

AES-LBBB: AES Mode for Lightweight and BBB-Secure Authenticated Encryption

Yusuke Naito¹, Yu Sasaki² and Takeshi Sugawara³

¹ Mitsubishi Electric Corporation, Kanagawa, Japan,

Naito.Yusuke@ce.MitsubishiElectric.co.jp

² NTT Secure Platform Laboratories, Tokyo, Japan, yu.sasaki.sk@hco.ntt.co.jp

³ The University of Electro-Communications, Tokyo, Japan, sugawara@uec.ac.jp

Abstract. In this paper, a new lightweight authenticated encryption scheme AES-LBBB is proposed, which was designed to provide backward compatibility with advanced encryption standard (AES) as well as high security and low memory. The primary design goal, backward compatibility, is motivated by the fact that AES accelerators are now very common for devices in the field; we are interested in designing an efficient and highly secure mode of operation that exploits the best of those AES accelerators. The backward compatibility receives little attention in the NIST lightweight cryptography standardization process, in which only 3 out of 32 round-2 candidates are based on AES. Our mode, LBBB, is inspired by the design of ALE in the sense that the internal state size is a minimum $2n$ bits when using a block cipher of length n bits for the key and data. Unfortunately, there is no security proof of ALE, and forgery attacks have been found on ALE. In LBBB, we introduce an additional feed from block cipher's output to the key state via a certain permutation λ , which enables us to prove beyond-birthday-bound (BBB) security. We then specify its AES instance, AES-LBBB, and evaluate its performance for (i) software implementation on a microcontroller with an AES coprocessor and (ii) hardware implementation for an application-specific integrated circuit (ASIC) to show that AES-LBBB performs better than the current state-of-the-art Remus-N2 with AES-128.

Keywords: AES, authenticated encryption, backward compatibility, beyond-birthday-bound security, lightweight, AES accelerator, AES coprocessor.

1 Introduction

Symmetric-key cryptography plays an important role in secure communications owing to its efficiency. In particular, block ciphers are useful primitives to build various symmetric-key schemes. Advanced encryption standard (AES) [Nat01] is arguably the most widely used block cipher worldwide.

A recent trend in the symmetric-key community is to design lightweight schemes that are suitable to implement in extremely resource-restricted environments. Although AES is also efficient for lightweight implementations, the community is seeking designs that are sharply optimized for such implementations. For example, NIST is currently organizing a lightweight cryptography standardization process (NIST LWC) [Nat18].

A main challenge in designing lightweight cryptography schemes is to ensure sufficient security while keeping the implementation efficiency light.

Regarding implementation efficiency, “lightweight” can be interpreted in several different ways, e.g., small memory size in microcontroller implementations, small circuit size in hardware implementations, and low energy consumption in hardware implementations.



Among them, memory size is a common metric for lightweight schemes. Low-memory means smaller circuit area in hardware implementation because register dominates the hardware cost. The benefit of smaller memory size becomes even larger when a side-channel attack is a concern; the common countermeasures such as masking multiply the memory usage for duplicating the sensitive state into several shares.

Achieving low memory is also crucial in software implementation for resource-restricted microcontrollers. In fact, there is a line of work studying the low-memory implementation of cryptography [MM13]. In particular, there are several real-world use cases in which volatile memory (e.g., random access memory (RAM)) is crucial. First, embedded system engineers always have a motivation to minimize memory footprint to reduce the per-chip cost. Second, some platforms have a special secure RAM for storing sensitive data, which is usually limited in size. For example, Microchip’s SAM L11 microcontroller has a 256-byte TrustRAM featuring address scrambling and instantaneous wiping [Mic18]. Third, RAM can be temporarily unavailable even in a resource-rich environment. For example, some boot loaders should run cryptography before the initialization of dynamic RAM (DRAM) when only a tiny cache memory is available.

Regarding security, there is a demand for improving the 64-bit security of the conventional AES-Galois/Counter Mode (GCM) and AES-Counter with CBC-MAC Mode (CCM). NIST LWC explicitly requires better security than AES [Nat01], and many candidates have 128-bit security or close to it, for example, PHOTON-Beetle [BCD⁺19], Xoodyak [DHP⁺19], ISAP [DEM⁺19], Romulus [IKMP20], and SKINNY-AEAD [BJK⁺19].

In the NIST LWC, as listed above, many new designs have been discussed because the goal is to determine a new standard. Among 32 round-2 algorithms in NIST’s process, there are only 3 AES-based schemes: COMET [GJN19], mixFeed [CN19], and SAEAES [NMS⁺19].

On the other hand, AES-based schemes still have a great practical value because the industry has already invested so much in AES accelerators and coprocessors and will continue the investment for backward compatibility. AES-New Instructions (NI) on the Intel processors is by far the most popular AES accelerator available on the millions of central processing units (CPUs) in the field [Gue10]. Many other CPUs, including ARM and RISC-V, have the AES instructions [MNP⁺20]. Besides extending a CPU with new instructions for AES, having an independent AES coprocessor is another way for acceleration, which is common among low-end microcontrollers [Mic20, EHR19]. In particular, some systems use a coprocessor as a root of trust to maintain security even after a main processor becomes compromised, e.g., secure hardware extensions (SHE) for Autosar [Aut19].

There is a line of research for taking advantage of those AES accelerators, and there is another line of work for designing efficient cryptographic algorithms using the AES round function as a building block [JN16, BMR⁺14, WP14]. More recently, researchers have been studying efficient cryptography using AES coprocessors to achieve better performance in embedded devices. Elbaz-Vincent et al. proposed a mode of operation for an AES coprocessor [EHR19]; meanwhile Unterstein et al. studied efficient leakage-resilient cryptography using an AES coprocessor as a building block [USS⁺20].

In this paper, we propose a new lightweight scheme by spotlighting the fact that “backward compatibility” with AES is as important as security and implementation efficiency for providing real-world usability. That is to say, considering that so many accelerators and coprocessors for AES have become widespread, we aim to design new lightweight and secure schemes that can utilize the AES execution ability of those devices, instead of replacing an encryption algorithm with a brand-new one.

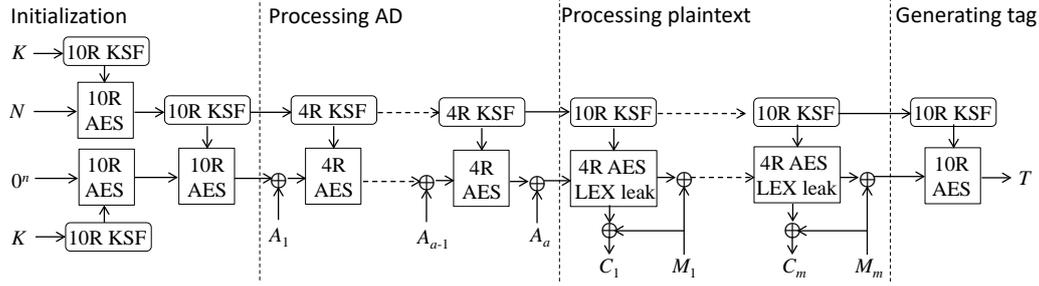


Figure 1: Encryption of ALE. N is a nonce, K is a secret key, A_i is an i -th associated data blocks, M_i/C_i is an i -th plaintext/ciphertext block, and T is a tag. i R KSF/AES iterates the AES key-schedule/round function i times. 4R AES LEX leak iterates the AES round function 4 times and leaks 32 bits of a 128-bit output of each round to encrypt a plaintext block.

Target Implementation Environment

Our primary target is microcontrollers having a coprocessor for computing AES (called AES coprocessor) and other lightweight devices that will communicate with them.

AES coprocessors commonly process the entire AES operation rather than its round function: it accepts a plaintext and a key as input and outputs a ciphertext. It does not output intermediate computing results, e.g., a value after every round. Hence a new scheme must be designed by assuming that devices only can call the entire AES computation.

Microcontrollers have a limited memory size, thus the memory for a new design needs to be as small as possible. Moreover, because the number of devices that will communicate with the devices with AES coprocessors will increase, efficiency in hardware implementations is also important for such a new design proposal.

Existing Designs

If the target security level is 64 bits, there are many existing block-cipher modes that can be instantiated with AES. For example, SAEAES [NMS⁺19], the SAEB mode instantiated with AES, is an AES-based design that is optimized for low-memory implementations. Regarding 128-bit security level, which is our target, the following designs exist.

- There are several beyond-birthday-bound (BBB) secure block-cipher based modes, and some achieve full-bit security. Those modes can be instantiated with AES. However, those modes require a large state or a key state, thus are obviously unsuitable for low-memory implementations.

Iwata et al. designed a block-cipher based mode Remus-N2 [IKMP20], which aims to achieve full-bit security in the ideal model and is suitable for low-memory implementations. The designers of Remus-N2 optimized the entire memory size, thus in their performance evaluation, the mode was instantiated with a lightweight (tweakable) block cipher SKINNY [BJK⁺16] without using a tweak. In the current state-of-the-art, Remus-N2 instantiated with AES is one of the most suitable schemes for our purpose.

- Another direction is to use dedicated AES-based designs. AES-Based Lightweight Authenticated Encryption (ALE) [BMR⁺14], shown in Fig. 1, claims to provide 128-bit security. After being initialized, ALE uses only 128 bits for the key state and 128 bits for the data state, 256 bits in total. To ensure 128-bit security against the birthday attack, this is a minimal requirement, thus ALE is lightweight in hardware

implementation. Besides, the key schedule function (KSF) is applied to the output of the KSF in the previous block. This enables saving of the implementation of KSF^{-1} . (To use the same key in every block while keeping the key state minimal, after $\text{KSF}(K)$ is computed in a certain block, KSF^{-1} needs to be computed to reproduce K for the next block). Moreover, the round transformation is 4 rounds of AES, thus if AES-NI is available, ALE can be computed fast.

However, as explained above, the AES coprocessors cannot produce intermediate values after each round, thus ALE cannot be implemented fast with the AES coprocessors. In addition, the 128-bit security claimed by the designers of ALE is not supported by security proofs. In fact, several researchers found forgery attacks [KR19, WWH⁺13]. Nevertheless, the design of ALE has many attractive features, and obtaining a provable security for ALE-like designs is an interesting challenge.

In summary, by using exiting schemes, Remus-N2 instantiated with AES would be the best choice for microcontrollers having the AES coprocessors. Investigating a new mode with a smaller memory requirement than Remus-N2 is of great interest. The ALE-like structure with provable security is also an interesting direction. Those features are summarized in Table 1.

To illustrate state-of-the-art clear, we also list several features of GCM, OCB3, COFB, and SAEB, which only offer birthday-bound level security in Table 1. GCM and OCB3 enable parallel implementations, thus can be fast but require a relatively large state. COFB and SAEB are sequential modes, thus do not require a large state. SAEB is the smallest but it does not achieve rate 1, while COFB does. Note that security of GCM, OCB3, COFB, and SAEB has been proved in the standard model, which could not achieved by ALE and Remus-N2.

Our Contributions

In this paper, we propose a new mode-of-operation for authenticated encryption that is efficiently implemented on the devices having an AES coprocessor and offers almost 128 bits of provable security. We call our mode LBBB and a particular instance for AES AES-LBBB. AES-128 uses a 128-bit data state and a 128-bit key state, thus we need to use at least a 256-bit state to implement it. To be optimized for lightweight implementations, LBBB is designed so that the state size can be minimal.

LBBB. LBBB, shown in Fig. 2, is a generic framework to construct authenticated encryption schemes with the minimum state size, which is $2n$ bits when using a block cipher of length n bits for the key and the data states. The $2n$ -bit internal state of LBBB consists of an n -bit key and an n -bit data states. The key state is always secret, whereas the data state is secret during the hash function computation but becomes public when encrypting plaintext blocks. LBBB is designed by extending the idea of ALE so that it achieves almost n -bit security. To achieve the security level for online queries (or data complexity), the size of the update state in every block must be $2n$ bits to avoid the birthday attack on the internal state. As ALE, the key state of LBBB is updated by an update-permutation π . Additionally, the key state is updated by a block cipher output via a permutation λ , which is the difference from ALE. This additional procedure ensures that a collision of the internal state requires a query complexity of at least 2^n , because for two data block sequences, after distinct data blocks are processed, the difference is propagated in the whole internal state of $2n$ bits. In addition to these two permutations, LBBB uses a permutation η for the domain separation, which distinguishes whether the last data block is full or not. In the mode level, exact specifications of π , λ , and η are not specified.

For efficiency, LBBB is highly efficient, similarly to the Remus-N2 mode, in the sense that both achieve rate-1 (a block cipher is performed for each n -bit plaintext block). On

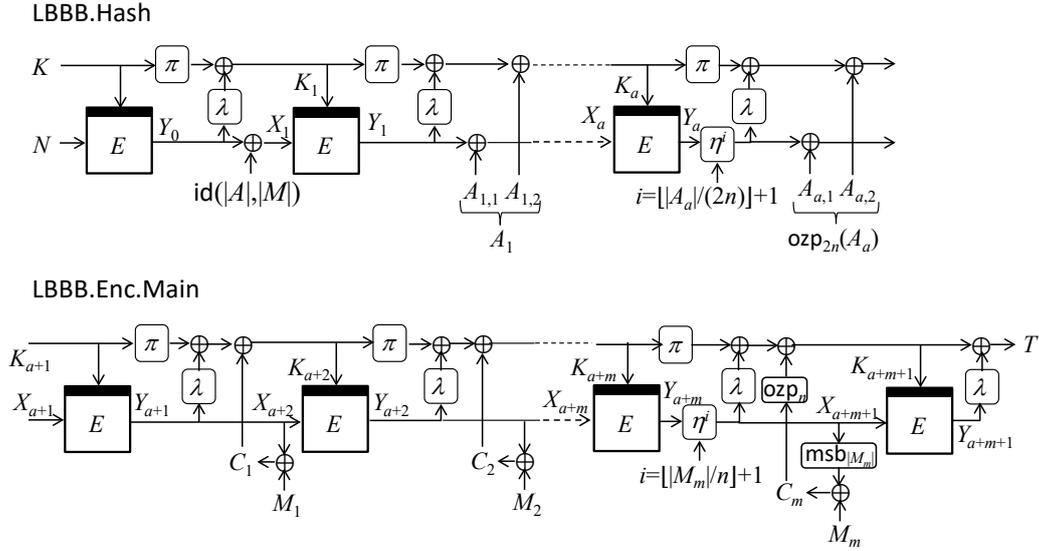


Figure 2: Encryption of LBBB. N , K , A and M are a nonce, a secret key, associated data, and a plaintext. $\text{id}(|A|, |M|)$ is a constant to distinguish whether $A = \varepsilon$ or not and whether $M = \varepsilon$ or not. $(A_{i,1}, A_{i,2})$ is an i -th pair of associated data blocks, M_i/C_i is a plaintext/ciphertext block, and T is a tag. π , λ , and η are permutations.

the other hand, LBBB has the minimum state size that is the advantage over Remus-N2.

For security, we prove that LBBB achieves $O(n - \log_2 n)$ -bit security. Hence, using a block cipher with $n = 128$, the resultant scheme achieves 121-bit security. In our proof, π , λ , and η are assumed to be chosen from a wide class of permutations with properties like a universal hash function, and such permutations can be realized using lightweight operations. Example permutations are powering-up-based schemes [Rog04] that use a multiplication by 2 over $\text{GF}(2^n)$. Note that in the security proof, the underlying block cipher is assumed to be ideal, but even if a practical block cipher is implemented, the construction is secure as long as it resists related-key attacks. Also note that to achieve almost n -bit security and have the minimum state size simultaneously, the key state must be updated as well as the data state because of the birthday attack on the internal state. The structural condition requires a proof in the ideal model.¹

AES-LBBB. For a concrete AES-based scheme called AES-LBBB, an instance of λ and π can be chosen so that it can be further optimized for low-memory software implementations or low-area hardware implementations depending on the design goal. This paper's main goal is for the construction to be efficiently implemented on the devices having the AES coprocessor and offer almost 128 bits of provable security. We use a software-friendly multiplication by 2^8 as λ and π , which can be implemented efficiently, particularly for byte-oriented ciphers like AES. The main computational cost for each message block is a single execution of AES, thus it can be implemented fast with the AES coprocessors.

We evaluate the performance of AES-LBBB in the two different platforms: (i) software implementation on a microcontroller with an AES coprocessor [Mic20] and (ii) hardware implementation for an application-specific integrated circuit (ASIC). We compare the results with the current state of the art (Remus-N2 [IKMP20] instantiated with AES-128) implemented with the same design policy.

¹The standard pseudo-random permutation (PRP) assumption requires a structure in which a key state is fixed to a secret key and is not updated.

Table 1: Comparison of the Properties of the Schemes. The underlying block ciphers are AES-128. SM and IC mean that the security is proven in the standard model and in the ideal model, respectively. BB and BBB mean birthday-bound security and beyond-birthday-bound security, respectively. Backward compatibility shows whether AES coprocessor is used or not. Rate shows a reciprocal of the number of the underlying primitive calls per 128-bit plaintext, which is asymptotic. Note that ALE does not require KSF^{-1} whereas other schemes require KSF^{-1} .

Scheme	Security proof	Security in bits	State size in bits	Backward compatibility	Rate	Parallel	Ref.
GCM	SM (BB)	64	640	Yes	1/2	Yes	[Nat07]
OCB3	SM (BB)	64	512	Yes	1	Yes	[KR11]
COFB	SM (BB)	64	320	Yes	1	No	[CIMN17]
SAEB	SM (BB)	64	256	Yes	1/2	No	[NMSS18]
ALE	No (broken)	—	384	No	1	No	[BMR ⁺ 14]
Remus-N2	IC (BBB)	128	384	Yes	1	No	[IKMP20]
LBBB	IC (BBB)	121	256	Yes	1	No	Ours

Design Extension: Provably Secure ALE-like Constructions and Suitable Primitives.

If we use a block cipher’s key schedule function KSF as π , the construction becomes similar to ALE with respect to the data flow. Different from ALE, the feed function λ slightly increases the implementation cost, but it enables us to prove the security of the construction. A limitation is that π needs to satisfy a certain property, which cannot be satisfied by non-linear KSF , including AES’s KSF . Hence AES’s KSF cannot be used as π .

Here, for an academic purpose, we discuss the design of KSF that achieves provably secure ALE-like modes or efficient implementations such that the implementation of KSF^{-1} can be removed. It is known that AES-192 and AES-256 are non-ideal in the related-key setting because of their KSF [BKN09, BK09], and several researchers have proposed alternative KSF s for AES [MHM⁺02, Nik10, KLPS17]. Among them, Khoo et al. [KLPS17] proposed a KSF that only permutes byte positions, and we show that LBBB can be instantiated efficiently with their KSF . We also show that the KSF of KATAN [CDK09] is very suitable for LBBB, which can achieve a provably secure ALE-like construction.

To avoid having misunderstandings, we would like to stress that our primary goal is reducing the memory size which is beneficial in resource-constrained devices; the BBB-secure parallel mode can be better if the memory size is not a matter. Discussions for design extensions are not limited to lightweight applications as they consider AES-NI.

Paper Outline

Section 2 introduces preliminary knowledge. Section 3 specifies our mode LBBB for a class of permutations λ and π . Section 4 describes security proofs of our mode. Section 5 specifies the AES-based instance AES-LBBB and explains its implementation results. Section 6 discusses a design extension. Section 7 concludes this paper.

2 Preliminaries

2.1 Basic Notation

Let ε be an empty string and $\{0, 1\}^*$ be the set of all bit strings. For an integer $i \geq 0$, let $\{0, 1\}^i$ be the set of all i -bit strings, $\{0, 1\}^0 := \{\varepsilon\}$, and $\{0, 1\}^{\leq i} := \{0, 1\}^1 \cup \{0, 1\}^2 \cup \dots \cup \{0, 1\}^i$. Let 0^i be the bit string of i -bit zeros. For integers $0 \leq i \leq j$, let $[i, j] := \{i, i+1, \dots, j\}$, $(j] := [0, j]$, and $[j := [1, j]$. For integers $0 \leq i \leq x$, let $(x)_i := x(x-1) \cdots (x-i+1)$. For a non-empty set \mathcal{T} , $T \stackrel{\$}{\leftarrow} \mathcal{T}$ means that an element is chosen uniformly at random from \mathcal{T} and assigned to T . The concatenation of two bit strings X and Y is written as $X\|Y$ or XY when no confusion is possible. For integers $0 \leq i \leq n$ and $X \in \{0, 1\}^n$, let $\text{msb}_i(X)$ resp. $\text{lsb}_i(X)$ be the most resp. least significant i bits of X , and let $|X|$ be the bit length of X , i.e., $|X| = n$. For an integer $n \geq 0$ and a bit string X , we denote the parsing into fixed-length n -bit strings as $(X_1, \dots, X_\ell) \stackrel{n}{\leftarrow} X$, where if $X \neq \varepsilon$, then $X = X_1\| \cdots \|X_\ell$, $|X_i| = n$ for $i \in [\ell-1]$, and $0 < |X_\ell| \leq n$; if $X = \varepsilon$, then $\ell = 1$ and $X_1 = \varepsilon$. For an integer $n > 0$, let $\text{ozp}_n : \{0, 1\}^{\leq n} \rightarrow \{0, 1\}^n$ be a one-zero padding function: for $X \in \{0, 1\}^{\leq n}$, $\text{ozp}_n(X) = X$ if $|X| = n$; $\text{ozp}_n(X) = X\|10^{n-1-|X|}$ if $|X| < n$.

2.2 Block Cipher and Block-Cipher-based AE

A block cipher (BC) is a set of permutations indexed by a key. Throughout this paper, the block and key sizes in bits are fixed to n . An encryption of BC is denoted by $E : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, and a decryption of BC is denoted by $E^{-1} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, where the first element is a key. Let \mathcal{BC} be the set of all encryptions of BC.

A nonce-based authenticated encryption (AE) scheme using an encryption of BC E and having a key K , denoted by $\Pi_K[E]$, is a pair of encryption and decryption algorithms $(\Pi.\text{Enc}_K[E], \Pi.\text{Dec}_K[E])$. $\mathcal{K}, \mathcal{N}, \mathcal{M}, \mathcal{C}, \mathcal{A}$, and \mathcal{T} are sets of keys, nonces, plaintexts, ciphertexts, associated data (AD), and tags of $\Pi_K[E]$, respectively. The encryption algorithm having a key $K \in \mathcal{K}$ takes $(N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ and returns, deterministically, $(C, T) \in \mathcal{C} \times \mathcal{T}$. The decryption algorithm having a key $K \in \mathcal{K}$ takes $(N, A, C, T') \in \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$ and returns, deterministically, either the distinguished invalid symbol **reject** $\notin \mathcal{M}$ or $M \in \mathcal{M}$. We require $|\Pi.\text{Enc}_K[E](N, A, M)| = |\Pi.\text{Enc}_K[E](N, A, M')|$ when these outputs are strings and $|M| = |M'|$, and $M = \Pi.\text{Dec}_K[E](N, A, C, T)$ if $(C, T) = \Pi.\text{Enc}_K[E](N, A, M)$.

We follow the security definition given in Namprempre, Rogaway and Shrimpton [NRS14], called **nae**-security, and prove the security of LBBB in the ideal cipher model. **nae**-security in the ideal cipher model is the indistinguishability between $(\Pi_K[E], E, E^{-1})$ and $(\$, \perp, E, E^{-1})$, where a key is defined as $K \stackrel{\$}{\leftarrow} \mathcal{K}$, an ideal cipher is defined as $E \stackrel{\$}{\leftarrow} \mathcal{BC}$, $\$$ is a random-bits oracle that has the same interface as $\Pi.\text{Enc}_K[E]$ and for an encryption query (N, A, M) returns a random bit string of length $|\Pi.\text{Enc}_K[E](N, A, M)|$; \perp is an oracle that returns **reject** for any decryption query. Throughout this paper, we call queries to $\Pi.\text{Enc}_K[E]/\$$ “encryption queries,” $\Pi.\text{Dec}_K[E]/\perp$ “decryption queries,” queries to E “forward offline queries,” and queries to E^{-1} “inverse offline queries.” The **nae**-security advantage function of an adversary \mathbf{A} that returns a decision bit $b \in \{0, 1\}$ after interacting with $\Pi.\text{Enc}_K[E], \Pi.\text{Dec}_K[E], E, E^{-1}$ in the real world or with $\$, \perp, E, E^{-1}$ in the ideal world is defined as

$$\text{Adv}_{\Pi_K[E]}^{\text{nae}}(\mathbf{A}) = \Pr[\mathbf{A}^{\Pi.\text{Enc}_K[E], \Pi.\text{Dec}_K[E], E, E^{-1}} = 1] - \Pr[\mathbf{A}^{\$, \perp, E, E^{-1}} = 1] ,$$

where $\mathbf{A}^{\Pi.\text{Enc}_K[E], \Pi.\text{Dec}_K[E], E, E^{-1}}$ resp. $\mathbf{A}^{\$, \perp, E, E^{-1}}$ is an output of \mathbf{A} in the real world resp. the ideal world, and the probabilities are taken over $K, E, \$$, and \mathbf{A} . We demand that \mathbf{A} is a nonce-respecting adversary, i.e., the same nonce is not repeated for encryption queries, never asks a trivial query, and never repeats a query. A trivial decryption query (N, A, C, T)

is that there is a prior encryption query (N, A, M) such that $(C, T) = \Pi.\text{Enc}_K[E](N, A, M)$, and a trivial forward (resp. inverse) offline query (\hat{K}, \hat{X}) (resp. (\hat{K}, \hat{Y})) is that there is a prior inverse (resp. forward) offline query (\hat{K}, \hat{Y}) (resp. (\hat{K}, \hat{X})) such that $\hat{X} = E^{-1}(\hat{K}, \hat{Y})$ (resp. $\hat{Y} = E(\hat{K}, \hat{X})$).

2.3 Multi-Collision

The upper bound of the number of multi-collision elements is used in our security proof. For a positive integer $s \geq 2$, let \mathcal{S} be a set of size s . For sampling $S_1, \dots, S_q \stackrel{\$}{\leftarrow} \mathcal{S}$, we denote by $N_{q,s}$ the maximum number of multi-collision elements, i.e., $N_{q,s} = \max_{S \in \mathcal{S}} |\{i : S_i = S\}|$. Let $\text{mcoll}(q, s)$ be the expected value of $N_{q,s}$. The upper bound of $\text{mcoll}(q, s)$ is given in Chakraborty et al. [CJN20] and the following lemma.

Lemma 1 (Proposition 1 in [CJN20]). *For $s \geq 4$, we have*

$$\text{mcoll}(q, s) \leq \begin{cases} \frac{4 \log_2 q}{\log_2 \log_2 q} & \text{if } 4 \leq q \leq s, \\ 5(\log_2 s) \lceil \frac{q}{s \log_2 s} \rceil & \text{if } s < q. \end{cases}$$

3 LBBB: Design and Security Bound

3.1 Design

We design a BC-based AE scheme, called LBBB. LBBB is highly secure, meaning almost n -bit security; is highly efficient, meaning rate-1 (a BC is processed once for each n -bit plaintext block); and has the minimum state size. The minimum state size is $2n$ bits when using a block cipher of n -bit data block and n -bit key.

As mentioned in Section 1, LBBB is designed by extending the ALE idea so that it achieves almost n -bit security. To achieve the security level for online queries (or data complexity), the size of the update state must be $2n$ bits due to the birthday attack on the internal state. As ALE, the key state of LBBB is updated by an update-permutation π . Additionally, the key state is updated by a block cipher output via a permutation λ , which is the difference from ALE. The additional procedure ensures that a collision of the internal state requires the 2^n online complexity, because for two data block sequences, after distinct data blocks are processed, the difference is propagated in the whole internal state of $2n$ bits. In addition to these two permutations, LBBB uses a permutation η for domain separation, which distinguishes whether the last data block is full or not.

The specification of LBBB is given in Algorithm 1 and is also depicted in Fig. 2. LBBB.Enc is the encryption of LBBB, and LBBB.Dec is the decryption. LBBB.Hash processes a secret key K , a nonce N , and AD A . LBBB.Enc.Main processes a plaintext M and generates a ciphertext C and a tag T . LBBB.Dec.Main processes a ciphertext C and checks the integrity. If the input is not forged, it returns the plaintext M . In LBBB.Hash, a flag $\text{id}(|A|, |M|)$ is used to distinguish whether A is empty or not and whether M/C is empty or not: $\text{id}(|A|, |M|) = 0^n$ if $|A| > 0 \wedge |M| > 0$; $\text{id}(|A|, |M|) = 0^{n-1}1$ if $|A| = 0 \wedge |M| > 0$; $\text{id}(|A|, |M|) = 0^{n-2}10$ if $|A| > 0 \wedge |M| = 0$; $\text{id}(|A|, |M|) = 0^{n-2}11$ if $|A| = 0 \wedge |M| = 0$. π , λ , and η are n -bit permutations.² For an n -bit permutation p , $X \in \{0, 1\}^n$, and an integer i , $p^i(X)$ denotes the output of i iterations of p for the input X . Note that $p^0(X) := X$ and $p^{-i}(X)$ are the i iterations of the inverse permutation p^{-1} .

²In LBBB, the flag is taken before processing AD blocks and plaintext/ciphertext blocks. After taking the flags, internal state values become distinct even when nonces are the same but the conditions for $|A|$ and $|M|$ are distinct, thereby making the security proof simpler. Moreover, the flag has an effect on preventing length extension type attacks.

Algorithm 1 LBBB**Encryption** LBBB.Enc_K[E](N, A, M)1: (KS, S) ← LBBB.Hash_K[E](N, A, M); **return** LBBB.Enc.Main_K[E](KS, S, M)**Decryption** LBBB.Dec_K[E](N, A, C, T')1: (KS, S) ← LBBB.Hash_K[E](N, A, C); **return** LBBB.Dec.Main_K[E](KS, S, C, T')**Hash** LBBB.Hash_K[E](N, A, M)

1: S ← E(K, N); KS ← π(K) ⊕ λ(S); S ← S ⊕ id(|A|, |M|); (A₁, ..., A_a) ←²ⁿ A;
2: **if** A = ε **then goto** Step 10
3: **for** i = 1, ..., a − 1 **do**
4: S ← E(KS, S); (A_{i,1}, A_{i,2}) ←ⁿ A_i
5: KS ← π(KS) ⊕ λ(S) ⊕ A_{i,2}; S ← S ⊕ A_{i,1};
6: **end for**
7: S ← E(KS, S); (A_{a,1}, A_{a,2}) ←ⁿ ozp_{2n}(A_a)
8: **if** |A_a| < 2n **then** S ← η(S); **else** S ← η²(S)
9: KS ← π(KS) ⊕ λ(S) ⊕ A_{a,2}; S ← S ⊕ A_{a,1}
10: **return** (KS, S)

Main Procedure of Encryption LBBB.Enc.Main_K[E](KS, S, M)

1: (M₁, ..., M_m) ←ⁿ M; **if** M = ε **then goto** Step 7
2: **for** i = 1, ..., m − 1 **do**
3: S ← E(KS, S); C_i ← S ⊕ M_i; KS ← π(KS) ⊕ λ(S) ⊕ C_i
4: **end for**
5: S ← E(KS, S); **if** |M_m| < n **then** S ← η(S); **else** S ← η²(S)
6: C_m ← msb_{|M_m|}(S) ⊕ M_m; KS ← π(KS) ⊕ λ(S) ⊕ ozp_n(C_m)
7: S ← E(KS, S); T ← π(KS) ⊕ λ(S)
8: C ← C₁ || ... || C_m; **return** (C, T)

Main Procedure of Decryption LBBB.Dec.Main_K[E](KS, S, C, T')

1: (C₁, ..., C_m) ←ⁿ C; **if** C = ε **then goto** Step 7
2: **for** i = 1, ..., m − 1 **do**
3: S ← E(KS, S); M_i ← S ⊕ C_i; KS ← π(KS) ⊕ λ(S) ⊕ C_i
4: **end for**
5: S ← E(KS, S); **if** |C_m| < n **then** S ← η(S); **else** S ← η²(S)
6: M_m ← msb_{|C_m|}(S) ⊕ C_m; KS ← π(KS) ⊕ λ(S) ⊕ ozp_n(C_m)
7: S ← E(KS, S); T ← π(KS) ⊕ λ(S)
8: **if** T = T' **then return** M ← M₁ || ... || M_m; **else return reject**

for the input X . In our proof, we assume that π , λ , and η have the following properties. Here, ℓ_{\max} denotes the maximum number of BC calls in LBBB.Enc or LBBB.Dec.

- π is linear and has the property that for any $Y \in \{0, 1\}^n$ and $i, j \in (\ell_{\max}]$ such that $i \neq j$, the equation $\pi^i(X) \oplus \pi^j(X) = Y$ offers a unique solution for X .
- λ is linear and has the property that for any $Y \in \{0, 1\}^n$, the equation $X \oplus \lambda(X) = Y$ offers a unique solution for X .
- η is linear and has the property that for any $Y \in \{0, 1\}^n$ and $i, j \in (2]$ such that $i \neq j$, the equation $\eta^i(X) \oplus \eta^j(X) = Y$ offers a unique solution for X .

Candidates of these permutations are discussed in Section 5.1.

We next explain the reasons the above properties are introduced.

- π : In the security proof, we need to evaluate the collision probability of the key state. Using the property, we can use the randomness of the key K for a collision of key state elements at distinct blocks, meaning that for $i \neq j$ and some Y , the collision is denoted by $\pi^i(K) \oplus \pi^j(K) = Y$.
- λ : The key state is updated as $\text{KS} \leftarrow \pi(\text{KS}) \oplus \lambda(S) \oplus C_i$ and the ciphertext block C_i is defined as $C_i \leftarrow S \oplus M_i$. The property of λ ensures that S , which is the output of E , is not canceled out in the key state update, thus the randomness of S can be used to evaluate the collision probability of key state elements.
- η : This property protects the length extension attack.

3.2 Security Bound

We show that LBBB achieves $(n - \log_2 n)$ -bit security. The security bound is given in the following theorem. The proof is given in Section 4.

Theorem 1. *Let \mathbf{A} be an adversary making at most q_p offline queries, q_e encryption queries with at most σ_e BC calls in total, and q_d decryption queries with at most σ_d BC calls in total such that $2q_p + \sigma \leq 2^n$. Let $\sigma := \sigma_e + \sigma_d$ the total number of BC calls by all queries. Then, we have*

$$\text{Adv}_{\text{LBBB}_K[E]}^{\text{nae}}(\mathbf{A}) \leq \frac{q_p + \sigma + 3q_p \text{mcoll}(\sigma_e, 2^n) + \sigma_d \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n} + \frac{2q_p \sigma + 2\sigma^2}{(2^n - \sigma)^2}.$$

By using Lemma 1, the numbers of multi-collision elements is upper bounded by $O(n)$. Thus, the advantage function is upper bounded by $O((n(q_p + \sigma_d) + \sigma)/2^n)$, and LBBB achieves $(n - \log_2 n)$ -bit security.

4 Proof of Theorem 1

4.1 Overview

We give an overview of the security proof. The goal of this proof is to upper-bound the probability of distinguishing between $(\text{LBBB.Enc}_K[E], \text{LBBB.Dec}_K[E], E, E^{-1})$ and $(\$, \perp, E, E^{-1})$. To have the distinguishing probability, we find structural differences between $(\text{LBBB.Enc}_K[E], \text{LBBB.Dec}_K[E])$ and $(\$, \perp)$, as $(\text{LBBB.Enc}_K[E], \text{LBBB.Dec}_K[E])$ define outputs using E whereas $(\$, \perp)$ are monolithic and don't use E , and upper-bound the probabilities that the differences occur.

The first difference comes from collisions in inputs to E . The collisions fall into the following two types.

- The first type is of collisions in inputs to E defined by online queries. The inputs are defined by $\text{LBBB.Enc}_K[E]$ or $\text{LBBB.Dec}_K[E]$. As $\text{LBBB.Enc}_K[E]$ and $\text{LBBB.Dec}_K[E]$ have structures of iterating E , an input collision propagates the output. On the other hand, $\$$ and \perp don't have such structures. Thus, the input collision makes a difference between the real and ideal worlds.
- The second type is of collisions between inputs of E defined by online and offline queries. As $(\$, \perp)$ are monolithic, a collision between inputs defined by online queries and defined by offline queries makes a difference between the real and ideal worlds.

Thus, the probability that the difference occurs is introduced in the distinguishing probability. Regarding inputs to E whose plaintext elements are revealed via the corresponding ciphertext blocks, we need to upper-bound the collision probability using only the randomnesses of the key elements of n bits. However, a naive birthday analysis on the key elements

degrades the security level to $n/2$ bits. To overcome the issue, we use the multi-collision technique on the ciphertext blocks. Regarding other inputs, the plaintext and key elements are not revealed, thus we can use the randomnesses of $2n$ bits. Then, we will show that the probability that the difference occurs is at most $O(nq_p/2^n + n\sigma_d/2^n + \sigma^2/2^{2n} + q_p\sigma/2^{2n})$.

The second difference comes from a key recovery in the real world, as using a key one can calculate all outputs of LBBB. Using the randomness of the key of n bits, we have the upper bound of the probability for the difference: $O((q_p + \sigma)/2^n)$.

The third difference comes from the difference between the randomnesses of outputs of $\text{LBBB.Enc}_K[E]$ and of $\$,$ where outputs of $\text{LBBB.Enc}_K[E]$ are defined using E whereas outputs of $\$$ are chosen uniformly at random. As an ideal cipher E , fixing the key element, becomes an n -bit random permutation, in $\text{LBBB.Enc}_K[E]$ if the same key element appears twice, the ciphertext elements are not truly random values. Thus, the probability that the difference occurs is introduced in the distinguishing probability. We will show that the probability is at most $O(\sigma^2/2^{2n})$.

The last difference comes from the difference between outputs of $\text{LBBB.Dec}_K[E]$ and of $\perp,$ as $\text{LBBB.Dec}_K[E]$ returns a plaintext if a forgery succeeds. Thus, the probability that the difference occurs is introduced in the distinguishing probability. We will show that the probability that the difference occurs (a forgery succeeds) is at most $O(q_d/2^{2n})$.

In the following, we give the detail of the security proof. Our proof uses the coefficient H technique [Pat08], where the above differences are defined as bad events.

4.2 Definition

Let $\sigma_{e,1}$ resp. $\sigma_{d,1}$ be the total number of BC calls in LBBB.Hash by encryption resp. decryption queries, $\sigma_{e,2} := \sigma_e - \sigma_{e,1}$ and $\sigma_{d,2} := \sigma_d - \sigma_{d,1}$.

Online query numbers 1 to q_e are assigned to encryption queries, and those $q_e + 1$ to $q_e + q_d$ are assigned to decryption queries. Hence, for $\alpha \in [q_e],$ α -th encryption query is said to be the α -th online query, and for $\beta \in [q_d],$ β -th decryption query is said to be the $(q_e + \beta)$ -th online query.

For $\alpha \in [q_e + q_d],$ values/variables defined at the α -th online query are denoted by using the superscript of $(\alpha),$ and the lengths a and m at the α -th online query are denoted by a_α and $m_\alpha,$ respectively. Let $\ell_\alpha := a_\alpha + m_\alpha$ be the length of data blocks at the α -th online query. The initial BC call whose input is $(K, N^{(\alpha)})$ is regarded as the 0-th BC call, and an input-output triple of an i -th BC call is denoted by $(K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}).$

For $\alpha \in [q_p],$ the input-output triple at the α -th offline query is denoted by $(\hat{K}^{(\alpha)}, \hat{X}^{(\alpha)}, \hat{Y}^{(\alpha)}),$ where $\hat{Y}^{(\alpha)} = E(\hat{K}^{(\alpha)}, \hat{X}^{(\alpha)})$ if the offline query is a forward one, and $\hat{X}^{(\alpha)} = E^{-1}(\hat{K}^{(\alpha)}, \hat{Y}^{(\alpha)})$ if the offline query is an inverse one.

For $\alpha \in [q_e + q_d],$ data blocks are denoted by $D_0^{(\alpha)} := N^{(\alpha)}, D_i^{(\alpha)} := A_i^{(\alpha)}$ for $i = 1, \dots, a_\alpha - 1,$ $D_{a_\alpha}^{(\alpha)} := \text{ozp}_{2n}(A_{a_\alpha}^{(\alpha)}), D_{a_\alpha+i}^{(\alpha)} := C_{a_\alpha+i}^{(\alpha)}$ for $i = 1, \dots, m_\alpha - 1,$ $D_{\ell_\alpha}^{(\alpha)} := \text{ozp}_n(C_{\ell_\alpha}^{(\alpha)}).$ For $i \in [a_\alpha],$ let $D_{i,1}^{(\alpha)} := \text{msb}_n(D_i^{(\alpha)})$ and $D_{i,2}^{(\alpha)} := \text{lsb}_n(D_i^{(\alpha)}).$ For $i \in [a_\alpha + 1, \ell_\alpha],$ let $D_{i,1} := 0^n$ and $D_{i,2} := D_i^{(\alpha)}.$

For $\alpha \in [q_e + q_d], i \in (\ell_\alpha],$ let $Y_i^{*(\alpha)} := \lambda(Y_i^{(\alpha)})$ if $i \neq a_\alpha$ and $i \neq \ell_\alpha;$ $Y_i^{*(\alpha)} := \lambda(\eta^{\lfloor |A_{a_\alpha}^{(\alpha)}|/(2n) \rfloor + 1}(Y_i^{(\alpha)}))$ if $i = a_\alpha;$ $Y_i^{*(\alpha)} := \lambda(\eta^{\lfloor |C_{m_\alpha}^{(\alpha)}|/n \rfloor + 1}(Y_i^{(\alpha)}))$ if $i = \ell_\alpha.$ For $\alpha \in [q_e + q_d], i \in (\ell_\alpha],$ let $\mathcal{Y}_i^{(\alpha)} := \{Y_0^{(\alpha)}, Y_1^{(\alpha)}, \dots, Y_i^{(\alpha)}\}.$

We consider an array of data blocks with distinguishing identifiers $\delta_i^{(\alpha)}$ for an α -th

Algorithm 2 Dummy Internal Values for Initial Blocks

```

1:  $K \xleftarrow{\$} \{0,1\}^n$  // dummy key
2: for  $\kappa \in \{0,1\}^n$  do  $E[\kappa] \leftarrow \emptyset$ 
3: for  $(\kappa, X) \in \{0,1\}^n \times \{0,1\}^n$  do  $E_{\text{Enc}}[\kappa, X] \leftarrow \varepsilon$ 
4: for  $\alpha \in [q_e + q_d]$  do
5:   if  $\exists \beta \in [\alpha - 1]$  s.t.  $N^{(\alpha)} = N^{(\beta)}$  then  $Y_0^{(\alpha)} \leftarrow Y_0^{(\beta)}$ 
6:   if  $\forall \beta \in [\alpha - 1] : N^{(\alpha)} \neq N^{(\beta)}$  then  $\{Y_0^{(\alpha)} \xleftarrow{\$} \{0,1\}^n \setminus E[K]; E[K] \xleftarrow{\cup} \{Y_0^{(\alpha)}\}\}$ 
7:    $K_1^{(\alpha)} \leftarrow \pi(K) \oplus \lambda(Y_0^{(\alpha)}); X_1^{(\alpha)} \leftarrow Y_0^{(\alpha)} \oplus \text{id}(|A^{(\alpha)}|, |M^{(\alpha)}|)$ 
8: end for

```

Algorithm 3 Dummy Internal Values for Encryption Queries

```

1: for  $\alpha \in [q_e]$  do
2:   //—— dummy internal values for LBBB.Hash ——
3:   for  $i \in [a_\alpha]$  do
4:      $Y_i^{(\alpha)} \xleftarrow{\$} \{0,1\}^n; E[K_i^{(\alpha)}] \xleftarrow{\cup} \{Y_i^{(\alpha)}\}$ 
5:      $tmp \leftarrow Y_i^{(\alpha)}$ ; if  $i = a_\alpha$  then  $tmp \leftarrow \eta^{\lfloor |A_{a_\alpha}^{(\alpha)}|/(2n) \rfloor + 1}(Y_i^{(\alpha)})$ 
6:      $K_{i+1}^{(\alpha)} \leftarrow \pi(K_i^{(\alpha)}) \oplus \lambda(tmp) \oplus A_{i,2}^{(\alpha)}; X_{i+1}^{(\alpha)} \leftarrow tmp \oplus A_{i,1}^{(\alpha)}$ 
7:   end for
8:   //—— dummy internal values for LBBB.Enc.Main ——
9:   for  $i \in [a_\alpha + 1, \ell_\alpha]$  do
10:     $tmp \xleftarrow{\$} \{0,1\}^{n - |C_{i-a_\alpha}^{(\alpha)}|}; X_{i+1}^{(\alpha)} \leftarrow (C_{i-a_\alpha}^{(\alpha)} \oplus M_{i-a_\alpha}^{(\alpha)}) \parallel tmp;$ 
11:     $Y_i^{(\alpha)} \leftarrow X_{i+1}^{(\alpha)}$ ; if  $i = \ell_\alpha$  then  $Y_i^{(\alpha)} \leftarrow \eta^{-(\lfloor |M_{\ell_\alpha}^{(\alpha)}|/n \rfloor + 1)}(X_{i+1}^{(\alpha)})$ 
12:     $K_{i+1}^{(\alpha)} \leftarrow \pi(K_i^{(\alpha)}) \oplus \lambda(X_{i+1}^{(\alpha)}) \oplus \text{ozp}_n(C_{i-a_\alpha}^{(\alpha)})$ 
13:     $E[K_i^{(\alpha)}] \xleftarrow{\cup} \{Y_i^{(\alpha)}\}; E_{\text{Enc}}[K_i^{(\alpha)}, X_i^{(\alpha)}] \leftarrow Y_i^{(\alpha)}$ 
14:   end for
15:    $Y_{\ell_\alpha+1}^{(\alpha)} \leftarrow \lambda^{-1}(T^{(\alpha)} \oplus \pi(K_{\ell_\alpha+1}^{(\alpha)}))$ 
16:    $E[K_{\ell_\alpha+1}^{(\alpha)}] \xleftarrow{\cup} \{Y_{\ell_\alpha+1}^{(\alpha)}\}; E_{\text{Enc}}[K_{\ell_\alpha+1}^{(\alpha)}, X_{\ell_\alpha+1}^{(\alpha)}] \leftarrow Y_{\ell_\alpha+1}^{(\alpha)}$ 
17: end for

```

online query: $((D_0^{(\alpha)}, \delta_0^{(\alpha)}), (D_1^{(\alpha)}, \delta_1^{(\alpha)}), \dots, (D_{\ell_\alpha}^{(\alpha)}, \delta_{\ell_\alpha}^{(\alpha)}))$ where

$$\delta_i^{(\alpha)} = \begin{cases} \text{id}(|A^{(\alpha)}|, |M^{(\alpha)}|) & \text{if } i = 0 \\ 0 & \text{if } (0 < i < a_\alpha) \vee (a_\alpha < i < \ell_\alpha) \\ 1 & \text{if } (i = a_\alpha \wedge |A_i^{(\alpha)}| < 2n) \vee (i = \ell_\alpha \wedge |M^{(\alpha)}| < n) \\ 2 & \text{if } (i = a_\alpha \wedge |A^{(\alpha)}| = 2n) \vee (i = \ell_\alpha \wedge |M^{(\alpha)}| = n), \end{cases}$$

where $\delta_i^{(\alpha)} = 0$ if $D_i^{(\alpha)}$ is not the last block; $\delta_i^{(\alpha)} = 1$ if $D_i^{(\alpha)}$ is the last block and a one-zero string is padded; $\delta_i^{(\alpha)} = 2$ if $D_i^{(\alpha)}$ is the last block and a one-zero string is not padded. For $i \in [\ell_\beta]$, the array up to the i -th block is denoted by $\text{pf}(\alpha, i) = ((D_0^{(\alpha)}, \delta_0^{(\alpha)}), (D_1^{(\alpha)}, \delta_1^{(\alpha)}), \dots, (D_i^{(\alpha)}, \delta_i^{(\alpha)}))$.

4.3 Proof Strategy and Upper Bound

4.3.1 Dummy Internal Values in Ideal World

In the ideal world, after making all queries, Algorithms 2, 3, and 4 are performed. First, Algorithm 2 defines a dummy key K , dummy outputs $Y_0^{(\alpha)}$ for the 0-th BC call, and dummy inputs $(K_1^{(\alpha)}, X_1^{(\alpha)})$ for the 1-st BC call. Then, Algorithm 3 defines dummy

Algorithm 4 Dummy Internal Values for Decryption Queries

```

1: for  $\alpha \in [q_e + 1, q_e + q_d]$  do
2:   //— dummy internal values for LBBB.Hash —
3:   for  $i \in [a_\alpha]$  do
4:     if  $\forall \beta \in [\alpha - 1] : \text{pf}(\alpha, i - 1) \neq \text{pf}(\beta, i - 1)$  then  $Y_i^{(\alpha)} \xleftarrow{\$} \{0, 1\}^n \setminus E[K_i^{(\alpha)}]$ 
5:     if  $\exists \beta \in [\alpha - 1]$  s.t.  $\text{pf}(\alpha, i - 1) = \text{pf}(\beta, i - 1)$  then  $Y_i^{(\alpha)} \leftarrow Y_i^{(\beta)}$ 
6:      $E[K_i^{(\alpha)}] \leftarrow \bigcup \{Y_i^{(\alpha)}\}; tmp \leftarrow Y_i^{(\alpha)}; \text{if } i = a_\alpha \text{ then } tmp \leftarrow \eta^{\lfloor |A_{a_\alpha}^{(\alpha)}|/(2n) \rfloor + 1}(Y_i^{(\alpha)})$ 
7:      $K_{i+1}^{(\alpha)} \leftarrow \pi(K_i^{(\alpha)}) \oplus \lambda(tmp) \oplus A_{i,2}^{(\alpha)}; X_{i+1}^{(\alpha)} \leftarrow tmp \oplus A_{i,1}^{(\alpha)}$ 
8:   end for
9:   //— dummy internal values for LBBB.Dec.Main —
10:  for  $i \in [a_\alpha + 1, \ell_\alpha + 1]$  do
11:    if  $\exists \beta \in [\alpha - 1]$  s.t.  $\text{pf}(\alpha, i - 1) = \text{pf}(\beta, i - 1)$  then  $Y_i^{(\alpha)} \leftarrow Y_i^{(\beta)}$ 
12:    if  $\forall \beta \in [\alpha - 1] : \text{pf}(\alpha, i - 1) \neq \text{pf}(\beta, i - 1)$  then
13:       $Y_i^{(\alpha)} \leftarrow E(K_i^{(\alpha)}, X_i^{(\alpha)})$ 
14:      if  $E_{\text{Enc}}[K_i^{(\alpha)}, X_i^{(\alpha)}] \neq \varepsilon$  then  $Y_i^{(\alpha)} \leftarrow E_{\text{Enc}}[K_i^{(\alpha)}, X_i^{(\alpha)}]$ 
15:    end if
16:     $X_{i+1}^{(\alpha)} \leftarrow Y_i^{(\alpha)}; \text{if } i = \ell_\alpha \text{ then } X_{i+1}^{(\alpha)} \leftarrow \eta^{\lfloor |C_{m_\alpha}^{(\alpha)}|/n \rfloor + 1}(Y_i^{(\alpha)})$ 
17:     $K_{i+1}^{(\alpha)} \leftarrow \pi(K_i^{(\alpha)}) \oplus \lambda(X_{i+1}^{(\alpha)}) \oplus \text{ozp}_n(C_{a_\alpha+i}^{(\alpha)})$ 
18:  end for
19:   $T^{(\alpha)} \leftarrow \pi(K_{\ell_\alpha+1}^{(\alpha)}) \oplus \lambda(Y_{\ell_\alpha+1}^{(\alpha)})$ 
20: end for

```

input-output triples of LBBB.Enc. Dummy outputs for LBBB.Hash are chosen uniformly at random from $\{0, 1\}^n$, and other dummy outputs are defined using ciphertext blocks or tags. Finally, Algorithm 4 defines dummy input-output triples of LBBB.Dec. Dummy outputs for LBBB.Hash are chosen uniformly at random from $\{0, 1\}^n \setminus E[K_i^{(\alpha)}]$, where $E[K_i^{(\alpha)}]$ keeps previous dummy outputs with the key element $K_i^{(\alpha)}$, except for outputs of E . Dummy outputs for LBBB.Dec.Main are defined using E or the table E_{Enc} , where E_{Enc} keeps dummy input-output triples for LBBB.Enc.Main defined in Algorithm 3.

4.3.2 Adversary's View

In this proof, after making all queries, an adversary is permitted to obtain all (dummy) internal values. Note that the revealed internal values do not reduce adversary's advantage. The adversary's view is summarized in a transcript τ ,³ which is equal to

$$\begin{aligned}
& \left\{ (N^{(\alpha)}, A^{(\alpha)}, M^{(\alpha)}, C^{(\alpha)}, T^{(\alpha)}) : \alpha \in [q_e] \right\} \cup \\
& \left\{ (N^{(q_e+\alpha)}, A^{(q_e+\alpha)}, C^{(q_e+\alpha)}, T^{(q_e+\alpha)}, rv^{(q_e+\alpha)}) : \alpha \in [q_d] \right\} \cup \\
& \left\{ (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e + q_d], i \in (\ell_\alpha + 1) \right\} \cup \\
& \left\{ (\hat{K}^{(\alpha)}, \hat{X}^{(\alpha)}, \hat{Y}_i^{(\alpha)}) : \alpha \in [q_p] \right\},
\end{aligned}$$

where $rv^{(q_e+\alpha)}$ is a response of the α -th decryption query: **reject** or a (decrypted) plaintext. Note that query-response tuples $\{(N^{(\alpha)}, A^{(\alpha)}, M^{(\alpha)}, C^{(\alpha)}, T^{(\alpha)}) : \alpha \in [q_e]\}$ and $\{(N^{(q_e+\alpha)}, A^{(q_e+\alpha)}, C^{(q_e+\alpha)}, T^{(q_e+\alpha)}) : \alpha \in [q_d]\}$ can be recovered from the (dummy) input-output tuples $\{(K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e + q_d], i \in (\ell_\alpha + 1)\}$.

³Defining the dummy internal values in the ideal world can be seen as a simulator that mimics the internal values in the real world.

4.3.3 Coefficient H Technique

This proof uses the coefficient H technique [Pat08]. Let T_R be a transcript in the real world obtained by sampling K and E . Let T_I be a transcript in the ideal world obtained by sampling $\$$ and dummy values. We call a transcript τ *valid* if $\Pr[\mathsf{T}_I = \tau] > 0$. Let \mathcal{T} be all valid transcripts. Then,

$$\mathbf{Adv}_{\text{LBBB}_\kappa[E]}^{\text{nae}}(\mathbf{A}) = \text{SD}(\mathsf{T}_R, \mathsf{T}_I) = \frac{1}{2} \sum_{\tau \in \mathcal{T}} |\Pr[\mathsf{T}_R = \tau] - \Pr[\mathsf{T}_I = \tau]|,$$

and the statistical distance $\text{SD}(\mathsf{T}_R, \mathsf{T}_I)$ can be upper-bounded using the following lemma. Here, \mathcal{T} is partitioned into two transcripts: good transcripts $\mathcal{T}_{\text{good}}$ and bad transcripts \mathcal{T}_{bad} .

Lemma 2 ([Pat08]). *Let $0 \leq \mu \leq 1$ be such that for all $\tau \in \mathcal{T}_{\text{good}}$, $\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]} \geq 1 - \mu$. Then, $\text{SD}(\mathsf{T}_R, \mathsf{T}_I) \leq \Pr[\mathsf{T}_I \in \mathcal{T}_{\text{bad}}] + \mu$.*

In the following proof, good and bad transcripts are defined. Then $\Pr[\mathsf{T}_I \in \mathcal{T}_{\text{bad}}]$ is upper-bounded, and $\frac{\Pr[\mathsf{T}_R = \tau]}{\Pr[\mathsf{T}_I = \tau]}$ is lower-bounded. Finally, an upper bound of $\mathbf{Adv}_{\text{LBBB}_\kappa[E]}^{\text{nae}}(\mathbf{A})$ is obtained, putting the bounds into the above lemma.

4.3.4 Definitions of Good and Bad Transcripts

A set of bad transcripts \mathcal{T}_{bad} satisfies one of the following bad events, and a set of good transcripts $\mathcal{T}_{\text{good}}$ is defined as $\mathcal{T}_{\text{good}} = \mathcal{T} \setminus \mathcal{T}_{\text{bad}}$.

In the ideal world, dummy internal values for online queries (except for LBBB.Dec.Main) are defined independently of E . On the other hand, in the real world, internal values and responses of offline queries are defined using E . Thus, bad events are defined so that the difference appears.

bad₁, bad₂, bad₃: In the ideal world, even if a collision occurs between dummy internal values and offline query-response triples, the output for the online query is defined independently of the output for the offline query. The events handle the collisions, and it can be ensured that if these bad events do not occur, an adversary cannot distinguish between the real and ideal worlds using the difference between online and offline queries.

bad₁ : $\exists \alpha \in [q_p]$ s.t. $\hat{K}^{(\alpha)} = K$.

bad₂ : $\exists \alpha \in [q_p], \beta \in [q_e], i \in [\ell_\beta + 1]$ s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge (\hat{X}^{(\alpha)} = X_i^{(\beta)} \vee \hat{Y}^{(\alpha)} = Y_i^{(\beta)})$.

bad₃ : $\exists \alpha \in [q_p], \beta \in [q_e + 1, q_e + q_d], i \in [a_\beta]$
s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge (\hat{X}^{(\alpha)} = X_i^{(\beta)} \vee \hat{Y}^{(\alpha)} = Y_i^{(\beta)})$.

bad₄, bad₅, bad₆: In the ideal world, for online queries, dummy outputs with distinct prefix data blocks are independently defined, even if a dummy input collision occurs. On the other hand, in the real world, if an input collision occurs for online queries, the outputs are the same, since all outputs are defined by E . Hence, the bad events handle the collisions for online queries, and it can be ensured that if the bad events do not occur, an adversary cannot distinguish between the real and ideal worlds using the difference for internal values defined by online queries.

bad₄ : $\exists \alpha, \beta \in [q_e], i \in [\ell_\alpha + 1], j \in [\ell_\beta + 1]$
s.t. $(\alpha, i) \neq (\beta, j) \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge (X_i^{(\alpha)} = X_j^{(\beta)} \vee Y_i^{(\alpha)} = Y_j^{(\beta)})$.

bad₅ : $\exists \alpha \in [q_e + q_d], \beta \in [q_e + 1, q_e + q_d]$,
 $(i, j) \in ([\ell_\alpha + 1] \times [\ell_\beta + 1]) \setminus ([a_\alpha + 2, \ell_\alpha + 1] \times [a_\beta + 2, \ell_\beta + 1])$
s.t. $\text{pf}(\alpha, i - 1) \neq \text{pf}(\beta, j - 1) \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge (X_i^{(\alpha)} = X_j^{(\beta)} \vee Y_i^{(\alpha)} = Y_j^{(\beta)})$.

bad₆ : $\exists \alpha \in [q_e + q_d], i \in [\ell_\alpha + 1]$ s.t. $K_i^{(\alpha)} = K$.

bad₇: The last event handles a forgery. In the ideal world, all responses of decryption queries are **reject**, whereas in the real world, a plaintext is sometimes returned. It can be ensured that if a bad event does not occur, an adversary cannot distinguish between the real and ideal worlds using a forgery.

bad₇ : $\exists \alpha \in [q_e + 1, q_e + q_d]$ s.t. $T^{(\alpha)} = T'^{(\alpha)}$.

As mentioned above, these bad events handle the differences between the real and ideal worlds. Thus, it can be ensured that an adversary cannot distinguish between the real and ideal worlds as long as no bad event occurs, and for any $\tau \in \mathcal{T}_{\text{good}}$, $\frac{\Pr[\mathbf{T}_R = \tau]}{\Pr[\mathbf{T}_I = \tau]} \geq 1$. Details of the analysis are given in Section 4.5. Analyses of the bad events (an evaluation of $\Pr[\mathbf{T}_I \in \mathcal{T}_{\text{bad}}]$) are given in Section 4.4.

4.4 Evaluating $\Pr[\mathbf{T}_I \in \mathcal{T}_{\text{bad}}]$

Without loss of generality, assume that an adversary aborts if one of the bad events occurs. Thus, for $i \in [7]$, **bad_i** occurs as long as other bad events have not occurred. Then, we have

$$\Pr[\mathbf{T}_I \in \mathcal{T}_{\text{bad}}] \leq \sum_{i=1}^7 \Pr[\text{bad}_i] .$$

These upper bounds are given in the following sections. Using the upper bounds, we have

$$\begin{aligned} \Pr[\mathbf{T}_I \in \mathcal{T}_{\text{bad}}] &\leq \frac{q_p}{2^n} + \frac{2q_p\sigma_{e,1}}{2^n(2^n - q_e)} + \frac{3q_p\text{mcoll}(\sigma_e, 2^n)}{2^n} + \frac{2q_p\sigma_{d,1}}{2^n(2^n - \sigma)} \\ &\quad + \frac{\sigma_e^2}{2^n(2^n - q_e)} + \frac{2\sigma\sigma_d}{(2^n - \sigma)^2} + \frac{\sigma}{2^n} + \frac{\sigma_d\text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n} \\ &\leq \frac{q_p + \sigma + 3q_p\text{mcoll}(\sigma_e, 2^n) + \sigma_d\text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n} + \frac{2q_p\sigma + 2\sigma^2}{(2^n - \sigma)^2} . \end{aligned}$$

4.4.1 Upper Bound of $\Pr[\text{bad}_1]$

Recall the definition of **bad₁** below.

$$\text{bad}_1 : \exists \alpha \in [q_p] \text{ s.t. } \hat{K}^{(\alpha)} = K .$$

As $K \xleftarrow{\$} \{0, 1\}^n$, for each $\alpha \in [q_p]$, we have $\Pr[\hat{K}^{(\alpha)} = K] \leq 1/2^n$, and thus

$$\Pr[\text{bad}_1] \leq \frac{q_p}{2^n} .$$

4.4.2 Upper Bound of $\Pr[\text{bad}_2]$

Recall the definition of **bad₂** below.

$$\text{bad}_2 : \exists \alpha \in [q_p], \beta \in [q_e], i \in [\ell_\beta + 1] \text{ s.t. } \hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge (\hat{X}^{(\alpha)} = X_i^{(\beta)} \vee \hat{Y}^{(\alpha)} = Y_i^{(\beta)}) .$$

We upper-bound $\Pr[\text{bad}_2]$ using the following sub-events.

- **bad_{2,1}**: $\exists \alpha \in [q_p], \beta \in [q_e], i \in [a_\beta + 2, \ell_\beta + 1]$ s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{X}^{(\alpha)} = X_i^{(\beta)}$, where $X_i^{(\beta)}$ is revealed via the ciphertext block $C_{i-a_\beta-1}^{(\beta)} (= X_i^{(\beta)} \oplus M_{i-a_\beta-1}^{(\beta)})$.
- **bad_{2,2}**: $\exists \alpha \in [q_p], \beta \in [q_e], i \in [a_\beta + 1, \ell_\beta]$ s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{Y}^{(\alpha)} = Y_i^{(\beta)}$, where $Y_i^{(\beta)}$ is revealed via the ciphertext block $C_{i-a_\beta}^{(\beta)} (= Y_i^{(\beta)} \oplus M_{i-a_\beta}^{(\beta)})$.

- **bad_{2,3}**: $\exists \alpha \in [q_p], \beta \in [q_e]$ s.t. $\hat{K}^{(\alpha)} = K_{\ell_{\beta+1}}^{(\beta)} \wedge \hat{Y}^{(\alpha)} = Y_{\ell_{\beta+1}}^{(\beta)}$, where $\pi(K_{\ell_{\beta+1}}^{(\beta)}) \oplus \lambda(Y_{\ell_{\beta+1}}^{(\beta)})$ is revealed via the tag $T^{(\beta)}$.
- **bad_{2,4}**: $\exists \alpha \in [q_p], \beta \in [q_e], i \in [a_{\beta} + 1]$ s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{X}^{(\alpha)} = X_i^{(\beta)}$, where $X_i^{(\beta)}$ is not revealed before finishing all queries.
- **bad_{2,5}**: $\exists \alpha \in [q_p], \beta \in [q_e], i \in [a_{\beta}]$ s.t. $\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{Y}^{(\alpha)} = Y_i^{(\beta)}$, where $Y_i^{(\beta)}$ is not revealed before finishing all queries.

For the collision $\hat{K}^{(\alpha)} = K_i^{(\beta)}$, as $K \stackrel{\$}{\leftarrow} \{0, 1\}^n$, for each α, β, i , we have $\Pr[\hat{K}^{(\alpha)} = K_i^{(\beta)}] \leq 1/2^n$.

bad_{2,4}: As $X_i^{(\beta)} = Y_{i-1}^{(\beta)} \oplus A_{i-1,1}^{(\beta)}$ and $Y_{i-1}^{(\beta)}$ is chosen uniformly at random from at least $2^n - q$ elements, we have $\Pr[(\hat{K}^{(\alpha)}, \hat{X}^{(\alpha)}) = (K_i^{(\beta)}, X_i^{(\beta)})] \leq 1/(2^n(2^n - q))$ for each α, β, i , and thus $\Pr[\text{bad}_{2,4}] \leq q_p \sigma_{e,1}/(2^n(2^n - q))$

bad_{2,5}: Similarly to **bad_{2,4}**, as $Y_i^{(\beta)}$ is chosen uniformly at random from at least $2^n - q$ elements, we have $\Pr[\text{bad}_{2,5}] \leq q_p \sigma_{e,1}/(2^n(2^n - q))$.

bad_{2,1}: Unlike the former analyses, we cannot use the randomness of $Y_{i-1}^{(\beta)}$ that is revealed via the corresponding ciphertext block. To overcome this issue, we use the number of multi-collision elements with $\hat{X}^{(\alpha)} = X_i^{(\beta)}$. For each $\alpha \in [q_p]$, the number of input blocks $X_i^{(\beta)}$ such that $\hat{X}^{(\alpha)} = X_i^{(\beta)}$ is at most $\text{mcoll}(\sigma_{e,2}, 2^n)$, thus we have $\Pr[\text{bad}_{2,1}] \leq q_p \text{mcoll}(\sigma_{e,2}, 2^n)/2^n$.

bad_{2,2}: Similarly to **bad_{2,1}**, using the number of the multi-collision elements with $\hat{Y}^{(\alpha)} = Y_i^{(\beta)}$. for each $\alpha \in [q_p]$, the number of output blocks $Y_i^{(\beta)}$ such that $\hat{Y}^{(\alpha)} = Y_i^{(\beta)}$ is at most $\text{mcoll}(\sigma_{e,2}, 2^n)$, thus we have $\Pr[\text{bad}_{2,2}] \leq q_p \text{mcoll}(\sigma_{e,2}, 2^n)/2^n$.

bad_{2,3}: Similarly to **bad_{2,1}**, using the number of multi-collision elements for the XOR values $T^{(\beta)} \oplus \lambda(Y_{\ell_{\beta+1}}^{(\beta)})$, we have $\Pr[\text{bad}_{2,3}] \leq q_p \text{mcoll}(q_e, 2^n)/2^n$.

Finally, summing the upper bounds, we have

$$\Pr[\text{bad}_2] \leq \frac{2q_p \sigma_{e,1}}{2^n(2^n - q)} + \frac{3q_p \text{mcoll}(\sigma_{e,2}, 2^n)}{2^n}.$$

4.4.3 Upper Bound of bad₃

Recall the definition of **bad₃** below.

$$\text{bad}_3 : \exists \alpha \in [q_p], \beta \in [q_e + 1, q_e + q_d], i \in [a_{\beta}] \text{ s.t. } \hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge (\hat{X}^{(\alpha)} = X_i^{(\beta)} \vee \hat{Y}^{(\alpha)} = Y_i^{(\beta)}).$$

For the collisions $\hat{X}^{(\alpha)} = X_i^{(\beta)}$ and $\hat{Y}^{(\alpha)} = Y_i^{(\beta)}$, $X_i^{(\beta)}$ and $Y_i^{(\beta)}$ are not revealed before finishing all queries. Thus, the analysis is the same as the analyses of **bad_{2,4}** and **bad_{2,5}**. For each α, β, i , we have $\Pr[\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{X}^{(\alpha)} = X_i^{(\beta)}] \leq 1/(2^n(2^n - \sigma))$, and $\Pr[\hat{K}^{(\alpha)} = K_i^{(\beta)} \wedge \hat{Y}^{(\alpha)} = Y_i^{(\beta)}] \leq 1/(2^n(2^n - \sigma))$. Summing the upper bound $2/(2^n(2^n - \sigma))$ for each α, β, i , we have

$$\Pr[\text{bad}_3] \leq \frac{2q_p \sigma_{d,1}}{2^n(2^n - \sigma)}.$$

4.4.4 Upper Bound of bad₄

Recall the definition of **bad₄** below.

$$\text{bad}_4 : \exists \alpha, \beta \in [q_e], i \in [\ell_{\alpha} + 1], j \in [\ell_{\beta} + 1] \text{ s.t. } (\alpha, i) \neq (\beta, j) \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge (X_i^{(\alpha)} = X_j^{(\beta)} \vee Y_i^{(\alpha)} = Y_j^{(\beta)}).$$

For $\alpha \neq \beta$, $K_1^{(\alpha)} \neq K_1^{(\beta)}$ is satisfied due to $N^{(\alpha)} \neq N^{(\beta)}$. We thus assume that $1 < i$ or $1 < j$ is satisfied.

Regarding the collision $X_i^{(\alpha)} = X_j^{(\beta)}$ ($\Leftrightarrow D_{i-1,1}^{(\alpha)} \oplus Y_{i-1}^{(\alpha)} = D_{j-1,1}^{(\beta)} \oplus Y_{j-1}^{(\beta)}$), for each α, β, i, j , as $Y_{i-1}^{(\alpha)}$ or $Y_{j-1}^{(\beta)}$ is chosen uniformly at random from $\{0, 1\}^n$, we have $\Pr[X_i^{(\alpha)} = X_j^{(\beta)}] \leq 1/2^n$.

Regarding the collision $Y_i^{(\alpha)} = Y_j^{(\beta)}$, for each α, β, i, j , as $Y_i^{(\alpha)}$ or $Y_j^{(\beta)}$ is chosen uniformly at random from $\{0, 1\}^n$, we have $\Pr[Y_i^{(\alpha)} = Y_j^{(\beta)}] \leq 1/2^n$.

The collision $K_i^{(\alpha)} = K_j^{(\beta)}$ equals

$$Y_{i-1}^{*(\alpha)} \oplus Y_{j-1}^{*(\beta)} \oplus \pi(K_{i-1}^{(\alpha)}) \oplus \pi(K_{j-1}^{(\beta)}) = D_{i-1,2}^{(\alpha)} \oplus D_{j-1,2}^{(\beta)},$$

where $K_{i-1}^{(\alpha)}$ resp. $K_{j-1}^{(\beta)}$ is defined using $Y_0^{(\alpha)}$ resp. $Y_0^{(\beta)}$. By $i \neq 1 \vee j \neq 1$, $Y_0^{(\alpha)}$ or $Y_0^{(\beta)}$ is defined independently of $Y_{i-1}^{(\alpha)}$ and $Y_{j-1}^{(\beta)}$, and is chosen uniformly at random from at least $2^n - q_e$ elements in $\{0, 1\}^n$. Using the randomness of $Y_0^{(\alpha)}$ or $Y_0^{(\beta)}$, for each α, β, i, j , we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)}] \leq 1/(2^n - q_e)$.

Summing the upper bound $2/2^n(2^n - q_e)$ for each α, β, i, j , we have

$$\Pr[\text{bad}_4] \leq \binom{\sigma_e}{2} \frac{2}{2^n(2^n - q_e)} \leq \frac{\sigma_e^2}{2^n(2^n - q_e)}.$$

4.4.5 Upper Bound of bad_5

Recall the definition of bad_5 below.

$$\begin{aligned} \text{bad}_5 : & \exists \alpha \in [q_e + q_d], \beta \in [q_e + 1, q_e + q_d], \\ & (i, j) \in ([\ell_\alpha + 1] \times [\ell_\beta + 1]) \setminus ([a_\alpha + 2, \ell_\alpha + 1] \times [a_\beta + 2, \ell_\beta + 1]) \\ & \text{s.t. } \text{pf}(\alpha, i - 1) \neq \text{pf}(\beta, j - 1) \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge (X_i^{(\alpha)} = X_j^{(\beta)} \vee Y_i^{(\alpha)} = Y_j^{(\beta)}). \end{aligned}$$

First, we consider the collisions $K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}$, which equal

$$Y_{i-1}^{*(\alpha)} \oplus Y_{j-1}^{*(\beta)} \oplus \pi(K_{i-1}^{(\alpha)}) \oplus \pi(K_{j-1}^{(\beta)}) = D_{i-1,2}^{(\alpha)} \oplus D_{j-1,2}^{(\beta)} \quad (1)$$

$$Y_{i-1}^{(\alpha)} \oplus Y_{j-1}^{(\beta)} = D_{i-1,1}^{(\alpha)} \oplus D_{j-1,1}^{(\beta)}. \quad (2)$$

If $\text{pf}(\alpha, i - 2) = \text{pf}(\beta, j - 2)$, then $K_{i-1}^{(\alpha)} = K_{j-1}^{(\beta)} \wedge Y_{i-1}^{(\alpha)} = Y_{j-1}^{(\beta)}$ is satisfied, and by $\text{pf}(\alpha, i - 1) \neq \text{pf}(\beta, j - 1)$, $K_i^{(\alpha)} \neq K_j^{(\beta)} \vee X_i^{(\alpha)} \neq X_j^{(\beta)}$ is satisfied. Thus, we assume that $\text{pf}(\alpha, i - 2) \neq \text{pf}(\beta, j - 2)$.

We upper-bound the collision probability using the following sub-events.

- $\text{bad}_{5,1}$: $\exists \alpha \in [q_e + q_d], \beta \in [q_e + 1, q_e + q_d], i \in [a_\alpha + 1], j \in [a_\beta + 1]$ s.t. $K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}$
- $\text{bad}_{5,2}$: $\exists \alpha \in [q_e + q_d], \beta \in [q_e + 1, q_e + q_d], i \in [a_\alpha + 1], j \in [a_\beta + 2, \ell_\beta + 1]$ s.t. $K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}$
- $\text{bad}_{5,3}$: $\exists \alpha \in [q_e + q_d], \beta \in [q_e + 1, q_e + q_d], i \in [a_\alpha + 2, \ell_\alpha + 1], j \in [a_\beta + 1]$ s.t. $K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}$

Note that regarding the collision $X_i^{(\alpha)} = X_j^{(\beta)}$ (Eq. (2)), $Y_{i-1}^{(\alpha)}$ or $Y_{j-1}^{(\beta)}$ is a dummy internal value in LBBB.Hash and thus is chosen uniformly at random from at least $2^n - \sigma$ elements.

Using the randomness of $Y_{i-1}^{(\alpha)}$ or $Y_{j-1}^{(\beta)}$, for each α, β, i, j , we have $\Pr[X_i^{(\alpha)} = X_j^{(\beta)}] \leq 1/(2^n - \sigma)$.

$\Pr[\text{bad}_{5,1}]$ is evaluated. Fix α, β, i, j . We consider the collision $K_i^{(\alpha)} = K_j^{(\beta)}$ (Eq. (1)). By $\text{pf}(\alpha, i-2) \neq \text{pf}(\beta, j-2)$, there exists an output block $Y \in \mathcal{Y}_{i-2}^{(\alpha)} \cup \mathcal{Y}_{j-2}^{(\beta)}$ such that Y is chosen independently of other output blocks in $\mathcal{Y}_{i-2}^{(\alpha)} \cup \mathcal{Y}_{j-2}^{(\beta)}$. As Y is chosen uniformly at random from at least $2^n - \sigma$ elements, we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)}] \leq 1/(2^n - \sigma)$. Summing the upper bound $1/(2^n - \sigma)^2$ for each α, β, i, j , we have $\Pr[\text{bad}_{5,1}] \leq (\sigma_{e,1} + \sigma_{d,1})\sigma_{d,1}/(2^n - \sigma)^2$.

$\Pr[\text{bad}_{5,2}]$ is evaluated. Consider two cases.

- $i = j$: For each $\alpha, \beta, i (= j)$, if $N^{(\alpha)} = N^{(\beta)}$, we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)}] \leq 1/(2^n - \sigma)$ due to $Y_{a_\beta}^{(\beta)}$, which is chosen uniformly at random from at least $2^n - \sigma$ elements and used to define $K_{j-1}^{(\beta)}$ in Eq. (1); if $N^{(\alpha)} \neq N^{(\beta)}$, we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)}] \leq 1/(2^n - q)$ due to $Y_0^{(\alpha)}$, which is chosen uniformly at random from at least $2^n - q$ elements and used to define $K_{i-1}^{(\alpha)}$ in Eq. (1). Thus, for each $\alpha, \beta, i (= j)$, we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}] \leq 1/(2^n - \sigma)^2$.
- $i \neq j$: For each $\alpha, \beta, i \neq j$, we have $\Pr[X_i^{(\alpha)} = X_j^{(\beta)}] \leq 1/2^n$ due to K and the property of π . Thus, for each $\alpha, \beta, i \neq j$, we have $\Pr[K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}] \leq 1/2^n(2^n - \sigma)$.

Summing the upper bounds for each α, β, i, j , we have $\Pr[\text{bad}_{5,2}] \leq (\sigma_{e,1} + \sigma_{d,1})\sigma_{d,2}/(2^n - \sigma)^2$.

$\Pr[\text{bad}_{5,3}]$ is evaluated. The analysis is similar to that of $\Pr[\text{bad}_{5,2}]$. We have $\Pr[\text{bad}_{5,3}] \leq (\sigma_{e,2} + \sigma_{d,2})\sigma_{d,1}/(2^n - \sigma)^2$.

Summing these upper bounds, we have

$$\Pr[\text{bad}_5 \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}] \leq \frac{\sigma\sigma_d}{(2^n - \sigma)^2} .$$

Regarding the collisions $K_i^{(\alpha)} = K_j^{(\beta)} \wedge Y_i^{(\alpha)} = Y_j^{(\beta)}$, the analysis is similar to that of the collisions $K_i^{(\alpha)} = K_j^{(\beta)} \wedge X_i^{(\alpha)} = X_j^{(\beta)}$ (replacing $X_i^{(\alpha)} = X_j^{(\beta)}$ with $Y_i^{(\alpha)} = Y_j^{(\beta)}$). We have

$$\Pr[\text{bad}_5 \wedge K_i^{(\alpha)} = K_j^{(\beta)} \wedge Y_i^{(\alpha)} = Y_j^{(\beta)}] \leq \frac{\sigma\sigma_d}{(2^n - \sigma)^2} .$$

Summing these upper bounds, we have

$$\Pr[\text{bad}_5] \leq \frac{2\sigma\sigma_d}{(2^n - \sigma)^2} .$$

4.4.6 Upper Bound of bad_6

Recall the definition of bad_6 below.

$$\text{bad}_6 : \exists \alpha \in [q_e + q_d], i \in [\ell_\alpha + 1] \text{ s.t. } K_i^{(\alpha)} = K.$$

As K is chosen uniformly at random from $\{0, 1\}^n$, we have

$$\Pr[\text{bad}_6] \leq \frac{\sigma}{2^n} .$$

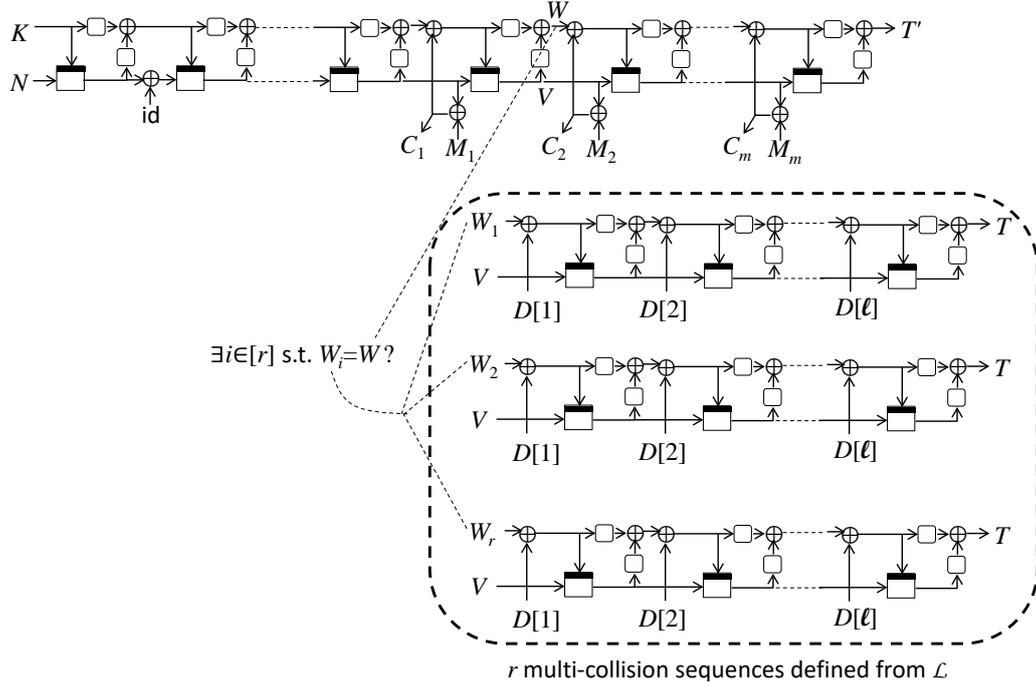


Figure 3: Multi-Collision Sequences.

4.4.7 Upper Bound of bad_7

Recall the definition of bad_7 below.

$$\text{bad}_7 : \exists \alpha \in [q_e + 1, q_e + q_d] \text{ s.t. } T^{(\alpha)} = T'^{(\alpha)}.$$

First, fix α and upper-bound $\Pr[T^{(\alpha)} = T'^{(\alpha)}]$, which is the probability of forging a tag at the α -th online query. Let $\mathcal{L} := \{(K, X, Y) : \mathbf{E}_{\text{Enc}}[K, X] = Y \neq \varepsilon\} \cup \{(\hat{X}^{(\alpha)}, \hat{K}^{(\alpha)}, \hat{Y}^{(\alpha)}) : \alpha \in [q_p]\}$ be input-output triples that might be used as dummy internal values of LBBB.Dec.Main . In this analysis, we take into account whether dummy internal values of LBBB.Dec.Main are defined in \mathcal{L} or not. We thus consider the following sub-events.

- $\text{bad}_{7,1}[\alpha]$: $T^{(\alpha)} = T'^{(\alpha)} \wedge (K_{a_\alpha + m_\alpha + 1}^{(\alpha)}, X_{\ell_\alpha + 1}^{(\alpha)}, Y_{\ell_\alpha + 1}^{(\alpha)}) \notin \mathcal{L}$.
- $\text{bad}_{7,2}[\alpha]$: $T^{(\alpha)} = T'^{(\alpha)} \wedge \exists i^* \in [a_\alpha + 1, \ell_\alpha] \text{ s.t. } (K_{i^*}^{(\alpha)}, X_{i^*}^{(\alpha)}, Y_{i^*}^{(\alpha)}) \notin \mathcal{L}$.
- $\text{bad}_{7,3}[\alpha]$: $T^{(\alpha)} = T'^{(\alpha)} \wedge \forall i \in [a_\alpha + 1, \ell_\alpha + 1] : (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) \in \mathcal{L}$.

$\text{bad}_{7,1}[\alpha]$: As $T^{(\alpha)}$ is defined using a dummy key $K \xleftarrow{\$} \{0, 1\}^n$, we have $\Pr[\text{bad}_{7,1}[\alpha]] \leq 1/2^n$.

$\text{bad}_{7,2}[\alpha]$: We assume that i^* is the maximum index such that $(K_{i^*}^{(\alpha)}, X_{i^*}^{(\alpha)}, Y_{i^*}^{(\alpha)}) \notin \mathcal{L}$. Then, $\text{bad}_{7,2}[\alpha]$ occurs if there exists $(K_{i^*+1}^{(\alpha)}, X_{i^*+1}^{(\alpha)}, Y_{i^*+1}^{(\alpha)}) \in \mathcal{L}$ such that $\pi(K_{i^*}^{(\alpha)}) \oplus \lambda(Y_{i^*}^{(\alpha)}) \oplus C_{i^*}^{(\alpha)} = K_{i^*+1}^{(\alpha)}$ and $Y_{i^*}^{(\alpha)} = X_{i^*+1}^{(\alpha)}$. As $Y_{i^*}^{(\alpha)}$ is chosen uniformly at random from at least $2^n - q_p - \sigma_d$ elements and $K_{i^*}^{(\alpha)}$ is defined using a dummy key K , we have $\Pr[\text{bad}_{7,2}[\alpha]] \leq q_p / (2^n(2^n - q_p - \sigma_d))$.

$\text{bad}_{7,3}[\alpha]$: In this case, all dummy internal values of LBBB.Dec.Main are not new, i.e., these internal values are in \mathcal{L} . To upper-bound $\Pr[\text{bad}_{7,3}]$, we need to take into account the fact that an adversary can obtain n -bit internal values via ciphertext blocks defined by

encryption queries. If there exist r multi-collision sequences of input-output triples in \mathcal{L} that start at a value $V \in \{0, 1\}^n$, end with a value $T \in \{0, 1\}^n$, and have the same data blocks $D[1], \dots, D[\ell] \in \{0, 1\}^n$, where V is the revealed internal value via the ciphertext block, $\Pr[\text{bad}_{7,3}[\alpha]]$ is upper-bounded by $r/2^n$. See also Fig. 3. Multi-collision sequences are defined as follows.

Definition 1. For two input-output triples $(X', K', Y'), (X^*, K^*, Y^*) \in \mathcal{L}$, the relation between these triples is denoted by

- $(X', K', Y') \xrightarrow[0]{D} (X^*, K^*, Y^*)$ if $Y' = X^*$,
- $(X', K', Y') \xrightarrow[1]{D} (X^*, K^*, Y^*)$ if $\eta(Y') = X^*$, and
- $(X', K', Y') \xrightarrow[2]{D} (X^*, K^*, Y^*)$ if $\eta^2(Y') = X^*$,

where $D = \pi(K') \oplus K^* \oplus \lambda(X^*)$. Then, for data blocks $(D[2], \dots, D[\ell])$, an integer $r \geq 2$, $i \in \{1, 2\}$, $V \in \{0, 1\}^n$ and $T \in \{0, 1\}^n$, r -multi-collision sequences of \mathcal{L}

$$(K_{u,1}, X_{u,1}, Y_{u,1}), \dots, (K_{u,\ell}, X_{u,\ell}, Y_{u,\ell}) : u \in [r]$$

have the following relations: for each $u \in [r]$,

$$(K_{u,1}, X_{u,1}, Y_{u,1}) \xrightarrow[0]{D[2]} \dots \xrightarrow[0]{D[\ell-1]} (K_{u,\ell-1}, X_{u,\ell-1}, Y_{u,\ell-1}) \xrightarrow[i]{D[\ell]} (K_{u,\ell}, X_{u,\ell}, Y_{u,\ell}),$$

$V = X_{u,1}$ and $T = \pi(Y_{u,\ell}) \oplus \lambda(K_{u,\ell})$. The set of the multi-collision sequences is denoted by $\mathcal{G}(V, T, (D[2], \dots, D[\ell]), i)$.

The upper bound of the number of multi-collision sequences is given in the following lemma.

Lemma 3. *Let ℓ be a positive integer. Then we have*

$$\begin{aligned} & \text{Exp} \left(\max_{Y, T, D[2], \dots, D[\ell], i} |\mathcal{G}(Y, T, (D[2], \dots, D[\ell]), i)| \right) \\ & \leq \max \left\{ \text{mcoll}(q_p, 2^n - q_p), \frac{q_p \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p} \right\} \\ & \quad + \max \left\{ \text{mcoll}(q_p + q_e, 2^n - q_p), \frac{q_p \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p} \right\} \\ & \quad + (\ell - 2) \cdot \frac{(q_p + \sigma_d) \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p}. \end{aligned}$$

Using the upper bound, denoted by r^* , and the randomness of a dummy key, we have $\Pr[\text{bad}_{7,3}[\alpha]] \leq r^*/2^n$.

Finally, summing the upper bounds for each α , we have

$$\begin{aligned}
\Pr[\text{bad}_7] &\leq \sum_{\alpha=1}^{q_d} \max \{ \Pr[\text{bad}_{7,1}[\alpha]], \Pr[\text{bad}_{7,2}[\alpha]], \Pr[\text{bad}_{7,3}[\alpha]] \} \\
&\leq \max \left\{ \frac{q_d}{2^n}, \frac{q_d q_p}{2^n(2^n - q_p - \sigma_d)} \right\}, \\
&\quad \frac{1}{2^n} \cdot \sum_{\alpha=1}^{q_d} \left(\max \left\{ \text{mcoll}(q_p, 2^n - q_p), \frac{q_p \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p} \right\} \right. \\
&\quad \quad \left. + \max \left\{ \text{mcoll}(q_p + q_e, 2^n - q_p), \frac{q_p \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p} \right\} \right. \\
&\quad \quad \left. + (m_\alpha - 2) \cdot \frac{(q_p + \sigma_e) \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n - q_p} \right) \}
\end{aligned}$$

Assuming $2q_p + \sigma_e + \sigma_d \leq 2^n$, we have

$$\Pr[\text{bad}_7] \leq \frac{\sigma_d \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)}{2^n}.$$

4.5 Analysis for Good Transcripts

Fix $\tau \in \mathcal{T}_{\text{good}}$. For $W \in \{I, R\}$ and a set V , $\mathbb{T}_W \vdash V$ denotes an event that \mathbb{T}_W is compatible with the values in V . Let

$$\begin{aligned}
\tau_0 &:= \left\{ (K, N^{(\alpha)}, Y_0^{(\alpha)}) : \alpha \in [q_e + q_d] \right\}, \\
\tau_{e,1} &:= \left\{ (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e], i \in [a_\alpha] \right\}, \\
\tau_{e,2} &:= \left\{ (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e], i \in [a_\alpha + 1, \ell_\alpha + 1] \right\}, \\
\tau_{d,1} &:= \left\{ (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e + 1, q_e + q_d], i \in [a_\alpha] \right\}, \\
\tau_{d,2} &:= \left\{ (K_i^{(\alpha)}, X_i^{(\alpha)}, Y_i^{(\alpha)}) : \alpha \in [q_e + 1, q_e + q_d], i \in [a_\alpha + 1, \ell_\alpha + 1] \right\}, \text{ and} \\
\tau_p &:= \left\{ (\hat{K}^{(\alpha)}, \hat{X}^{(\alpha)}, \hat{Y}^{(\alpha)}) : \alpha \in [q_p] \right\}.
\end{aligned}$$

Let $\tau_e := \tau_{e,1} \cup \tau_{e,2}$. Then, we have

$$\begin{aligned}
\Pr[\mathbb{T}_W = \tau] &= \Pr[\mathbb{T}_W \vdash \tau_0] \cdot \Pr[\mathbb{T}_W \vdash \tau_e | \mathbb{T}_W \vdash \tau_0] \cdot \Pr[\mathbb{T}_W \vdash \tau_{d,1} | \mathbb{T}_W \vdash \tau_0 \cup \tau_e] \\
&\quad \times \Pr[\mathbb{T}_W \vdash \tau_p \cup \tau_{d,2} | \mathbb{T}_W \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}].
\end{aligned}$$

4.5.1 Evaluating $\Pr[\mathbb{T}_W \vdash \tau_0]$ for $W \in \{I, R\}$

Let N_0 be the number of distinct nonces for all online queries. As $K \xleftarrow{\$} \{0, 1\}^n$ and for each $\alpha \in [q_e + q_d]$ $Y_0^{(\alpha)}$ is chosen uniformly at random from $\{0, 1\}^n \setminus \{Y_0^{(1)}, \dots, Y_0^{(\alpha-1)}\}$ if $\forall \beta \in [\alpha - 1] : N^{(\alpha)} \neq N^{(\beta)}$, we have

$$\Pr[\mathbb{T}_I \vdash \tau_0] = \Pr[\mathbb{T}_R \vdash \tau_0] = \frac{1}{2^n \cdot (2^n)_{N_0}} \text{ and } \frac{\Pr[\mathbb{T}_R \vdash \tau_0]}{\Pr[\mathbb{T}_I \vdash \tau_0]} = 1,$$

where $(2^n)_{N_0}$ is $2^n(2^n - 1) \dots (2^n - N_0 + 1)$ as defined in Subsection 2.1.

4.5.2 Evaluating $\Pr[\mathbf{T}_W \vdash \tau_e | \mathbf{T}_W \vdash \tau_0]$ for $W \in \{I, R\}$

As all input-output triples in τ_e are distinct by $\neg\text{bad}_4$, we have $|\tau_e| = \sigma_e - q_e$. Let $N_e[K'] := |\{(K^*, X^*, Y^*) \in \tau_e : K^* = K'\}|$ be the number of input-output triples in τ_e whose key elements equal K' .

$\Pr[\mathbf{T}_I \vdash \tau_e | \mathbf{T}_I \vdash \tau_0]$ is evaluated. For each $\alpha \in [q_e]$, $i \in [\ell_\alpha + 1]$, $Y_i^{(\alpha)}$ is chosen uniformly at random from $\{0, 1\}^n$, and thus we have $\Pr[\mathbf{T}_I \vdash \tau_e | \mathbf{T}_I \vdash \tau_0] = (1/2^n)^{\sigma_e - q_e}$.

$\Pr[\mathbf{T}_R \vdash \tau_e | \mathbf{T}_R \vdash \tau_0]$ is evaluated. Note that all key elements in τ_e are not K . As $\sum_{K' \in \{0, 1\}^n} N_e[K'] = \sigma_e - q_e$, we have

$$\Pr[\mathbf{T}_R \vdash \tau_e | \mathbf{T}_R \vdash \tau_0] = \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n)^{N_e[K']}} \geq \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n)^{N_e[K']}} = \frac{1}{2^{n(\sigma_e - q_e)}} .$$

Thus, we have

$$\frac{\Pr[\mathbf{T}_R \vdash \tau_e | \mathbf{T}_R \vdash \tau_0]}{\Pr[\mathbf{T}_I \vdash \tau_e | \mathbf{T}_I \vdash \tau_0]} \geq 1 .$$

4.5.3 Evaluating $\Pr[\mathbf{T}_W \vdash \tau_{d,1} | \mathbf{T}_W \vdash \tau_0 \cup \tau_e]$ for $W \in \{I, R\}$

Let $N_{d,1}[K'] := |\{(K^*, X^*, Y^*) \in \tau_{d,1} : K^* = K' \wedge (K^*, X^*, Y^*) \notin \tau_e\}|$ be the number of input-output triples in $\tau_{d,1}$ whose key elements equal K' and that are not in τ_e .

In both real and ideal worlds, for each $\alpha \in [q_d]$, $i \in [a_\alpha - 1]$, $Y_i^{(\alpha)}$ is distinct from other outputs in $\tau_e \cup \tau_{d,1}$ whose key elements equal $K_i^{(\alpha)}$. Thus, for each $W \in \{I, R\}$, we have

$$\Pr[\mathbf{T}_W \vdash \tau_{d,1} | \mathbf{T}_W \vdash \tau_0 \cup \tau_e] = \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n - N_e[K'])^{N_{d,1}[K']}}$$

and

$$\frac{\Pr[\mathbf{T}_R \vdash \tau_{d,1} | \mathbf{T}_R \vdash \tau_0 \cup \tau_e]}{\Pr[\mathbf{T}_I \vdash \tau_{d,1} | \mathbf{T}_I \vdash \tau_0 \cup \tau_e]} = 1 .$$

4.5.4 Evaluating $\Pr[\mathbf{T}_W \vdash \tau_p \cup \tau_{d,2} | \mathbf{T}_W \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}]$ for $W \in \{I, R\}$

Let $N_{p,d,2}[K'] := |\{(K^*, X^*, Y^*) \in \tau_p \cup \tau_{d,2} : K^* = K' \wedge (K^*, X^*, Y^*) \notin \tau_e\}|$ be the number of input-output triples in $\tau_p \cup \tau_{d,2}$ whose key elements equal K' and that are not in τ_e .

In the ideal world, we have

$$\Pr[\mathbf{T}_I \vdash \tau_p \cup \tau_{d,2} | \mathbf{T}_I \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}] = \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n)^{N_{p,d,2}[K']}}$$

In the real world, we have

$$\begin{aligned} \Pr[\mathbf{T}_R \vdash \tau_p \cup \tau_{d,2} | \mathbf{T}_R \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}] &= \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n - N_e[K'] - N_{d,1}[K'])^{N_{p,d,2}[K']}} \\ &\geq \prod_{K' \in \{0, 1\}^n} \frac{1}{(2^n)^{N_{p,d,2}[K']}} . \end{aligned}$$

Thus, we have

$$\frac{\Pr[\mathbf{T}_R \vdash \tau_p \cup \tau_{d,2} | \mathbf{T}_R \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}]}{\Pr[\mathbf{T}_I \vdash \tau_p \cup \tau_{d,2} | \mathbf{T}_I \vdash \tau_0 \cup \tau_e \cup \tau_{d,1}]} \geq 1 .$$

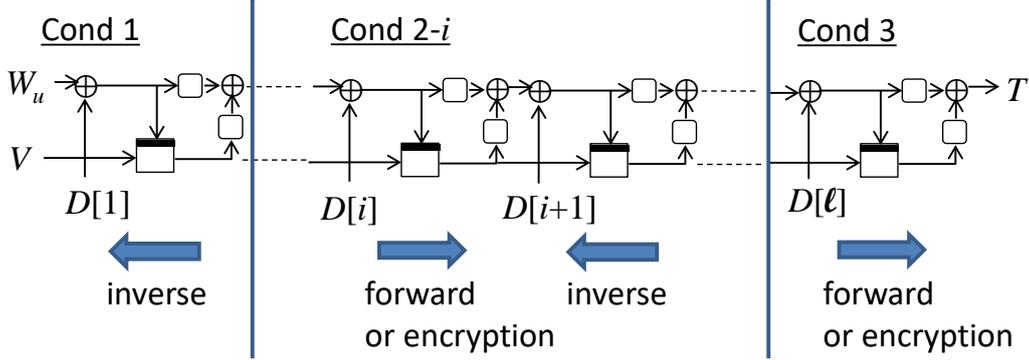


Figure 4: Conditions of Multi-Collision Sequences.

4.5.5 Lower Bound of $\frac{\Pr[\mathbf{T}_R = \tau]}{\Pr[\mathbf{T}_I = \tau]}$

By the above results, we have

$$\frac{\Pr[\mathbf{T}_R = \tau]}{\Pr[\mathbf{T}_I = \tau]} \geq 1 .$$

4.6 Proof of Lemma 3

Each sequence $\{(K_{u,1}, X_{u,1}, Y_{u,1}), \dots, (K_{u,\ell}, X_{u,\ell}, Y_{u,\ell})\} \in \mathcal{G}(V, T, (D[1], \dots, D[\ell]), i)$ in \mathcal{L} satisfies at least one of the following conditions.

- **Cond 1:** $(K_{u,1}, X_{u,1}, Y_{u,1})$ is defined by an inverse offline query.
- **Cond 2- i** where $i \in [1, \ell - 1]$: $(K_{u,i}, X_{u,i}, Y_{u,i})$ is defined by a forward offline query or at an encryption query, and $(K_{u,i+1}, X_{u,i+1}, Y_{u,i+1})$ is defined by an inverse offline query.
- **Cond 3:** $(K_{u,\ell}, X_{u,\ell}, Y_{u,\ell})$ is defined by a forward offline query or at an encryption query.

See also Fig. 4. We upper-bound the number of multi-collision sequences for each condition.

Cond 1. The first blocks $(K_{u,1}, X_{u,1}, Y_{u,1})$ are defined by inverse offline queries and the $X_{u,1}$ values are the same. As each $X_{u,1}$ is chosen uniformly at random from at least $2^n - q_p$ elements in $\{0, 1\}^n$, the number of multi-collision sequences with **Cond 1** is at most $\text{mcoll}(q_p, 2^n - q_p)$.

Cond 3. The last blocks $(K_{u,\ell}, X_{u,\ell}, Y_{u,\ell})$ are defined by forward offline queries or at an encryption query, and the XOR values $\pi(K_{u,\ell}) \oplus \lambda(Y_{u,\ell})$ are the same. As each $Y_{u,\ell}$ is chosen uniformly at random from at least $2^n - q_p$ elements in $\{0, 1\}^n$, the number of multi-collision sequences with **Cond 3** is at most $\text{mcoll}(q_p + q_e, 2^n - q_p)$.

Cond 2- i . The number of multi-collision sequences with this condition is upper-bounded by v : the number of pairs of query-response triple $(K_j, X_j, Y_j), (K'_j, X'_j, Y'_j) \in \mathcal{L} : j \in [v]$ such that $\exists D \in \{0, 1\}^n, s \in [2]$ s.t. $\forall j \in [v] : (K_j, X_j, Y_j) \xrightarrow{D, s} (K'_j, X'_j, Y'_j)$, (K_j, X_j, Y_j) is defined by a forward offline query or at an encryption query, and (K'_j, X'_j, Y'_j) is defined by an inverse offline query. Fix D, s, j and consider the following cases.

- (K_j, X_j, Y_j) is defined after (K'_j, X'_j, Y'_j) is defined. For each (K_j, X_j, Y_j) , the number of inverse offline queries such that the XOR values $K'_j \oplus \lambda(X'_j)$ are the same is at most $\text{mcoll}(q_p, 2^n - q_p)$. Thus, the probability that the output Y_j equals one of the values X'_j is at most $\text{mcoll}(q_p, 2^n - q_p) / (2^n - q_p)$.

- (K_j, X_j, Y_j) is defined before (K'_j, X'_j, Y'_j) is defined. Similarly to the former case, for each (K'_j, X'_j, Y'_j) , the number of multi-collisions for XOR values defined by forward offline queries or at encryption queries is at most $\text{mcoll}(q_p + \sigma_e, 2^n - q_p)$. Thus, the probability that the output X'_j equals one of the Y_j values is at most $\text{mcoll}(q_p + \sigma_e, 2^n - q_p)/(2^n - q_p)$.

Thus, the number of multi-collision sequences with **Cond 2-i** is at most $(q_p + \sigma_e) \cdot \text{mcoll}(q_p + \sigma_e, 2^n - q_p)/(2^n - q_p)$.

Finally, using the upper bounds of the number of offline sequences for each case, we obtain the upper bound in Lemma 3.

5 AES-LBBB: Design and Performance Evaluation

We describe the instantiation of LBBB with AES-128, namely AES-LBBB, followed by its performance evaluation under two different platforms: (i) software implementation on a microcontroller with an AES accelerator [Mic20] and (ii) hardware implementation for ASIC using the NanGate 45-nm standard cell library [Nan].

5.1 Specification of AES-LBBB

To achieve the BBB security using AES-128, we design the concrete instantiation of AES-LBBB. This section discusses the considerations in choosing the π , λ , and η functions for AES-LBBB.

5.1.1 The π and λ functions

π should satisfy the following properties:

- π is linear and has the property that for any $Z \in \{0, 1\}^n$ and $i, j \in (\ell_{\max}]$ such that $i \neq j$, the equation $\pi^i(S) \oplus \pi^j(S) = Z$ offers a unique solution for S .

Notably, the period of π should be at least ℓ_{\max} : the number of maximum blockcipher calls in LBBB. λ should satisfy the same properties except the period:

- λ is linear and has the property that for any $Z \in \{0, 1\}^n$, the equation $S \oplus \lambda(S) = Z$ offers a unique solution for S .

Many symmetric key algorithms, including PMAC1 and its variants, use the functions with the same properties as π . Multiplication over $\text{GF}(2^n)$, e.g., $\pi(S) = S \times 2$, has been used in those conventional works because of the long period $(2^n - 1)$ and efficient hardware implementation. Another advantage of choosing the linear function is that we can unify π and λ by choosing $\pi = \lambda$ because

$$\pi(K_i) \oplus \lambda(Y_i) = \pi(K_i \oplus Y_i). \quad (3)$$

With the above considerations, we choose $\pi = \lambda = \times 2^8$ for AES-LBBB. We choose $\times 2^8$ instead of $\times 2$ because the bitwise operation is more software friendly. Changing the multiplier shortens the maximum number of message blocks to $2^{120} - 1$ but has no practical impact. More specifically, we use the finite field determined by the AES-GCM's irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$ [Nat07] for further backward compatibility.

Other promising choices are bitwise LFSRs, which are given in for example [Sar09]. The LFSR in [Sar09] uses a tower field representation with the following irreducible polynomials: the irreducible polynomial of the sub-field $\text{GF}(2^8)$ is $\alpha^8 + \alpha^7 + \alpha^3 + \alpha^2 + 1$ and the irreducible polynomial of the field $\text{GF}(2^{128})$ over the sub-field is $x^{16} + x^7 + x + \alpha$.

5.1.2 The η function

η should satisfy the same properties as π except for the period:

- η is linear and has the property that for any $Z \in \{0, 1\}^n$ and $i, j \in (2)$ such that $i \neq j$, the equation $\eta^i(S) \oplus \eta^j(S) = Z$ offers a unique solution for S .

Unlike λ , the benefit of choosing the field multiplication is small for η , and it is more meaningful to choose a more efficient function exploiting the shorter period. For AES-LBBB, we choose the following function for η :

$$\eta(S) = (S_2 \oplus S_3) \| S_3 \| S_4 \| \cdots \| S_{16} \| S_1, \text{ where } n = 128 \text{ and } (S_1, S_2, \dots, S_{16}) \stackrel{8}{\leftarrow} S,$$

which is efficient both in software and hardware implementations.

For other choices, bitwise LFSRs, for example [Sar09], are promising choices as well as π and λ .

5.2 Target for comparison

To make a consistent performance comparison, we instantiate the state-of-the-art authenticated encryption with associated data (AEAD) with AES-128 and implement it with the same design policy as AES-LBBB: we set Remus-N2 [IKMP20] as the competitor, which is the blockcipher-based variant with the BBB security from the Romulus/Remus family. More specifically, we implement Remus-N2 instantiated with AES-128, namely Remus-N2-AES⁴.

Table 2 compares the memory sizes of AES-LBBB and Remus-N2-AES in bits. AES-LBBB’s main advantage is its smaller memory footprint: we can implement AES-LBBB with 256 bits of memory, which is smaller than that of Remus-N2-AES by 128 bits. With those memory capacities, we need to overwrite and reuse the memory space for the secret key during the operation and to feed the same secret key for the next operation. To preserve the secret key for the next operations, we need another 128-bit memory, and AES-LBBB and Remus-N2-AES use 384 and 512 bits in total, respectively. Table 2 also shows the memory size needed with TI, which we will discuss later in Section 5.6.

Table 2: Memory/register sizes in bits for AES-LBBB and Remus-N2-AES with and without threshold implementation (TI)

Profile	Target	AES state	AES key	Others	Total	Total w/ key storage
Normal	AES-LBBB	128	128	0	256	384
Normal	Remus-N2-AES	128	128	128	384	512
3-share TI	AES-LBBB	384	384	0	768	896
3-share TI	Remus-N2-AES	384	384	256	1,024	1,152

5.3 Software Implementation

5.3.1 Target Platform

We implement the target algorithms on Microchip’s SAM L11 microcontroller [Mic20], which is an ARM Cortex-M23 microcontroller with a hardware AES accelerator, on an

⁴We replace the Remus-N2-AES’s $\times 2$ operations to $\times 2^8$ in the same way as AES-LBBB (see Section 5.1) for efficient implementation.

evaluation board (SAM L11 Xplained Pro Evaluation Kit [Mic19]). The basic building block is the user-accessible AES function wrapping the hardware AES accelerator:

```
crya_aes_encrypt(key, 4, src, dst), (4)
```

which the microcontroller provides in its mask read only memory (ROM). This line of code implements a single AES-128 call⁵: it encrypts a 16-byte message in `src` with the secret key in `key` and writes the 16-byte ciphertext to `dst`. We can set the same address for `src` and `dst` to overwrite the message with ciphertext in place.

5.3.2 Interface

The implementations are compliant with the SUPERCOP’s interface for AEAD [lab20] and provide the `crypto_aead_encrypt` and `crypto_aead_decrypt` functions. The depth of the function calls is important for ROM and RAM sizes because nested functions reduce the code size at the cost of increasing stack usage. For a rigorous optimization, we limit the depth from the top-level functions (`crypto_aead_encrypt` and `crypto_aead_decrypt`) to one level.

We allocate a space in global memory (cf. in the stack) for storing the sensitive intermediate values assuming that there is a special secure region in memory (e.g., TrustRAM in SAM L11 [Mic18]). Meanwhile, we design the sub-function interfaces so the sensitive intermediate data will not stay in the stack: sensitive data is treated only in the last-level functions that do not use the stack for local variables.

5.4 Software Performance Evaluation

Procedure. We describe the codes in C without an assembly-level optimization and compile them using `gcc` version 6.3.1 on Atmel Studio 7.0⁶. We use the `size` command for evaluating static memory allocation. Meanwhile, we evaluate the stack usage by inspecting the generated object code in between the SUPERCOP’s interface and the AES function (`crya_aes_encrypt`).

We measure the execution time by running the implementations on the target chip (the SAM L11 microcontroller) because the simulator with a cycle counter does not support the hardware AES accelerator. We assert a general-purpose input/output (GPIO) pin during the execution and measure the pulse width using an oscilloscope. Then, we calculate the number of cycles by multiplying the time duration with a clock frequency. The target microcontroller runs at 16 MHz, the maximum frequency of the chip’s internal oscillator⁷. We evaluate the execution time for a particular test vector composed of a 16-byte associated data and a 256-byte message; both AES-LBBB and Remus-N2-AES call AES 19 times for processing the test vector.

Results. Table 3 shows the software performances of our AES-LBBB and Remus-N2-AES implementations. AES-LBBB and Remus-N2-AES use 32 and 48 RAM bytes for storing the sensitive intermediate values as predicted in Table 2. Meanwhile, both implementations use 88 bytes in the stack for storing non-sensitive data, e.g., preserved general-purpose registers, function arguments, and loop counters. Reducing the sensitive data’s memory size by 16 bytes can be considerable because a special memory region for sensitive data is sometimes very limited in size. For example, the SAM L11 microcontroller provides TrustRAM featuring address scrambling and instantaneous wiping [Mic18], which is limited to 256 bytes only. The proposed design achieves a smaller RAM size than AES-GCM,

⁵The second argument of `crya_aes_encrypt` represents the key size in 32-bit word: 4 for AES-128.

⁶We use `-Os` for size optimization, and `-mthumb` for generating the Thumb instructions.

⁷The oscilloscope captures the traces at 1 GSa/s, which is sufficiently larger than the microcontroller’s 16 MHz clock.

as summarized in Table 1, even considering the SAM L11 microcontroller’s AES-GCM acceleration; the microcontroller provides an accelerator for $GF(2^{128})$ multiplication, which does not contribute to reducing the RAM size.

Table 3: Software performance evaluation of AES-LBBB and Remus-N2-AES. Speed is measured with a 16-byte associated data and a 256-byte message.

Target	RAM [byte]	Stack [byte]	ROM [byte]	Speed [cycle]
AES-LBBB	32	88	1,422	32,816
Remus-N2-AES	48	88	1,946	39,664

Table 3 shows the speed in the number of cycles to finish the entire operation for processing the test vector, including initialization, MAC processing, and encryption. It involves 19 AES calls: 1 for initialization, 1 for processing AD, 16 for processing message, and 1 for generating tag. AES-LBBB achieved roughly 32,816 cycles, which is 83% of Remus-N2-AES. Since the number of AES calls is the same between AES-LBBB and Remus-N2-AES, the difference comes from the non-AES operations such as $\times 2^8$, XOR, and the ρ function. A single AES call takes 915 cycles⁸, i.e., 57.2 cycle/byte, and the AES occupies 17,385 ($= 915 \times 19$) cycles in total. In other words, non-AES operations occupy 15,431 cycles or 47.0% of the total execution time in the AES-LBBB implementation. Less frequent calls to non-AES operations are also advantageous to make the code smaller for the limited function depth, and AES-LBBB’s ROM size (1,422 bytes) is smaller than that of Remus-N2-AES by 27%.

Comparison with software implementation of lightweight primitives We set out to discuss the benefit of using an AES coprocessor compared to optimized software implementation of the state-of-the-art lightweight block ciphers. First, we discuss the memory size, which is the AES-LBBB’s primary goal; using a coprocessor provides substantial advantages in memory size over a software implementation of newer lightweight primitives. The ROM needed for using a coprocessor is just a simple interface, which is smaller than a complete block cipher implementation. The coprocessor approach also achieves a smaller RAM because there is no need for intermediate variables and nested function calls.

Second, we discuss the speed. That is not the AES-LBBB’s primary goal, and the other mode/primitive can be better, especially when rich memory is available. AES coprocessors’ performance depends on a particular chip, but chip vendors typically design them for average use cases, and rigorously optimized software implementation can outperform them. Although the speed comes at the cost of memory, the fixslicing Skinny-128-128 implementation achieves 58 cycle/byte [AP21], which is almost the same as the SAM L11’s AES coprocessor with 57 cycle/byte based on the real measurement.

We finally discuss briefly software performance in high-end processors. AES-LBBB enjoys the AES instructions such as AES-NI [Gue10, MNP⁺20]. Meanwhile, ROM and RAM are cheaper in those processors, and trading memory size with speed using a parallel mode (e.g., OCB) could be better for those targets.

5.5 Hardware Implementation

Although the legacy devices with AES coprocessors enjoy the AES-LBBB’s backward compatibility, the newer devices have the opportunity to use upgraded coprocessors to communicate with those legacy devices more efficiently. We set out to discuss the design of such upgraded AES coprocessors.

⁸Based on real measurement of 57.2 μ s for finishing `crya_aes_encrypt(key, 4, src, dst)`

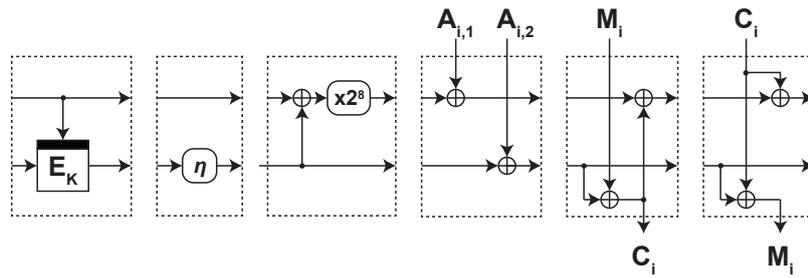


Figure 5: Breakdown of AES-LBBB into several commands

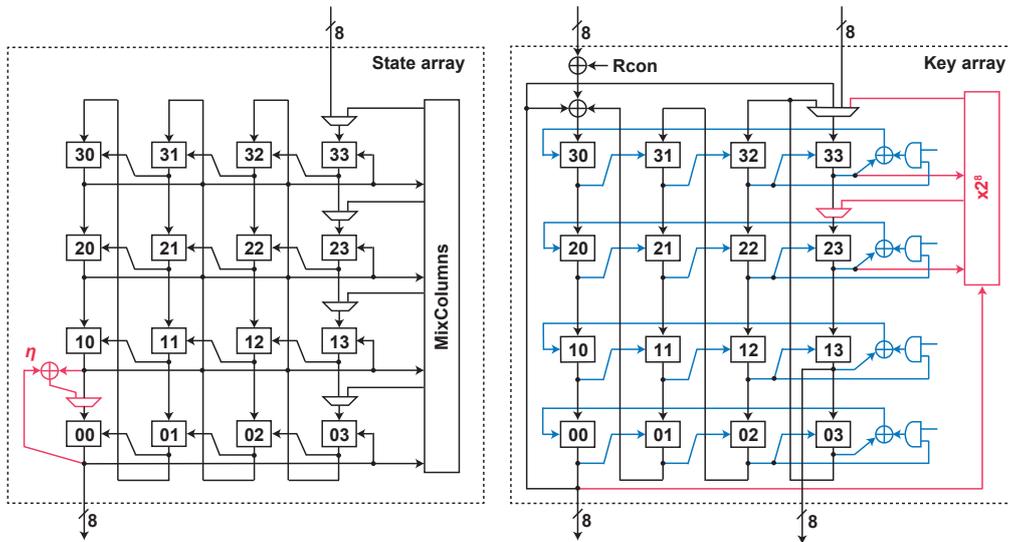


Figure 6: The state and key arrays used in the AES-LBBB implementation. We extended the basic arrays for column-oriented serialization (shown in black) with (i) the inverse key schedule (shown in blue, see Section 5.5.2) and (ii) the linear-update functions for the mode of operation (shown in red, η and $\times 2^8$).

5.5.1 Interface

We design coprocessors that provide a set of commands for processing a block at a time, which is common in the previous works [NS20]. We decompose the target algorithm into the operational units, as shown in Figure 5. We can realize the AES-LBBB’s AD processing, encryption, and decryption by combining those commands. We assume an external main controller that feeds message blocks and dispatches the commands in an appropriate sequence.

5.5.2 AES Implementation

Our design follows the byte-serial architecture [MPL⁺11] commonly used for the conventional compact AES implementations. Figure 6 shows the state and key arrays, flip-flops arranged in the 4×4 array, that efficiently realize the AES’s operations in place. We use a particular variant that makes the column-oriented serialization [Sug20], which respects the AES’ native byte order.

The on-the-fly key schedule is common among compact hardware implementations. As a downside, however, the on-the-fly key schedule overwrites the key register in place, and thus we lose the original AES key after calling an AES encryption. This is a problem for AES-LBBB and Remus-N2-AES that use the AES key for processing the next blocks. A straightforward workaround is to add an extra 128-bit register storing the AES secret key, but it increases the number of registers and devastates the low-memory advantage of our scheme.

Another way is to implement the inverse key schedule that reverts the final-round key to the initial one. Although implementing such an inverse key schedule is simple and efficient for the linear key schedule [NS20, NSS20], which is not the case for the AES’s non-linear key schedule. We efficiently address the problem by integrating the inverse key schedule into the key array as shown in Figure 6. First, we describe the AES key schedule as

$$c_0 \leftarrow f(c_3) \oplus c_0; \quad c_1 \leftarrow c_0 \oplus c_1; \quad c_2 \leftarrow c_1 \oplus c_2; \quad c_3 \leftarrow c_2 \oplus c_3 \quad (5)$$

wherein c_0, c_1, c_2 , and c_3 are 32-bit registers storing each column of an AES key state, and $f(\cdot)$ is the non-linear function composed of `RotWord`, `SubWord`, and `rcon` addition. The idea is to revert Equation 5 with the following procedure:

$$c_3 \leftarrow c_2 \oplus c_3; \quad c_2 \leftarrow c_1 \oplus c_2; \quad c_1 \leftarrow c_0 \oplus c_1; \quad c_0 \leftarrow f(c_3) \oplus c_0. \quad (6)$$

Since the function f is common, we can realize Equation 6 just by adding several XOR gates as shown in Figure 6. The datapath for Equation 6 efficiently fits the key array’s horizontal connection, which was unused in the original column-oriented serialization. The key array finishes Equation 6 in 8 cycles (4 cycles for `SubWord` and another 4 cycles for the XOR between the columns), and the entire inverse key schedule takes 80 cycles. The entire AES operation takes 324 cycles⁹, and the inverse key schedule occupies its 25% fraction.

The arrays also integrates the circuits for performing the $\times 2^8$ and η operations as highlighted with red in Figure 6. With this integration we can perform $\times 2^8$ and η in one cycle, otherwise we need 16 cycles and additional 8-bit register.

5.5.3 Circuit Architecture

Figure 7 shows the datapath architecture of our AES-LBBB implementation. The dashed line indicates the region for AES, composed of the state and key arrays indicated by (C_{ST}) and (C_K) . We use Canright’s design [Can05] for efficiently implementing S-box (C_S) .

We use several XOR and AND gates, in addition to the $\times 2^8$ and η operations integrated into (C_{ST}) and (C_K) , to extend the AES implementation to AES-LBBB. The AND gates regulate the data flow by using the control signals from a state machine and realize the operations for each command shown in Figure 5 in a byte-serial manner. It also supports padding on the incoming message, which is indicated by `0x80` in Figure 7.

Figure 8 shows the datapath diagram of our Remus-N2-AES implementation that uses the same components, namely (C_{ST}) , (C_K) , and (C_S) ¹⁰. To store the Remus-N2-AES’s larger state, we use a 128-bit shift register with embedded $\times 2^8$ operation indicated by (C_{Ext}) .

5.6 Hardware Performance Evaluation

Procedure. We describe the designs at the register-transfer level except the scan flip-flop essential for the state and key arrays [MPL⁺11, NMSS18]: we manually instantiate the

⁹A single AES round takes 23 cycles, and the entire AES without the inverse key schedule takes 244 cycles.

¹⁰We removed the unnecessary η circuit from in the state array in the Remus-N2-AES implementation.

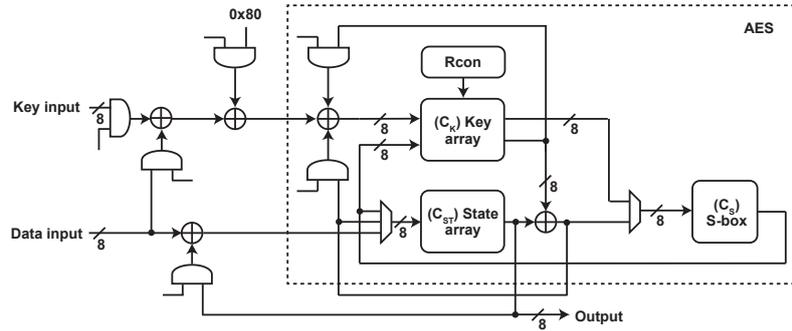


Figure 7: Datapath diagram of the AES-LBBB hardware implementation

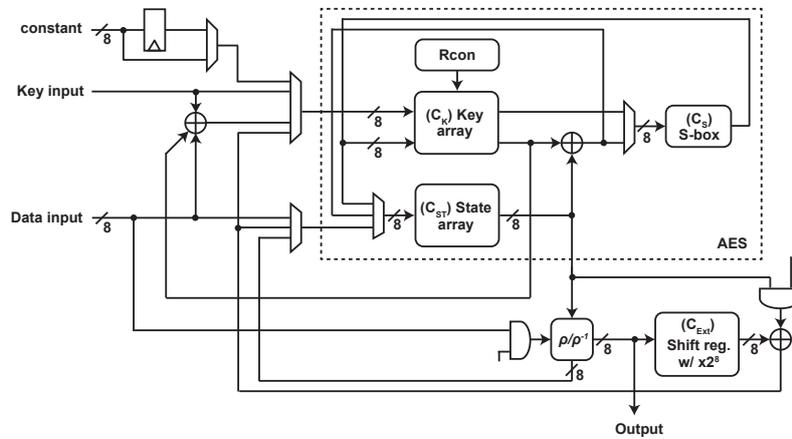


Figure 8: Datapath diagram of the Remus-N2-AES hardware implementation

scan flip-flop standard cells in the code. We synthesize the designs with the NanGate 45-nm standard cell library [Nan] using Synopsys Design Compiler. To obtain the component-wise circuit area, we preserve the module hierarchy at the boundary of the components, i.e., (C_{ST}) , (C_K) , (C_{Ext}) , and (C_S) .

Results. Table 4 shows the post-synthesis circuit area in gate equivalent (GE). Each row corresponds to the implementation of AES-LBBB and Remus-N2-AES. For comparison, the table also shows the performance of the baseline AES implementation composed of the same (C_{ST}) , (C_K) , and (C_S) . The columns represent the total and component-wise performances.

Our AES-LBBB implementation achieved 3,635 GE, which is smaller than that of Remus-N2-AES by 1,200 GE. This advantage mostly comes from the smaller state size: the 128-bit shift register (C_{Ext}) occupies 1,023 GE for Remus-N2. This confirms along with the conventional works that the register dominates the circuit area. The cost for the mode of operation is 271 GE only compared with 3,364 GE for the baseline AES implementation; we can upgrade an AES coprocessor for supporting AES-LBBB at a very small cost.

Secret Key Storage. As discussed in Section 5.2, our AES-LBBB and Remus-N2-AES implementations overwrite the AEAD’s secret key during the operation, in the same way as some previous implementations [GWDE15, IKMP20]. Meanwhile, the other implementations preserve the key so that we can process another message/ciphertext without feeding

Table 4: Post-synthesis circuit area of AES-LBBB, Remus-N2-AES, and the baseline AES implementation shown in gate equivalent (GE).

	Total	(C_{ST}) State array	(C_K) Key array	(C_{Ext}) Shift register	(C_S) S-box
AES-LBBB	3,635	1,373	1,278	—	281
Remus-N2-AES	4,835	1,343	1,309	1,023	272
Baseline AES	3,364	1,343	1,215	—	277

the same key again [NSS20, NS20]. To compare our results with those implementations, we need to add the cost for an additional 128-bit register. With the approximation of 7 GE/bit, a 128-bit register uses 896 GE, and our AES-LBBB uses 4,531 GE.

Threshold implementation. There is a line of research for optimizing the mode of operation for side-channel attack countermeasures [NS20, NSS20] for resource-constrained devices, which have probed the advantage of the low-memory schemes.

The popular countermeasures based on the multi-party computation multiplies the state size by the number of shares, which virtually multiplies the hardware cost. Table 2 also compares the memory sizes of AES-LBBB and Remus-N2-AES with 3-share threshold implementation. In this setting, the AES-LBBB’s advantage becomes even larger because Remus-N2-AES’s extra state should be doubled: AES-LBBB is smaller by 256 bits or 1,792 GE (for 7 GE/bit).

6 Design Extension

Recall the design philosophy of ALE such that the sequential execution of KSF is efficient in hardware implementations and the use of the AES round function enables AES-NI to be used, thus is also efficient in software implementations. More explanations for the sequential execution of KSF are as follows. To use the same key in every block-cipher call while keeping the size of the key state minimal, after computing $KSF(K)$ in some block, we need to compute KSF^{-1} to the value of $KSF(K)$ to reproduce K for the next block, which requires an additional cost for KSF^{-1} . In Sect. 5, we optimized our AES-LBBB implementation by integrating KSF^{-1} into the key array (see Figure 6), but the cost is still non-negligible.

In ALE, the key input to the next block is exactly the output of the key schedule function in the current block, thus no additional circuit is required to bypass multiple rounds. We believe that this idea of ALE deserves further investigation, and we call the scheme that does not require the implementation of KSF^{-1} in this way an “ALE-like mode.” Note that the attractive feature of ALE was obtained by giving up security proofs.

This section discusses suitable primitives for LBBB so that the cost of KSF^{-1} can be reduced. Note that the goal of this section is to make academic progress for achieving an ALE-like mode with provable security, and independent of the goal in the previous sections to provide backward compatibility for AES co-processors. Because we no longer focus on AES, we do not assume the hardware support of cryptographic operations like AES-NI. Hence we aim to propose a scheme that is optimized for hardware implementations.

ALE-like Mode for a Block Cipher with Suitable KSF. Suppose that there is a dedicated block cipher in which KSF satisfies the property required for π . Then, LBBB becomes an ALE-like mode by using KSF as π . The construction achieves the same efficiency as ALE for the key processing part. The construction needs to implement λ , which is an overhead

from ALE, but it enables the construction to achieve almost full-bit security in a provable way.

A lightweight block cipher **KATAN** [CDK09] is an example of existing designs in which the KSF satisfies the property required for π . **KATAN** takes an 80-bit key as input. **KATAN**'s KSF consists of a linear feedback shift register (LFSR). Let K_i be the i -th bit of the user-provided 80-bit key K and let k_i be the i -th bit of the expanded key. K is first set to the 80-bit key state and generates the expanded key bits as follows.

$$k_i = \begin{cases} K_i & \text{for } i = 0, \dots, 79 \\ k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13} & \text{Otherwise} \end{cases}$$

Specifically, 1-clock of KSF is equivalent to the multiplication by 2 with an irreducible polynomial of $x^{80} + x^{61} + x^{50} + x^{13} + 1$. Note that the designers of **KATAN** chose this irreducible polynomial to have the minimal Hamming weight of 5 (there are no primitive polynomials of degree 80 with only 3 monomials). Hence the design is very suitable for LBBB not only for the property for π but also its design policy.

Block Cipher with Light Key Schedule. Suppose that there is a block cipher in which KSF and its inverse are very light or essentially negligible, but the property of π cannot be satisfied. A KSF that only applies a permutation of the key bits is an example. Such a block cipher can be used in LBBB efficiently. We use a multiplication by 2^8 for π . After computing each block, we need to compute KSF^{-1} to reproduce K without using an extra state. The assumption here is that the cost of the inverse KSF is very light or negligible. Thus, the only cost is the multiplication by 2^8 , which is a reasonable trade for obtaining provable security.

Note that there exists several designs that can be used in LBBB for achieving almost 128-bit security. **GIFT-128** [BPP⁺17] is an example, in which the block size is 128 bits, the key size is 128 bits, and its KSF is only the bit permutation of the key bits.

AES Variant for AES-NI. Suppose that we still aim to exploit AES-NI to efficiently compute the round function. Unfortunately AES's KSF does not satisfy the property for π . However, only by replacing AES's KSF, AES-NI can still be used for computing the round function. There are several studies that investigate a new KSF for AES [MHM⁺02, Nik10, KLPS17]. Among them, Khoo et al. [KLPS17] proposed a KSF that only permutes byte positions. This is exactly the case with "block cipher with light key schedule" explained in the previous paragraph. Hence AES with KSF of Khoo et al. [KLPS17] is efficient for this construction both for hardware and software implementations.

7 Conclusion

In this paper, we proposed a new mode, LBBB, and its AES instance, AES-LBBB, which provides backward compatibility with AES coprocessors as well as high security and low memory. The core idea of LBBB is to introduce a feed computation from block cipher's output to the key state via a permutation λ . This enabled us to prove BBB security of LBBB, particularly 121-bit security for AES-LBBB. λ is a software friendly multiplication by 2^8 and the main computational cost for each message block is a single execution of AES, thus AES-LBBB can be implemented fast with the AES coprocessors. The state size is a minimum $2n$ bits, thus AES-LBBB is also low memory in ASIC implementation. We actually implemented AES-LBBB to evaluate its performance for (i) software implementation on a microcontroller with an AES coprocessor and (ii) hardware implementation for ASIC. The results showed that AES-LBBB outperforms the current state-of-the-art **Remus-N2** instantiated with AES-128 implemented with the same policy. We also discussed several

choices of primitives suitable for LBBB, particularly to point out that a block cipher with an LFSR-based KSF is very suitable for LBBB.

We conclude this paper by mentioning several future directions. The first possible direction is to extend the security proof for a more general class of the key updating function π . Ideally, π should be any function including non-linear update so that π can be any KSF of any cipher, which would remove the necessity of implementing KSF^{-1} in LBBB. The second direction is to extend the framework so that a key of a larger size than the block size can be covered. This would enable us to support AES-192 and AES-256 in the framework. The last direction we want to mention is the challenge in the standard model. As long as the state size is $2n$ bits, which is minimal to ensure n -bit security, the ideal-cipher model is required. From a different viewpoint, how small we can go with a standard model is an interesting research direction.

References

- [AP21] Alexandre Adomnicai and Thomas Peyrin. Fixslicing aes-like ciphers new bitsliced AES speed records on arm-cortex M and RISC-V. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):402–425, 2021.
- [Aut19] Autosar. Specification of secure hardware extensions. Available at https://www.autosar.org/fileadmin/user_upload/standards/foundation/19-11/AUTOSAR_TR_SecureHardwareExtensions.pdf, 2019.
- [BCD⁺19] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. PHOTON-Beetle Authenticated Encryption and Hash Family. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [BJK⁺16] 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 *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.
- [BJK⁺19] 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. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [BK09] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, 2009.
- [BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.
- [BMR⁺14] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In *FSE 2013*, volume 8424 of *LNCS*, pages 447–466. Springer, 2014.
- [BPP⁺17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, 2017.

- [Can05] David Canright. A Very Compact S-Box for AES. In *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005.
- [CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.
- [CIMN17] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-Based Authenticated Encryption: How Small Can We Go? In *CHES 2017*, volume 10529 of *LNCS*, pages 277–298. Springer, 2017.
- [CJN20] Bishwajit Chakraborty, Ashwin Jha, and Mridul Nandi. On the Security of Sponge-type Authenticated Encryption Modes. *IACR Trans. Symmetric Cryptol.*, 2020(2):93–119, 2020.
- [CN19] Bishwajit Chakraborty and Mridul Nandi. mixFeed. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [DEM⁺19] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. ISAP. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [DHP⁺19] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Kee. Xoodoo, a lightweight cryptographic scheme. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [EHR19] Philippe Elbaz-Vincent, Cyril Hugouenq, and Sébastien Riou. SPAE a mode of operation for AES on low-cost hardware. *IACR Cryptol. ePrint Arch.*, 2019:1007, 2019.
- [GJN19] Shay Gueron, Ashwin Jha, and Mridul Nandi. COMET: COUNTER Mode Encryption with authentication Tag. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [Gue10] Shay Gueron. White paper: Intel Advanced Encryption Standard (AES) New Instructions Set Revision 3.0. Available at <https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf>, 2010.
- [GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up! - Made-to-Measure Hardware Implementations of ASCON. In *DSD 2015*, pages 645–652. IEEE Computer Society, 2015.
- [IKMP20] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Trans. Symmetric Cryptol.*, 2020(1):43–120, 2020.
- [JN16] Jérémy Jean and Ivica Nikolic. Efficient Design Strategies Based on the AES Round Function. In *FSE 2016*, volume 9783 of *LNCS*, pages 334–353. Springer, 2016.
- [KLPS17] Khoongming Khoo, Eugene Lee, Thomas Peyrin, and Siang Meng Sim. Human-readable Proof of the Related-Key Security of AES-128. *IACR Trans. Symmetric Cryptol.*, 2017(2):59–83, 2017.

- [KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011.
- [KR19] Dmitry Khovratovich and Christian Rechberger. The LOCAL Attack: Cryptanalysis of the Authenticated Encryption Scheme ALE. In *SAC 2013*, volume 8282 of *LNCS*, pages 174–184. Springer, 2019.
- [lab20] The VAMPIRE lab. SUPERCOP. Available at <https://bench.cr.yp.to/supercop.html>, 2020.
- [MHM⁺02] Lauren May, Matthew Henricksen, William Millan, Gary Carter, and Ed Dawson. Strengthening the Key Schedule of the AES. In *ACISP 2002*, volume 2384 of *LNCS*, pages 226–240. Springer, 2002.
- [Mic18] Microchip. AN5365: SAM L11 security reference guide. Available at <http://ww1.microchip.com/downloads/en/AppNotes/SAM-L11-Security-ReferenceGuide-AN-DS70005365A.pdf>, 2018.
- [Mic19] Microchip. SAM L10/L11 Xplained Pro user guide (DS70005359C). Available at <http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-L10-L11-Xplained-Pro-User-Guide-70005359C.pdf>, 2019.
- [Mic20] Microchip. Sam 110/111 family datasheet: Ultra low-power, 32-bit cortex-m23 mcus with trustzone, crypto, and enhanced ptc (ds60001513f). Available at <http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-L10L11-Family-DataSheet-DS60001513F.pdf>, 2020.
- [MM13] Mitsuru Matsui and Yumiko Murakami. Minimalism of Software Implementation - Extensive Performance Analysis of Symmetric Primitives on the RL78 Microcontroller. In *FSE 2013*, volume 8424 of *LNCS*, pages 393–409. Springer, 2013.
- [MNP⁺20] Ben Marshall, G. Richard Newell, Dan Page, Markku-Juhani O. Saarinen, and Claire Wolf. The design of scalar AES Instruction Set Extensions for RISC-V. *IACR Cryptol. ePrint Arch.*, 2020:930, 2020.
- [MPL⁺11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.
- [Nan] NanGate. NanGate FreePDK45 open cell library. <http://www.nangate.com>.
- [Nat01] National Institute of Standards and Technology (NIST). Announcing the advanced encryption standard (aes). FIPS PUB 197, 2001. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.
- [Nat07] National Institute of Standards and Technology (NIST). Recommendation for block cipher modes of operation: Galois/Counter mode (gcm) and gmac. NIST Special Publication 800-38D, 2007.
- [Nat18] National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the lightweight cryptography standardization process. Available at <https://csrc.nist.gov/Projects/lightweight-cryptography>, 2018.

- [Nik10] Ivica Nikolić. Tweaking AES. In *Selected Areas in Cryptography, SAC 2010*, volume 6544 of *LNCS*, pages 198–210. Springer, 2010.
- [NMS⁺19] Yusuke Naito, Mitsuru Matsui, Yasuyuki Sakai, Daisuke Suzuki, Kazuo Sakiyama, and Takeshi Sugawara. SAEAES. Submitted to NIST Lightweight Cryptography Standardization Process, 2019.
- [NMSS18] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A Lightweight Blockcipher-Based AEAD Mode of Operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.
- [NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering Generic Composition. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 257–274. Springer, 2014.
- [NS20] Yusuke Naito and Takeshi Sugawara. Lightweight Authenticated Encryption Mode of Operation for Tweakable Block Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):66–94, 2020.
- [NSS20] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, volume 12106 of *LNCS*, pages 705–735. Springer, 2020.
- [Pat08] Jacques Patarin. The "Coefficients H" Technique. In *SAC 2008*, volume 5381 of *LNCS*, pages 328–345. Springer, 2008.
- [Rog04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, 2004.
- [Sar09] Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IACR Cryptol. ePrint Arch.*, 2009:217, 2009.
- [Sug20] Takeshi Sugawara. Hardware Performance Evaluation of Authenticated Encryption SAEAES with Threshold Implementation. *Cryptography*, 4(3):23, Aug 2020.
- [USS⁺20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting Leakage Resilient Authenticated Encryption to Microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.
- [WP14] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. In *SAC 2013*, volume 8282 of *LNCS*, pages 185–201. Springer, 2014.
- [WWH⁺13] Shengbao Wu, Hongjun Wu, Tao Huang, Mingsheng Wang, and Wenling Wu. Leaked-State-Forgery Attack against the Authenticated Encryption Algorithm ALE. In *ASIACRYPT 2013*, volume 8269 of *LNCS*, pages 377–404. Springer, 2013.