 security.

5.2 Future Work

In this section, we attempt to identify and present some of the future research directions spanning the topics discussed in this thesis.

- We presented all the ways to create authenticated encryption scheme with associated data using *xor* operations and a tweakable blockcipher, where the tweak size is twice the block size. This leads to some imperative questions like what other operations can be used to create such a family of AEAD? What about other variants of the tweakable blockcipher, for example such that the tweak size is equal to the block size etc. Further, what properties of the lightweight category can still hold for the new variants. Is it possible to add several more properties? Is possible to provide beyond birthday bound (or BBB in short) under new set of operations?

- Lynx currently lacks hardware implementation and it would be a great direction to investigate and profile the performance of lynx family on FPGA. Such implementations would require to minimization of the table lookup as well the miniaturization of code. A comparison with existing hardware implementations will show the true applicability of lynx in the lightweight category.

- The methodology of creating an AEAD using encode-then-encipher paradigm has not gain much attention from the research community, and it has only recently started to gain lot of popularity. We showed successful CMT-4 attack with time complexity $O(1)$ on four different tweakable wide blockcipher schemes under EtE paradigm. This provides a future direction about other tweakable wide blockcipher schemes, where one can perform CMT-4 analysis of other schemes under EtE paradigm. Further, we proposed the notion of tweakable stream cipher to counter the above attack. This opens up another front one can ask question that: What are other ways to counter such attacks?

- Committing security is one of the most active area of research in the cryptographic community. Most of the standardization procedures are demanding this additional feature. Hence, this leads to the research direction of providing committing security analysis of the AEAD.

In this thesis, we explore the idea of specific setup for the analysis of committing security of an AEAD. As a feature work, it would be good to analyze the generic approach for the security analysis and design as outlined in [49].

References

[1] "Tweakable block cipher based cryptography," https://thomaspeyrin.github.io/web/assets/docs/invited/FSE2020_slides.pdf, Accessed: October 12, 2024.

[2] M. Hasan and D. Chang, "Context-committing authenticated encryptions using tweakable stream cipher," *IEEE Access*, 2024.

[3] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, "Farfalle: parallel permutation-based cryptography," *Cryptology ePrint Archive*, 2016.

[4] P. Crowley, "Mercy: A fast large block cipher for disk sector encryption," in *Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings 7*. Springer, 2001, pp. 49–63.

[5] S. R. Fluhrer, "Cryptanalysis of the mercy block cipher," in *Fast Software Encryption: 8th International Workshop, FSE 2001 Yokohama, Japan, April 2–4, 2001 Revised Papers 8*. Springer, 2002, pp. 28–36.

[6] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable Block Ciphers," in *Annual International Cryptology Conference*. Springer, 2002, pp. 31–46.

[7] "Lightweight Cryptography," https://csrc.nist.gov/Projects/Lightweight-Cryptography, 2018.

[8] "National Institute of Standards and Technology," https://www.nist.gov/.

[9] T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin, "Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms," *IACR Transactions on Symmetric Cryptology*, pp. 43–120, 2020.

[10] "Proposal of Requirements for an Accordion Mode," https://csrc.nist.gov/files/pubs/other/2024/04/10/proposal-of-requirements-for-an-accordion-mode-dis/iprd/docs/proposal-of-requirements-for-an-accordion-mode-discussion-draft.pdf, 2024.

[11] M. Hasan and D. Chang, "Lynx: Family of lightweight authenticated encryption schemes based on tweakable blockcipher," *IEEE Internet of Things Journal*, 2023.

[12] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 531–545.

[13] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 98–107.

[14] C. S. Jutla, "Encryption modes with almost free message integrity," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 529–544.

[15] V. D. Gligor and P. Donescu, "Fast encryption and authentication: Xcbc encryption and xecb authentication modes," in *International Workshop on Fast Software Encryption*. Springer, 2002, pp. 92–108.

[16] P. Rogaway, M. Bellare, and J. Black, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[17] "ADVANCED ENCRYPTION STANDARD (AES)," https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=901427, 2001.

[18] "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf, 2007.

[19] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of computer and system sciences*, vol. 22, no. 3, pp. 265–279, 1981.

[20] A. Chakraborti, N. Datta, A. Jha, and M. Nandi, "Structural classification of authenticated encryption schemes."

[21] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT-COFB," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 738, 2020.

[22] P. Rogaway and T. Shrimpton, "A Provable-Security Treatment of the Key-Wrap Problem," in *Advances in Cryptology - EUROCRYPT 2006*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 373–390.

[23] "AES-GCM-SIV: Nonce misuse-resistant authenticated encryption," https://www.rfc-editor.org/rfc/rfc8452.html, 2019.

[24] S. Gueron and Y. Lindell, "GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 109–119.

[25] V. T. Hoang, T. Krovetz, and P. Rogaway, "Robust Authenticated-Encryption AEZ and the Problem That It Solves," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 15–44.

[26] L. Akhmetzyanova, E. Alekseev, A. Babueva, A. Bozhko, and S. Smyshlyaev, "Misuse-resistant mgm2 mode," *International Journal of Open Information Technologies*, vol. 10, no. 1, pp. 6–14, 2021.

[27] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda, "How to securely release unverified plaintext in authenticated encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 105–125.

[28] "TinyJAMBU," https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/TinyJAMBU-spec.pdf, 2019.

[29] A. Chakraborti, N. Datta, A. Jha, C. Mancillas-López, M. Nandi, and Y. Sasaki, "Int-rup secure lightweight parallel ae modes," *IACR Transactions on Symmetric Cryptology*, pp. 81–118, 2019.

[30] M. Bellare and B. Tackmann, "The multi-user security of authenticated encryption: Aes-gcm in tls 1.3," in *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I 36*. Springer, 2016, pp. 247–276.

[31] S. B. Sadkhan and A. O. Salman, "A survey on lightweight-cryptography status and future challenges," in *2018 International Conference on Advance of Sustainable Engineering and its Application (ICASEA)*. IEEE, 2018, pp. 105–108.

[32] "Lightweight cryptography lounge," http://cryptolux.org/index.php/Lightweight_Cryptography, 2015.

[33] H. Yoshida, D. Watanabe, and N. Mouha, "On the status of techniques and standardization regarding lightweight cryptography–iso/iec jtc1/sc27/wg2 status report," *IEICE Technical Report; IEICE Tech. Rep.*, vol. 114, no. 340, pp. 25–30, 2014.

[34] N. Mouha, "The design space of lightweight cryptography," *Cryptology ePrint Archive*, 2015.

[35] "Caesar: Competition for Authenticated Encryption: Security, Applicability, and Robustness," http://competitions.cr.yp.to/caesar.html.

[36] Z. Bao, A. Chakraborti, N. Datta, J. Guo, M. Nandi, T. Peyrin, and K. Yasuda, "PHOTON-Beetle Authenticated Encryption and Hash Family," https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/photon-beetle-spec-round2.pdf, 2019.

[37] H. Wu, "ACORN v3," https://competitions.cr.yp.to/round3/acornv3.pdf, 2016.

[38] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2: Lightweight Authenticated Encryption and Hashing," *Journal of Cryptology*, vol. 34, no. 3, pp. 1–42, 2021.

[39] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," 2020.

[40] C. Beierle, A. Biryukov, L. C. dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, Q. Wang, and A. Biryukov, "Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family," *NIST round*, vol. 2, 2019.

[41] "Elephant v1.1," https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/elephant-spec-round2.pdf, 2019.

[42] "ISAP v2.0," https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/isap-spec-round2.pdf, 2019.

[43] "AET-LR: Rate-1 Leakage-Resilient AEAD based on the Romulus Family," https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/AET-LR-lwc2020.pdf, 2020.

[44] "OpenSSL," https://www.openssl.org/, 2022.

[45] "BoringSSL," https://github.com/boringssl/boringssl, 2022.

[46] "wolfSSL," https://www.wolfssl.com/, 2022.

[47] "Apple CryptoKit," https://developer.apple.com/documentation/cryptokit/, 2022.

[48] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda, "How to Securely Release Unverified Plaintext in Authenticated Encryption," in *Advances in Cryptology – ASIACRYPT 2014*, P. Sarkar and T. Iwata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 105–125.

[49] A. Bhattacharjee, R. Bhaumik, and C. Dhar, "Universal context commitment without ciphertext expansion," *Cryptology ePrint Archive*, 2024.

[50] P. Rogaway, "Nonce-based symmetric encryption," in *International workshop on fast software encryption*. Springer, 2004, pp. 348–358.

[51] Y. Dodis, P. Grubbs, T. Ristenpart, and J. Woodage, "Fast message franking: From invisible salamanders to encryptment," in *Advances in Cryptology–CRYPTO 2018: 38th*

Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38. Springer, 2018, pp. 155–186.

[52] A. Albertini, T. Duong, S. Gueron, S. Kölbl, A. Luykx, and S. Schmieg, "How to abuse and fix authenticated encryption without key commitment," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3291–3308.

[53] J. Len, P. Grubbs, and T. Ristenpart, "Partitioning oracle attacks," in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 195–212.

[54] P. Rogaway and T. Shrimpton, "A provable-security treatment of the key-wrap problem," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2006, pp. 373–390.

[55] M. Bellare, P. Rogaway, and D. Wagner, "The eax mode of operation," in *Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers 11*. Springer, 2004, pp. 389–407.

[56] M. Dworkin, "Block cipher modes of operation: The ccm mode for authentication and confidentiality," *NIST special publication*, vol. 800, p. 38C, 2003.

[57] S. Menda, J. Len, P. Grubbs, and T. Ristenpart, "Context discovery and commitment attacks: How to break ccm, eax, siv, and more," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 379–407.

[58] M. Bellare and V. T. Hoang, "Efficient schemes for committing authenticated encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 845–875.

[59] M. Bellare and P. Rogaway, "Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 317–330.

[60] S. Halevi and P. Rogaway, "A Tweakable Enciphering Mode," in *Annual International Cryptology Conference*. Springer, 2003, pp. 482–499.

[61] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 173–201.

[62] M. Bellare, "Practice-oriented provable-security," in *International workshop on information security*. Springer, 1997, pp. 221–231.

[63] M. Liskov, R. L. Rivest, and D. A. Wagner, "Tweakable Block Ciphers," in *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442. Springer, 2002, pp. 31–46.

[64] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The skein hash function family," *Submission to NIST (round 3)*, vol. 7, no. 7.5, p. 3, 2010.

[65] D. Goldenberg, S. Hohenberger, M. Liskov, E. C. Schwartz, and H. Seyalioglu, "On tweaking luby-rackoff blockciphers," in *Advances in Cryptology–ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. Proceedings 13*. Springer, 2007, pp. 342–356.

[66] T. Peyrin, "Tweaks and keys for block ciphers: the tweakey framework."

[67] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS," in *Advances in Cryptology – CRYPTO 2016*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 123–153.

[68] R. Avanzi, O. Dunkelman, and K. Minematsu, "Matter: A wide-block tweakable block cipher," *Cryptology ePrint Archive*, 2024.

[69] J. Jean, I. Nikolić, T. Peyrin, and Y. Seurin, "The deoxys aead family," *Journal of Cryptology*, vol. 34, no. 3, p. 31, 2021.

[70] M. Khairallah, S. Yadhunathan, and S. Bhasin, "Lightweight leakage-resilient prng from tbcs using superposition," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2024, pp. 197–217.

[71] H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?)," in *Annual International Cryptology Conference*. Springer, 2001, pp. 310–331.

[72] M. Katagi, S. Moriai *et al.*, "Lightweight cryptography for the internet of things," *Sony Corporation*, vol. 2008, pp. 7–10, 2008.

[73] "On Lightweightness," https://cryptolux.org/index.php/On_Lightweightness, 2015.

[74] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, "A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues," *Journal of Network and Computer Applications*, vol. 58, pp. 73–93, 2015.

[75] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–18, 2017.

[76] V. A. Thakor, M. A. Razzaque, and M. R. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," *IEEE Access*, vol. 9, pp. 28 177–28 193, 2021.

[77] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Radiogatún, a belt-and-mill hash function." *IACR Cryptol. ePrint Arch.*, vol. 2006, p. 369, 2006.

[78] G. Bertoni, J. Daemen, M. Peeters, and G. Assche, "Sponge Functions," *ECRYPT Hash Workshop 2007*, 01 2007.

[79] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda, "Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 218–241, 2018.

[80] S. Banik, S. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A Small Present," 08 2017, pp. 321–345.

[81] M. Hell, T. Johansson, W. Meier, J. Sönnerup, and H. Yoshida, "Grain-128AEAD-A lightweight AEAD stream cipher," *NIST Lightweight Cryptography, Round*, vol. 1, 2019.

[82] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain family of stream ciphers," in *New Stream Cipher Designs*. Springer, 2008, pp. 179–190.

[83] P. Crowley and E. Biggers, "{Adiantum}: length-preserving encryption for entry-level processors," *Cryptology ePrint Archive*, 2018.

[84] P. Crowley, N. Huckleberry, and E. Biggers, "Length-preserving encryption with HCTR2," *Cryptology ePrint Archive*, 2021.

[85] A. Gunsing, J. Daemen, and B. Mennink, "Deck-based wide block cipher modes," *Cryptology ePrint Archive*, 2022.

[86] W. F. Ehrsam, C. H. Meyer, J. L. Smith, and W. L. Tuchman, "Message verification and transmission error detection by block chaining," Feb. 14 1978, uS Patent 4,074,066.

[87] "Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices," https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38e.pdf, 2010.

[88] "Setting the Bitlocker encryption algorithm for Autopilot devices," https://learn.microsoft.com/en-us/autopilot/bitlocker, 2023.

[89] R. Fujita, T. Isobe, and K. Minematsu, "Ace in chains: How risky is cbc encryption of binary executable files?" in *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part I 18*. Springer, 2020, pp. 187–207.

[90] H. Wu and B. Preneel, "AEGIS: A fast authenticated encryption algorithm," in *Selected Areas in Cryptography–SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers 20*. Springer, 2014, pp. 185–201.

[91] T. Isobe and M. Rahman, "Key Committing Security Analysis of AEGIS," *Cryptology ePrint Archive*, 2023.

[92] Y. Naito, Y. Sasaki, and T. Sugawara, "Committing Security of Ascon: Cryptanalysis on Primitive and Proof on Mode," *IACR Transactions on Symmetric Cryptology*, vol. 2023, no. 4, pp. 420–451, 2023.

[93] ——, "KIVR: Context-Committing Authenticated Encryption Using Plaintext Redundancy and Application to GCM and Variants," *NIST*, 2023.

[94] J. Krämer, P. Struck, and M. Weishäupl, "Committing Authenticated Encryption: Sponges vs. block-ciphers in the case of the nist lwc finalists," *Cryptology ePrint Archive*, 2023.

[95] S. Gueron, "Key Committing AEADs," *Cryptology ePrint Archive*, 2020.

[96] D. Chang, N. Datta, A. Dutta, B. Mennink, M. Nandi, S. Sanadhya, and F. Sibleyras, "Release of unverified plaintext: Tight unified model and application to ANYDAE," *IACR Transactions on Symmetric Cryptology*, pp. 119–146, 2019.

[97] P. Wang, D. Feng, and W. Wu, "HCTR: A variable-input-length enciphering mode," in *International Conference on Information Security and Cryptology*. Springer, 2005, pp. 175–188.

[98] H. Krawczyk, "Lfsr-based hashing and authentication," in *Annual International Cryptology Conference*. Springer, 1994, pp. 129–139.

[99] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "SKINNY-AEAD and SKINNY-Hash," *IACR Transactions on Symmetric Cryptology*, pp. 88–131, 2020.

[100] E. Andreeva, V. Lallemand, A. Purnal, R. Reyhanitabar, A. Roy, and D. Vizár, "ForkAE v.1," *Submission to NIST Lightweight Cryptography Project*, 2019.

[101] A. Inoue, C. Guo, and K. Minematsu, "Nonce-misuse resilience of romulus-n and gift-cofb," *Cryptology ePrint Archive*, 2022.

[102] "Lightweight Cryptography Workshop 2020," https://csrc.nist.gov/Events/2020/lightweight-cryptography-workshop-2020, 2020.

[103] "Lightweight Cryptography Requirements," https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf, 2018.

[104] "Lightweight crypto, heavyweight protection," https://www.nist.gov/comment/97756, 2021.

[105] J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from PGV," in *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2442. Springer, 2002, pp. 320–335. [Online]. Available: https://iacr.org/archive/crypto2002/24420321/24420321.pdf

[106] M. Bellare and P. Rogaway, "Code-Based Game-Playing Proofs and the Security of Triple Encryption, Eurocrypt 2006, lncs vol. 4004," 2006.

[107] D. Chang, S. Lee, M. Nandi, and M. Yung, "Indifferentiable security analysis of popular hash functions with prefix-free padding," in *Advances in Cryptology – ASIACRYPT 2006*, X. Lai and K. Chen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 283–298.

[108] "Raspberry Pi OS," https://www.raspberrypi.com/software/, 2022.

[109] "Exynos 9820," https://semiconductor.samsung.com/processor/mobile-processor/exynos-9-series-9820/, 2019.

[110] "Raspberry Pi v3 Hardware," https://www.raspberrypi.com/products/raspberry-pi-3-model-b/, Accessed: October 12, 2024.

[111] Android Open Source Project, ""Enabling Adiantum on Android"." https://source.android.com/docs/security/features/encryption/adiantum, 2024.

[112] A. O. S. Project, "File-Based Encryption." https://source.android.com/docs/security/features/encryption/file-based, 2024.

[113] C. Dobraunig, K. Matusiewicz, B. Mennink, and A. Tereschenko, "Efficient instances of docked double decker with aes," *Cryptology ePrint Archive*, 2024.

[114] "The third nist workshop on block cipher modes of operation 2023," https://csrc.nist.gov/events/2023/third-workshop-on-block-cipher-modes-of-operation.

[115] "Key committing security of aez and more," https://csrc.nist.gov/csrc/media/Presentations/2023/key-committing-security-of-aez/images-media/sess-7-chen-bcm-workshop-2023.pdf.

[116] P. Grubbs, J. Lu, and T. Ristenpart, "Message franking via committing authenticated encryption," in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III 37*. Springer, 2017, pp. 66–97.

[117] B. Schneier and J. Kelsey, "Unbalanced feistel networks and block cipher design," in *International Workshop on Fast Software Encryption*. Springer, 1996, pp. 121–144.

[118] D. J. Bernstein *et al.*, "ChaCha, a variant of Salsa20," in *Workshop record of SASC*, vol. 8. Citeseer, 2008, pp. 3–5.

[119] N. I. of Standards and T. (NIST), "Advanced Encryption Standard (AES)," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf, 2001.

[120] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "Umac: Fast and secure message authentication," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer, 1999, pp. 216–233.

[121] D. J. Bernstein, "The Poly1305-AES message-authentication code," in *International workshop on fast software encryption*. Springer, 2005, pp. 32–49.

[122] Y. L. Chen, A. Flórez-Gutiérrez, A. Inoue, R. Ito, T. Iwata, K. Minematsu, N. Mouha, Y. Naito, F. Sibleyras, and Y. Todo, "Key committing security of aez and more," *IACR Transactions on Symmetric Cryptology*, vol. 2023, no. 4, pp. 452–488, 2023.

[123] J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer, "The design of Xoodoo and Xoofff," *IACR Transactions on Symmetric Cryptology*, pp. 1–38, 2018.

[124] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," *NIST*, 2015.

[125] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the Indifferentiability of the Sponge Construction," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 181–197.

[126] "FIPS 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf, 2015.

[127] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, R. Van Keer, and B. Viguier, "TurboSHAKE," *Cryptology ePrint Archive*, 2023.