tHyENA: Making HyENA Even Smaller

Avik Chakraborti[1], Nilanjan Datta[2], Ashwin Jha[3], Cuauhtemoc Mancillas-López[4], and Mridul Nandi[5]

[1] University of Exeter, UK
avikchkrbrti@gmail.com
[2] Institute for Advancing Intelligence, TCG CREST, Kolkata, India
nilanjan.datta@tcgcrest.org
[3] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
ashwin.jha@cispa.de
[4] Computer Science Department, CINVESTAV-IPN, Mexico
cuauhtemoc.mancillas@cinvestav.mx
[5] Indian Statistical Institute, Kolkata, India
mridul.nandi@gmail.com

Abstract. This paper proposes a lightweight short-tweak tweakable block-cipher (tBC) based authenticated encryption (AE) scheme tHyENA, a tweakable variant of the high profile NIST LWC competition submission HyENA. tHyENA is structurally similar to HyENA, however, proper usage of short-tweaks for the purpose of domain separation, makes the design much simpler compact. We know that HyENA already achieves a very small hardware footprint, and tHyENA further optimizes it. To realize our claim, we provide NIST API compliant hardware implementation details and benchmark for tHyENA against HyENA and several other well-known sequential feedback-based designs. The implementation results depict that when instantiated with the tBC TweGIFT, tHyENA achieves an extremely low hardware footprint - consuming only around 680 LUTs and 260 slices while maintaining the full rate and the almost birthday bound security. To the best of our knowledge, this figure is significantly better than all the known implementation results of other lightweight ciphers with sequential structures.

Keywords: Authenticated Encryption, lightweight, tBC, HyENA, Feedback based AE, TweGIFT

1 Introduction

In the last few years, lightweight cryptography has seen a growing popularity due to increasing security demands for lightweight IoT applications such as sensor networks, healthcare applications, distributed control systems, cyber-physical systems, etc., where highly resource-constrained devices communicate and need to be operated with the low hardware area, low power or low energy. Lightweight cryptography is about developing cryptographic solutions for these resource-constrained environments and this research direction has been triggered by the

ongoing NIST Lightweight Standardization Competition (LWC) [20] followed by CAESAR [12]. As a result, in recent years, the cryptographic community has witnessed a rise in various lightweight authenticated encryption proposals.

One popular approach to design lightweight AE schemes is to use a sequential structure as it consumes lesser hardware footprint. Blockcipher (BC) based sequential AE schemes typically use a feedback function on the previous blockcipher output, an auxiliary secret state, and the current input (message or associated data) block. It outputs the next blockcipher feedback, updates the auxiliary secret state and the current output block (in case of message blocks). Thus, blockcipher-based sequential AE schemes can be well described by the underlying blockcipher, the auxiliary secret state, and the feedback function. Consequently, the efficiency and the hardware footprint of the AE scheme also largely depend on these three components (the underlying blockcipher, the feedback function, and the auxiliary secret state). In the following context, we assume that we have an ultra-lightweight and efficient primitive to instantiate the AE scheme. The efficiency of a construction is primarily dependent upon the *rate*, the number of data blocks processed per primitive call. It is well known that a trivial upper bound on the rate is 1. In this paper, we concentrate only on rate-1 authenticated encryptions with a small hardware footprint such that we can achieve a lightweight construction along with a high throughput.

1.1 Rate-1 Feedback Based Authenticated Encryption

Zhang et al. in [26], proposed a plaintext feedback-based mode iFEED that has rate 1. However, it requires a large state size of $(3n + k)$ bits, where n is the underlying blockcipher's state size, and k is the key size. CPFB by Montes et al. [19] is a notable scheme that reduces the state size to $(2n + k)$ bits, at the cost of a reduction in the rate to 3/4. In CHES 2017, Chakraborti et al. [7] proposed COFB the first feedback-based AE scheme that achieves rate-1 with an impressive state size of just $1.5n + k$ bits. The main feature of COFB is a novel feedback function, called *combined feedback*.

In [8, 10], Chakraborti et al. studied a generalized feedback-based rate-1 AE scheme and showed that it is a necessity for any rate-1 feedback-based AE mode to have an auxiliary state of $n/2$-bit to achieve security up to $2^{n/2}$ queries, depicting the optimality of COFB in the auxiliary state size. However, they have observed that the use of the combined feedback requires $2.5n$-bit XORs, which could be improved further using a *hybrid feedback* HyFB, which results in a hybrid of plaintext feedback and ciphertext feedback. Based on the HyFB feedback function, Chakraborti et al. in [5] proposed a rate-1 AE mode called HyENA, that uses an $n/2$-bit auxiliary secret state, but significantly reduces the XOR count from $2.5n$-bit to $1.5n$-bit. This seems to achieve the smallest footprint for any rate 1 blockcipher based authenticated cipher owing to the fact that it requires the optimal auxiliary state, optimal linear operations. The optimality of the linear operations can be conjectured from the facts that (i) to implement a feedback function that takes $2n$-bit input and produces n-bit output, one needs

at least n-bit binary operation, and (ii) XOR is one of the most simple operations, and hence n-bit XORs seem to achieve the trivial lower bound of any feedback function.

1.2 Our Contribution

In this paper, we primarily focus on highly optimizing HyENA, i.e. significantly minimizing the hardware footprint. To achieve the goal, we use a short tweak tweakable blockcipher based on the well-known blockcipher GIFT [1] to propose a tweakable variant of HyENA, dubbed as tHyENA. This new variant is structurally much simpler and removes the redundant operations to reduce the hardware footprint. tHyENA inherits all the desirable properties of HyENA: (i) single-pass (one primitive call per data block), (ii) inverse-free (no need for blockcipher decryption), (iii) extremely low state size and XOR count. Precisely, the use of tweaks for the purpose of domain separation makes the construction simpler and removes all the constant field multiplications making the mode even lighter. We instantiate tHyENA with an ultra-lightweight short-tweak tweakable blockcipher TweGIFT (designed over the blockcipher GIFT [1]). We also provide concrete hardware implementation details of tHyENA. The hardware results depicts that tHyENA with TweGIFT consumes the least hardware area among all the feedback type (tweakable) block cipher based designs.

1.3 tHyENA in DSCI Light-weight Competition

In 2020, National CoE, the joint initiative of the Data security council of India and the Ministry of Electronics and IT (MeitY), announced a lightweight cryptography competition named "Lightweight Cipher Design Challenge 2020" [21]. One of the primary objectives of the challenge is to design new lightweight authenticated ciphers, and the best designs will be considered for developing the prototype for ready industry implementation. The algorithm tHyENA has been nominated as one of the top three candidates in the challenge and has been selected for the final round. Interestingly, the construction achieves the lowest hardware footprint, making it to be the most light-weight design, in terms of area, in the competition.

2 Preliminaries

For $n \in \mathbb{N}$, we write $\{0,1\}^*$ and $\{0,1\}^n$ to denote the set of all binary strings (including the empty string λ), and the set of all n-bit strings, respectively. Throughout we fix even integer n as the block size in bits, and often refer to n-bit strings as *blocks*. For all $X \in \{0,1\}^*$, $|X|$, referred as the length of X, denotes the number of bits in X. For any $X \in \{0,1\}^n$, X_L and X_R denote the most and least significant $n/2$ bits of X, respectively. For all practical purposes, we use the little endian format for representing binary strings, i.e., the least significant bit is the right most bit. We use the notation $\oplus$ to denote binary addition. For

two strings $X, Y \in \{0,1\}^*$, $X\|Y$ denotes the concatenation of X and Y. We use the notation $(X_{\ell-1}, \ldots, X_0) \xleftarrow{n} X$ to denote parsing of the string X into ℓ blocks such that for $0 \leq i \leq \ell-2$, $|X_i| = n$ and $1 \leq |X_{\ell-1}| \leq n$. For any predicate $\mathcal{E}$, the expression $\mathcal{E}?a : b$ evaluates to a if $\mathcal{E}$ is true, and b otherwise. For any binary string X with $|X| \leq n$, we define the padding function Pad as

$$\mathsf{Pad}(X) = \begin{cases} X & \text{if } |X| \bmod n = 0 \\ 0^{n-|X|-1}\|1\|X & \text{otherwise.} \end{cases}$$

For any binary string X, the truncate function $\mathsf{Trunc}_i(X)$ returns the least significant i bits of X.

The set $\{0,1\}^{n/2}$ can be viewed as the finite field $\mathbb{F}_{2^{n/2}}$ consisting of $2^{n/2}$ elements. We interchangeably think of an element $A \in \mathbb{F}_{2^{n/2}}$ in any of the following ways: (i) as an $n/2$-bit string $a_{\frac{n}{2}-1} \ldots a_1 a_0 \in \{0,1\}^{n/2}$; (ii) as a polynomial $A(x) = a_{\frac{n}{2}-1}x^{n/2-1} + a_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + \cdots + a_1 x + a_0$ over the field $\mathbb{F}_2$; (iii) a non-negative integer $a < 2^{n/2}$; (iv) an abstract element in the field. Addition in $\mathbb{F}_{2^{n/2}}$ is just bitwise XOR of two $n/2$-bit strings, and hence denoted by $\oplus$. $P(x)$ denotes the primitive polynomial used to represent the field $\mathbb{F}_{2^{n/2}}$, and α denotes the primitive element in this representation. The multiplication of $A, B \in \mathbb{F}_{2^{n/2}}$ is defined as $A \odot B := A(x) \cdot B(x) \pmod{P(x)}$, i.e. polynomial multiplication modulo $P(x)$ in $\mathbb{F}_2$.

For a finite set $\mathcal{X}$, $\mathsf{X} \leftarrow \mathcal{X}$ denotes the uniform at random sampling of X from $\mathcal{X}$. Let $\gamma = (\gamma[1], \ldots, \gamma[s])$ be a tuple of equal-length strings. We define $\mathsf{mColl}(\gamma) = m$ if there exist distinct $i_1, \ldots, i_m \in [1..s]$ such that $\gamma[i_1] = \cdots = \gamma[i_m]$ and m is the maximum of such integer. We say that $\{i_1, \ldots, i_m\}$ is an m-multi-collision set for γ.

Tweakable Blockcipher: For $n, \tau, \kappa \in \mathbb{N}$, E-$n/\kappa/\tau$ denotes a tweakable blockcipher family E, parameterized by the block length n, key length κ, and tweak length τ. For $K \in \{0,1\}^\kappa$, $T \in \{0,1\}^\tau$, and $M \in \{0,1\}^n$, we use $E_K^T(M) := E(K,T,M)$ to denote invocation of the encryption function of E on key K, tweak T, and input M.

2.1 Authenticated Encryption

An authenticated encryption (AE) is a symmetric-key primitive that provides both data confidentiality (or privacy), and authenticity of the input plaintext. Often, practical scenarios additionally require authenticity for some associated data. In this case, we extend the ambit of AE to AE with associated data functionality or AEAD, which guarantees privacy for the input message and authenticity for the input message and associated data.

Formally, an AEAD scheme AE is a tuple of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ defined over the key space $\mathcal{K}$, nonce space $\mathcal{N}$, associated data space $\mathcal{A}$, message and ciphertext space $\mathcal{M}$, and tag space $\mathcal{T}$, where:

$$\mathsf{AE.Enc} : \mathcal{K} \times \mathcal{A} \times \mathcal{N} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T} \qquad \mathsf{AE.Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\},$$

and $\perp$ denotes the error symbol indicating authentication failure.

The encryption function $\mathsf{AE.Enc}$ instantiated with key $K \in \mathcal{K}$, takes a nonce $N \in \mathcal{N}$ (which is usually a unique value for each invocation), an associated data $A \in \mathcal{A}$, and a plaintext $M \in \mathcal{M}$ as input, and outputs a tagged-ciphertext (C, T) where $|C| = |M|$. The corresponding decryption function $\mathsf{AE.Dec}$ instantiated with key K, takes $(N', A', C', T') \in \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T}$, and returns a decrypted plaintext M' when (N, A, C, T) authenticates successfully, and it returns the error symbol $\perp$ otherwise. For all key $K \in \mathcal{K}$, we write $\mathsf{AE.Enc}_K(\cdot,\cdot) := \mathsf{AE.Enc}(K,\cdot,\cdot)$ and $\mathsf{AE.Dec}_K(\cdot,\cdot,\cdot) := \mathsf{AE.Enc}(K,\cdot,\cdot,\cdot)$. For correctness in decryption, it is required that $\mathsf{AE.Dec}(K, N, A, \mathsf{AE.Enc}(K, N, A, M)) = M$ for all $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$.

In addition to the block size n, we fix positive even integers κ and η to denote the *key size* and*nonce size*, respectively, in bits. Throughout this document, we fix $n = 128$, $\kappa = 128$, $\eta = 96$, and tag size $= n$.

2.2 Security Definitions

Adversary: A (q, t)-adversary $\mathcal{A}$ is an interactive algorithm with access to an oracle, that runs in time at most t, and makes at most q oracle queries. By convention, $t = \infty$ denotes computationally unbounded (information-theoretic) and deterministic adversaries. Throughout, we make the plausible assumption that the adversary is *non-trivial*, i.e., it never makes a duplicate query. Whenever, the adversarial queries are allowed to be of arbitrary length, we parametrize the adversary with additional parameters. For example, an adversary that makes queries of length at most ℓ blocks, and total length of all queries at most σ blocks is referred as (q, ℓ, σ, t)-adversary. We write $\mathcal{A}^{\mathcal{O}} \Rightarrow x$ to denote the compound operation: "adversary $\mathcal{A}$ outputs x after interacting with oracle $\mathcal{O}$".

Tweakable Blockcipher Security: The security of any tweakable blockcipher family is formalized in terms of the notion of *tweakable pseudorandom permutation*. Formally, the tweakable pseudorandom permutation or *TPRP advantage* of any adversary $\mathcal{A}$ against tweakable blockcipher E is defined as

$$\mathbf{Adv}^{\mathsf{tprp}}_{E}(\mathcal{A}) := \left|\Pr[\mathcal{A}^{E_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\Pi} \Rightarrow 1]\right|,$$

where Π is a uniform at random tweakable permutation sampled from the set of all tweakable permutations over $\{0,1\}^n$ with tweak space $\{0,1\}^\tau$. For $\epsilon \geq 0$, the blockcipher E is called a (q, t, ϵ)-TPRP if

$$\mathbf{Adv}^{\mathsf{tprp}}_{E}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{prp}}_{E}(\mathcal{A}) \leq \epsilon,$$

where the maximum is taken over all (q, t)-adversary.

AEAD Security: The security of any nonce-based AEAD scheme can be modeled in terms of the NAEAD notion,. In this model, the adversary is nonce-respecting, i.e., assuming single-key setting, no pair of distinct encryption queries

share the same public nonce value. Formally, the NAEAD advantage of any adversary $\mathcal{A}$ against AEAD scheme AE is defined as

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{AE}}(\mathcal{A}) := \left|\Pr[\mathcal{A}^{\mathsf{AE.Enc}_K,\mathsf{AE.Dec}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\$,\perp} \Rightarrow 1]\right|,$$

where \$ returns an independent and uniform at random string of length $\tau + |M|$ for each queried message M. Note that, we overload the $\perp$ notation to denote the "*always fail*" oracle, which returns $\perp$ in all cases, except when $\mathcal{A}$ makes a query (N, A, C, T), such that there exists an earlier query (N, A, M) to \$ and (C, T) is the corresponding response, in which case the always fail oracle returns M. For $\epsilon \geq 0$, the AEAD AE is called a $(q_e, q_v, \ell_e, \ell_v, \sigma_e, \sigma_v, t, \epsilon)$-NAEAD if

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{AE}}(q_e, q_v, \ell_e, \ell_v, \sigma_e, \sigma_v, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{naead}}_{\mathsf{AE}}(\mathcal{A}) \leq \epsilon,$$

where the maximum is taken over all $(q_e, q_v, \ell_e, \ell_v, \sigma_e, \sigma_v, t)$-adversary, i.e., all adversary $\mathcal{A}$ such that

- the number of encryption queries is bounded by q_e; each encryption query length is at most ℓ_e blocks; and the total length across all encryption queries is at most σ_e blocks.
- the number of decryption queries is bounded by q_v; each decryption query length is at most ℓ_v blocks; and the total length across all decryption queries is at most σ_v blocks.

In addition, we let $q = q_e + q_v$, $\ell = \ell_e + \ell_v$ and $\sigma = \sigma_e + \sigma_v$.

Note on AEAD Security Conventions: It is worth noting here that the NAEAD security notion subsumes [14, 24] the conventional security notions such as privacy and integrity, and provides a combined and uniform security argument for the concerned AEAD scheme. Although our security analysis will follow the NAEAD notion, we briefly define the conventional notions as we present our security claims in terms of the conventional notions.

Privacy Security: We define the *privacy advantage* of any adversary $\mathcal{A}$ against AEAD scheme AE as

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{AE}}(\mathcal{A}) := \left|\Pr[\mathcal{A}^{\mathsf{AE.Enc}_K} = 1] - \Pr[\mathcal{A}^{\$} = 1]\right|.$$

Integrity Security: We say that any adversary $\mathcal{A}$ *forges* an AEAD scheme AE, if $\mathcal{A}$ is able to compute a tuple (N, A, C, T) satisfying $\mathsf{AE.Dec}_K(N, A, C, T) \neq \perp$, without querying (N, A, M) for some M to $\mathsf{AE.Enc}_K$ and receiving (C, T), i.e., (N, A, C, T) is a non-trivial forgery. The *forging advantage* for $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AE}}(\mathcal{A}) := \Pr[\mathcal{A}^{\mathsf{AE.Enc}_K,\mathsf{AE.Dec}_K} \text{ forges}].$$

3 tHyENA Authenticated Encryption Mode

The tHyENA authenticated encryption mode receives an encryption key $K \in \{0,1\}^\kappa$, a nonce $N \in \{0,1\}^r$, an associated data $A \in \{0,1\}^*$, and a message

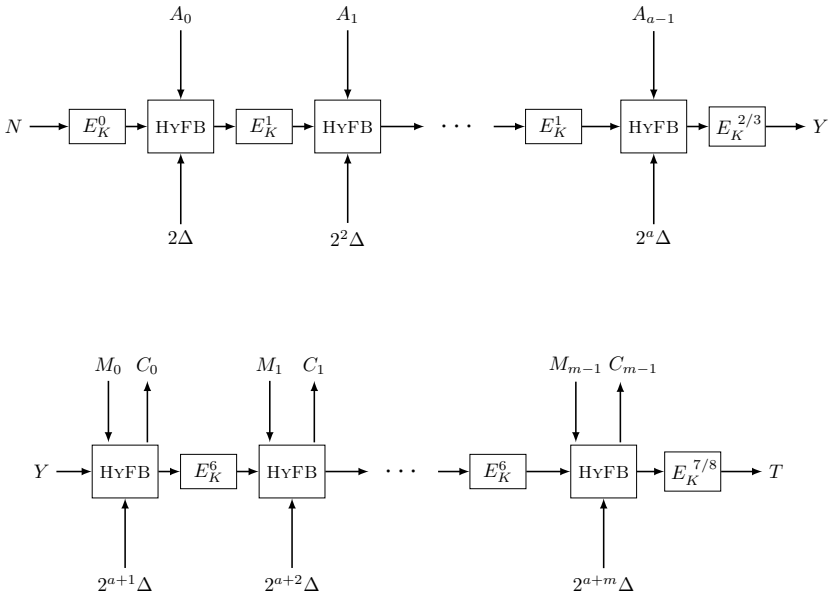


Fig. 1: tHyENA authenticated encryption mode for a block associated data and m block message.

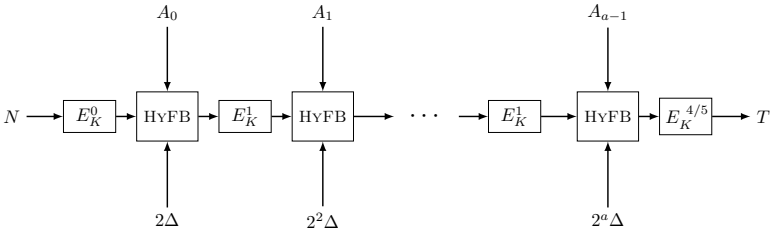


Fig. 2: tHyENA authenticated encryption mode for a block associated data and empty message.

$M \in \{0,1\}^*$ as inputs, and returns a ciphertext $C \in \{0,1\}^{|M|}$ and a tag $T \in \{0,1\}^n$.

The decryption algorithm receives a key $K \in \{0,1\}^\kappa$, an associated data $A \in \{0,1\}^*$, a nonce $N \in \{0,1\}^r$, a ciphertext $C \in \{0,1\}^*$ and a tag $T \in \{0,1\}^n$ as inputs and return the plaintext $M \in \{0,1\}^{|C|}$, corresponding to the ciphertext

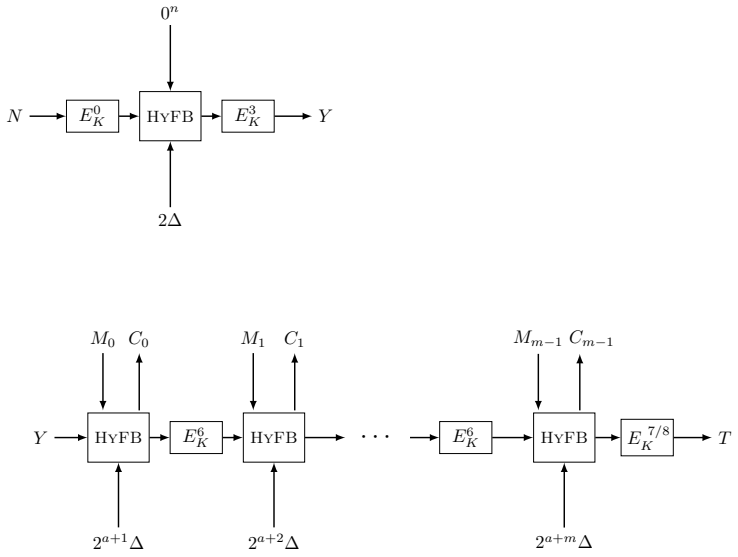


Fig. 3: tHyENA authenticated encryption mode for empty associated data and m block message.

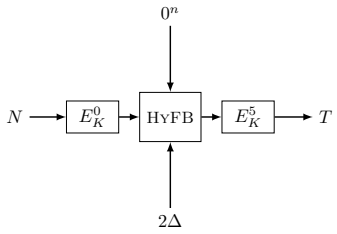


Fig. 4: tHyENA authenticated encryption mode for empty associated data and empty message.

C, if the tag T authenticates. Complete specification of tHyENA is presented in Algorithm 5 and the corresponding pictorial description can be found in Figure 1, 2, 3, 4. We use the same hybrid feedback function as defined in [5]. For completeness, we have given the corresponding pictorial representation in Figure 6 and 7.

Algorithm tHyENA-Enc(K, N, A, M)

1. $Y \leftarrow E_K^0(N)$
2. $\Delta \leftarrow Y_R$
3. $\Delta \leftarrow 2 \odot \Delta$
4. $(Y, \Delta) \leftarrow$ Proc-AD(Y, A)
5. **if** $|M| \neq 0$ **then**
6. $\quad (Y, C) \leftarrow$ Proc-TXT$(Y, \Delta, M, +)$
7. $T \leftarrow Y$
8. **return** (C, T)

Algorithm tHyENA-Dec(K, N, A, C, T)

1. $Y \leftarrow E_K^0(N)$
2. $\Delta \leftarrow Y_R$
3. $\Delta \leftarrow 2 \odot \Delta$
4. $(Y, \Delta) \leftarrow$ Proc-AD(Y, A)
5. **if** $|C| \neq 0$ **then**
6. $\quad (Y, M) \leftarrow$ Proc-TXT$(Y, \Delta, C, -)$
7. $T' \leftarrow Y$
8. **if** $T' = T$ **then return** M
9. **else return** $\perp$

Algorithm HyFB(Y, Δ, M)

1. $C \leftarrow \mathsf{Trunc}_{|M|}(Y) \oplus M$
2. $\overline{M} \leftarrow \mathsf{Pad}(M)$, $\overline{C} \leftarrow \mathsf{Pad}(C)$
3. $B \leftarrow \left(\overline{M}_L \| (\overline{C}_R \oplus \Delta)\right)$
4. $X \leftarrow B \oplus Y$
5. **return** (X, C)

Algorithm HyFB$^-(Y, \Delta, C)$

1. $M \leftarrow \mathsf{Trunc}_{|C|}(Y) \oplus C$
2. $\overline{M} \leftarrow \mathsf{Pad}(M)$, $\overline{C} \leftarrow \mathsf{Pad}(C)$
3. $B \leftarrow \left(\overline{M}_L \| (\overline{C}_R \oplus \Delta)\right)$
4. $X \leftarrow B \oplus Y$
5. **return** (X, M)

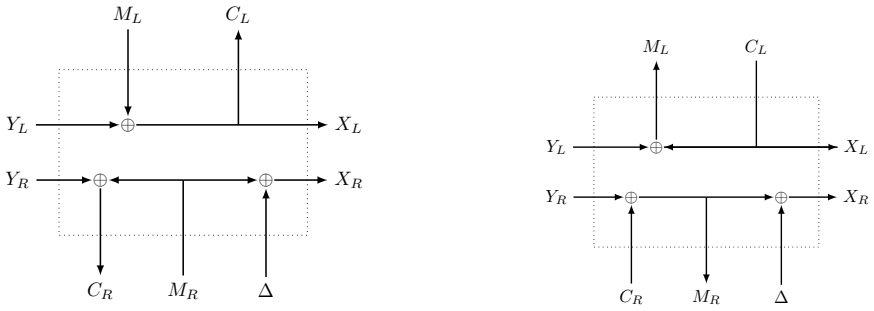
Algorithm Proc-AD(Y, A)

1. **if** $|A| = 0$ **then**
2. $\quad (X, \star) \leftarrow$ HyFB$(Y, \Delta, 0^n)$
3. $\quad t \leftarrow (|M| \neq 0)?\ 3 : 5$
4. $\quad Y \leftarrow E_K^t(X)$
5. **else**
6. $\quad (A_{a-1}, \ldots, A_0) \xleftarrow{n} A$
7. $\quad$ **for** $i = 0$ **to** $a - 2$
8. $\quad\quad (X, \star) \leftarrow$ HyFB(Y, Δ, A_i)
9. $\quad\quad Y \leftarrow E_K^1(X)$
10. $\quad\quad \Delta \leftarrow 2 \odot \Delta$
11. $\quad (X, \star) \leftarrow$ HyFB(Y, Δ, A_{a-1})
12. $\quad t \leftarrow ((|M| \neq 0)$ and $(|A_{a-1}| = n))?\ 2 : 3 : 4 : 5$
13. $\quad Y \leftarrow E_K^t(X)$
14. $\quad \Delta \leftarrow 2 \odot \Delta$
15. $\quad$ **return** (Y, Δ)

Algorithm Proc-TXT$(Y, \Delta, D, \mathsf{dir})$

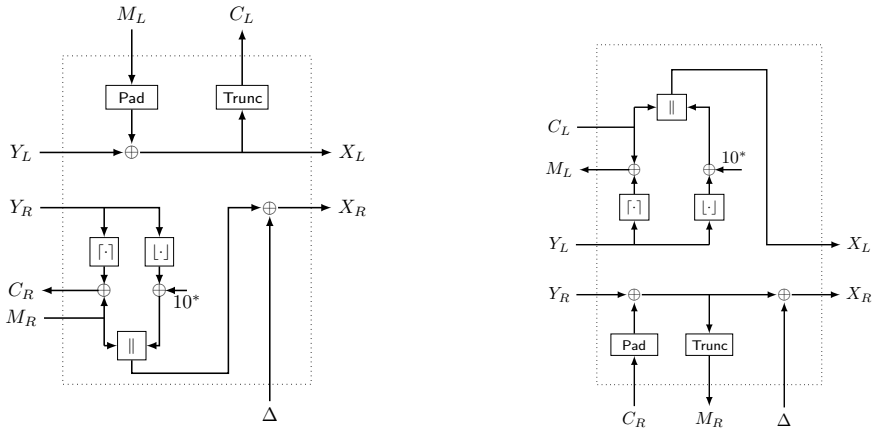
1. $(D_{d-1}, \ldots, D_0) \xleftarrow{n} D$
2. **for** $i = 0$ **to** $d - 2$
3. $\quad$ **if** $\mathsf{dir} = +$ **then**
4. $\quad\quad (X, O_i) \leftarrow$ HyFB(Y, Δ, D_i)
5. $\quad$ **else**
6. $\quad\quad (X, O_i) \leftarrow$ HyFB$^-(Y, \Delta, D_i)$
7. $\quad \Delta \leftarrow 2 \odot \Delta$
8. $\quad Y \leftarrow E_K^6(X)$
9. **if** $\mathsf{dir} = +$ **then**
10. $\quad (X, O_{d-1}) \leftarrow$ HyFB(Y, Δ, D_{d-1})
11. **else**
12. $\quad (X, O_{d-1}) \leftarrow$ HyFB$^-(Y, \Delta, D_{d-1})$
13. $t \leftarrow (|D_{d-1}| = n)?\ 7 : 8$
14. $Y \leftarrow E_K^t(X)$
15. **return** $(Y, (O_{d-1} \| \ldots \| O_0))$

Fig. 5: Formal Specification of tHyENA Authenticated Encryption and Decryption algorithm. For any n-bit string S, we define S_L (and S_R) as the most (and least) significant $n/2$ bits of S i.e. $(S_L, S_R) \xleftarrow{n/2} S$. We use the notation $\star$ to denote values that we do not care.



(a) HyFB+ module. **(b)** HyFB- module.

Fig. 6: HyFB+ and HyFB- module of tHyENA for full data blocks.



(a) HyFB+ module. **(b)** HyFB- module.

Fig. 7: HyFB+ and HyFB- module of tHyENA for partial data blocks.

3.1 Features and Design Rationale

Here, we summarize the salient features and design rationale of tHyENA:

(i) Inverse-Free: tHyENA is an inverse-free authenticated encryption algorithm. Both encryption and verified decryption of the algorithm do not require any decryption call to the underlying block cipher. This reduces the overall hardware footprint significantly, especially in the combined encryption-decryption implementations.

(ii) Optimal: tHyENA requires $(a + m + 1)$ many block cipher invocations to process an a block associated-data and m block message. In [6], it has been shown that this is the optimal number of non-linear primitive calls required for any nonce based authenticated encryption. This feature is particularly important for short messages from the perspective of energy consumption, which is directly dependent upon the number of non-linear[6] primitive calls.

(iii) Low State-size: tHyENA requires a state size as low as $3n/2$-bits along with the key state.

(iv) Low XOR Count: To achieve optimal, inverse-free authenticated ciphers with low state, a possible direction is to use the combined feedback approach where (i) the previous block cipher output is XORed with the plaintext to generate the ciphertext, and (ii) the next block cipher input is defined as the XOR of the plaintext with some linear function of the previous block cipher output. This technique was used in the popular authenticated encryption mode COFB [9]. It is easy to see that such combined feedback function require at least $2n$ bits of XOR operations (when operated on n bit data), along with some additional XOR operations required for the linear function mentioned above. On the contrary, in tHyENA, we use the concept of hybrid feedback or HyFB, where the block cipher input is defined partially via ciphertext feedback and partially via plaintext feedback. This reduces the number of the XOR operations to only n bits.

(v) No Swap or Field Multiplication by 3 or 3^2: tHyENA does not require the constant field multiplications by 3 or 3^2, or the swap operation during finalization, as required in HyENA. It uses the short tweaks efficiently to handle all the necessary domain separations. This is extremely helpful in having very low area footprint.

3.2 HyENA vs tHyENA

There are two significant changes in the tHyENA design over HyENA [5].

- The first significant difference between tHyENA and HyENA is that tHyENA uses Tweakable Blockcipher (TBC) whereas HyENA uses Blockcipher (BC). This change significantly optimizes the hardware area and throughput. It also improves the proof structure (the proof is more well structured now).
- The second significant difference is the secret internal state update. We are simplifying the secret state update and can reduce the update overheads significantly by making the design more compact.

The above-mentioned modifications ensure that the new design tHyENA has the following advantages over HyENA:

(i) First, HyENA uses 3 : 1 128-bit Multiplexors and other field multiplications with 3 (68-bit XOR and 1-bit left) or 3^2 (uses field multiplication by 3 twice in

[6] In general, non-linear primitives consume significantly more energy as compared to linear counterparts

sequence and this reduces the throughput) that acquire a significant amount of hardware area and reduce the throughput. The main reason behind this usage of multiplexors and field multiplication is the domain separation of the inputs (that means, differentiating nonce, message, associated data, and the number of data bits in the last input block for each data type). We use a novel technique of using a TBC instead of a BC (and additional operations). Generally, in most of the TBC-based algorithms, the tweak is used as a counter. In this design, we take a completely different approach and we use the tweak to separate the domains. In general, there are few domains in most of the designs and a small 4-bit tweak is sufficient to separate the domains (e.g, the 4-bit tweak can separate $2^4 = 16$ domains) and circuit area for tweak updates can be reduced a lot. In tHyENA, we use a TBC that deals with small 4-bit tweaks and the area for this tweak processing circuit is very small (only a few 4-bit XORs) as compared to $3:1$ 128-bit Multiplexors plus field multiplication by 3 or 3^2. This technique, can significantly reduce the hardware area and increase the throughput a bit.

(ii) Second, using a TBC makes the design simpler and removes several avoidable operations (for example, constant field multiplications with 3 or 3^2, and the swap operation can be avoided in tHyENA). This makes the construction cleaner, more modular (e.g, we can now replace the blockcipher and several avoidable operations with a simple TBC). The security proof is now much simpler, well structured, and easily readable. To be precise, we adopt the Coefficient H technique for the security proof and HyENA has 8 bad cases to consider. However, due to the simpler structure of tHyENA, we need only 3 bad cases for the security proof of tHyENA.

3.3 Instantiation of tHyENA with TweGIFT

We instantiate tHyENA with tBC TweGIFT-128, or simply TweGIFT, the 128-bit tweakable block cipher with 4-bit tweak and 128-bit key. As the name suggests, it is a tweakable variant of the GIFT [1] block cipher. TweGIFT is composed of 40 rounds and each round is composed of five operations: SubCells, PermBits, AddRoundKey, AddRoundConstant, and AddTweak. The first four operations are identical to that of GIFT. In AddTweak, the 4-bit tweak is first expanded to a 32-bit value:

$$(x_1, x_2, x_3, x_4) \rightarrow (X, X, X, X)\,,\ X \leftarrow (x_1, x_2, x_3, x_4, S{\oplus}x_1, S{\oplus}x_2, S{\oplus}x_3, S{\oplus}x_4),$$

where $S = x_1 \oplus x_2 \oplus x_3 \oplus x_4$. Then the 32-bit value is XORed to the state at an interval of 5 rounds. Technically speaking, it adds the expanded 32-bit tweak to bit positions $4i + 3$, $i = 0 \ldots 31$. A detailed description can be found in [2].

4 On the Security of tHyENA Mode of Operation

In this section, we prove the NAEAD security of tHyENA in shape of the following theorem.

Theorem 1. *Let $Q = q_e + \sigma_e + q_v + \sigma_v$. For $Q \leq 2^{\frac{n}{2}-1}$, we have*

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathit{tHyENA}}(q_e, q_v, \ell_e, \ell_v, \sigma_e, \sigma_v, t) \leq \mathbf{Adv}^{\mathsf{tprp}}_{E}(Q, t') + \frac{2\sigma_e}{2^{n/2}} + \frac{2\sigma_e^2}{2^n} + \frac{q_v}{2^n} + \frac{2n\sigma_v}{2^{n/2}}.$$

where $t' = t + O(Q)$.

Without loss of generality, we can assume that $Q \leq 2^{n/2-1}$, since otherwise the result is vacuously true. First, we replace the block cipher E_K with a uniform random tweakable permutation $\mathsf{\Pi}$ using the standard hybrid argument, and then replace $\mathsf{\Pi}$ with a uniform random tweakable function $\mathsf{\Gamma} : \{0,1\}^\tau \times \{0,1\}^n \to \{0,1\}^n$ using the standard TRP-TRF switching lemma. The cost of these transitions is accounted in the first two terms of our bound. We will employ the Coefficient-H technique for the rest of the proof. Before delving further into the proof we briefly describe this technique in the next subsection.

4.1 Coefficient-H Technique

The coefficient-H technique by Patarin [22, 23] is a tool to upper bound the distinguishing advantage of any deterministic and computationally unbounded distinguisher $\mathcal{A}$ in distinguishing the real oracle $\mathcal{R}$ from the ideal oracle $\mathcal{I}$. We describe the technique in context of the NAEAD security game, i.e., we take $\mathcal{R} = (\mathsf{AE.Enc}, \mathsf{AE.Dec})$ and $\mathcal{I} = (\$, \perp)$.

The collection of all queries and responses that $\mathcal{A}$ made and received to and from the oracle is called the transcript of $\mathcal{A}$, denoted as ω. Let $\mathsf{\Lambda_{re}}$ and $\mathsf{\Lambda_{id}}$ denote the transcript random variable induced by $\mathcal{A}$'s interaction with $\mathcal{R}$ and $\mathcal{I}$, respectively. Let Ω be the set of all transcripts. A transcript $\tau \in \Omega$ is said to be *attainable* if $\Pr[\mathsf{\Lambda_{id}} = \omega] > 0$, i.e., it can be realized by $\mathcal{A}$'s interaction with $\mathcal{I}$.

Theorem 2. *For $\epsilon_1, \epsilon_2 \geq 0$, suppose there is a set $\Omega_{\mathsf{bad}} \subseteq \Omega$, that we call the set of bad transcripts, such that the following conditions hold:*

- $\Pr[\mathsf{\Lambda_{id}} \in \Omega_{\mathsf{bad}}] \leq \epsilon_1$*; and*
- *For any $\tau \notin \Omega_{\mathsf{bad}}$, τ is attainable and* $\dfrac{\Pr[\mathsf{\Lambda_{re}} = \tau]}{\Pr[\mathsf{\Lambda_{id}} = \tau]} \geq 1 - \epsilon_2$.

Then, for any computationally unbounded and deterministic distinguisher $\mathcal{A}$, we have

$$\mathbf{Adv}^{\mathsf{naead}}_{\mathsf{AE}}(\mathcal{A}) \leq \epsilon_1 + \epsilon_2.$$

We skip the proof of Theorem 2 as it is readily available in several previous works including [11, 18].

4.2 Notations and Initial Setup

Fix a $(q_e, q_v, \ell_e, \ell_v, \sigma_e, \sigma_v, \infty)$-adversary $\mathcal{A}$ that interacts with either the real oracle, i.e.,

$$\mathcal{R} := (\mathsf{tHyENA.Enc}, \mathsf{tHyENA.Dec}),$$

or the ideal oracle, i.e., $\mathcal{I} := (\$, \perp)$, making at most

1. q_e encryption queries $(N_i^+, A_i^+, M_i^+)_{i=1..q_e}$, each of length $l_i^+ \leq \ell_e$, with an aggregate of total σ_e many blocks, and
2. attempts to forge with q_v many queries $(N_i^-, A_i^-, C_i^-, T_i^-)_{i=1..q_v}$, each of length $l_i^- \leq \ell_v$, having a total of σ_v many blocks.

We assume that for $1 \leq i \leq q_e$, M_i^+ and A_i^+ have m_i^+ and a_i^+ blocks respectively, and for $1 \leq j \leq q_v$, C_j^- and A_j^- have m_j^- and a_j^- blocks respectively. So, $l_i^+ = m_i^+ + a_i^+$ and $l_j^- = m_j^- + a_j^-$. We use the notation X, Y to denote the intermediate variables. Let

$$(S_i[0], S_i[1], \cdots, S_i[l_i^+ - 1]) \leftarrow (A_i^+[0], \cdots, A_i^+[a_i^+ - 1], M_i^+[0], \cdots, M_i^+[m_i^+ - 1]).$$

Let $(\lambda_i^+[j] : 1 \leq i \leq q_e, 0 \leq j \leq l_i^+)$ denote the tweak sequence in the encryption queries. Similarly, we have the tweak sequence $(\lambda_i^-[j] : 1 \leq i \leq q_v, 0 \leq j \leq l_i^-)$ in the decryption queries. Note that, one can easily deduce these tweak sequences just by observing the query inputs.

If bitwise representation of n-bit string G is $(G_{n-1} \cdots G_0)$. Then for $u > w$, we denote $(G_u \cdots G_w)$ by G_{u-w} which is a $u - w + 1$-bit substring of G from u^{th} bit to w^{th} bit of G.

4.3 Overview of Attack Transcript

We begin with a description of the ideal oracle which consists of two phases.

- **Online phase**: For the i^{th} encryption query $(N_i^+, A_i^+ = (A_i[0], \ldots, A_i^+[a_i - 1]), M_i^+ = (M_i^+[0], \ldots, M_i^+[m_i - 1]))$, the oracle samples $(Y_i^+[a_i^+], \ldots, Y_i^+[l_i^+]) \leftarrow_{\$} \{0,1\}^{n(m_i^+ + 1)}$ independently. It next sets the tag $T_i^+ = Y_i^+[l_i^+]$ and $C_i^+ = (C_i^+[0], \ldots, C_i^+[m_i^+ - 1])$ where $C_i^+[j] = Y_i^+[j + a_i^+] \oplus M_i^+[j]$ for $0 \leq j \leq m_i^+ - 1$ and returns (C_i^+, T_i^+) to $\mathcal{A}$.
- **Offline phase**: After $\mathcal{A}$ makes all the queries the oracle samples other Y^+ values as $Y_i^+[j] \leftarrow_{\$} \{0,1\}^n$, for $0 \leq j \leq a_i - 1$.

For convenience, we slightly modify the experiment where we reveal to the adversary $\mathcal{A}$ (after $\mathcal{A}$ made all its queries and obtains corresponding responses but before it outputs its decision) the Y^+-values and now the adversary can set all intermediate values $X_i^+[j]$ using $S_i[j]$ and $Y_i^+[j]$. Note that $\Delta_i^+ = \lfloor Y_i^+[0] \rfloor$ and $\Delta_i^- = \lfloor Y_i^-[0] \rfloor$.
Overall, the transcript of the adversary $\omega := (\omega_e, \omega_v)$ be the list of queries and responses of $\mathcal{A}$ that constitutes the query response transcript of $\mathcal{A}$, where

- $\omega_e = (N_i^+, A_i^+, M_i^+, X_i^+, Y_i^+, T_i^+)_{i=1..q_e}$,
- $\omega_v = (N_j^-, A_j^-, C_j^-, T_j^-, \perp)_{j=1..q_v}$.

A prefix for a decryption query is defined as the common prefix blocks between the decryption query input string and an encryption query (if any) output string prepended with the nonce and the associated data. The length of the longest common prefix for the i^{th} decryption query is denoted as p_i. Note that if the decryption query uses a fresh nonce (not occurred during encryption queries), then it does not share any common prefix with any of the encryption queries then we set $p_i = -1$.

4.4 Identifying and Bounding Bad Events

We say that a transcript is bad if one of the following conditions is satisfied:

B1: $\mathtt{mcoll}(\lceil X^+ \rceil) > n$, where $\lceil X^+ \rceil := (\lceil X_i^+[j] \rceil : 1 \le i \le q_e, 1 \le j \le l_i^+)$.

B2: there exists $(i,j) \neq (i',j')$, such that, $\lambda_i^+[j] = \lambda_{i'}^+[j'] \wedge X_i^+[j] = X_{i'}^+[j']$.

B3: there exists $i, (i',j')$, such that, $\lambda_i^-[p_i{+}1] = \lambda_{i'}^+[j'] \wedge X_i^-[p_i{+}1] = X_{i'}^+[j']$.

The following lemma bounds the probability of bad transcripts in ideal oracle.

Lemma 1. *For $\sigma_e \le 2^{\frac{n}{2}-1}$, we have*

$$\Pr[\Lambda_{\mathsf{id}} \in \Omega_{\mathsf{bad}}] \le \frac{2\sigma_e}{2^{n/2}} + \frac{2\sigma_e^2}{2^n} + \frac{nq_v}{2^{n/2}}.$$

Proof. By definition of bad transcripts, we have

$$\begin{aligned} \Pr[\Lambda_{\mathsf{id}} \in \Omega_{\mathsf{bad}}] &= \Pr[\mathtt{B1} \vee \mathtt{B2} \vee \mathtt{B3}] \\ &\le \Pr[\mathtt{B1}] + \Pr[\mathtt{B2}] + \Pr[\mathtt{B3} | \neg \mathtt{B1}]. \end{aligned} \tag{1}$$

Now, we bound the probability terms on the right hand side one by one:

Bounding $\Pr[\mathtt{B1}]$: The event B1 is a multicollision event for uniformly chosen n many $n/2$-bit strings out of σ_e many $n/2$-bit strings. As the Y^+-values are sampled uniformly and independently in the ideal game, we have,

$$\Pr[\mathtt{B1}] \le \frac{\binom{\sigma_e}{n}}{2^{n/2(n-1)}} \le \left(\frac{2\sigma_e}{2^{n/2}}\right)^n \le \frac{2\sigma_e}{2^{n/2}}. \tag{2}$$

The last inequality follows from the assumption that $\sigma_e \le 2^{\frac{n}{2}-1}$.

Bounding $\Pr[\mathtt{B2}]$: For any $(i,j) \neq (i',j')$, $\lambda_i^+[j] = \lambda_{i'}^+[j']$ and $j, j' > 0$, we have the following two possibilities:

(a) $j < l_i^+, j' < l_{i'}^+$: for any $(i,j) \neq (i',j')$,the event $X_i^+[j] = X_{i'}^+[j']$ is nothing but two non-trivial linear equations. One is on $\lceil Y_i^+[j-1] \rceil$ & $\lceil Y_{i'}^+[j'-1] \rceil$ and other is on $\delta_j \odot \Delta_i^+$ & $\delta_{j'} \odot \Delta_{i'}^+$ for some constants δ_j & $\delta_{j'}$. For $i \neq i'$, we have $\lceil Y_i^+[j-1] \rceil, \lceil Y_{i'}^+[j'-1] \rceil, \Delta_i^+$ and $\Delta_{i'}^+$ are independent and uniformly distributed. For $i = i'$, we have $\delta_j \neq \delta_{j'}$ and $\lceil Y_i^+[j-1] \rceil, \lceil Y_{i'}^+[j'-1] \rceil$ are independent and uniformly distributed. Hence this event has probability at most 2^{-n}. Therefore,

$$\Pr[X_i^+[j] = X_{i'}^+[j']] \le \frac{(\sigma_e - q_e)^2}{2^n}.$$

(b) $j = l_i^+, j' = l_{i'}^+$: This can be handled in a similar manner as case (a). So, we have

$$\Pr[X_i^+[j] = X_{i'}^+[j']] \le \frac{q_e^2}{2^n}.$$

Therefore,

$$\Pr[\mathtt{B2}] \leq \frac{(\sigma_e - q_e)^2 + q_e^2}{2^n} \leq \frac{2\sigma_e^2}{2^n}. \tag{3}$$

Bounding $\Pr[\mathtt{B3}|\neg\mathtt{B1}]$: The condition $\neg\mathtt{B1}$ implies that there are at most n possible choices for (i', j') for any fixed choice of i'. Once we fix (i', j'), we get an equality relation in the least significant $n/2$ bits. Now, we have have the following cases:

(a) $p_i = -1$: This case is actually not possible. As $\lambda_i^-[0] = \lambda_{i'}^+[j]$ if and only if $j = 0$. But $N_i^- \neq N_{i'}^+$ (since $p_i = -1$), whence $X_i^-[0] \neq X_{i'}^+[j']$.

(b) $0 \leq p_i < l_i^- - 1$: Since $p_i \geq 0$, we have $N_i^- = N_k^+$ for some k. Suppose $k \neq i'$. Then we obtain a non-trivial linear equation on $\Delta_{i'}^+$. Therefore, the probability in this case is at most $\frac{nq_v}{2^{n/2}}$. Now, suppose $k = i'$. Then we must have $j' \neq p_i + 1$. Otherwise we get $C_i^-[p_i] = C_k^+[p_i]$ which contradicts the definition of p_i. Hence we get the probability at most $\frac{q_v}{2^{n/2}}$.

(c) $p_i = l_i^- - 1$: In this case, j' must equal to $l_{i'}^+$ (as $\lambda_i^-[p_i + 1] = \lambda_{i'}^+[j']$). Following similar line of argument as in case (b), we get a bound of $\frac{nq_v}{2^{n/2}}$.

$$\Pr[X_i^-[l_i^-] = X_{i'}^+[j'] \wedge \neg\mathtt{B1}] \leq \frac{nq_v}{2^{n/2}}.$$

By accumulating all the cases above, we get

$$Pr[\mathtt{B3}|\neg\mathtt{B1}] \leq \frac{nq_v}{2^{n/2}}. \tag{4}$$

The result follows from Eq. (1)–(4). □

4.5 Good Transcript Analysis

We fix $\omega \in \Omega_{\mathsf{good}}$. Let $\omega = (\omega_e, \omega_v)$, where

$$\omega_e = (N_i^+, A_i^+, M_i^+, X_i^+, Y_i^+, T_i^+)_{i=1..q_e},$$

and

$$\omega_v = (N_i^-, A_i^-, C_i^-, T_i^-, \perp)_{i=1..q_v}.$$

First, it is easy to see that

$$\Pr[\Lambda_{\mathsf{id}} = \omega] = 1/2^{n(\sigma_e+q_e)} \tag{5}$$

Next, we consider the real world. As $\neg\mathtt{B2}$ holds, all the inputs of the tweakable random function are distinct and hence all the Y^+-values are independent and uniformly distributed. Therefore, $\Pr[\Lambda_{\mathsf{re}e} = \omega_e] = \frac{1}{2^{n(\sigma_e+q_e)}}$. Now, we have

$$\begin{aligned}
\Pr[\Lambda_{\mathsf{re}} = \omega] &= \Pr[(\Lambda_{\mathsf{re}e}, \Lambda_{\mathsf{re}v}) = (\omega_e, \omega_v)] \\
&= \Pr[\Lambda_{\mathsf{re}v} = \omega_v | \Lambda_{\mathsf{re}e} = \omega_e] \times \Pr[\Lambda_{\mathsf{re}e} = \omega_e] \\
&= \frac{1}{2^{n(\sigma_e+q_e)}} \times \Pr[\Lambda_{\mathsf{re}v} = \omega_v | \Lambda_{\mathsf{re}e} = \omega_e] \\
&= \frac{1}{2^{n(\sigma_e+q_e)}} \times (1 - \Pr[\Lambda_{\mathsf{re}v} \neq \omega_v | \Lambda_{\mathsf{re}e} = \omega_e])
\end{aligned} \tag{6}$$

Let $\mathtt{E}$ be the event that $\forall 1 \leq i \leq q_v, p_i + 1 < j \leq l_i^-$, $X_i^-[j] = X_{i'}^+[j']$ and $\lambda_i^-[j] = \lambda_{i'}^+[j']$ for some (i', j'). *Here we remark that, in case of* HyENA *one needs to consider another type of collision, where* $X_i^-[j] = X_i^-[j'']$ *for some* $j' < j''$*. However, this is not required in our case due to dedicated tweak for tag generation.* As the event $\neg\mathtt{B3}$ holds for the good transcript, $Y_i^-[p_i + 1]$ is uniformly random. Due to the property of feedback function, $X_i^-[p_i + 2]$ is also uniformly random.
Now we need to calculate $\Pr[\Lambda_{\mathsf{re}v} = \omega_v | \Lambda_{\mathsf{re}e} = \omega_e]$.

$$\begin{aligned}\Pr[\Lambda_{\mathsf{re}v} = \omega_v | \Lambda_{\mathsf{re}e} = \omega_e] &= 1 - \Pr[\Lambda_{\mathsf{re}v} \neq \omega_v | \Lambda_{\mathsf{re}e} = \omega_e] \\ &= 1 - (\Pr[\Lambda_{\mathsf{re}v} \neq \omega_v, \mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e] + \Pr[\neg\mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e])\end{aligned} \tag{7}$$

Here, $\Pr[\Lambda_{\mathsf{re}v} \neq \omega_v, \mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e]$ is the probability that $\exists 1 \leq i \leq q_v$ such that T_i^- is correct. But $T_i^- = Y_i^-[l_i^-]$ and the event $\mathtt{E}$ implies that $Y_i^-[l_i^-]$ is uniformly random. Hence $\Pr[\Lambda_{\mathsf{re}v} \neq \omega_v, \mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e]$ is the probability of guessing T_i^- correctly. Therefore,

$$\Pr[\Lambda_{\mathsf{re}v} \neq \omega_v, \mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e] \leq \frac{q_v}{2^n} \tag{8}$$

Now, consider $\Pr[\neg\mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e]$. The event $\neg\mathtt{E}$ can be described as: for all $1 \leq i \leq q_v$ and $p_i + 1 \leq j \leq l_i^-$, $X_i^-[j] = X_{i_1}^+[j_1]$ for some i_1, j_1. The event $\neg\mathtt{B1}$ holds for good transcripts. Hence (i_1, j_1) can take at most n values. Then for a fixed i, we have
$\Pr[X_i^-[j] = X_{i_1}^+[j_1]] \leq \frac{n.l_i^-}{2^{n/2}}$. Since $\sum_{1 \leq i \leq q_v}(l_i^-) \leq \sigma_v$, summing over all $1 \leq i \leq q_v$, we have

$$\Pr[\Lambda_{\mathsf{re}v} \neq \omega_v, \neg\mathtt{E} | \Lambda_{\mathsf{re}e} = \omega_e] \leq \frac{n\sigma_v}{2^{n/2}} \tag{9}$$

From Eq. (5)-(9), we get

$$\begin{aligned}\Pr[\Lambda_{\mathsf{re}} = \omega] &\geq \frac{1}{2^{n(\sigma_e + q_e)}} \times \left(1 - \frac{q_v}{2^n} - \frac{n\sigma_v}{2^{n/2}}\right) \\ &\geq \Pr[\Lambda_{\mathsf{id}} = \omega] \times \left(1 - \frac{q_v}{2^n} - \frac{n\sigma_v}{2^{n/2}}\right).\end{aligned}$$

The result follows from Coefficient-H Theorem 2 in combination with Lemma 1.

A Note on the Security of TweGIFT. In our AEAD algorithm, we utilize the tweakable pseudorandom permutation security of TweGIFT. Since TweGIFT is an extension of GIFT-128 blockcipher – a well-known and studied cipher – it benefits from the extensive analysis [13, 17, 25, 27] already present for GIFT-128. Indeed, for tweak value 0 (the setting for majority of TweGIFT calls made in our algorithm), TweGIFT is exactly similar to GIFT-128 blockcipher, whence all the cryptanalytic results directly translate to this case. In addition, TweGIFT-128 has also been analyzed as a dedicated tweakable blockcipher in [2–4].

Table 1: Clock cycles per message byte for tHyENA

	Message length (Bytes)											
	16	32	64	128	256	512	1024	2048	4096	16384	32768	262144
cpb	10.3125	6.469	4.547	3.586	3.105	2.865	2.745	2.685	2.655	2.633	2.629	2.625

5 Hardware Implementation Results

tHyENA aims to achieve a lightweight implementation on low resource devices. tHyENA has a simple structure with a blockcipher and a few linear operations. It has a small state size and the complete circuit size is dominated by the underlying blockcipher. In this section we provide hardware implementation details of tHyENA instantiated with the GIFT blockcipher.

5.1 Clock Cycle Analysis

We provide a conventional way for speed estimation, i.e, the number of clock cycles to process input bytes. Since tHyENA processes at least one associated data (AD) block (one dummy block when AD is empty), we calculate the cpb assuming one AD block and m message blocks. We use 40 round GIFT and need 40 cycles for the GIFT module. We use 2 more cycles to compute the feedback and update the Δ value. Overall, tHyENA needs $(42(m+1)+81)$ cycles. Table 1 shows the number of average cycles per input message bytes, which we call cycles per byte (cpb). The cpb is $(42(m+1)+81)/16m$ and it converges to 2.625 for very large m.

5.2 Hardware Architecture

tHyENA is based on E-t-M paradigm and the message blocks are processed along with the associated data blocks to generate the ciphertext blocks and the tag. We use the same circuit for both the associated data and ciphertext processing as they are computed similarly. Only a change in the blockcipher tweak value for the two types of input data is required to distinguish. We provide the hardware architecture and briefly describe the individual components of this architecture. For the sake of simplicity, we remove the control unit from Fig. 8 and present a separate control unit in Fig. 9. The main components in the hardware circuit are briefly described below.

State Registers. The hardware circuit consists of two registers. The primary state register is used to store the internal state, this register is internally used by the TweGIFT module. The Δ register is used to store the Δ value, this register is internally used by the X2 module.

Module TweGIFT. The module TweGIFT is used to compute one round of the underlying tweakable blockcipher. TweGIFT uses an n-bit internal register to

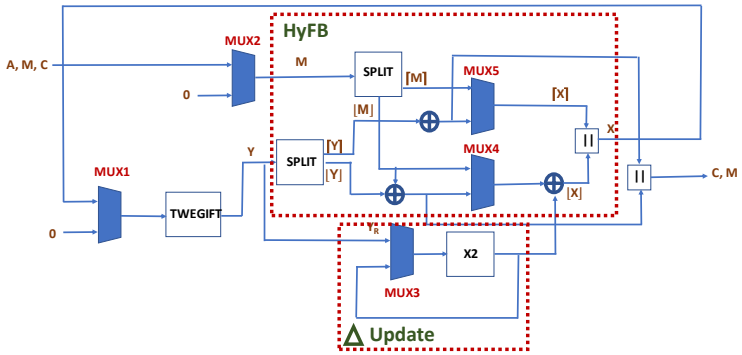


Fig. 8: Hardware Architecture Diagram

hold the blockcipher internal state. This register is updated with a new state value whenever the state is updated with the round function or other operations. TweGIFT also uses an internal control unit, that we are omitting from the description for simplicity.

Other Modules. Apart from the above two main components, we have a $||$ module that concatenates two strings into one, a SPLIT module that splits the internal state into two parts, and X2 module that multiplies Δ by 2. The SPLIT module is mainly used in the hybrid feedback function.

Remark 1. (Combined Encryption and Decryption) In this implementation, we mainly focus on a combined encryption-decryption circuit. We observe that we can also implement encryption-only circuits even with a small decrease in hardware area and with the same throughput.

5.3 Control Unit

We also describe the control unit in our implementation. We first list the control signals and next we describe the states in the control unit.

□ **Data Signals**. The hardware circuit uses several internal data signals controlled by the finite state machine (FSM). The circuit uses the following signals.

- **Start**: This signal signifies the start of the circuit. This signal makes a transition from the **WAIT** to the **COMP_E$_K$(N)** state.
- **Rdy**: This signal signifies the start of the corresponding module.
- **Empty AD**: This signal signifies whether the associated data is empty or not.
- **Last AD**: This signal signifies whether the current associated data block is the last or not.

- **Last Msg**: This signal signifies whether the current message block is the last or not.

□ **FSM**. This module controls the hardware circuit for tHyENA. **FSM** generates and controls signals to operate the state transitions. It generates and sends signals to different modules and divides the functionalities of the circuit into several states. This is depicted in Fig. 9. Note that, for the sake of simplicity we omit the control unit of the TweGIFT module. This module uses its own architecture and control unit.

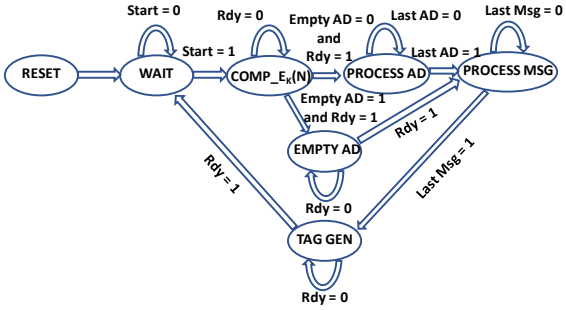


Fig. 9: Finite State Machine

- **RESET**: This state resetss all the circuit parameters.
- **WAIT**: This state signifies the start of the circuit. **COMP_E_K(N)**. This state corresponds to the first blockcipher call
- **PROCESS AD**: This state corresponds to the processing of the associated data blocks.
- **PROCESS MSG**: This state corresponds to the processing of the message blocks.
- **EMPTY AD**: This state signifies that the associated data is empty and the control goes to the message processing phase.
- **TAG GEN**: This state corresponds to the tag generation phase and the control goes to this state from the **PROCESS MSG** state after the last message is processed.

Note that, for decryption, the process is the same, with just a few changes in some control signals.

5.4 Implementation Results

We implement tHyENA on Virtex 7 (xc7v585tffg1761-3), using VHDL and VIVADO. The implementation follows the NIST LWC API. The result includes all

Table 2: FPGA implementation results of tHyENA and HyENA

Design (Platform)	Slice Registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/LUT	Mbps/Slice
tHyENA (Virtex 7)	472	679	261	555	1.73	2.548	6.628
HyENA (Virtex 7)	470	725	280	555	1.73	2.386	6.179

the overheads caused by this API. Table 2 presents the implementation results. We follow the RTL approach and a basic iterative type architecture with 128-bit datapath. The areas are provided in the number of LUTs and slices. Frequency (MHz), Throughput (Gbps), and throughput-area efficiencies are also reported in addition to the hardware areas. Table 2 presents the mapped hardware results of tHyENA.

We have also made our own implementation for the NIST submission HyENA under the same setup and using the NIST LWC API. The results for HyENA is given in Table 2 below. The results reveal that both tHyENA and HyENA achieve the same frequency but tHyENA is significantly better than HyENA in hardware area.

5.5 Benchmarking with Feedback Based Constructions

We benchmark our implemented results using the existing FPGA results (note that, all the other benchmarking candidates are feedback-based similar as tHyENA) on Virtex 7. We provide comparisons with the implementations from the references cited in Table 3 below.

Table 3: Comparison of Lightweight Feedback based AE Modes on Virtex 7. '-' denotes results not available.

Scheme	Rate	LUT	Slices	T'put (GBps)	Mbps / LUT	Mbps / Slice
tHyENA-TweGIFT128	**1**	**679**	**261**	**1.73**	**2.548**	**6.628**
HyENA-GIFT128	1	725	280	1.73	2.386	6.179
COFB[GIFT] [8, 10]	1	771	316	2.230	2.892	6.623
COFB[GIFT]-CAESAR-API [8, 10]	1	1041	355	1.164	1.174	2.604
COFB[AES] [8, 10]	1	1440	564	2.933	2.031	5.191
COFB[AES]-CAESAR-API [8, 10]	1	1496	579	2.747	1.842	4.395
ESTATE-TweGIFT128 [4]	1/2	681	263	0.84	1.23	3.20
SUNDAE-GIFT128 [4]	1/2	931	310	0.84	0.90	2.71
CLOC-AES [15]	1/2	3145	891	2.996	0.488	1.724
CLOC-TWINE [15]	1/2	1689	532	0.343	0.203	0.645
CLOC-AES-Optimized [15, 16]	1/2	-	595	0.695	-	1.17
SILC-AES [15]	1/2	3066	921	4.040	1.318	4.387
SILC-LED [15]	1/2	1685	579	0.245	0.145	0.422
SILC-PRESENT [15]	1/2	1514	548	0.407	0.269	0.743

Acknowledgements. The authors would like to thank all the anonymous reviewers of Indocrypt 2021 for their valuable comments. Prof. Mridul Nandi is supported by the project "Study and Analysis of IoT Security" by NTRO under the Government of India at R.C.Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata. Dr. Ashwin Jha's work was carried out in the framework of the French-German-Center for Cybersecurity, a collaboration of CISPA and LORIA.

References

1. 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 *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.
2. Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. Elastic-tweak: A framework for short tweak tweakable block cipher. *IACR Cryptol. ePrint Arch.*, 2019:440, 2019.
3. Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. INT-RUP secure lightweight parallel AE modes. *IACR Trans. Symmetric Cryptol.*, 2019(4):81–118, 2019.
4. Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. ESTATE: A lightweight and low energy authenticated encryption mode. *IACR Trans. Symmetric Cryptol.*, 2020(S1):350–389, 2020.
5. Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Snehal Mitragotri, and Mridul Nandi. From combined to hybrid: Making feedback-based AE even smaller. *IACR Trans. Symmetric Cryptol.*, 2020(S1):417–445, 2020.
6. Avik Chakraborti, Nilanjan Datta, and Mridul Nandi. On the optimality of non-linear computations for symmetric key primitives. *J. Mathematical Cryptology*, 12(4):241–259, 2018.
7. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 277–298, 2017.
8. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? *IACR Cryptol. ePrint Arch.*, 2017:649, 2017.
9. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *CHES 2017*, pages 277–298, 2017.
10. Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? *J. Cryptol.*, 33(3):703–741, 2020.
11. Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014.

12. CAESAR Committee. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html/.
13. Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 115–138. Springer, 2018.
14. Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *FSE 2012*, pages 196–215, 2012.
15. Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. Submission to CAESAR. 2016. https://competitions.cr.yp.to/round3/clocsilcv3.pdf.
16. Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. A comprehensive performance analysis of hardware implementations of CAESAR candidates. *IACR Cryptology ePrint Archive*, 2017:1261, 2017.
17. Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In Julian Jang-Jaccard and Fuchun Guo, editors, *Information Security and Privacy - 24th Australasian Conference, ACISP 2019, Christchurch, New Zealand, July 3-5, 2019, Proceedings*, volume 11547 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2019.
18. Bart Mennink and Samuel Neves. Encrypted davies-meyer and its dual: Towards optimal security using mirror theory. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 556–583. Springer, 2017.
19. Miguel Montes and Daniel Penazzi. AES-CPFB v1. Submission to CAESAR. 2015. https://competitions.cr.yp.to/round1/aescpfbv1.pdf.
20. NIST. Lightweight cryptography. https://csrc.nist.gov/Projects/Lightweight-Cryptography.
21. National Centre of Excellence. Light-weight Cipher Design Challenge. https://www.dsci.in/ncoe-light-weight-cipher-design-challenge-2020/.
22. J. Patarin. Etude de Géńerateurs de Permutations Basés sur les Schémas du DES. Ph. Thesis. Inria, Domaine de Voluceau, France., 1991.
23. Jacques Patarin. The "Coefficients H" Technique. In *SAC 2008*, pages 328–345, 2008.
24. Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.
25. Yu Sasaki. Integer linear programming for three-subset meet-in-the-middle attacks: Application to gift. In Atsuo Inomata and Kan Yasuda, editors, *Advances in Information and Computer Security*, pages 227–243, Cham, 2018. Springer International Publishing.
26. Liting Zhang, Wenling Wu, Han Sui, and Peng Wang. iFeed[AES] v1. Submission to CAESAR, 2014. https://competitions.cr.yp.to/round1/ifeedaesv1.pdf.
27. Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced GIFT. In Mitsuru Matsui, editor, *Topics in Cryptology - CT-RSA 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 372–390. Springer, 2019.