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

Munawar Hasan[1,2] and Donghoon Chang[1,2,3]

[1] National Institute of Standards and Technology, USA
{munawar.hasan,donghoon.chang}@nist.gov
[2] Indraprastha Institute of Information Technology, India
{munawarh,donghoon}@iiitd.ac.in
[3] Strativia, USA

Abstract. The widespread deployment of low-power and handheld devices opens an opportunity to design lightweight authenticated encryption schemes. The schemes so proposed must also prove their resilience under various security notions. Romulus-N1 is an authenticated encryption scheme with associated data based on a tweakable blockcipher, a primary variant of Romulus-N family which is NIST (National Institute of Standards and Technology) lightweight cryptography competition finalist; provides beyond birthday bound security for integrity security in nonce respecting scenario but fails to provide the integrity security in nonce misuse and RUP (release of unverified plaintext) scenarios. In this paper, we propose lynx, a family with 14 members of 1-pass and rate-1 lightweight authenticated encryption schemes with associated data based on a tweakable blockcipher, that provides birthday bound security for integrity security in both nonce respecting as well as nonce misuse and RUP scenarios and birthday bound security for privacy in nonce respecting scenario. For creating such a family of schemes we propose a family of function $\mathcal{F}$ that provides a total of 72 cases out of which we show that only 14 of them can be used for creating authenticated encryption schemes. We provide the implementation of one of the members of lynx family on six different hardware platforms and compare it with Romulus-N1. The comparison clearly shows that the lynx member outperforms Romulus-N1 on all the six platforms.

Keywords: Authenticated encryption, Tweakable blockcipher, Lightweight cryptography, Lightweight authenticated encryption scheme, Security proofs

1 Introduction

Authenticated Encryption scheme with associated data (or AEAD) provides both privacy and authenticity based on various factors. 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.
CAESAR competition [1] 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 [2] and Ascon [3] were selected as the winners in the lightweight category of the competition. National Institute of Standards and Technology (NIST) [4], 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 [5]. 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 [6, 7, 8, 9]. 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 1 candidates. As of writing this paper, 10 candidates have been selected for the final round.
One of the finalists of the NIST lightweight competition, Romulus [10], utilizes tweakable blockcipher as the underlying premitive, for their AEAD algorithm. Tweakable blockcipher was introduced by Liskov *et al.* at CRYPTO 2002 [11]. 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. Since the inception of the tweakable blockciphers, they have been widely studied under various security scenarios for creating authenticated encryption schemes, for example Romulus [10], SKINNY-AEAD [12], ForkAE [13] etc.

1.1 Motivation

Romulus [10] which is 1-pass and rate-1 design provides beyond birthday bound (or BBB) security in nonce respecting scenario and is only nonce resilient (not nonce resistant) in nonce misuse and RUP scenario [14]. Our motivation is to target the area of nonce resistant AEAD constructions, under nonce misuse and RUP scenarios. In 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 [15], an AEAD scheme called AET-LR [16], 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 scenario 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.
In this paper, we present a family of lightweight authenticated encryption schemes called lynx [1] (1-pass and rate-1) based on a tweakable blockcipher (with tweak size as twice the

[1] Lynx [pronounced: lɪŋks] 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 1: Summary of Lynx-A family. There are 10 members in lynx-A family with each member constructed using a function from F^A family. For more details see section 4

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

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 4 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-A10 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, F, which is used to create the lynx family. For the lynx-A family, the function family is referred to as F^A and for the lynx-B family, the function family is referred to as F^B respectively. There are 36 cases of F^A (refer table 5) and 36 cases of F^B (refer table 6) enumerated as $F^A_{1...36}$ and $F^B_{1...36}$ respectively, thereby making a total of 72 cases. Internally, both the function families utilizes simple *xor* operations (refer section 4.3 and section 4.4). We perform analysis of each of these 72 cases in the section 5 and sort them into the correct and the incorrect groups. Out of the 36 cases of F^A, 10 of them fall into the correct group while out of the 36 cases of 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 F^A and 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 $\overline{F^A}$ and $\overline{F^B}$ for the respective function families.

In table 1, we present a mapping of all the members of the lynx-A family with the cases of the function family F^A. Similarly, in table 2, we present a mapping of all the members of the lynx-B family with the cases of the function family F^B.

Table 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 F^B family. For more details see section 4

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

1.2 Contributions

- We propose family of 1-pass and rate-1 AEAD scheme called lynx (table 1 and table 2) based on a tweakable blockcipher where tweak size is twice the block size. We divide the lynx family into two sub-family; 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 (figure 1 and algorithm 1) and lynx-B (figure 2 and algorithm 2).
- We propose a family of functions called F (sub-family F^A and F^B) and present 72 cases (table 5 and table 6) of F (36 cases of F^A and 36 cases of F^B).
- We present correctness criteria of F and then we analyze each case of 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 (table 7), based on the issues in the internal construction of F.
- We present formal security proofs of lynx-A and lynx-B in the information-theoretic model (section 7) 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 (section 8). 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.

1.3 Organization of this paper:

The rest of the paper is organized as follows. Section 2 describes the related work in the direction of lightweight cryptography and specifically in the area of lightweight authenticated encryption schemes. Section 3 delivers the preliminaries used in the paper. This section also covers the definitions and security notions. Section 4 presents the authenticated encryption scheme proposed in this paper. This section also gives the internal construction of the family of functions F^A, F^B, $\overline{F^A}$ and $\overline{F^B}$, with its 72 cases. In section 5, we present the analysis of function family F^A and F^B, and show the cases of F^A and F^B families that can be used to construct lynx. We present design rationale of lynx in section 6 followed by security proofs in section 7. Section 8 shows the implementation of one

of the member of lynx family and its comparison to Romulus-N1 followed by conclusion in section 9. The appendix provides several supporting mathematical proofs for our proposed algorithms.

2 Related Work

The inception of authenticated encryption with associated data [17] simulated research in the area of authenticated encryption modes. OCB [18] became as one of the early proposals for efficient authenticated encryption scheme and was later patented. This was followed by several new modes of operation [19, 20, 21, 22, 23, 24]. In 2007, NIST (National Institute of Standards and Technology) choose GCM [22] as a standard for authenticated encryption scheme with associated data. Soon AES [25] blockcipher became widespread underlying primitive for the GCM mode and hence the name AES-GCM.
In the recent years, there has been a lot of traction in the area of lightweight cryptographic algorithms. Such algorithms are tailored for implementation in resource constrained environments including RFID tags, sensors, contactless smart cards, health-care devices etc. [26, 27, 28, 29] discussed a general overview of design strategy of lightweight cryptographic primitives. [30, 31] presents a comparative overview of several cryptographic primitives. We primarily try to focus on lightweight authenticated encryption schemes.
Several authenticated encryption schemes that rely on sponge function [32, 33], utilize duplex mode of operation. The LWC competition at NIST [4] has seen wide range of submissions based on sponge functions that are in lightweight category. Sparkle [34] and Xoodyak [35] both NIST LWC competition finalists are based on sponge function. Ascon [3] a finalist of NIST LWC and the winner of CAESAR [1] competition uses the duplex mode of authenticated encryption. PHOTON-Beetle [36] based on Beetle [37] which is a sponge based mode and ISAP [38], 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 [39], a blockcipher based authenticated encryption scheme with associated data uses GIFT blockcipher [40] as the underlying primitive. GIFT-COFB is also one of the finalists of NIST lightweight cryptography competition. Another finalist, Grain-128AEAD [41], uses stream cipher from the Grain [42] 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 [10], another NIST lightweight cryptography competition finalist uses SKINNY [43] tweakable blockcipher for their authenticated encryption design. AET-LR [16] is also based on a tweakbale 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 [44] is based on keyed permutation while elephant [45] is based on cryptographic permutation masked using LFSRs. Both TinyJAMBU and elephant are the finalists of NIST lightweight cryptography competition.

3 Preliminaries

3.1 Notations

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: $1 \quad 0^i$ or 10^i refers to bit 1 followed by i zero bits. $X \quad Y$ denotes the bitwse xor between X and Y where $|X| = |Y|$. For a given $X \quad \{0,1\}^+$, 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 $\nu \quad \{0,1\}$ as suffix of $X \quad \{0,1\}$ 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 $X \quad \{0,1\}^+$, we define X in x blocks (or block length of X is x) in the following way: $(X[1] \quad X[2] \quad ... \quad X[x]) = X)$ where $x = \frac{|X|}{b}$ 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 $X \quad \{0,1\}^{<b}$:

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

3.2 Definitions

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=(E, D) that consists of an encryption sub-routine E and a decryption sub-routine D defined over a key $K \quad \{0,1\}^k$, a nonce $N \quad \{0,1\}^n$, an associated data $A \quad \{0,1\}$, a message $M \quad \{0,1\}$, a ciphertext $C \quad \{0,1\}$ and a tag $T \quad \{0,1\}^{tag}$ such that:

$$\begin{aligned} \mathsf{E} &: (K, N, A, M) \rightarrow (C, T) \\ \mathsf{D} &: (K, N, A, C, T) \rightarrow M/ \end{aligned} \tag{2}$$

  where:

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

  Alternatively using the notation from [14], we can define RUP security. In the RUP setting, the real-world constitutes both the encryption E_K sub-routine and the decryption D_K sub-routine over a random key K. Further, the decryption sub-routine 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=($\mathsf{E}, \mathsf{D}, \mathsf{V}$) as follows:

$$\begin{aligned} \mathsf{E} &: (K, N, A, M) \rightarrow (C, T) \\ \mathsf{D} &: (K, N, A, C, T) \rightarrow M \\ \mathsf{V} &: (K, N, A, C, T) \rightarrow \quad / \end{aligned} \tag{3}$$

  where:

$$\mathsf{D}(K, N, A, \mathsf{E}(K, N, A, M)) \rightarrow M$$
$$\mathsf{V}(K, N, A, \mathsf{E}(K, N, A, M)) \rightarrow$$

  From the equation (3), 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, symbol denotes always a false outcome while a symbol denotes always a true outcome. In an alternative notation, we denote an authenticated encryption scheme by following sub-routines: ($\mathsf{E}, \mathsf{D}, \mathsf{V}$), where V is the verification sub-routine.

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

$$\begin{aligned} \widetilde{E} &: (K, T, M) \rightarrow C \\ \widetilde{D} &: (K, T, C) \rightarrow M \end{aligned} \tag{4}$$

where:

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

A tweakable blockcipher should be efficient i.e., both encryption $\widetilde{E}$ and $\widetilde{D}$ should be easy to compute [11]. Occasionally we also denote a tweakable blockcipher by following notation: $(\widetilde{E}_K, \widetilde{D}_K)$

3.3 Security Notion

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 [46]. 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 E_K of AEAD against a random function \$ i.e., given AEAD=(E, D), we define \$ $\xleftarrow{r}$ $\mathrm{F}(n, \ , \ )$ as a random function that depends on the encryption sub-routine of the AEAD.

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

$$\mathrm{CONF}_{\Pi}(A) = [Pr[A^{E_K} \rightarrow 1] - Pr[A^{\$} \rightarrow 1]] \tag{5}$$

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 V_K returns true (or) 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 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 V_K.

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

$$\mathrm{INT}_{\Pi}(A) = Pr[A^{E_K, V_K} forges] \tag{6}$$

The INT advantage measures the ability of an adversary A to generate a valid tag for a new input i.e., for an input, A has not seen before.
It is often desirable to combine the equations (5) and (6) in a single security notion as defined below:

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

$$\mathrm{AE}_{\Pi}(A) = [Pr[A^{E_K, V_K} \rightarrow 1] - Pr[A^{\$,} \rightarrow 1]] \tag{7}$$

Formal definition of RUP (releasing unverified plaintext) security is provided by [14]. In definition 4, we take the notion of RUP setting from [14, 46]

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

$$\text{INT-RUP}_{\Pi}(A) = Pr[A^{E_K, D_K, V_K} forges] \tag{8}$$

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 $\widetilde{\mathrm{F}}(t, b)$ informally as the set of all the functions that takes a t bit tweak and b bit message and produces b bit output. Let $\widetilde{\$} \xleftarrow{r} \widetilde{\mathrm{F}}(t, b)$ 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 A for a randomly chosen key K is given by:

$$\begin{aligned} IND_{\widetilde{E}_K}(A) &= [Pr[A^{\widetilde{E}_K} \rightarrow 1] - Pr[A^{\widetilde{\$}} \rightarrow 1]] \\ IND_{\widetilde{E}_K, \widetilde{D}_K}(A) &= [Pr[A^{\widetilde{E}_K, \widetilde{D}_K} \rightarrow 1] \\ &\quad - Pr[A^{\widetilde{\$}, \widetilde{\$}^{-1}} \rightarrow 1]] \end{aligned} \tag{9}$$

Using the results from Rogaway and Shrimpton [47], we can write the co-relation between (5), (6) and (7) as:

Lemma 1. *[Adapted from [46]] Let $\Pi = (E, D, V)$ be an authenticated encryption scheme with associated data. Let A_1 be an adversary with query complexity as q_1 (encryption oracle), A_2 be an adversary with query complexity as q_2 (encryption +verification oracle) and A_3 be an adversary with query complexity as q_3 (encryption + verification oracle):*

$$\begin{aligned} \mathrm{CONF}_{\Pi}(A_1) &\leq \mathrm{AE}_{\Pi}(B_1) \\ \mathrm{INT}_{\Pi}(A_2) &\leq \mathrm{AE}_{\Pi}(B_2) \\ \mathrm{AE}_{\Pi}(A_3) &\leq \mathrm{CONF}_{\Pi}(B_3) + \mathrm{INT}_{\Pi}(B_4) \end{aligned} \tag{10}$$

where B_1, B_2, B_3 and B_4 are adversaries with query complexities as q_1, q_2 and q_3 respectively.

4 Lynx

We propose family of 1-pass rate-1 authenticated encryption schemes with associated data called lynx-A and lynx-B. We use E to denote the encryption sub-routine for the members of lynx-A and lynx-B families, and 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 F^A family that fall into the correct group. Figure 1 shows the construction of lynx-A family and algorithm 1 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 F^B family that fall into the correct group. In figure 2, we show the construction of lynx-B family and the algorithm 2 presents the encryption and decryption sub-routine of the lynx-B family.

In table 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 [48].

Next, we describe the lynx-A and lynx-B in detail followed by the internal constructions of F^A and F^B.

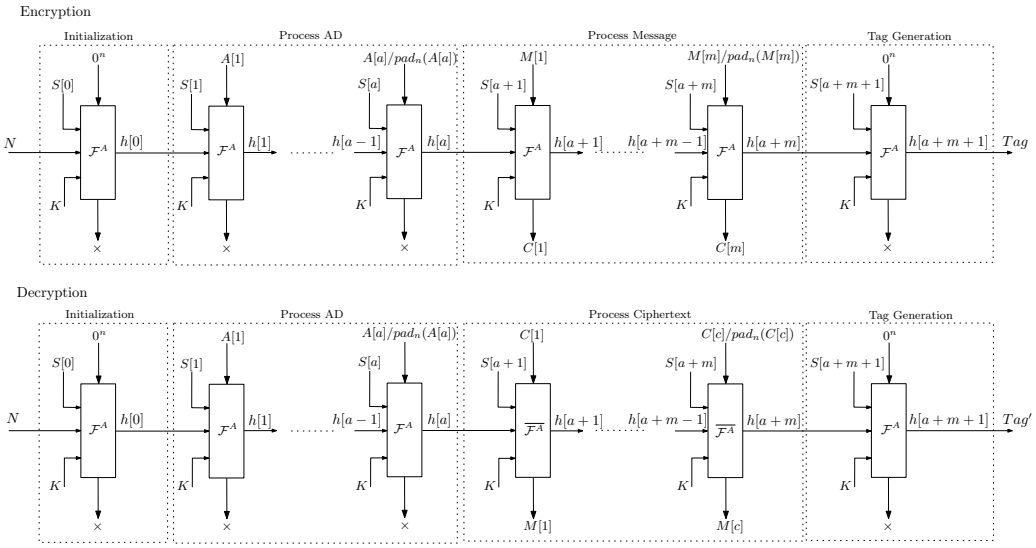

Figure 1: Encryption and Decryption sub-routine of Lynx-A family. F^A in below figure is used from the cases that fall into correct group (refer table 1 for list)

4.1 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 1). The initialization phase and the tag generation phase require an input with a

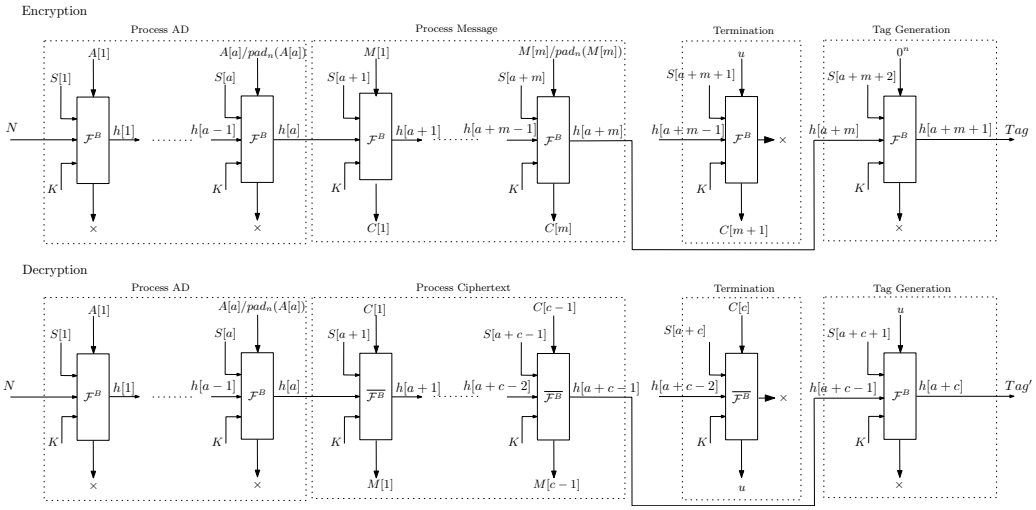

Figure 2: Encryption and Decryption sub-routine of Lynx-B family. F^B in below figure is used from the cases that fall into correct group (refer table 2 for list)

Table 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** ($t = 2 \quad b$)	**Tag Size** (tag)
Lynx-A Family	128	128	256	128
Lynx-B Family	128	128	256	128

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 (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. The length of the ciphertext produced by lynx-A is equal to the length of the message and hence lynx-A is length preserving.

4.2 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 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 1 and algorithm 1) while lynx-B executes only the tag generation phase (refer figure 2 and algorithm 2).

4.3 $\mathcal{F}^A$

The function family F^A takes in three inputs and returns two outputs (refer figure 3) as described by the following equation:

$$F^A(K, h[i-1], S[i], V[i]) \rightarrow (h[i], W[i]) \tag{11}$$

where K $\{0,1\}^k$, $h[i-1]$ $\{0,1\}^b$, $S[i]$ $\{0,1\}^s$, $V[i]$ $\{0,1\}^b$, $h[i]$ $\{0,1\}^b$ and $W[i]$ $\{0,1\}^b$. Like F^A, $\overline{F^A}$ also takes in three inputs and returns two outputs (refer figure 3):

$$\overline{F^A}(K, h[i-1], S[i], W[i]) \rightarrow (h[i], V[i]) \tag{12}$$

We present the internal construction of F^A and $\overline{F^A}$ in figure 3 and present their pseudocode in algorithm 3.
All the 36 cases of F^A (and $\overline{F^A}$) are shown in table 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 5 for details). Hence, the members of lynx-A family are created only using these 10 cases of F^A (and $\overline{F^A}$). As stated earlier, F^A is used during the invocation of encryption sub-routine (E) while $\overline{F^A}$ is used during the invocation of decryption sub-routine (D) of *lynx*-A respectively. Further, in equation (11), $V[i]$ 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 (12), $V[i]$ is the original message corresponding to the ciphertext $W[i]$.

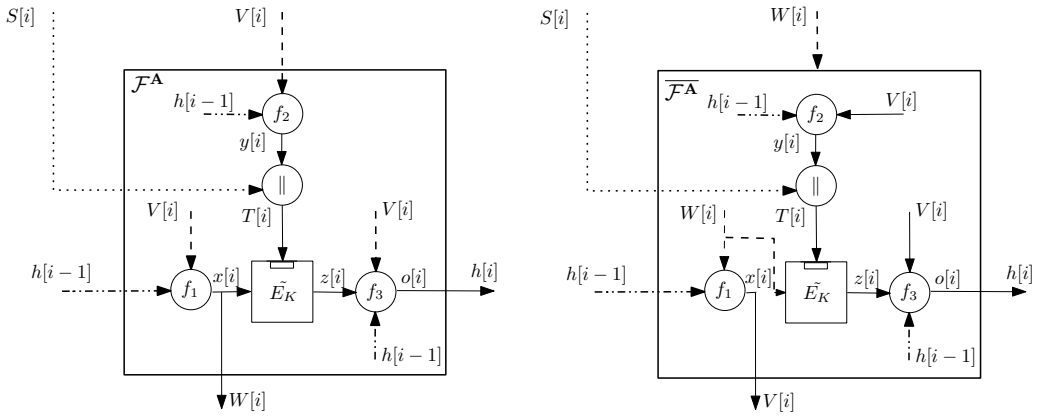

Figure 3: Internal construction of the function family F^A / $\overline{F^A}$. Internally it is composition of three sub-functions: f_1 and f_2 takes in two inputs and returns one output while f_3 takes in three input and returns one output

4.4 $\mathcal{F}^B$

Similar to the F^A, F^B takes in three inputs and returns two outputs (refer figure 4) as described by the following equation:

$$F^B(K, h[i-1], S[i], V[i]) \rightarrow (h[i], W[i]) \tag{13}$$

Algorithm 1 : Lynx-A encryption and decryption

$E(K, N, A, M)$
Note: F^A is used from the cases that fall into correct group (refer table 1)

$(A[1] \quad A[2] \quad ... \quad A[a]) = A$, $(M[1] \quad M[2] \quad ... \quad M[m]) = M$, $l_{7...0} \leftarrow 0$, $K \xleftarrow{\$} \{0,1\}^k$, $S[0] \leftarrow bin_{120}(0) \quad l_{7...0}$
$(h[0], \times) \leftarrow F^A(K, N, S[0], 0^b)$ ▷ *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
  $S[i] \leftarrow bin_{120}(i) \quad l_{7...0}$
  $(h[i], \times) \leftarrow F^A(K, h[i-1], S[i], A[i])$
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
  $S[a+i] \leftarrow bin_{120}(a+i) \quad l_{7...0}$
  $(h[a+i], C[i]) \leftarrow F^A(K, h[a+i-1], S[a+i], M[i])$
end for
$l_{3...0} \leftarrow 1$ ▷ *Tag Generation*
$S[a+m+1] \leftarrow bin_{120}(a+m+1) \quad l_{7...0}$
$(h[a+m+1], \times) \leftarrow F^A(K, h[a+m], S[a+m+1], 0^n)$
$Tag \leftarrow h[a+m+1]$, $C \leftarrow C[0] \quad C[1] \quad ... \quad \text{Trunc}_\nu(C[m])$
Return (C, Tag)

$D(K, N, A, C, Tag)$
$(A[1] \quad A[2] \quad ... \quad A[a]) = A$, $(C[1] \quad C[2] \quad ... \quad C[c]) = C$, $l_{7...0} \leftarrow 0$, $S[0] \leftarrow bin_{120}(0) \quad l_{7...0}$
$(h[0], \times) \leftarrow F^A(K, N, S[0], 0^b)$ ▷ *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
  $S[i] \leftarrow bin_{120}(i) \quad l_{0...7}$
  $(h[i], \times) \leftarrow F^A(K, h[i-1], S[i], A[i])$
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
  $S[a+i] \leftarrow bin_{120}(a+i) \quad l_{7...0}$
  $(h[a+i], M[i]) \leftarrow \overline{F^A}(K, h[a+i-1], S[a+i], C[i])$
end for
$l_{3...0} \leftarrow 1$ ▷ *Tag Generation*
$S[a+m+1] \leftarrow bin_{120}(a+m+1) \quad l_{7...0}$
$(h[a+m+1], \times) \leftarrow F^A(K, h[a+m], S[a+m+1], 0^n)$
$Tag \leftarrow h[a+m+1]$, $M \leftarrow M[0] \quad M[1] \quad ... \quad \text{Trunc}_\nu(M[c])$
if $Tag = Tag$ **then**
  Return M
else
  Return
end if

where $K \quad \{0,1\}^k$, $h[i-1] \quad \{0,1\}^b$, $S[i] \quad \{0,1\}^s$, $V[i] \quad \{0,1\}^b$, $h[i] \quad \{0,1\}^b$ and $W[i] \quad \{0,1\}^b$. $\overline{F^B}$ takes in three inputs and returns two outputs (refer figure 4):

$$\overline{F^B}(K, h[i-1], S[i], W[i]) \rightarrow (h[i], V[i]) \tag{14}$$

Algorithm 2 : Lynx-B encryption and decryption

$E(K, N, A, M)$
Note: F^B is used from the cases that fall into correct group (refer table 2)

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

$D(K, N, A, C, Tag)$

$(A[1] \quad A[2] \quad ... \quad A[a]) = A$, $(C[1] \quad C[2] \quad ... \quad C[c]) = C$, $l_{7...0} \leftarrow 0$, $l_3 \leftarrow 1$, $h[0] \leftarrow N$
for $i = 1$ to a **do** ▷ *Process Associated Data*
  if $|A[i]| \pmod b = 0$ **then**
    $l_0 \leftarrow 1$, $S[i] \leftarrow bin_{120}(i) \quad l_{7...0}$
  else
    $l_1 \leftarrow 1$, $A[i] \leftarrow pad_n(A[i])$, $S[i] \leftarrow bin_{120}(i) \quad l_{7...0}$
  end if
  $(h[i], \times) \leftarrow F^B(K, h[i-1], S[i], A[i])$
end for
for $i = 1$ to $c - 2$ **do** ▷ *Process Ciphertext*
  $l_0, l_1 \leftarrow 1$, $S[a+i] \leftarrow bin_{120}(a+i) \quad l_{7...0}$
  $(h[a+i], M[i]) \leftarrow \overline{F}_B(K, h[a+i-1], S[a+i], C[i])$
end for
if $C = \epsilon$ **then**
  $l_0, l_1 \leftarrow 0$, $S[a+c] \leftarrow bin_{120}(a+c) \quad l_{7...0}$ ▷ *Termination*
  $(\times, M[c]) \leftarrow \overline{F}_B(K, h[a+c-2], S[a+c], C[c])$
  if $M[c] = 10^{b-1}$ **then** ▷ *Process Last Ciphertext Block*
    $l_0, l_1 \leftarrow 1$, $S[a+c-1] \leftarrow bin_{120}(a+c-1) \quad l_{7...0}$
    $(h[a+c-1], M[c-1]) \leftarrow \overline{F}_B(K, h[a+c-2], S[a+c-1], C[c-1])$
  else
    $l_2 \leftarrow 1$, $S[a+c-1] \leftarrow bin_{120}(a+c-1) \quad l_{7...0}$
    $\nu \leftarrow 10$ ▷ *Padded according to Eq. 1*

$(h[a+c-1],\ M[c-1]) \leftarrow \overline{F}_B(K, h[a+c-2], S[a+c-1], C[c-1])$
$M[c-1] \leftarrow \text{Extract}_\nu(M[c-1]))$
end if
end if
$l_{3\ldots0} \leftarrow 1,\ S[a+c+1] \leftarrow bin_{120}(a+c+1) \quad l_{7\ldots0}$ ▷ *Tag Generation*
$(h[a+c], \times) \leftarrow F^B(K, h[a+c-1], S[a+c+1], 0^n)$
$Tag \leftarrow h[a+c],\ M \leftarrow M[0] \quad M[1] \quad \ldots \quad M[c-1]$
if $Tag = Tag$ **then**
Return M
else
Return
end if

Algorithm 3 : F^A and $\overline{F^A}$

$F^A(K, h[i-1], S[i], V[i])$
$x[i] \leftarrow f_1(h[i-1], V[i]) \mid x[i] \quad \{h[i-1], V[i], h[i-1] \quad V[i]\}$
$W[i] \leftarrow x[i]$
$y[i] \leftarrow f_2(h[i-1], V[i]) \mid y[i] \quad \{h[i-1], V[i], h[i-1] \quad V[i]\}$
$T[i] \leftarrow S[i] \quad y[i]$
$z[i] \leftarrow \widetilde{E}(K, T[i], x[i])$
$o[i] \leftarrow f_3(z[i], h[i-1], V[i]) \mid o[i] \quad \{z[i], z[i] \quad h[i-1], z[i] \quad V[i], z[i] \quad h[i-1] \quad V[i]\}$
$h[i] \leftarrow o[i]$
Return $(h[i], W[i])$

$\overline{F^A}(K, h[i-1], S[i], W[i])$
$x[i] \leftarrow f_1(h[i-1], W[i]) \mid x[i] \quad \{h[i-1], W[i], h[i-1] \quad W[i]\}$
$V[i] \leftarrow x[i]$
$y[i] \leftarrow f_2(h[i-1], W[i]) \mid y[i] \quad \{h[i-1], W[i], h[i-1] \quad W[i]\}$
$T[i] \leftarrow S[i] \quad y[i]$
$z[i] \leftarrow \widetilde{E}(K, T[i], x[i])$
$o[i] \leftarrow f_3(z[i], h[i-1], W[i]) \mid o[i] \quad \{z[i], z[i] \quad h[i-1], z[i] \quad W[i], z[i] \quad h[i-1] \quad W[i]\}$
$h[i] \leftarrow o[i]$
Return $(h[i], V[i])$

We present the internal construction of F^B and $\overline{F^B}$ in figure 4 and present their pseudocode in algorithm 4.
Table 6 gives the 36 cases of F^B (and $\overline{F^B}$). The 4 cases marked in bold red font can only be used to create an authenticated encryption scheme (check section 5 for details). These 4 cases are used to create *lynx*-B family.
The fundamental difference between F^A and F^B family is based on the idea of generating the ciphertext. In F^A, the ciphertext is generated after the *xor* operation i.e., $C[i] = x[i]$ (refer $W[i]$ in figure 3) while in case of F^B, the ciphertext is produced from the output of the tweakable blockcipher i.e., $C[i] = z[i]$ (refer $W[i]$ in figure 4). Hence, in case of F^A, a single bit difference causes one bit difference in the ciphertext, while, a single bit difference in case of F^B, affects all the bits of the ciphertext (given a secure tweakable blockcipher).

To establish the correctness of $F^A/\overline{F^A}$ and $F^B/\overline{F^B}$, we present following criteria:
Correctness: For a given K, $h[i-1]$, $V[i]$ and $S[i]$, a valid construction is defined as:

$$\begin{aligned} F^A(K, h[i-1], S[i], V[i]) &\rightarrow (h[i], W[i]) \\ \overline{F^A}(K, h[i-1], S[i], W[i]) &\rightarrow (h\ [i], V\ [i]) \\ \text{where } (h\ [i] = h[i]) &\quad (V\ [i] = V[i]) \end{aligned} \tag{15}$$

and

$$\begin{aligned} F^B(K, h[i-1], S[i], V[i]) &\rightarrow (h[i], W[i]) \\ \overline{F^B}(K, h[i-1], S[i], W[i]) &\rightarrow (h\ [i], V\ [i]) \\ \text{where } (h\ [i] = h[i]) &\quad (V\ [i] = V[i]) \end{aligned} \tag{16}$$

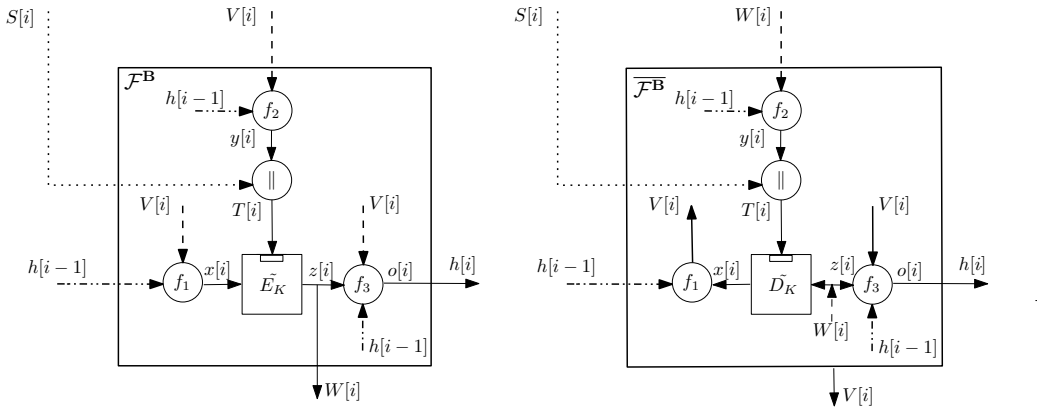

Figure 4: Internal construction of the function family F^B / $\overline{F^B}$. Internally it is composition of three sub-functions: f_1 and f_2 takes in two inputs and returns one output while f_3 takes in three input and returns one output

Algorithm 4 : F^B and $\overline{F^B}$

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

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

4.5 Domain Separation

We use one byte flag to provide domain separation in various scenarios. The flag we use is represented by following notation: $l_{7\ldots0}$. The four most significant bits of flag is always 0 i.e., $l_{7\ldots4} \leftarrow 0$. Rest of the four bits i.e., the four least significant bits provide domain separation for 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 4 shows the values taken by the flag bits. The bit l_3 serves as domain separator between the two families of the lynx.

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

As stated earlier, only 10 cases of F^A and 4 cases of F^B can be used for creating authenticated encryption scheme (marked in bold red font in table 5 and table 6). In

Table 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** $\lvert A[i]\rvert \mod b = 0$
	Process Message	$0^4 0011$ **if** $\lvert M[i]\rvert \mod b = 0$
		$0^4 0100$ **if** $\lvert M[i]\rvert \mod b = 0$
	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** $\lvert A[i]\rvert \mod b = 0$
	Process Message	$0^4 1011$ **if** $\lvert M[i]\rvert \mod b = 0$
		$0^4 1100$ **if** $\lvert M[i]\rvert \mod b = 0$
	Termination	$0^4 1000$
	Tag Generation	$0^4 1111$

Table 5: F^A and $\overline{F^A}$: Enumerated from 1 to 36. The bold letters in red color in the table denote the cases that fall into correct group

$F^{\mathbf{A}}$: $W[i] = x[i]$ And $\overline{F^{\mathbf{A}}}$: $V[i] = x[i]$					
$x[i]$	$y[i]$	$o[i]$			
		$z[i]$	$z[i] \quad V[i]$	$z[i] \quad h[i-1]$	$z[i] \quad V[i]$ $h[i-1]$
$h[i-1]$ / $h[i-1]$	$V[i]$ / $W[i]$	F_1^A / $\overline{F_1^A}$	F_2^A / $\overline{F_2^A}$	F_3^A / $\overline{F_3^A}$	F_4^A / $\overline{F_4^A}$
	$h[i-1]$ / $h[i-1]$	F_5^A / $\overline{F_5^A}$	F_6^A / $\overline{F_6^A}$	F_7^A / $\overline{F_7^A}$	F_8^A / $\overline{F_8^A}$
	$V[i] \quad h[i-1]$ / $W[i] \quad h[i-1]$	F_9^A / $\overline{F_9^A}$	F_{10}^A / $\overline{F_{10}^A}$	F_{11}^A / $\overline{F_{11}^A}$	F_{12}^A / $\overline{F_{12}^A}$
$V[i]$ / $W[i]$	$V[i]$ / $W[i]$	F_{13}^A / $\overline{F_{13}^A}$	F_{14}^A / $\overline{F_{14}^A}$	F_{15}^A / $\overline{F_{15}^A}$	F_{16}^A / $\overline{F_{16}^A}$
	$h[i-1]$ / $h[i-1]$	F_{17}^A / $\overline{F_{17}^A}$	F_{18}^A / $\overline{F_{18}^A}$	F_{19}^A / $\overline{F_{19}^A}$	F_{20}^A / $\overline{F_{20}^A}$
	$V[i] \quad h[i-1]$ / $W[i] \quad h[i-1]$	F_{21}^A / $\overline{F_{21}^A}$	F_{22}^A / $\overline{F_{22}^A}$	F_{23}^A / $\overline{F_{23}^A}$	F_{24}^A / $\overline{F_{24}^A}$
$h[i-1] \quad V[i]$ / $h[i-1] \quad W[i]$	$V[i]$ / $V[i]$	$F_{\mathbf{25}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{25}}^{\mathbf{A}}}$	$F_{\mathbf{26}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{26}}^{\mathbf{A}}}$	$F_{\mathbf{27}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{27}}^{\mathbf{A}}}$	$F_{\mathbf{28}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{28}}^{\mathbf{A}}}$
	$h[i-1]$ / $h[i-1]$	$F_{\mathbf{29}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{29}}^{\mathbf{A}}}$	$F_{\mathbf{30}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{30}}^{\mathbf{A}}}$	$F_{\mathbf{31}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{31}}^{\mathbf{A}}}$	$F_{\mathbf{32}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{32}}^{\mathbf{A}}}$
	$V[i] \quad h[i-1]$ / $V[i] \quad h[i-1]$	F_{33}^A / $\overline{F_{33}^A}$	$F_{\mathbf{34}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{34}}^{\mathbf{A}}}$	$F_{\mathbf{35}}^{\mathbf{A}}$ / $\overline{F_{\mathbf{35}}^{\mathbf{A}}}$	F_{36}^A / $\overline{F_{36}^A}$

this section, we analyze each case of F^A and 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

Table 6: F^B and $\overline{F^B}$: Enumerated from 1 to 36. The bold letters in red color in the table denote the cases that fall into correct group

$F^{\mathbf{B}}$, $W[i] = z[i]$ And $\overline{F^{\mathbf{B}}}$: $V[i] = x[i]$					
$x[i]$	$y[i]$	$o[i]$			
		$z[i]$	$z[i]$ $V[i]$	$z[i]$ $h[i-1]$	$z[i]$ $V[i]$ $h[i-1]$
$h[i-1]$ / $h[i-1]$	$V[i]$ / $W[i]$	F_1^B / $\overline{F_1^B}$	F_2^B / $\overline{F_2^B}$	F_3^B / $\overline{F_3^B}$	F_4^B / $\overline{F_4^B}$
	$h[i-1]$ / $h[i-1]$	F_5^B / $\overline{F_5^B}$	F_6^B / $\overline{F_6^B}$	F_7^B / $\overline{F_7^B}$	F_8^B / $\overline{F_8^B}$
	$V[i]$ $h[i-1]$ / $W[i]$ $h[i-1]$	F_9^B / $\overline{F_9^B}$	F_{10}^B / $\overline{F_{10}^B}$	F_{11}^B / $\overline{F_{11}^B}$	F_{12}^B / $\overline{F_{12}^B}$
$V[i]$ / $V[i]$	$V[i]$ / $W[i]$	F_{13}^B / $\overline{F_{13}^B}$	F_{14}^B / $\overline{F_{14}^B}$	F_{15}^B / $\overline{F_{15}^B}$	F_{16}^B / $\overline{F_{16}^B}$
	$h[i-1]$ / $h[i-1]$	F_{17}^B / $\overline{F_{17}^B}$	$\mathbf{F_{18}^B}$ / $\overline{\mathbf{F_{18}^B}}$	F_{19}^B / $\overline{F_{19}^B}$	$\mathbf{F_{20}^B}$ / $\overline{\mathbf{F_{20}^B}}$
	$V[i]$ $h[i-1]$ / $W[i]$ $h[i-1]$	F_{21}^B / $\overline{F_{21}^B}$	F_{22}^B / $\overline{F_{22}^B}$	F_{23}^B / $\overline{F_{23}^B}$	F_{24}^B / $\overline{F_{24}^B}$
$h[i-1]$ $V[i]$ / $h[i-1]$ $V[i]$	$V[i]$ / $W[i]$	F_{25}^B / $\overline{F_{25}^B}$	F_{26}^B / $\overline{F_{26}^B}$	F_{27}^B / $\overline{F_{27}^B}$	F_{28}^B / $\overline{F_{28}^B}$
	$h[i-1]$ / $h[i-1]$	F_{29}^B / $\overline{F_{29}^B}$	$\mathbf{F_{30}^B}$ / $\overline{\mathbf{F_{30}^B}}$	F_{31}^B / $\overline{F_{31}^B}$	$\mathbf{F_{32}^B}$ / $\overline{\mathbf{F_{32}^B}}$
	$V[i]$ $h[i-1]$ / $W[i]$ $h[i-1]$	F_{33}^B / $\overline{F_{33}^B}$	F_{34}^B / $\overline{F_{34}^B}$	F_{35}^B / $\overline{F_{35}^B}$	F_{36}^B / $\overline{F_{36}^B}$

Table 7: Incorrect cases of F^A and F^B that fall into implausible case or non-confidential case or non-integrity case

Implausible Cases	F_1^A, F_2^A, F_3^A, F_4^A, F_5^A, F_6^A, F_7^A, F_8^A F_9^A, F_{10}^A, F_{11}^A, F_{12}^A, F_1^B, F_2^B, F_3^B, F_4^B F_5^B, F_6^B, F_7^B, F_8^B, F_9^B, F_{10}^B, F_{11}^B, F_{12}^B F_{13}^B, F_{14}^B, F_{15}^B, F_{16}^B, F_{21}^B, F_{22}^B, F_{23}^B, F_{24}^B F_{25}^B, F_{26}^B, F_{27}^B, F_{28}^B, F_{33}^B, F_{34}^B, F_{35}^B, F_{36}^B
Non-Confidential Cases	F_{13}^A, F_{14}^A, F_{15}^A, F_{16}^A, F_{17}^A, F_{18}^A, F_{19}^A, F_{20}^A F_{21}^A, F_{22}^A, F_{23}^A, F_{24}^A
Non-Integrity Cases	F_{33}^A, F_{36}^A, F_{17}^B, F_{19}^B, F_{29}^B, F_{31}^B

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 F^A and F^B into these sub-cases of the incorrect group. In this section, we use notation $F_{i\ldots k}^A$ to cover all the cases of F^A between F_i^A (inclusive) and F_k^A (inclusive), thereby making a total of $k - i + 1$ cases. We use similar notation for F^B as well.

- **Implausible Cases:** Such cases of F^A and 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 F_1^A. From table 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 $M[i]$. $F_{2\ldots12}^A$ fall into this sub-class as well.

In case of 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, $F^B_{1...4}$, $F^B_{9...12}$, $F^B_{13...16}$, $F^B_{21...24}$, $F^B_{25...28}$ and $F^B_{33...36}$ fall into this category as well.
Now consider 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. $F^B_{6...8}$ 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 F^A_{13}, one can see that the message block $M[i]$ becomes the ciphertext block, there by making input message public. $F^A_{14...24}$ fall into this sub-class as well.

- **Non-Integrity Cases:** Consider F^A_{33}, the input to the tweakable blockcipher, $x[i] = h[i-1] \quad M[i] = C[i]$. Further, the tweak to the tweakable blockcipher, $T[i] = S[i] \quad h[i-1] \quad M[i] = S[i] \quad C[i]$. 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 F^A_{36} which has similar design to F^A_{33}. Input to the tweakable blockcipher, $x[i] = h[i-1] \quad M[i] = C[i]$ and tweak to the tweakable blockcipher, $T[i] = S[i] \quad h[i-1] \quad M[i] = S[i] \quad C[i]$. Further, in case of F^A_{36}, $h[i] = z[i] \quad h[i-1] \quad M[i] = z[i] \quad C[i]$. Hence, similar to the case of F^A_{33}, the dependency is only on the ciphertext.
In 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. F^B_{29} has similar problem as well.
In 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. F^B_{31} has similar problem as well.

Table 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 7).

6 Design Rationale

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

6.1 Initialization in Lynx-A

Lynx-A starts with the initialization phase (refer figure 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 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.

6.2 Termination in Lynx-B

In Lynx-B, we use termination phase (refer figure 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 (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.

6.3 Tag Generation

The lynx family uses tag generation phase (refer figure 1 for lynx-A and figure 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 F_{18}^B, F_{20}^B, F_{30}^B and F_{32}^B that are used for constructing lynx-B1, lynx-B2, lynx-B3 and lynx-B4 respectively. In case of F_{18}^B and F_{30}^B, if there is no tag generation phase then from table 6 we know that the output from F_{18}^B and 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 F_{20}^B and 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 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.

6.4 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.

6.5 Design of Flag

The choice of one byte flag was done to facilitate stream processing (see section 6.4). The bits provide domain separation but at the same time doesn't require extra knowledge about the upcoming blocks.

6.6 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.

6.7 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) [49]. We next present the security proofs.

7 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 F^A and F^B respectively. There are 10 members of lynx-A namely lynx-A1, lynx-A2, ..., 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$\{1 \dots 10\}$ and lynx-B family as lynx-B$\{1 \dots 4\}$. Figure 1 and figure 2 shows the internal construction of lynx-A and lynx-B while the internal construction of the function F^A and F^B is shown in figure 3 and figure 4. Table 1 and table 2 lists down F^A and 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 6** and **definition 8**. 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 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 F^A (for lynx-A) or F^B (for lynx-B) respectively. $x[i]$ and $T[i]$ denotes the two inputs (message and tweak) to the tweakable blockcipher of F^A (for lynx-A) or F^B (for lynx-B) for the i^{th} invocation of F^A and F^B respectively.

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

Proof. Refer appendix A.1 for proof. □

Lemma 3. *Lynx-A9 and Lynx-A10*

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 $(h[i-1] = h'[i-1]) \quad (M[i] = M'[i])$ then $(x[i], T[i]) = (x'[i], T'[i])$*

   Proof. Refer appendix A.2 for proof. □

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 $(h[i-1] = h'[i-1]) \quad (M[i] = M'[i])$ then $(x[i], T[i]) = (x'[i], T'[i])$*

   Proof. Refer appendix A.2 for proof. □

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 6. **[h-Coll]** If there exists an i, such that $h[i] = h [i]$ when $(x[i], T [i]) = (x [i], T [i])$, then the h-Coll event has occurred for the two tuples (N, A, M, C, T) and (N , A , M , C , T)

Definition 7. **[h*-Coll]** If there exists an i, such that $h[i - 1] = h [i - 1]$ 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 8. **[input-Coll]** If there exists an i, such that $h[i - 1] = h [i - 1]$ but $(x[i], T [i]) = (x [i], T [i])$, 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_E, q_D and q_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 F^A and the block length of a query for lynx-B is defined as the total number of invocations of F^B respectively.
We can now present **lemma 4** and **lemma 5** that gives the probability bound of **h-Coll** for lynx-A$\{1 \dots 8\}$ and lynx-B$\{1 \dots 4\}$, and **lemma 5** that gives the probability bound of **h-Coll** for lynx-A9 and lynx-A10.

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

Proof. Refer appendix A.3 for proof. □

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

Proof. Refer appendix A.4 for proof. □

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

Proof. Refer appendix A.5 for proof. □

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

Proof. Refer appendix A.6 for proof. □

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

7.1 Integrity in RUP

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

$$\text{INT-RUP}_{\Pi}(A) = Pr[A^{E_K, D_K, V_K} forges]$$

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

Theorem 1. [***INT-RUP*** *in* ***Lynx-A***$\{1 \dots 8\}$ *and* ***Lynx-B***$\{1 \dots 4\}$] *Let* Π *represent any one of the twelve authenticated encryption schemes from* lynx-$A\{1 \dots 8\}$ *and* lynx-$B\{1 \dots 4\}$ *where underlying tweakable blockcipher is replaced by random tweakable permutation. Let* A *be an adversary under nonce misuse scenario that makes* q_E *number of encryption queries,* q_D *number of decryption queries and* q_V *number of verification queries to* Π *such that* $q = q_E + q_D + q_V$*. The number of blocks for each query is at most* l*. Then*

$$\text{INT-RUP}_{\Pi}(A) \leq l \cdot \frac{q^2}{2^b} + \frac{q_V}{2^{b-1}} \tag{17}$$

Proof. Let h-Coll denote the event as described in **definition 6**. Let $Pr[\text{h-Coll}^A]$ be the probability of the occurrence of h-Coll event while $Pr[\overline{\text{h-Coll}}^A]$ be the probability of no h-coll event. Then the INT-RUP advantage of an adversary A against Π can be described using h-Coll event by the following generic expression of INT-RUP:

$$\begin{aligned} \text{INT-RUP}_{\Pi}(A) &= Pr[(A \ forges)] \\ &= Pr[\text{h-Coll}^A \quad (A \ forges)] \\ &\quad + Pr[\overline{\text{h-Coll}}^A \quad (A \ forges)] \end{aligned} \tag{18}$$

We know that $Pr[\text{h-Coll}^A \quad (A \ forges)] \leq Pr[\text{h-Coll}^A]$. From **lemma 4**, we know the that $Pr[\text{h-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$ (please refer appendix A.3 for proof). Hence, the expression $Pr[\overline{\text{h-Coll}}^A \quad (A \ forges)]$ is now only left to evaluate from the equation (18), we next try to find its bound. Using the laws of probability, we can write $Pr[\overline{\text{h-Coll}}^A \quad (A \ forges)] = Pr[(A \ forges) \mid \overline{\text{h-Coll}}^A] \cdot Pr[\overline{\text{h-Coll}}^A]$. But we know that $Pr[(A \ forges) \mid \overline{\text{h-Coll}}^A] \cdot Pr[\overline{\text{h-Coll}}^A] \leq Pr[(A \ forges) \mid \overline{\text{h-Coll}}^A]$. 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 6**, there exists i such that when the inputs to the tweakable blockcipher is different i.e., when $(x[i], T[i]) = (x'[i], T'[i])$ then $h[i] = h'[i]$. Further, from **lemma 2**, we know that if $h[i] = h'[i]$ then if there exists next input, then the next input to the tweakable blockcipher is different .i.e, $(x[i+1], T[i+1]) = (x'[i+1], T'[i+1])$. In this way, we use **definition 6**

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 $Pr[(\mathsf{A}\ forges) \mid \overline{\text{h-Coll}}^{\mathsf{A}}]$. We begin with the assumption that out of the two given tuples (N, A, M, C, T) and $(N\ , A\ , M\ , C\ , T\ )$, there is at least one bit difference between the (N, A, M, C) and the $(N\ , A\ , M\ , C\ )$, since a difference only in the respective tags i.e., $T = T$ 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, A can perform q_V such trials. Hence, $Pr[(\mathsf{A}\ forges) \mid \overline{\text{h-Coll}}^{\mathsf{A}}] \leq \frac{q_V}{2^{b-1}}$. Using this result we can write the equation (18) in the following way:

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

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-*A*9 *and* lynx-*B*10 *where underlying tweakable blockcipher is replaced by random tweakable permutation. Let* A *be an adversary that makes* q_E *number of encryption queries,* q_D *number of decryption queries and* q_V *number of verification queries to* Π *such that* $q = q_E + q_D + q_V$*. The number of blocks for each query is at most* l*. Then*

$$\text{INT-RUP}_{\Pi}(\mathsf{A}) \leq l \cdot \frac{q^2}{2^b} + \frac{q_V}{2^{b-1}} \tag{19}$$

Proof. We know from **lemma 5** that for lynx-A9 and lynx-A10, the $Pr[\text{h-Coll}^{\mathsf{A}}] \leq l \cdot \frac{q^2}{2^b}$. 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 $(h[i-1] = h\ [i-1])\quad (M[i] = M\ [i])$, then the inputs to the tweakable blockcipher is different i.e., $(x[i], T\ [i]) = (x[i]\ , T\ \ [i])$. In case of no h-Coll event, we know from **definition 6**, that when inputs to the tweakable blockcipher is different i.e., when $(x[i], T\ [i]) = (x\ [i], T\ \ [i])$ then $h[i] = h\ [i]$. In the second case, if $(h[i - 1] = h\ [i - 1])\quad (M[i] = M\ [i])$, then using **lemma 3** (specifically 3.2), we know that the inputs to the tweakable blockcipher i.e., $(x[i], T\ [i])$ and $(x[i]\ , T\ \ [i])$ are different and hence we can use the **definition 6** in the similar way as in the previous case to infer that $h[i] = h\ [i]$. In the third and the last case, for the two tuples (N, A, M, C, T) and $(N\ , A\ , M\ , C\ , T\ )$, when $(h[i - 1] = h\ [i - 1])\quad (M[i] = M\ [i])$, then due to the construction of F_A^{34} and 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., $(x[i], T\ [i]) = (x\ [i], T\ \ [i])$.

But we know from **definition 7**, that if $h[i-1] = h\,[i-1]$ then $h[i] = h\,[i]$, hence, this case is bounded by h -Coll of **lemma 6** i.e. $Pr[\text{h*-Coll}^{A}] \leq l \cdot \frac{q^2}{2^b}$. 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., $(x[i], T\,[i]) = (x\,[i], T\;[i])$. We begin with the first trial or the first verification query by the adversary 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:

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

□

7.2 Integrity

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

$$\text{INT}_{\Pi}(A) = Pr[A^{E_K, V_K}\, forges]$$

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 $\text{INT}_{\Pi}(A) \leq \text{INT-RUP}_{\Pi}(A)$. Since the bound are similar to the previous proof, we omit the proof for brevity.

7.3 Confidentiality

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

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

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 A be a nonce respecting adversary that makes a total of q queries to the encryption sub-routine of Π. Then*

$$\text{CONF}_{\Pi}(A) \leq 2 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}} \tag{20}$$

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_{Real} denote the h-Coll event (from **definition 6**) induced by the encryption sub-routine of Π i.e., for any i and pairs $(h[i], x[i], T\,[i])$ and $(h\,[i], x\,[i], T\;[i])$, h-Coll_{Real} occurs if $h[i] = h\,[i]$ when $(x[i], T\,[i]) = (x\,[i], T\;[i])$. From **lemma 4**, we know that: $Pr[\text{h-Coll}^{A}_{Real}] \leq l \cdot \frac{q^2}{2^{b+1}}$
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 A is nonce respecting). Hence, the probability of h-Coll$^{\mathsf{A}}_{Ideal}$ i.e., for any i and pairs $(h[i], x[i], \mathsf{T}[i])$ and $(h[i], x[i], \mathsf{T}[i])$, h-Coll$^{\mathsf{A}}_{Ideal}$ occurs if $h[i] = h[i]$ when $(x[i], \mathsf{T}[i]) = (x[i], \mathsf{T}[i])$. 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 $Pr[\text{h-Coll}^{\mathsf{A}}_{Ideal}] \leq l \cdot \frac{q^2}{2^b}$.

Let $\mathsf{E}^{\widetilde{P}}_K$ denote the encryption sub-routine of lynx using tweakable random permutation ($\widetilde{P}$) as the underlying primitive. Then we have: $Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{P}}_K} \rightarrow 1] = Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{P}}_K} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Real}] + Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{P}}_K} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Real}]$ and $Pr[\mathsf{A}^{\$} \rightarrow 1] = Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Ideal}] + Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Ideal}]$. Hence, we can write the CONF advantage of a nonce respecting adversary A for Π in the following way:

$$
\begin{aligned}
&Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{P}}_K} \rightarrow 1] - Pr[\mathsf{A}^{\$} \rightarrow 1] \leq \\
&\quad (Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{F}}_K} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Real}] \\
&\qquad + Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{F}}_K} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Real}]) \\
&\quad - (Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Ideal}] \\
&\qquad + Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Ideal}]) \\
&\quad + l \cdot \frac{\binom{q}{2}}{2^b}
\end{aligned}
\tag{21}
$$

where we replace $\mathsf{E}^{\widetilde{P}}_K$ with $\mathsf{E}^{\widetilde{F}}_K$ i.e., we switch from tweakbale tweakable random permutation to tweakable random function for the underlying primitive of lynx (Π) and hence, $l \cdot \frac{\binom{q}{2}}{2^b}$ comes from random tweakable permutation-tweakable random function (TRP-TRF) switching ([50]). 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 TRP-TRF 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:

- $Pr[\mathsf{A}^{\mathsf{E}^{\widetilde{F}}_K} \rightarrow 1 \mid \overline{\text{h-Coll}}^{\mathsf{A}}_{Real}] = Pr[\mathsf{A}^{\$} \rightarrow 1 \mid \overline{\text{h-Coll}}^{\mathsf{A}}_{Ideal}]$

Proof. In case of confidentiality security, we know that the adversary A is nonce respecting. Since, for given two tuples (N, A, M, C, T) and (N', A', M', C', T'), we have $N = N'$, then for lynx-A, $h[0] = h'[0]$ and for lynx-B $N = N'$. From **lemma** 2, we know that if $h[i-1] = h'[i-1]$ then the inputs to the tweakable blockcipher are different i.e., $(x[i], \mathsf{T}[i]) = (x'[i], \mathsf{T}'[i])$. But from **definition** 6, we know that if $(x[i], \mathsf{T}[i]) = (x'[i], \mathsf{T}'[i])$, then the output from F^A (for lynx-A) and F^B (for lynx-B) are different .i.e., $h[i] = h'[i]$. Hence, using **lemma** 2 and **definition** 6 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 3.3). This completes the proof of our claim. □

We apply the above results in equation (21) and also use the results from *Chang et.al.* [51] to solve the CONF advantage of the adversary.

$$\begin{aligned}
&(Pr[\mathsf{A}^{\widetilde{E_K^F}} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Real}] \\
&\qquad + Pr[\mathsf{A}^{\widetilde{E_K^F}} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Real}]) \\
&- (Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \text{h-Coll}^{\mathsf{A}}_{Ideal}] \\
&\qquad + Pr[\mathsf{A}^{\$} \rightarrow 1 \quad \overline{\text{h-Coll}}^{\mathsf{A}}_{Ideal}]) \\
&\leq 2 \cdot max\{Pr[\text{h-Coll}^{\mathsf{A}}_{Real}], Pr[\text{h-Coll}^{\mathsf{A}}_{Ideal}]\}
\end{aligned} \tag{22}$$

We know from **lemma 4** that $Pr[\text{h-Coll}^{\mathsf{A}}_{Real}] = l \cdot \frac{q^2}{2^b}$ while $Pr[\text{h-Coll}^{\mathsf{A}}_{Ideal}] = l \cdot \frac{q^2}{2^{b+1}}$. Hence,

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

□

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 blockcipher is replaced by random tweakable permutation. Let* A *be a nonce respecting adversary that makes a total of* q *queries to the encryption sub-routine of* Π*. Then*

$$\text{CONF}_{\Pi}(\mathsf{A}) \leq 4 \cdot l \cdot \frac{q^2}{2^b} + l \cdot \frac{q^2}{2^{b+1}} \tag{23}$$

Proof. In case of lynx-A9 and lynx-A10, the CONF security is influenced by both the collision events, **h-Coll** (**definition** 6) as well as **input-Coll** (**definition** 8). As in **theorem** 3, we consider nonce respecting adversary, hence if the one bit difference in inputs to the tweakable blockcipher i.e., $(x[i], \mathsf{T}[i]) = (x[i], \mathsf{T}[i])$, then from **definition** 6, we know that $h[i] = h[i]$. But from **definition** 8, we know that if $h[i] = h[i]$ then $(x[i+1], \mathsf{T}[i+1]) = (x[i+1], \mathsf{T}[i+1])$. 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 (21) using h-Coll$^{\mathsf{A}}$ and input-Coll$^{\mathsf{A}}$ as

follows:

$$
\begin{aligned}
Pr[A^{E_K^{\widetilde{F}}} \rightarrow 1] - Pr[A^{\$} \rightarrow 1] \leq \\
&(Pr[A^{E_K^{\widetilde{F}}} \rightarrow 1 \quad \text{h-Coll}^A_{Real}] \\
&\quad + Pr[A^{E_K^{\widetilde{F}}} \rightarrow 1 \quad \overline{\text{h-Coll}}^A_{Real}]) \\
&- (Pr[A^{\$} \rightarrow 1 \quad \text{h-Coll}^A_{Ideal}] \\
&\quad + Pr[A^{\$} \rightarrow 1 \quad \overline{\text{h-Coll}}^A_{Ideal}]) \\
&+ (Pr[A^{E_K^{\widetilde{F}}} \rightarrow 1 \quad \text{input-Coll}^A_{Real}] \\
&\quad + Pr[A^{E_K^{\widetilde{F}}} \rightarrow 1 \quad \overline{\text{input-Coll}}^A_{Real}]) \\
&- (Pr[A^{\$} \rightarrow 1 \quad \text{input-Coll}^A_{Ideal}] \\
&\quad + Pr[A^{\$} \rightarrow 1 \quad \overline{\text{input-Coll}}^A_{Ideal}]) \\
&+ l \cdot \frac{\binom{q}{2}}{2^b}
\end{aligned}
\tag{24}
$$

The solution of equation (24) is similar to the results of **theorem 3** and hence

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

□

8 Experiments and Performance Analysis

In this section, we present the implementation of lynx, 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 [10] for each of these six platforms. We want to point out that the underlying primitive of lynx-A1 is same as that of Romulus-N1 i.e., SKINNY tweakable blockcipher [43] 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 8 through table 13. For table 8 and table 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 10, we present the comparison on arduino uno platfrom (8 bit microcontroller). Table 11 shows the comparison on arduino due i.e., 32 bit arm microcontroller. In case of table 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 [52] runs in 32 bit mode on the hardware. In table 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 [53] 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 13 are only for the C code.

Table 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

Table 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 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 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

Table 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 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

All the implementations were run 100 times, and an average was taken of these 100 runs and reported in the table 8 through table 13. It can be clearly seen from the tables (table 8 - table 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 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 chose of lynx, one can see in table10, 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 authenticated 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.

9 Conclusion

We propose family of 1-pass and rate-1 lightweight authenticated encryption scheme called lynx based on tweakable blockcipher. Lynx has two sub-family, namely, lynx-A and lynx-B. Lynx has several advantages in lightweight category like stream processing, computationally menial and simplistic operations. Further, to create such an authenticated encryption scheme, we propose a family of functions called F. We present 72 cases of F and divide each of these F into incorrect and correct groups and show that only 14 of these cases can be used to create an authenticated encryption schemes (a.k.a lynx) using a tweakable blockcipher. The integrity security of lynx under nonce respecting as well as under nonce

misuse and RUP scenario is $\leq l \cdot \frac{q^2}{2^b} + \frac{qv}{2^{b-1}}$. 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, due to the simplistic structure of lynx, lynx-A1 outperforms Romulus-N1 on all the six platforms as shown in the experiments section.

A Mathematical Proofs of Lynx

A.1 Lemma 2

Lemma 2: In lynx-A{1...8} and lynx-B{1...4}, given two input pairs (N, A, M, C, Tag) and (N', A', M', C', Tag') and for any i if $h[i-1] = h'[i-1]$ then $(x[i], T[i]) = (x'[i], T'[i])$

Proof. We want to show that if $h[i-1] = h'[i-1]$, then the respective inputs to the tweakable blockcipher ($x[i]$ and $T[i]$) is also different. Hence, it is suffice to show that for an i either $x[i] = x'[i]$ or $T[i] = T'[i]$ or both $x[i] = x'[i]$ and $T[i] = T'[i]$. Equation (25) describes the inputs to the tweakable blockcipher

$$\begin{aligned} z[i] &= \widetilde{E}(K, T[i], x[i]) \\ x[i] &= \widetilde{D}(K, T[i], z[i]) \\ T[i] &= bin(i) \quad l_{7\ldots0} \quad y[i] \\ y[i] &= \{h[i-1], M[i], h[i-1] \quad M[i]\} \end{aligned} \tag{25}$$

There are two inputs $x[i]$ and $T[i]$ to the tweakable blockcipher. Based on these two inputs and the construction of function F^A and F^B, the proof can be divided into four separate sections. We use two cases $C[i] = C'[i]$ and $C[i] = C'[i]$ to prove the given lemma.

A.1.1 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

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

Lynx-A2

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

Lynx-A3

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

Lynx-A4

- $F_{28}^A : x[i] = h[i-1] \quad M[i],\ y[i] = M[i],\ o[i] = z[i] \quad M[i] \quad h[i-1]$, where $x[i] = C[i]$

å $\overline{F_{28}^{A}} : M[i] = h[i-1] \quad x[i], y[i] = M[i], o[i] = z[i] \quad M[i] \quad h[i-1]$

- **Case a:** $C[i] = C\,[i]$
  Since $x[i] = C[i], \quad (x[i] = x\,[i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\,[i] = C\,[i] \quad h\,[i-1]$. Hence, $M[i] = M\,[i]$. The tweaks are given by: $(T\,[i] = bin(i) \quad l_{7\ldots0} \quad M[i]) \quad (T\,[i] = bin(i) \quad l_{7\ldots0} \quad M\,[i])$, hence $T\,[i] = T\,[i]$
  Hence, $(x[i], T\,[i]) = (x\,[i], T\,[i])$
- **Case b:** $C[i] = C\,[i]$
  - $C[i] \quad h[i-1] = C\,[i] \quad h\,[i-1]$
    Since, $(x[i] = C[i]) \quad (x[i] = x\,[i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\,[i] = C\,[i] \quad h\,[i-1]$, Hence, $M[i] = M\,[i]$. The tweaks are given by: $(T\,[i] = bin(i) \quad l_{7\ldots0} \quad M[i])$ and $(T\,[i] = bin(i) \quad l_{7\ldots0} \quad M\,[i])$, hence $T\,[i] = T\,[i]$.
    $(x[i], T\,[i]) = (x\,[i], T\,[i])$
  - $C[i] \quad h[i-1] = C\,[i] \quad h\,[i-1]$
    Since, $(x[i] = C[i]) \quad (x[i] = x\,[i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\,[i] = C\,[i] \quad h\,[i-1]$, Hence, $M[i] = M\,[i]$. The tweaks are given by: $(T\,[i] = bin(i) \quad l_{7\ldots0} \quad M[i])$ and $(T\,[i] = bin(i) \quad l_{7\ldots0} \quad M\,[i])$, hence $T\,[i] = T\,[i]$.
    $(x[i], T\,[i]) = (x\,[i], T\,[i])$

A.1.2 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

å $F_{29}^{A} : x[i] = h[i-1] \quad M[i], y[i] = h[i-1], o[i] = z[i]$, where $x[i] = C[i]$

å $\overline{F_{29}^{A}} : M[i] = h[i-1] \quad x[i], y[i] = h[i-1], o[i] = z[i]$

Lynx-A6

å $F_{30}^{A} : x[i] = h[i-1] \quad M[i], y[i] = h[i-1], o[i] = z[i] \quad M[i]$, where $x[i] = C[i]$

å $\overline{F_{30}^{A}} : M[i] = h[i-1] \quad x[i], y[i] = h[i-1], o[i] = z[i] \quad M[i]$

Lynx-A7

å $F_{31}^{A} : x[i] = h[i-1] \quad M[i], y[i] = h[i-1], o[i] = z[i] \quad h[i-1]$, where $x[i] = C[i]$

å $\overline{F_{31}^{A}} : M[i] = h[i-1] \quad x[i], y[i] = h[i-1], o[i] = z[i] \quad h[i-1]$

Lynx-A8

å $F_{32}^{A} : x[i] = h[i-1] \quad M[i], y[i] = h[i-1], o[i] = z[i] \quad M[i] \quad h[i-1]$, where $x[i] = C[i]$

å $\overline{F_{32}^{A}} : M[i] = h[i-1] \quad x[i], y[i] = h[i-1], o[i] = z[i] \quad M[i] \quad h[i-1]$

- **Case a:** $C[i] = C\ [i]$
  Since $x[i] = C[i]$, hence $(x[i] = x\ [i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\ [i] = C\ [i] \quad h\ [i-1]$
  Hence, $M[i] = M\ [i]$. The tweaks are given by: $(T\ [i] = bin(i) \quad l_{7\ldots0} \quad h[i-1])$ and $(T\ \ [i] = bin(i) \quad l_{7\ldots0} \quad h\ [i-1])$, hence $T\ [i] = T\ \ [i]$.
  Hence, $(x[i], T\ [i]) = (x\ [i], T\ \ [i])$

- **Case b:** $C[i] = C\ [i]$

  - $C[i] \quad h[i-1] = C\ [i] \quad h\ [i-1]$
    Since, $(x[i] = C[i]) \quad (x[i] = x\ [i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\ [i] = C\ [i] \quad h\ [i-1]$, Hence, $M[i] = M\ [i]$. The tweaks are given by: $(T\ [i] = bin(i) \quad l_{7\ldots0} \quad h[i-1])$ and $(T\ \ [i] = bin(i) \quad l_{7\ldots0} \quad h\ [i-1])$, hence $T\ [i] = T\ \ [i]$.
    $(x[i], T\ [i]) = (x\ [i], T\ \ [i])$
  - $C[i] \quad h[i-1] = C\ [i] \quad h\ [i-1]$
    Since, $(x[i] = C[i]) \quad (x[i] = x\ [i])$. In case of lynx-A: $M[i] = C[i] \quad h[i-1]$ and $M\ [i] = C\ [i] \quad h\ [i-1]$, Hence, $M[i] = M\ [i]$. The tweaks are given by: $(T\ [i] = bin(i) \quad l_{7\ldots0} \quad h[i-1])$ and $(T\ \ [i] = bin(i) \quad l_{7\ldots0} \quad h\ [i-1])$, hence $T\ [i] = T\ \ [i]$.
    $(x[i], T\ [i]) = (x\ [i], T\ \ [i])$

A.1.3 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

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

Lynx-B2

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

Lynx-B3

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

Lynx-B4

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

Since, $h[i-1] = h\ [i-1] \quad T\ [i] = T\ \ [i]$.
$(x[i], T\ [i]) = (x\ [i], T\ \ [i])$ □

A.2 Lemma 3

Lemma 3.1: In lynx-A and lynx-B, given two input pairs (N, A, M, C, T) and (N', A', M', C', T') and for any i if $(h[i-1] = h'[i-1]) \quad (M[i] = M'[i])$ then $(x[i], T[i]) = (x'[i], T'[i])$

Proof. Similar to **lemma 2** (A.1), we want to show that the inputs to the tweakable blockcipher is different if $(h[i-1] = h'[i-1]) \quad (M[i] = M'[1])$. In case of lynx-A, one can check that $x[i] = h[i-1] \quad M[i]$, but from the lemma we know that $h[i-1] = h'[i-1]$ and $M[i] = M'[i]$. Hence, for lynx-A $x[i] = x'[i]$.
In case of lynx-B1 and lynx-B2, $x[i] = M[i]$ while in case of lynx-B3 and lynx-B4, $x[i] = h[i-1] \quad M[i]$. Hence, for lynx-B as well $x[i] = x'[i]$.
$(x[i], T[i]) = (x'[i], T'[i])$ □

Lemma 3.2: In lynx-A9 and lynx-A10, given two input pairs (N, A, M, C, T) and (N', A', M', C', T') and for any i if $(h[i-1] = h'[i-1]) \quad (M[i] = M'[i])$ then $(x[i], T[i]) = (x'[i], T'[i])$

Proof. The proof is similar to **lemma 3.1** (A.2). In case of lynx-A, $x[i] = h[i-1] \quad M[i]$. Since $h[i-1] = h'[i-1]$ then $x[i] = x'[i]$. In case of lynx-B, $y[i] = h[i-1]$ and $T[i] = bin(i) \quad l_{7\ldots0} \quad y[i]$ but since $h[i-1] = h'[i-1]$ then $T[i] = T'[i]$
$(x[i], T[i]) = (x'[i], T'[i])$ □

A.3 Lemma 4

Lemma 4: Let A be an adversary with q_E encryption queries, q_D decryption queries and q_V verification queries where maximum block length of any query is l, $q = q_E + q_D + q_V$ and $q \leq 2^{b-1}$ then for lynx-A$\{1\ldots8\}$ and lynx-B$\{1\ldots4\}$ the $Pr[\text{h-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$, where h-Coll^A is the occurrence of h-Coll event when the adversary 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, T) and (N', A', M', C', T') from q possible queries. From **definition 6**, we know that for the two pairs $(h[i], x[i], T[i])$ and $(h'[i], x'[i], T'[i])$, if $(x[i], T[i]) = (x'[i], T'[i])$, 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: $Pr[\text{h-Coll}^A] \leq l \cdot \frac{1}{2^{b-1}} \cdot \binom{q}{2} \leq l \cdot \frac{q^2}{2^b}$. □

A.4 Lemma 5

Lemma 5: Let A be an adversary with q_E encryption queries, q_D decryption queries and q_V verification queries where maximum block length of any query is l, $q = q_E + q_D + q_V$ and $q \leq 2^{b-1}$ then for lynx-A9 and lynx-A10 the $Pr[\text{h-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$, where h-Coll^A is the occurrence of h-Coll event when the adversary 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^A bound using the function F^A. Further let K be a randomly chosen key. The proof has been adapted from [54]
For some i, we know $F_{34}^A(K, h[i-1], S[i], M[i]) \rightarrow (h[i], C[i])$. For the sake of proof, we omit $C[i]$. The adversary asks q queries to the function F^A and constructs a tuple set of following form: $\{(K, h_1[i-1], S_1[i], M_1[i], h_1[i]) \ldots (K, h_q[i-1], S_q[i], M_q[i], h_q[i])\}$. The adversary 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:

- $((K, h[i-1], S[i], M[i], h[i]) =$
  $(K, h\,[i-1], S\,[i], M\,[i], h\,[i]))$ $(h[i] = h\,[i])$
  where $h[i] = \widetilde{E}(K, T\,[i], M[i])$ and $h\,[i] = \widetilde{E}(K, T\;[i], M\,[i])$, $T\,[i] = S[i]$ $(M[i]$
  $h[i-1])$, $T\;[i] = S[i]$ $(M\,[i]$ $h\,[i-1])$

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

A.5 Lemma 6

Lemma 6: Let A be an adversary with q_E encryption queries, q_D decryption queries and q_V verification queries where maximum block length of any query is l, $q = q_E + q_D + q_V$ and $q \leq 2^{b-1}$ then for Lynx-A9 and Lynx-A10 the $Pr[\text{h*-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$, where h*-Coll^A is the occurrence of h*-Coll event when the adversary 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 $h[i] = h\,[i]$. Hence, $Pr[\text{h*-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$
- $(h[i-1], h\,[i-1])$ difference and $(M[i], M\,[i])$ difference are not equal. In this case, the proof is same as the **lemma 5** (A.4) i.e., $Pr[\text{h*-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$

□

A.6 Lemma 7

Lemma 7: Let A be an adversary with q_E encryption queries, q_D decryption queries and q_V verification queries where maximum block length of any query is l, $q = q_E + q_D + q_V$ and $q \leq 2^{b-1}$ then for Lynx-A9 and Lynx-A10 the $Pr[\text{input-Coll}^A] \leq l \cdot \frac{q^2}{2^b}$, where input-Coll^A is the occurrence of input-Coll event when the adversary A is interacting with the Lynx.

Proof. The proof for confidentiality is provided in nonce respecting scenario. Hence, the adversary A will get the h values only after all the ciphertext is generated. Clearly, the adversary A cannot change any message in between. We take the notation from **lemma 5** (A.4) and add a *xor* with the next message block to obtain the input collision: $\widetilde{E}(K, T_j[i], M_j[i])$ $M_j[i]$ $M_j[i+1] = \widetilde{E}(K, T_k[i], M_k[i])$ $M_k[i]$ $M_k[i+1]$. Hence, the proof now is exactly same as **lemma 5** (A.4) and hence we omit it for brevity. □

References

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

[2] Hongjun Wu. ACORN v3. https://competitions.cr.yp.to/round3/acornv3.pdf, 2016.

[3] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2: Lightweight Authenticated Encryption and Hashing. *Journal of Cryptology*, 34(3):1–42, 2021.

[4] National Institute of Standards and Technology. https://www.nist.gov/.

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

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

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

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

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

[10] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Transactions on Symmetric Cryptology*, pages 43–120, 2020.

[11] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable Block Ciphers. In *Annual International Cryptology Conference*, pages 31–46. Springer, 2002.

[12] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash. *IACR Transactions on Symmetric Cryptology*, pages 88–131, 2020.

[13] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. ForkAE v.1. *Submission to NIST Lightweight Cryptography Project*, 2019.

[14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 105–125, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

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

[16] 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.

[17] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?). In *Annual International Cryptology Conference*, pages 310–331. Springer, 2001.

[18] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):365–403, 2003.

[19] Mihir Bellare, Phillip Rogaway, and David Wagner. A conventional authenticated-encryption mode. *manuscript, April*, 2003.

[20] Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf, 2004.

[21] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A High-performance Conventional Authenticated Encryption Mode. In *International Workshop on Fast Software Encryption*, pages 408–426. Springer, 2004.

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

[23] Shay Gueron and Yehuda 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*, pages 109–119, 2015.

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

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

[26] Masanobu Katagi, Shiho Moriai, et al. Lightweight cryptography for the internet of things. *Sony Corporation*, 2008:7–10, 2008.

[27] Nicky Mouha. The design space of lightweight cryptography. *Cryptology ePrint Archive*, 2015.

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

[29] Bassam J Mohd, Thaier Hayajneh, and Athanasios V Vasilakos. A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues. *Journal of Network and Computer Applications*, 58:73–93, 2015.

[30] Saurabh Singh, Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–18, 2017.

[31] Vishal A Thakor, Mohammad Abdur Razzaque, and Muhammad RA Khandaker. Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities. *IEEE Access*, 9:28177–28193, 2021.

[32] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Radiogatún, a belt-and-mill hash function. *IACR Cryptol. ePrint Arch.*, 2006:369, 2006.

[33] G.M. Bertoni, Joan Daemen, Michael Peeters, and Gilles Assche. Sponge Functions. *ECRYPT Hash Workshop 2007*, 01 2007.

[34] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Qingju Wang, and Alex Biryukov. Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family. *NIST round*, 2, 2019.

[35] Joan Daemen, Seth Hoffert, Michaël Peeters, G Van Assche, and R Van Keer. Xoodyak, a lightweight cryptographic scheme. 2020.

[36] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan 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] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 218–241, 2018.

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

[39] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. *IACR Cryptol. ePrint Arch.*, 2020:738, 2020.

[40] Subhadeep Banik, Sumit Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present. pages 321–345, 08 2017.

[41] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEAD-A lightweight AEAD stream cipher. *NIST Lightweight Cryptography, Round*, 1, 2019.

[42] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain family of stream ciphers. In *New Stream Cipher Designs*, pages 179–190. Springer, 2008.

[43] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In *Advances in Cryptology – CRYPTO 2016*, pages 123–153, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

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

[46] Donghoon Chang, Nilanjan Datta, Avijit Dutta, Bart Mennink, Mridul Nandi, Somitra Sanadhya, and Ferdinand Sibleyras. Release of unverified plaintext: Tight unified model and application to ANYDAE. *IACR Transactions on Symmetric Cryptology*, pages 119–146, 2019.

[47] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 373–390, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

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

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

[50] Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption, Eurocrypt 2006, lncs vol. 4004, 2006.

[51] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, pages 283–298, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

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

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

[54] John Black, Phillip Rogaway, and Thomas 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*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.