Light-OCB: Parallel Lightweight Authenticated Cipher with Full Security

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 authenticated encryption (AE) scheme, called Light-OCB, which can be viewed as a lighter variant of the CAESAR winner OCB as well as a faster variant of the high profile NIST LWC competition submission LOCUS-AEAD. Light-OCB is structurally similar to LOCUS-AEAD and uses a nonce-based derived key that provides optimal security, and short-tweak tweakable blockcipher (tBC) for efficient domain separation. Light-OCB improves over LOCUS-AEAD by reducing the number of primitive calls, and thereby significantly optimizing the throughput. To establish our claim, we provide FPGA hardware implementation details and benchmark for Light-OCB against LOCUS-AEAD and several other well-known AEs. The implementation results depict that, when instantiated with the tBC TweGIFT64, Light-OCB achieves an extremely low hardware footprint - consuming only around 1128 LUTs and 307 slices (significantly lower than that for LOCUS-AEAD) while maintaining a throughput of 880 Mbps, which is almost twice that of LOCUS-AEAD. To the best of our knowledge, this figure is significantly better than all the known implementation results of other lightweight ciphers with parallel structures.

Keywords: Authenticated Encryption, lightweight, tBC, Light-OCB, parallel

1 Introduction

From the recent past, lightweight cryptography is enjoying high popularity due to an increase in the demands of security for lightweight IoT applications such as healthcare applications, sensor-based applications, banking applications, etc., where resource-constrained devices communicate and need to be implemented with a low resource. Lightweight cryptography involves providing security in these resource-constrained environments. The importance of this research domain has been addressed by the ongoing NIST Lightweight Standardization Competition (LWC) [17] followed by the CAESAR [8]. Hence, in recent years, the cryptographic research community has seen a surge in new lightweight authenticated encryption proposals.

One popular design approach for lightweight AE schemes is to use a blockcipher based parallel structure as it can be efficient for both lightweight and faster implementations. Blockcipher (BC) based parallel AE schemes popularly use XEX structure. It processes all the inputs in parallel and finally integrates them. Thus, blockcipher-based parallel AE schemes can be well described by the underlying blockcipher, and the final integration function. Consequently, the efficiency and the hardware footprint of the AE scheme also largely depend on these two components. In the following part, we assume that the underlying blockcipher is ultra-lightweight and efficient 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 where the upper bound on the rate value is one. Here, we only concentrate on rate-1 authenticated encryptions with a small hardware footprint such that we can achieve a lightweight construction as well as a high throughput construction.

1.1 Parallel Authenticated Encryption

Parallel AE modes, such as OCB[21], OTR[15], COPA[1], ELmD[7] have mainly been designed to exploit the advantage of parallel computations needed for several high performance computing environments. These constructions mainly concentrate on efficiency in software as well as on faster implementations in hardware. Among them, OCB and OTR are efficient, achieve rate one, and COPA and ELmD achieve rate half. One of the disadvantages of such schemes are large state size. For example, OCB has $3n + k$ and OTR, COPA and ELmD require $4n + k$-bit state size where n and k are the block size and the key size respectively. Furthermore, both of them are only birthday bound secure in the block size n. This means they need at least an 128-bit block cipher to satisfy the NIST criteria (which says that when the key size is 128 bits, any cryptanalytic attack should need at least 2^{112} computations in a single key setting). This large state size makes these designs inefficient for lightweight applications. One possible way out is to improve the security, and thereby instantiate with 64-bit primitives such as PRESENT[6], SKINNY[4], or GIFT[3] that have ultra lightweight implementation with decent throughputs. In [16], Naito proposed a variant of OCB, called ΘCB+, which offers beyond the birthday bound security but does not meet NIST's security criteria if instantiated with 64-bit primitive. This raises the important question of whether it is possible to design AE schemes using 64-bit primitives that satisfies the NIST criteria.

1.2 LOCUS-AEAD

Chakraborti et al. answers the above question in a positive direction by proposing LOCUS-AEAD [10] that employs OCB style encryption with nonce-based derived key that boosts the design by providing full security, and hence realizable by 64-bit primitives. Additionally, the novel use of short-tweak tweakable block ciphers handles all the domain separation, and makes the design simple and compact. To achieve RUP security, the construction uses two primitives in a sequential manner to process each message block, which degrades the throught, and hence the speed of the cipher, which is the primary focus for parallel constructions. So, we ask the question

"Can we design a rate-1 parallel authenticated cipher with full security?"

1.3 Our Contribution

We answer the above question in an affirmative way by presenting a new *rate one* parallel, nonce based authenticated encryption mode of operation with full security named Light-OCB. As the name suggests, Light-OCB follows the general design paradigms of popular NAEAD modes OCB [13,12]. However, we update Light-OCB introduces several key changes (see section 3.1 for more details) in order to add new features. Some of the important changes include nonce-based rekeying and short-tweak based domain separation similarly as used in [10].

Light-OCB achieves higher NAEAD security bounds with lighter primitives. It allows close to 2^{64} data and 2^{128} time limit when instantiated by a block cipher with 64-bit block and 128-bit key. Light-OCB is a single pass, online, fully parallelizable, rate-1 authenticated encryption mode. This mode is extremely versatile, in the sense that, it is equally suitable for lightweight memory constrained environments, as well as high-performance applications. We provide concrete AE security proof for Light-OCB in the ideal-cipher model.

We instantiate Light-OCB with TweGIFT-64 [9,10], a tweakable variant of the GIFT-64-128 [3] block cipher. TweGIFT-64 is a dedicated design, built upon the original GIFT-64-128 block cipher, for efficient processing of small tweak values of size 4-bit. TweGIFT-64 provides sufficient security while maintaining the lightweight features of GIFT-64-128. We propose Light-OCB [TweGIFT-64], the TweGIFT-64 based instantiation of Light-OCB.

Finally, we also provide our own hardware implementation results for Light-OCB on FPGA. The implementation result depicts Light-OCB is significantly better than LOCUS-AEAD in throughput (twice the value for LOCUS-AEAD). In fact, Light-OCB also improves the hardware area over LOCUS-AEAD.

1.4 Applications and Use Cases

The most important feature of Light-OCB is its scope of applicability. At one end of the spectrum, the parallelizability of Light-OCB make them a perfect candidate for applications in high-performance infrastructures. On the other end, there overall state size is competitively small with respect to many existing lightweight candidates, which makes them suitable for low-area hardware implementations. We would like to emphasize that Light-OCB is inherently parallel and can be implemented in a fully pipelined manner keeping a comparable area-efficient implementation. Hence, it is well-suited for protocols that require both lightweight and high-performance implementations e.g, lightweight clients interacting with high performance servers (e.g, LwM2M protocols [19]). Some real life applications, where our proposed mode would best fit includes vehicular applications and memory encryptions.

1.5 Light-OCB 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" [18]. 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 Light-OCB has been nominated as one of the top three candidates in the challenge and has been selected for the final round. Interestingly, this is the only construction that supports full pipelined implementation alongside a very hardware footprint, making it to be the most versatile design in the competition.

2 Preliminaries

2.1 Notations and Conventions.

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 binary strings, respectively. For $A \in \{0,1\}^*$, $|A|$ denotes the length (number of the bits) of A, where $|\lambda| = 0$ by convention. For all practical purposes, we use the little-endian format for representing binary strings, i.e., the least significant bit is the rightmost bit. For any non-empty binary string X, $(X_{k-1},\ldots,X_0) \xleftarrow{n} x$ denotes the n-bit block parsing of X, where $|X_i| = n$ for $0 \leq i \leq k-2$, and $1 \leq |X_{k-1}| \leq n$. For $A, B \in \{0,1\}^*$ and $|A| = |B|$, we write $A \oplus B$ to denote the bitwise XOR of A and B.

We use the notation $\widetilde{\mathsf{E}}$ to denote a tweakable block cipher. For $K \in \{0,1\}^\kappa$, $T \in \{0,1\}^\tau$, and $M \in \{0,1\}^n$, we use $\widetilde{\mathsf{E}}_{K,T}(M) := \widetilde{\mathsf{E}}(K,T,M)$ to denote invocation of the encryption function of $\widetilde{\mathsf{E}}$ on input K, T, and M. The decryption function is analogously defined as $\widetilde{\mathsf{D}}_{K,T}(M)$. We fix positive even integers n, τ, κ, r, and t to denote the *block size*, *tweak size*, *key size*, *nonce size*, and *tag size*, respectively, in bits. Throughout this document, we fix $n = 64$, $\tau = 4$, and $\kappa = 128$, $r = \kappa$, and $t = n$.

We sometimes use the terms (*complete*) *blocks* for n-bit strings, and *partial blocks* for m-bit strings, where $m < n$. Throughout, we use the function ozs, defined by the mapping

$$\forall X \in \bigcup_{m=1}^{n} \{0,1\}^m, \quad X \mapsto \begin{cases} 0^{n-|X|-1} \| 1 \| X & \text{if } |X| < n, \\ X & \text{otherwise,} \end{cases}$$

as the padding rule to map partial blocks to complete blocks. Note that the mapping is injective over partial blocks. For any $X \in \{0,1\}^+$ and $0 \leq i \leq |X|-1$, x_i denotes the i-th bit of X. The function chop takes a string X and an integer $i \leq |X|$, and returns the least significant i bits of X, i.e., $x_{i-1} \cdots x_0$.

The set $\{0,1\}^\kappa$ can be viewed as the finite field $\mathbb{F}_{2^\kappa}$ consisting of 2^κ elements. Addition in $\mathbb{F}_{2^\kappa}$ is just bitwise XOR of two κ-bit strings, and hence denoted by $\oplus$. $P(x)$ denotes the primitive polynomial used to represent the field $\mathbb{F}_{2^\kappa}$, and α denotes the primitive element in this representation. The multiplication of $A, B \in \mathbb{F}_{2^\kappa}$ 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 $\kappa = 128$, we fix the primitive polynomial

$$P(x) = x^{128} + x^7 + x^2 + x + 1.$$

2.2 (Ideal) Tweakable Blockcipher

The notion of tweakable blockciphers was first formalized by Liskov et al. [14]. Additional to a plaintext X and a key K, it takes a third input - a tweak T, which is generally public. An ideal tweakable blockcipher provides an independent permutation for each new key and tweak pair (K, T). More formally,

$$\widetilde{\mathscr{E}} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n.$$

Here, $\mathcal{K}$ and $\mathcal{T}$ are called the keyspace and the tweak space respectively. To ease the notation, for a fix K and T we denote $\widetilde{\mathscr{E}}(K, T, \cdot)$ by $\widetilde{\mathscr{E}}_K^T(\cdot)$. Hence, according to the definition $\widetilde{\mathscr{E}}_K^T$ is a permutation (for K and T.

2.3 Authenticated Encryption in the Ideal Cipher Model

Authenticated encryption (AE) is a cryptographic scheme that provides both privacy of the message and authenticity of both the message, the nonce, and the associated data. It takes as input, a plaintext $M \in \{0,1\}^*$, a nonce $N \in \{0,1\}^n$ (typically one block data) and an associated data $A \in \{0,1\}^*$, such that the encryption function of AE, $\mathcal{E}_K$, outputs a ciphertext-tag pair (C, T) such that $|C| = |M|$ and $|T| = t$ (t is called the tag length). Throughout the paper, we assume that t is fixed and $n = t$. There is a corresponding decryption function, $\mathcal{D}_K$, that takes (N, A, C, T) as the inputs and outputs the corresponding ciphertext M if (N, A, C, T) is successfully verified, otherwise $\mathcal{D}_K$ rejects the (N, A, C, T) tuple denoted by the symbol $\perp$.

Privacy in the Ideal Cipher Model. Given an adversary $\mathcal{A}$, we define the *privacy-advantage* of $\mathcal{A}$ against $\mathcal{AE}$ in the ideal cipher model as

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathcal{AE}[\widetilde{\mathscr{E}}]}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{AE}_K, \widetilde{\mathscr{E}}^{\pm}} = 1] - \Pr[\mathcal{A}^{\$, \widetilde{\mathscr{E}}^{\pm}} = 1]|,$$

where \$ returns a random string of the same length as the output length of $\mathcal{AE}_K$. where the maximum is taken over all adversaries running in time t and making q_e many queries to the encryption oracle with an aggregate of σ_e blocks and and q_p many the primitive queries.

INT-CTXT Security in the Ideal Cipher Model. We say that an adversary $\mathcal{A}$ *forges* an AE scheme $(\mathcal{AE}, \mathcal{AD})$ in the INT-CTXT security settings in the ideal cipher model if $\mathcal{A}$ is able to compute a tuple (N, A, C, T) satisfying $\mathcal{AD}_K(N, A, C, T) \neq \perp$, without querying (N, A, M) for some M to $\mathcal{AE}_K$ and receiving (C, T), i.e., (N, A, C, T) is a non-trivial forgery. The *forging advantage* for an adversary $\mathcal{A}$ is written as

$$\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathcal{AE}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{AE}_K, \mathcal{AD}_K, , \widetilde{\mathscr{E}}^{\pm}} \text{ forges}],$$

and the maximum forging advantage for all adversaries running in time t, making q_e encryption queries with an aggregate of σ_e blocks, q_p ideal cipher oracle queries and q_v forgery attempts with an aggregate of σ_v blocks is denoted by

$$\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathcal{AE}}((q_e, q_v, q_p), (\sigma_e, \sigma_v), t) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathcal{AE}}(\mathcal{A}).$$

2.4 Coefficients-H Technique

We briefly describe the Coefficients-H technique proposed by Patarin [20]. This technique is used to find the upper bound of the statistical distance between the outputs of two interactive systems. This is traditionally used to prove the information theoretic pseudo randomness of constructions. Here, we assume a computationally unbounded adversary (hence deterministic) $\mathcal{A}$ that interacts with either the real oracle, i.e., the construction of our interest, or the ideal oracle which is usually considered to be a uniform random function or permutation. The tple of all the interactive queries and responses that $\mathcal{A}$ made and received to and from the oracle, is called a *transcript* of $\mathcal{A}$, which is typically denoted by ω. We often let the oracle release additional information $\mathcal{A}$ only if $\mathcal{A}$ is done with all its queries and replies but before it outputs its decision bit.

Let $\mathsf{\Lambda}_1$ and $\mathsf{\Lambda}_0$ denote the probability distributions of the transcript ω induced by the real oracle and the ideal oracle respectively. The probability of realizing a transcript ω in the ideal oracle (i.e., $\Pr[\mathsf{\Lambda}_0 = \omega]$) is called the *ideal interpolation probability.* Similarly, one can define the *real interpolation probability.* A transcript ω is said to be *attainable* with respect to $\mathcal{A}$ if the ideal interpolation probability is non-zero (i.e., $\Pr[\mathsf{\Lambda}_1 = \omega] > 0$). We denote the set of all attainable transcripts by Ω. Following these notations, we state the main lemma of H-Coefficient Technique as follows:

Lemma 1. *Suppose we have a set of transcripts, $\Omega_{\mathsf{bad}} \subseteq \Omega$, which we call* bad *transcripts, and the following conditions hold:*

1. *The probability of getting a transcript in Ω_{bad} the ideal oracle $\mathcal{O}_0$ is at most ϵ_1,*
2. *For any transcript $\omega \in \Omega \setminus \Omega_{\mathsf{bad}}$, we have $\Pr[\Lambda_1 = \omega)] \geq (1-\epsilon_2) \cdot \Pr[\Lambda_0 = \omega]$.*

Then, we have

$$|\Pr[\mathcal{A}^{\mathcal{O}_0} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1} = 1]| \leq \epsilon_1 + \epsilon_2. \tag{1}$$

Proof of this lemma can be found in [22].

3 Specification

We propose a short-tweak tweakable block cipher based authenticated encryption algorithm Light-OCB instantiated with the underlying tweakable block cipher TweGIFT-64. The instantiation is denoted by Light-OCB [TweGIFT-64].

3.1 Light-OCB Mode

During the encryption phase, Light-OCB mode receives an encryption key $K \in \{0,1\}^\kappa$, an associated data $A \in \{0,1\}^*$, a nonce $N \in \{0,1\}^\kappa$, and a message $M \in \{0,1\}^*$ as inputs and generates a ciphertext $C \in \{0,1\}^{|M|}$, and a tag $T \in \{0,1\}^n$ pair. The corresponding decryption function receives a key $K \in \{0,1\}^\kappa$, an associated data $A \in \{0,1\}^*$, a nonce $N \in \{0,1\}^\kappa$, a ciphertext $C \in \{0,1\}^*$, and a tag $T \in \{0,1\}^n$ pair as inputs, and returns the corresponding plaintext $M \in \{0,1\}^{|C|}$, if T is matched. Light-OCB is tweakable block cipher based with an underlying primitive $\widetilde{\mathsf{E}}$. The tweaks are very short with 4-bit length, and are mainly used for domain separation. The 4-bit tweaks vary from 0 to 13. Light-OCB can process data with a maximum $2^{64} - 1$ block message and a maximum $2^{64} - 1$ block AD.

Initialization. In the initialization phase, a κ-bit nonce N is added to the κ-bit master secret key K to output a κ-bit nonce-based encryption key denoted by K_N. Next, Δ_N, a nonce dependent masking key is generated by double encrypting a constant 0^n with K and K_N successively with $\widetilde{\mathsf{E}}$.

Associated Data Processing. In this phase, we divide the data into n-bit blocks and the blocks are processed following the hash layer of PMAC [5]. For each of the associated data blocks, we first update the current key value by field multiplying it by 2. Next, we add this block with Δ_N and encrypt it with $\widetilde{\mathsf{E}}$ under the fixed tweak 0010 (also denoted by 2 in integer representation) and K_N. The encrypted output is finally accumulated by adding it to the previous checksum value. For domain separation, if the final block is partial the tweak 0011 is used (also denoted by 3) to process it. Output of this associated data processing phase is denoted as AD checksum. This phase is described in Fig. 1 and Algorithm 1.

Plaintext Processing. In this phase, we divide the message into n-bit blocks and the blocks are processed following OCB's [21] message processing. Each message block is first masked and then encrypted with the tBC. Next, it is again masked to compute the ciphertext block. The Δ_N masking along a query is done following OCB and plaintext checksum is computed by adding all the message blocks. For the last plaintext block, we first apply XEX on the block length (instead of applying it on the last plaintext block) and add the output with the last plaintext block. This ensures a similar process technique of the complete or incomplete last blocks. Note the, we also update the key by multiplying it by 2 before each block processing. It is described in Fig. 2.

Tag Generation. In this phase, tweak 0100 and 0101 (4 and 5 respectively) are used for non final and final blocks respectively. Here, XEX transformation is applied on the sum of the plaintext checksum and AD checksum. Algorithm 1 describes this phase in detail.

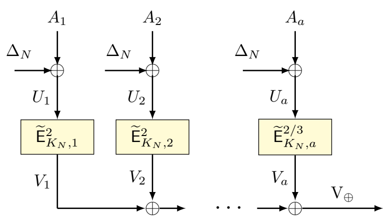
A_1
A_2
A_a
Δ_N
Δ_N
Δ_N
U_1
U_2
U_a
$\widetilde{\mathsf{E}}^2_{K_N,1}$
$\widetilde{\mathsf{E}}^2_{K_N,2}$
$\widetilde{\mathsf{E}}^{2/3}_{K_N,a}$
V_1
V_2
V_a
$V_\oplus$
...

Fig. 1. Associated Data Processing for Light-OCB. Here $\widetilde{\mathsf{E}}^t_{K_N,i}$ denotes invocation of $\widetilde{\mathsf{E}}$ with key $2^{i+1} \odot K_N$ and tweak i. For the final associated data block, the use of $\widetilde{\mathsf{E}}^{2/3}_{K_N,a}$ indicates invocation of $\widetilde{\mathsf{E}}$ with key $2^a \odot K_N$ and tweak 2 or 3 depending on whether the final block is full or partial.

3.2 Features

Here we discuss the salient features of our proposal and possible applications:

1. **High Security**: Light-OCB provides beyond the birthday bound security as it uses nonce-based encryption key and masking key. Actually, Light-OCB provides the optimal security with the level $DT = O(2^{n+\kappa})$, such that D and T are the data and time complexity, respectively. We assume, $D < 2^n$, and $T < 2^\kappa$.
2. **Lightweight**: Light-OCB satisfies all the lightweight requirements of the NIST lightweight standardization process. In fact, Light-OCB uses a 64-bit block cipher and optimizes the state size. TweGIFT-64 can perfectly fit with Light-OCB.

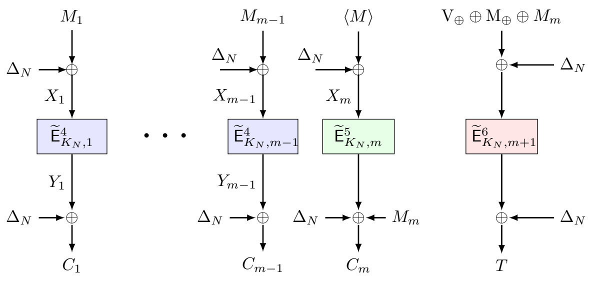
M_1
Δ_N
X_1
$\widetilde{\mathsf{E}}^4_{K_N,1}$
Y_1
Δ_N
C_1
M_{m-1}
Δ_N
X_{m-1}
$\widetilde{\mathsf{E}}^4_{K_N,m-1}$
Y_{m-1}
Δ_N
C_{m-1}
$\langle M \rangle$
Δ_N
X_m
$\widetilde{\mathsf{E}}^5_{K_N,m}$
Δ_N
M_m
C_m
$V_\oplus \oplus M_\oplus \oplus M_m$
Δ_N
$\widetilde{\mathsf{E}}^6_{K_N,m+1}$
Δ_N
T

Fig. 2. Processing of an m block message M and tag generation for Light-OCB. $\langle \text{len} \rangle_n$ denotes the n bit representation of the size of the final block in bits. $M_\oplus$ denotes the plaintext checksum value and $V_\oplus$ denotes the AD checksum value. $\widetilde{\mathsf{E}}^t_{K_N,i}$ is defined in a similar manner as in Fig. 1.

3. **Parallel**: Light-OCB has a parallel structure and hence the implementation of Light-OCB can be fully pipelined. This can help to achieve high throughput.
4. **Single Pass**: Light-OCB makes only one pass through the data while maintaining both confidentiality and authenticity. This, in turn reduces the computational cost by a factor of two as compared to two-pass schemes.
5. **Rate-**1: The *rate* of an AEAD is defined as the number of blocks of the message (plaintext) processed per non-linear (block-cipher, field multiplication, etc.) operation. Constructions with higher rates have shorter latency and achieve high speed. The maximum rate that a secure AEAD construction can achieve is 1, and our mode Light-OCB achieves the rate. This signifies extremely high speed and low latency which is ideal for high-speed applications.
6. **Optimal:** An authenticated encryption scheme is called *optimal* if the number of non-linear operations it uses is the minimum possible. For nonce based AEAD, the minimum number of non-linear operations required to process a data with a block associated data and m block plaintext is $(a + m + 1)$ [11]. Light-OCB is an optimal construction, and hence it performs excellent especially for short messages.
7. **Versatility**: As mentioned already, the parallelizability and small state size together makes the design extremely versatile.

3.3 Recommended Instantiation

We instantiate Light-OCB with the short-tweak tweakable block cipher TweGIFT-64. Here, the key size is 128 bits, nonce size is 128 bits, and tag size is 64 bits. Here we briefly describe the tBC TweGIFT-64 or more formally TweGIFT-64/4/128 [9]. It is a 64-bit tweakable block cipher with 4-bit tweak and 128-bit

Fig. 3. The encryption and verification-decryption algorithms of Light-OCB.

```
1: function Light-OCB_$\widetilde{\mathsf{E}}$.Enc$(K, N, A, M)$
2:     $C \leftarrow \perp,\ M_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
3:     $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
4:     if $|A| \neq 0$ then
5:         $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
6:     if $|M| \neq 0$ then
7:         $(K_N, M_\oplus, C) \leftarrow \mathsf{proc\_pt}(K_N, \Delta_N, M)$
8:     $T \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, M_\oplus)$
9:     return $(C, T)$

10: function init$(K, N)$
11:     $Y \leftarrow \widetilde{\mathsf{E}}^0_K(0^n)$
12:     $K_N \leftarrow K \oplus N$
13:     $\Delta_N \leftarrow \widetilde{\mathsf{E}}^1_{K_N}(Y)$
14:     return $(K_N, \Delta_N)$

15: function proc_ad$(K_N, \Delta_N, A)$
16:     $L \leftarrow K_N$
17:     $(A_{a-1}, \ldots, A_0) \xleftarrow{n} A$
18:     for $i = 0$ to $a - 2$ do
19:         $U \leftarrow A_i \oplus \Delta_N$
20:         $L \leftarrow L \odot 2$
21:         $V \leftarrow \widetilde{\mathsf{E}}^2_L(U)$
22:         $V_\oplus \leftarrow V_\oplus \oplus V$
23:     $U \leftarrow \mathsf{ozs}(A_{a-1}) \oplus \Delta_N$
24:     $L \leftarrow L \odot 2$
25:     $V \leftarrow (|A_{a-1}| = n)?\ \widetilde{\mathsf{E}}^2_L(U) : \widetilde{\mathsf{E}}^3_L(U)$
26:     $V_\oplus \leftarrow V_\oplus \oplus V$
27:     return $(L, V_\oplus)$

28: function proc_pt$(K_N, \Delta_N, M)$
29:     $L \leftarrow K_N$
30:     $(M_{m-1}, \ldots, M_0) \xleftarrow{n} M$
31:     for $j = 0$ to $m - 2$ do
32:         $M_\oplus \leftarrow M_\oplus \oplus M_j$
33:         $X \leftarrow M_j \oplus \Delta_N$
34:         $L \leftarrow L \odot 2$
35:         $Y \leftarrow \widetilde{\mathsf{E}}^4_L(X)$
36:         $C_j \leftarrow W \oplus \Delta_N$
37:     $L \leftarrow L \odot 2$
38:     $X \leftarrow \langle |M_{m-1}| \rangle_n \oplus \Delta_N$
39:     $Y \leftarrow \widetilde{\mathsf{E}}^5_L(X)$
40:     $C_{m-1} \leftarrow \mathsf{chop}(Y \oplus \Delta_N, |M_{m-1}|) \oplus M_{m-1}$
41:     $M_\oplus \leftarrow M_\oplus \oplus M_{m-1}$
42:     $C \leftarrow (C_{m-1}, \ldots, C_0)$
43:     return $(L, M_\oplus, C)$
```

```
1: function Light-OCB_$\widetilde{\mathsf{E}}$.Dec$(K, N, A, C, T)$
2:     $M \leftarrow \perp,\ M_\oplus \leftarrow 0,\ V_\oplus \leftarrow 0$
3:     $(K_N, \Delta_N) \leftarrow \mathsf{init}(K, N)$
4:     if $|A| \neq 0$ then
5:         $(K_N, V_\oplus) \leftarrow \mathsf{proc\_ad}(K_N, \Delta_N, A)$
6:     if $|C| \neq 0$ then
7:         $(K_N, M_\oplus, M) \leftarrow \mathsf{proc\_ct}(K_N, \Delta_N, C)$
8:     $T' \leftarrow \mathsf{proc\_tg}(K_N, \Delta_N, V_\oplus, M_\oplus)$
9:     if $T' = T$ then
10:         return $M$
11:     else
12:         return $\perp$

13: function proc_ct$(K_N, \Delta_N, A, C, T)$
14:     $L \leftarrow K_N$
15:     $(C_{m-1}, \ldots, C_0) \xleftarrow{n} C$
16:     for $j = 0$ to $m - 2$ do
17:         $Y \leftarrow C_j \oplus \Delta_N$
18:         $L \leftarrow L \odot 2$
19:         $X \leftarrow \widetilde{\mathsf{D}}^4_L(Y)$
20:         $M_j \leftarrow X \oplus \Delta_N$
21:         $M_\oplus \leftarrow M_\oplus \oplus M_j$
22:     $L \leftarrow L \odot 2$
23:     $X \leftarrow \langle |C_{m-1}| \rangle_n \oplus \Delta_N$
24:     $Y \leftarrow \widetilde{\mathsf{E}}^5_L(W)$
25:     $M_{m-1} \leftarrow \mathsf{chop}(Y \oplus \Delta_N, |C_{m-1}|) \oplus C_{m-1}$
26:     $M_\oplus \leftarrow M_\oplus \oplus M_{m-1}$
27:     $M \leftarrow (M_{m-1}, \ldots, M_0)$
28:     return $(L, M_\oplus, M)$

29: function proc_tg$(K_N, \Delta_N, V_\oplus, M_\oplus)$
30:     $L \leftarrow K_N \odot 2$
31:     $T \leftarrow \widetilde{\mathsf{E}}^6_L(V_\oplus \oplus M_\oplus \oplus \Delta_N) \oplus \Delta_N$
32:     return $T$
```

key. As the name suggests, it is a tweakable variant of GIFT-64-128 [3] block cipher. TweGIFT-64 is composed of 28 rounds and each round consists following operations:

SubCells: TweGIFT-64 uses a 4-bit S-box as GIFT-64-128 and paralelly applies it to each nibble of the cipher state. Table 1 below defines the S-box.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$GS(x)$	1	A	4	C	6	F	3	9	2	D	B	7	5	0	8	E

Table 1. The GIFT S-Box GS. Each value is a hexadecimal number.

PermBits: PermBits also uses the same permutation as used in GIFT-64-128. It maps the i^{th} of the cipher state to the $GP(i)^{th}$ bit, where

$$GP(i) = 4\lfloor i/16 \rfloor + 16\Big(\big(3\lfloor (i \bmod 16)/4 \rfloor + (i \bmod 4)\big) \bmod 4\Big) + (i \bmod 4).$$

AddRoundKey: Here a 32-bit round key is extracted from the master key and added to the cipher state. This also follows the same as used in GIFT.

AddRoundConstant: A single bit "1" and the bits of a 6-bit round constant are added to the cipher state at the 63^{rd}, 23^{rd}, 19^{th}, 15^{th}, 11^{th}, 7^{th} and 3^{rd}-bit respectively. The 6-bit constants are generated using a 6-bit affine LFSR (same as that of SKINNY [4] and GIFT-64-128 [3]).

AddTweak: The 4-bit tweak is first expanded to a 16-bit expanded tweak by the linear code Exp and then XOR this expanded tweak to the state at an interval of 4 rounds (starting from the 4^{th} round) at bit positions $4i+3$, for $i = 0, ..., 15$. Exp takes as input a 4-bit tweak $t = t_1\|t_2\|t_3\|t_4$ and outputs a 16-bit expanded tweak $t_e = t\|t'\|t\|t'$, such that $t' = s \oplus t_1\|s \oplus t_2\|s \oplus t_3\|s \oplus t_4$ and $s = t_1 \oplus t_2 \oplus t_3 \oplus t_4$.

3.4 Design Rationale

In this section, we briefly describe the various design choices and rationale for our proposals.

Choice of the Mode Light-OCB . We mainly target to design a lightweight AEAD that should be efficient as well as provides high-performance capability. The AEAD can be efficent by incorporating one-pass data process. It can be parallelizable to provide high-performance capability.

We take the approach to start with the well-known mode OCB. OCB is on-line, one-pass as well as fully parallelizable. OCB also provides birthday bound security, and thus we need 128-bit blocks that satisfy the NIST criteria. This in turn, increases the state size while keeping the main features intact.

The associated data is processed following the hash layer of PMAC, and the computation is fully parallel in order to maximize the performance in parallel computing environments.

Light-OCB mainly updates the use of nonce and position dependent keys. OCB achieves only the birthday bound security level. This is due to the fact that collision probability at any two distinct block cipher calls (as two calls share the same encrytion key). Light-OCB overcomes this by changing the key and tweak tuple for each block cipher invocation. Hence, even if there is a collision the security remains intact as the key, tweak tuples are distinct. Actually, Light-OCB achieve full security up to a data complexity 2^n, and time complexity 2^κ, and combined data-time complexity up to $2^{n+\kappa}$ (see the security analysis in Sect. 4). This helps us to design a secure aead using a ultra lightweight block cipher TweGIFT-64.

Choice of short Tweakable Block Cipher: TweGIFT-64 . We choose a twekable block cipher TweGIFT-64, that can handle short tweaks. This is essential for instantiating our mode Light-OCB. There are several efficient tweakable ciphers like SKINNY [4] but they are designed to handle general purpose tweaks and are optimized for handling short tweaks. On the other hand, TweGIFT-64 is designed over GIFT-64-128 to handle such short tweaks.

Tweak expansion is done using a simple (only 7 XORs are needed) high distance linear code (distance 4) to convert a 4-bit tweak value into a 16-bit codeword. This high distance code ensures strong differential characteristics for TweGIFT-64.

The expanded codeword is XORed to the block cipher state to the third bit of each nibble. The choice of this position has been made due to the fact that the other three positions are already masked by the round key and round constant bits. In addition, tweak addition after 4 rounds ensures low differential probability for TweGIFT-64.

3.5 Light-OCB vs LOCUS-AEAD

In this section, we briefly discuss how Light-OCB differs from LOCUS-AEAD. We first discuss on the difference between the two modes from specification point of view, and then demonstrate the advantages of Light-OCB over LOCUS-AEAD.

Structurally, both the modes are very similar in the sense that they both are parallel authenticated encryption schemes following the XEX paradigm, both use nonce based derived key, and short tweak tweakable block cipher. However, the modes have the following subtle differences:

1. While processing the message, Light-OCB uses one primitive call per message block, while LOCUS-AEAD requires two primitive calls per message block.
2. Light-OCB computes plain text (or message) checksum for generating the tag, while LOCUS-AEAD uses an intermediate checksum to generate the tag.

Due to the above two structural modification, Light-OCB achieves the following advantages over LOCUS-AEAD:

High Throughput: In LOCUS-AEAD, processing each message block requires 2 block-cipher invocations. On the contrary, Light-OCB requires only 1 block-cipher invocation per message. Hence, to process a message block of 64-bits Light-OCB requires only 1 clock-cycle as compared to 2 clock cycles required for LOCUS-AEAD. This gives a double speed-up, and improves the overall throughput by a factor of 2.

Efficient Short-Message Processing: Another notable advantage of Light-OCB as compared to LOCUS-AEAD is efficiency in short message processing. Light-OCB requires only $(a + m + 1)$ primitive invocations to process a message of m blocks with a block associated data, and this is the optimal number of non-linear invocations for any nonce based authenticated encryption scheme. The optimality ensures that the construction achieves extremely high throughput even for very short messages.

We would like to point out that these advantages are obtained at the cost of RUP security. However, we emphasize the fact that the RUP security is only required in some unconventional settings, and practical applications where RUP setting is necessary are limited. On the other hand, these modification allows the design to boost the speed and throughput up to a factor of 2, which is critical, specially for high-speed applications such as memory encryption, and vehicular security applications.

4 Security Analysis of Light-OCB

4.1 Privacy Security of Light-OCB

Theorem 1. *Let $\mathcal{A}$ be a non-trivial nonce-respecting adversary against* Light-OCB$[\widetilde{\mathscr{E}}]$ *that makes q_e many encryption queries (with an aggregate of σ_e many blocks) to the construction and q_p many queries to the primitive. The privacy advantage of $\mathcal{A}$ in the ideal cipher model satisfies*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]}(\mathcal{A}) \leq \frac{q_p}{2^k} + \frac{4q_pq_e}{2^{n+k}} + \frac{4q_p\sigma_e}{2^{n+k}}.$$

Proof. We employ the coefficient-H technique to prove Theorem 1. In the same spirit, we assume that $\mathcal{A}$ is deterministic. As per convention, $\mathcal{A}$ makes q_e many encryption queries to the construction oracle, with an aggregate of total ν_e many associated data blocks and μ_e many message blocks. We write $\sigma_e = \mu_e + \nu_e$. In addition, $\mathcal{A}$ makes q_f many forward queries and q_b many backward queries to the underlying primitive oracle, i.e the tweakable ideal cipher $\widetilde{\mathscr{E}}$. We write $q_p = q_f + q_b$ to denote the total number of primitive queries. The proof is given in the rest of this subsection.

Oracle Description The two oracles at hand are: $\mathcal{O}_1 := (\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}], \widetilde{\mathscr{E}}^{\pm})$, the real oracle, and $\mathcal{O}_0 := (\$, \widetilde{\mathscr{E}}^{\pm})$, the ideal oracle. We consider a stronger version of these oracles, the one in which they release some additional information.

Description of the real oracle, $\mathcal{O}_1$: The real oracle $\mathcal{O}_1$ has access to $\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]$ and $\widetilde{\mathscr{E}}^{\pm}$. We denote the transcript random variable generated by $\mathcal{A}$'s interaction with $\mathcal{O}_1$ by the usual notation Λ_1, which is a collection of construction and primitive query-response tuples. For $i \in [q_e]$, initially, the i-th construction query-response tuple is of the form $(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i, \mathsf{T}^i)$, where N^i is the i-th nonce, A^i is the i-th associated data consisting of a_i many blocks, M^i is the i-th message consisting of ℓ_i many blocks, C^i is the i-th ciphertext consisting of ℓ_i many blocks and T^i is the i-th tag value. Clearly, $\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}](\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i) = (\mathsf{C}^i, \mathsf{T}^i)$ for all $i \in [q_e]$. For $i \in [q_p]$, the i-th primitive query-response tuple is of the form $(\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)$, where $\hat{\mathsf{K}}^i$: the i-th key, $\hat{\mathsf{T}}^i$: the i-th tweak, $\hat{\mathsf{X}}^i$: the i-th input of $\widetilde{\mathscr{E}}$ and $\hat{\mathsf{Y}}^i$: the i-th output of $\widetilde{\mathscr{E}}$. Clearly, $\widetilde{\mathscr{E}}(\mathsf{K}^i, \mathsf{T}^i, \mathsf{X}^i) = (\mathsf{Y}^i)$ for all $i \in [q_p]$. Once the query-response phase is over $\mathcal{O}_1$ releases the secret key K, and the internal variables, $(\mathsf{U}^i, \mathsf{V}^i, \mathsf{X}^i, \mathsf{W}^i, \mathsf{Y}^i, \mathsf{V}^i_{\oplus}, \mathsf{W}^i_{\oplus}, \mathsf{K}^{\mathsf{i}}_{\mathsf{N}}, \Delta^{\mathsf{i}}_{\mathsf{N}})_{i\in[q_e]}$, which are defined analogously as in Fig. 3. Additionally it also releases $\varDelta_0 = \widetilde{\mathscr{E}}(\mathsf{K}, (0,0), 0)$. Finally, we have

$$\Lambda_1 = \left\{(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i, \mathsf{T}^i, \mathsf{U}^i, \mathsf{V}^i, \mathsf{X}^i, \mathsf{Y}^i, \mathsf{V}^i_{\oplus}, \mathsf{M}^i_{\oplus}, \mathsf{K}^{\mathsf{i}}_{\mathsf{N}}, \Delta^{\mathsf{i}}_{\mathsf{N}}, \Delta_0, \mathsf{K})_{i\in[q_e]}, (\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)_{i\in[q_p]}\right\}.$$

Description of the ideal oracle, $\mathcal{O}_0$: The ideal oracle $\mathcal{O}_0$ has access to \$ and $\widetilde{\mathscr{E}}^{\pm}$. The ideal transcript random variable Λ_0, is also a collection of construction and primitive query-response tuples. For $i \in [q_e]$, initially, the i-th construction query-response tuple is of the form $(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i, \mathsf{T}^i)$, where N^i is the i-th nonce, A^i is the i-th associated data consisting of a_i many blocks, M^i is the i-th message consisting of ℓ_i many blocks, $\mathsf{C}^i \leftarrow_{\$} \{0,1\}^{n\ell_i}$ is the i-th ciphertext consisting of ℓ_i many blocks and $\mathsf{T}^i \leftarrow_{\$} \{0,1\}^n$ is the i-th tag value. For $i \in [q_p]$, the i-th primitive query-response tuple is of the form $(\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)$, defined analogously as in the real world. Once the query-response phase is over $\mathcal{O}_0$ defines the internal variables in the following order

1. $\mathsf{K} \leftarrow_{\$} \{0,1\}^k$ and $\Delta_0 \leftarrow_{\$} \{0,1\}^n$.
2. $\mathsf{K}^{\mathsf{i}}_{\mathsf{N}} = \mathsf{K} \oplus \mathsf{N}^i$.
3. $\Delta^{\mathsf{i}}_{\mathsf{N}} \leftarrow_{\$} \{0,1\}^n$.
4. $\forall i \in [q_e]$, $j \in [a_i]$, $\mathsf{U}^i_j = \mathsf{A}^i_j \oplus \Delta^{\mathsf{i}}_{\mathsf{N}}$, and $\mathsf{V}^i_j \leftarrow_{\$} \{0,1\}^n$.
5. $\forall i \in [q_e]$, $j \in [\ell_i - 1]$, $\mathsf{X}^i_j = \mathsf{M}^i_j \oplus \Delta^{\mathsf{i}}_{\mathsf{N}}$, $\mathsf{Y}^i_j = \mathsf{C}^i_j \oplus \Delta^{\mathsf{i}}_{\mathsf{N}}$.
6. $\forall i \in [q_e]$, $\mathsf{X}^i_{\ell_i} = \langle \ell_i \rangle \oplus \Delta^{\mathsf{i}}_{\mathsf{N}}$, $\mathsf{Y}^i_{\ell_i} = \mathsf{C}^i_j \oplus \Delta^{\mathsf{i}}_{\mathsf{N}} \oplus \mathsf{M}^i_{\ell_i}$.
7. $\forall i \in [q_e]$, $\mathsf{V}_{\oplus} = \bigoplus_{j\in[a_i]} \mathsf{V}^i_j$, $\mathsf{M}_{\oplus} = \bigoplus_{j\in[\ell_i]} \mathsf{M}^i_j$, and $\mathsf{CS}^i = \mathsf{M}^i_{\oplus} \oplus \mathsf{V}^i_{\oplus}$.

At this point, we have the complete ideal transcript random variable, i.e.,

$$\Lambda_0 = \left\{(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i, \mathsf{T}^i, \mathsf{U}^i, \mathsf{V}^i, \mathsf{X}^i, \mathsf{Y}^i, \mathsf{V}^i_{\oplus}, \mathsf{M}^i_{\oplus}, \mathsf{K}^{\mathsf{i}}_{\mathsf{N}}, \Delta^{\mathsf{i}}_{\mathsf{N}}, \Delta_0, \mathsf{K})_{i\in[q_e]}, (\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)_{i\in[q_p]}\right\}.$$

Note that, for brevity we used identical notations to describe the real and ideal random variables. Since we never consider the joint probability of Θ_1 and Θ_0, this abuse of notation should not cause any confusion.

Bad Transcripts: Definition and Analysis Let Ω be the set of all attainable transcripts. We say that a transcript

$$\omega = \left\{ (N^i, A^i, M^i, C^i, T^i, U^i, V^i, X^i, Y^i, V_\oplus^i, M_\oplus^i, K_N^i, \Delta_N^i, \Delta_0, K)_{i \in [q_e]}, (\hat{K}^i, \hat{T}^i, \hat{X}^i, \hat{Y}^i)_{i \in [q_p]} \right\},$$

is bad, denoted as $\omega \in \Omega_{\mathtt{bad}}$, if one of the following three cases occurs.

Case 1: Key-guessing primitive query: We say that a primitive query-response tuple is key guessing if the following condition is true:

$$\exists\ i \in [q_p],\ \text{such that } \hat{K}^i = K.$$

Case 2: Inconsistent primitive-construction queries: We say that the primitive and construction query-response tuple is inconsistent if one of the following conditions is true for some $i \in [q_e]$ and $i' \in [q_p]$:

- $\mathtt{P}_1$: $\exists\ j \in [a_i]$, such that $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{X}^{i'}) = (K_N^i, (0, j), U_j^i)$.
- $\mathtt{P}_2$: $\exists\ j \in [\ell_i]$, such that $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{X}^{i'}) = (K_N^i, (1, j), X_j^i)$.
- $\mathtt{P}_3$: $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{X}^{i'}) = (K_N^i, (0, 0), CS^i)$.
- $\mathtt{P}_4$: $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{Y}^{i'}) = (K_N^i, (1, 0), \Delta_0)$.
- $\mathtt{P}_5$: $\exists\ j \in [a_i]$, such that $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{Y}^{i'}) = (K_N^i, (0, j), V_j^i)$.
- $\mathtt{P}_6$: $\exists\ j \in [\ell_i]$, such that $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{Y}^{i'}) = (K_N^i, (1, j), Y_j^i)$.
- $\mathtt{P}_7$: $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{Y}^{i'}) = (K_N^i, (0, 0), T^i \oplus \Delta_N^i)$.
- $\mathtt{P}_8$: $(\hat{K}^{i'}, \hat{T}^{i'}, \hat{Y}^{i'}) = (K_N^i, (1, 0), \Delta_N^i)$.

Let $\mathtt{B}_1$, and $\mathtt{B}_2$ denote the event that cases 1, and 2, respectively, are satisfied for Λ_0. Therefore, the probability that $\Lambda_0 \in \Omega_{\mathtt{bad}}$, is given by

$$\Pr[\Lambda_0 \in \Omega_{\mathtt{bad}}] = \Pr[\mathtt{B}_1 \vee \mathtt{B}_2] \leq \Pr[\mathtt{B}_1] + \Pr[\mathtt{B}_2] \tag{2}$$

Upper bound on $\Pr[\mathtt{B}_1]$: For a fixed i, $\hat{\mathsf{K}}^i = \mathsf{K}$, happens with at most 2^{-k} probability (K is uniform and independent of $\hat{\mathsf{K}}^i$). There are at most q_p many choices for i. So, we get $\Pr[\mathtt{B}_1] \leq q_p 2^{-k}$.

Upper bound on $\Pr[\mathtt{B}_2]$: By the definition of $\mathtt{B}_2$, we have

$$\Pr[\mathtt{B}_2] \leq \sum_{i=1}^{8} \Pr[\mathtt{P}_i] \leq \frac{4 q_p q_e}{2^{n+k}} + \frac{4 q_p \sigma_e}{2^{n+k}} \tag{3}$$

The proof of Eq. (3) is given in Appendix A.

$$\Pr[\Lambda_0 \in \Omega_{\mathtt{bad}}] \leq \frac{q_p}{2^k} + \frac{4 q_p q_e}{2^{n+k}} + \frac{4 q_p \sigma_e}{2^{n+k}}. \tag{4}$$

Good Transcript Analysis Let us fix a transcript $\omega \in \Omega \setminus \Omega_{\mathtt{bad}}$, where ω has the usual form. In Eq. (5), we claim that the ratio of real to ideal world interpolation probabilities is at least 1. The proof of this claim is available in Appendix B.

$$\frac{\Pr[\Lambda_1 = \omega]}{\Pr[\Lambda_0 = \omega]} \geq 1. \tag{5}$$

Theorem 1 follows by using Eq. (4) and (5) in Eq. (1) of the coefficient-H technique. □

4.2 INT-CTXT Security of $\mathsf{Light\text{-}OCB}$

Theorem 2. *Let $\mathcal{A}$ be a non-trivial nonce-respecting forger against* $\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]$ *that makes q_e and q_d many encryption and decryption, respectively, queries (with an aggregate of σ_e many encryption query blocks) to the construction and q_p many queries to the primitive. The INT-CTXT advantage of $\mathcal{A}$ in the ideal cipher model satisfies*

$$\mathbf{Adv}^{\textit{int-ctxt}}_{\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]}(\mathcal{A}) \leq \frac{2}{2^n} + \frac{q_p}{2^k} + \frac{16q_p}{2^{n+k}} + \frac{4q_pq_e}{2^{n+k}} + \frac{4q_p\sigma_e}{2^{n+k}}.$$

Proof. As per convention, $\mathcal{A}$ that makes at most q_e many encryption queries and at most q_d many RUP queries to $\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]$. In addition $\mathcal{A}$ also makes q_f many forward queries and q_b many backward queries to $\widetilde{\mathscr{E}}$. We write $q_p = q_f + q_b$ to denote the total number of primitive queries. We will reuse the notations used in Sect. 4.1. In particular, the i-th encryption query-response tuple is denoted by $(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i, \mathsf{T}^i)$, and the i-th primitive query-response tuple is denoted by $(\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)$. The intermediate variables are also analogously defined. In addition the i-th decryption query-response tuple is denoted by $(\mathsf{N}'^i, \mathsf{A}'^i, \mathsf{M}'^i, \mathsf{C}'^i)$, where A'^i, and M'^i contains a'_i, and ℓ'_i many blocks, respectively. Note that, $(\mathsf{N}^i, A^i, \mathsf{M}^i, \mathsf{C}^i) \neq (\mathsf{N}'^j, \mathsf{A}'^j, \mathsf{M}'^j, \mathsf{C}'^j)$ for all $i \in [q_e]$ and $j \in [q_d]$.

Finally, $\mathcal{A}$ tries to forge with $(\mathsf{N}^\star, \mathsf{A}^\star, \mathsf{C}^\star, \mathsf{T}^\star) \neq (\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i)$ for $i \in [q_e]$. Let $\mathtt{Forge}$ denotes the event that $\mathcal{A}$ submits valid forgery. We define the transcript random variable corresponding to $\mathcal{A}$'s interaction with $\mathsf{Light\text{-}OCB}[\widetilde{\mathscr{E}}]$ as

$$\Lambda_1 := \left\{(\mathsf{N}^i, \mathsf{A}^i, \mathsf{M}^i, \mathsf{C}^i)_{i\in[q_e]}, (\mathsf{N}'^i, \mathsf{A}'^i, \mathsf{M}'^i, \mathsf{C}'^i)_{i\in[q_d]}, (\hat{\mathsf{K}}^i, \hat{\mathsf{T}}^i, \hat{\mathsf{X}}^i, \hat{\mathsf{Y}}^i)_{i\in[q_p]}, (\mathsf{N}^\star, \mathsf{A}^\star, \mathsf{C}^\star, \mathsf{T}^\star)\right\}.$$

Let $\mathsf{multi}(x)$ denote the number of $i \in [q_p]$ such that $\hat{\mathsf{K}}^i = x$. Let $\mathtt{Bad}$ denote the event that Λ_1 satisfies one of the following properties:

- $\mathtt{G}_1$: $\exists i \in [q_p]$, such that $\hat{\mathsf{K}}^i = \mathsf{K}$.
- $\mathtt{G}_2$: $\exists i \in [q_p]$, such that $(\hat{\mathsf{K}}^i, \hat{\mathsf{X}}^i) = (\mathsf{K}^\star_\mathsf{N}, \widetilde{\mathscr{E}}_\mathsf{K}(0))$.
- $\mathtt{G}_3$: $\mathsf{multi}(\mathsf{K}^\star_\mathsf{N}) \geq 2^{n-1}$.
- $\mathtt{G}_4$: $\exists i \in [q_p]$, such that $(\hat{\mathsf{K}}^i, \hat{\mathsf{Y}}^i) = (\mathsf{K}^\star_\mathsf{N}, \mathsf{T}^\star \oplus \Delta^\star_\mathsf{N})$.
- $\mathtt{G}_5$: $\exists i \in [q_p]$, such that $(\hat{\mathsf{K}}^i, \hat{\mathsf{X}}^i) = (\mathsf{K}^\star_\mathsf{N}, \mathsf{U}^\star_{a_\star})$.
- $\mathtt{G}_6$: $\exists i \in [q_p]$, such that $(\hat{\mathsf{K}}^i, \hat{\mathsf{Y}}^i) = (\mathsf{K}^\star_\mathsf{N}, \mathsf{Y}^\star_{\ell_\star})$.
- $\mathtt{G}_7$: $\exists i \in [q_e], i' \in [q_p]$, such that $(\hat{\mathsf{K}}^{i'}, \hat{\mathsf{X}}^{i'}) = (\mathsf{K}^\mathsf{i}_\mathsf{N}, \widetilde{\mathscr{E}}_\mathsf{K}(0))$.
- $\mathtt{G}_8$: $\exists i \in [q_e]$, such that $\mathsf{multi}(\mathsf{K}^i) \geq 2^{n-1}$.
- $\mathtt{G}_9$: $\exists i \in [q_e], j \in [a_i], i' \in [q_p]$, such that $(\hat{\mathsf{K}}^{i'}, \hat{\mathsf{X}}^{i'}) = (\mathsf{K}^\mathsf{i}_\mathsf{N}, \mathsf{U}^i_j)$.
- $\mathtt{G}_{10}$: $\exists i \in [q_e], j \in [\ell_i], i' \in [q_p]$, such that $(\hat{\mathsf{K}}^{i'}, \hat{\mathsf{Y}}^{i'}) = (\mathsf{K}^\mathsf{i}_\mathsf{N}, \mathsf{Y}^i_j)$.

Observe that if $\mathtt{Bad}$ is satisfied, then $\mathcal{A}$ can forge with very high probability. On the other hand, we will show that given that $\mathtt{Bad}$ is not satisfied, $\mathcal{A}$ cannot succeed with significant probability. Formally, we have

$$\Pr[\mathtt{Forge}] \leq \Pr[\mathtt{Bad}] + \Pr[\mathtt{Forge}|\neg\mathtt{Bad}]. \tag{6}$$

We make two claims, given as follows

$$\Pr[\texttt{Bad}] \leq \frac{q_p}{2^k} + \frac{16q_p}{2^{n+k}} + \frac{4q_p q_e}{2^{n+k}} + \frac{4q_p \sigma_e}{2^{n+k}} \tag{7}$$

$$\Pr[\texttt{Forge}|\neg\texttt{Bad}] \leq \frac{2}{2^n}. \tag{8}$$

The proof for claims in Eq. (7) and (8) are given in Appendix C and D, respectively. The result follows from Eq. (6), (7), and (8). □

5 Hardware Implementation

In this section, we describe a lightweight implementation of Light-OCB. Light-OCB is structurally simple with tweakable blockcipher and a few XORs. In this section we provide hardware implementation details of Light-OCB instantiated with the TweGIFT64 blockcipher.

5.1 Clock Cycle Analysis

We provide a conventional way for speed estimation, i.e, the number of clock cycles per byte (*cpb*). This is a theoretical way to estimate the speed of the architecture. We consider round-based architecture with 64 bit datapath. To process a data block of $d = a + m$ blocks (a is the number of associated data blocks and m is the number of message blocks), we need $29d$ clock cycles. We use one TweGIFT64 call to process one data block. Our block cipher is optimized to process a bulk data, and the reset is required only to indicate that the stream processing starts. We observe that the *cpb* values for different sized data are constant as there is no initialization overhead and the overhead for the tag generation (constant small number of clock cycles) is negligible for long messages. Our design accept 64-bit or 8-byte data blocks and hence the *cpb* is $29d/8d = 3.625$.

5.2 Hardware Architecture

Light-OCB is based on E-t-M paradigm and the message blocks are processed in parallel to generate the ciphertext blocks and the tag. Here, the blockcipher tweak values for the three types of input data (N, A and M) are required to distinguish. Below, we provide brief hardware architecture details. For simplicity, we omit the control unit from Fig. 4. The main components in the hardware circuit are as follows.

State Registers. The architecture for Light-OCB contains four registers.

- A 64-bit state register to store the encryption state,
- an 128-bit register to store the blockcipher master secret key,
- a 64-bit register to store the checksum and
- the 64-bit Δ register to store Δ_N.

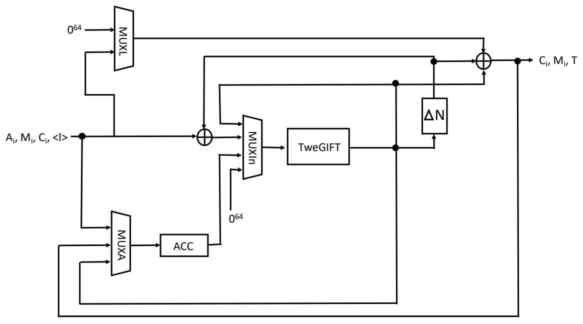

Fig. 4. Hardware Architecture Diagram

Module TweGIFT. The TweGIFT module computes one round of the underlying tweakable blockcipher. This module internally uses a 64-bit register for the blockcipher internal state and an 128-bit register fr the master key. In addition, TweGIFT also uses internally a control unit. We are omitting this for the sake of simplicity.

Accumulator Module. The accumulator module *ACC* is used to compute the checksum of the ECB layer and the last block for computing the tag.

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 Implementation Results

We implement Light-OCB on Virtex 7 (xc7v585tffg1761-3), using VHDL and the VIVADO tool. We use exactly the same implementation for TweGIFT64 as used in the implementation of LOCUS-AEAD [10]. The results are presented in Table 2. The implementation follows the RTL approach and a basic iterative type architecture with 64-bit datapath. The areas are reported in the number of flipflops, LUTs and slices. We also report the Frequency (MHz), Throughput (Gbps), and throughput-area efficiencies. The mapped hardware results are reported in Table 2. For the sake of comparison, we also provide the implementation results for LOCUS-AEAD taken from [10].

5.4 Benchmarking

We benchmark our implemented results using the existing FPGA results on Virtex 7. We provide comparisons with the implementation results of the well

Table 2. FPGA implementation comparison between Light-OCB and LOCUS-AEAD

Design (Platform)	Slice Registers	LUTs	Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/LUT	Mbps/Slice
Light-OCB (Virtex 7)	428	1128	307	400	0.88	0.780	2.866
LOCUS-AEAD (Virtex 7)	430	1154	439	392.20	0.44	0.38	1. 002

Table 3. Comparison of Parallel AEAD on Virtex 7 [2].

Scheme	# LUTs	# Slices	Gbps	Mbps/ LUT	Mbps/ Slice
LIGHT-OCB	1128	307	0.88	0.780	2.866
LOCUS-AEAD [10]	1154	439	0.44	0.38	1.00
LOTUS-AEAD [10]	865	317	0.48	0.55	1.50
CLOC-TWINE [2]	1552	439	0.432	0.278	0.984
SILC-AES [2]	3040	910	4.365	1.436	4.796
SILC-LED [2]	1682	524	0.267	0.159	0.510
SILC-PRESENT [2]	1514	484	0.479	0.316	0.990
JAMBU-SIMON [2]	1200	419	0.368	0.307	0.878
AES-OTR [2]	4263	1204	3.187	0.748	2.647
OCB [2]	4269	1228	3.608	0.845	2.889
AES-COPA [2]	7795	2221	2.770	0.355	1.247
AES-GCM [2]	3478	949	3.837	1.103	4.043
CLOC-AES [2]	3552	1087	3.252	0.478	1.561
ELmD [2]	4490	1306	4.025	0.896	3.082

known designs in Table 3 below. Note that, all the candidates for benchmarking in Table 3 either parallel in structure or can have almost parallel implementation. We did not consider the blockcipher based feedback designs or sponge based feedback designs.

References

1. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In *ASIACRYPT (1)*, volume 8269 of *LNCS*, pages 424–443. Springer, 2013.
2. Authenticated Encryption FPGA Ranking. `https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view`.
3. 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, Proceedings*, pages 321–345, 2017.
4. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016, Proceedings, Part II*, pages 123–153, 2016.
5. John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Advances in Cryptology - EUROCRYPT 2002, Proceedings*, pages 384–397, 2002.

6. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, pages 450–466, 2007.
7. Lilian Bossuet, Nilanjan Datta, Cuauhtemoc Mancillas-López, and Mridul Nandi. Elmd: A pipelineable authenticated encryption and its hardware implementation. *IEEE Trans. Computers*, 65(11):3318–3331, 2016.
8. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`.
9. 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 Cryptology ePrint Archive*, 2019:440, 2019.
10. 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.
11. Avik Chakraborti, Nilanjan Datta, and Mridul Nandi. On the optimality of non-linear computations for symmetric key primitives. *J. Math. Cryptol.*, 12(4):241–259, 2018.
12. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.
13. Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR, 2016. `https://competitions.cr.yp.to/round3/ocbv11.pdf`.
14. Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In *CRYPTO 2002*, pages 31–46, 2002.
15. Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR, 2016. `https://competitions.cr.yp.to/round3/aesotrv31.pdf`.
16. Yusuke Naito. Tweakable blockciphers for efficient authenticated encryptions with beyond the birthday-bound security. *IACR Trans. Symmetric Cryptol.*, 2017(2):1–26, 2017.
17. NIST. Lightweight cryptography. `https://csrc.nist.gov/Projects/Lightweight-Cryptography`.
18. National Centre of Excellence. Light-weight Cipher Design Challenge. `https://www.dsci.in/ncoe-light-weight-cipher-design-challenge-2020/`.
19. OMA-SpecWorks. Lightweight-M2M, 2019. `https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/`.
20. Jacques Patarin. The "Coefficients H" Technique. In *SAC 2008*, pages 328–345, 2008.
21. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
22. Serge Vaudenay. Decorrelation: A Theory for Block Cipher Security. *J. Cryptology*, 16(4):249–286, 2003.

Appendix

A Proof of Eq. (3)

We bound the probabilities of $\mathtt{P}_1$, $\mathtt{P}_3$, and $\mathtt{P}_4$, while the other probabilities can be similarly bounded.

- Upper bound on $\Pr[\mathsf{P}_1]$: Fix $i' \in [q_p]$, $i \in [q_e]$, $j \in [a_i]$. Then, we have a system of two equations
$$\mathsf{K} = \hat{\mathsf{K}}^{i'} \oplus \mathsf{N}^i, \quad \Delta^{\mathsf{i}}_{\mathsf{N}} = \hat{\mathsf{X}}^{i'} \oplus \mathsf{A}^i_j.$$
Here K and $\Delta^{\mathsf{i}}_{\mathsf{N}}$ are uniform and independent of each other as well as $\hat{\mathsf{K}}^{i'}$, N^i, A^i_j, and $\hat{\mathsf{X}}^{i'}$. So conditioning on $\hat{\mathsf{K}}^{i'}, \mathsf{N}^i, \mathsf{A}^i_j, \hat{\mathsf{X}}^{i'}$, the two equations satisfy with probability at most 2^{-n-k}. Since we have at most $q_p\nu_e$ many such i', i, j indices, we get $\Pr[\mathsf{P}_1] \le q_p\nu_e 2^{-n-k}$. Similarly, $\Pr[\mathsf{P}_2] \le q_p\mu_e 2^{-n-k}$, $\Pr[\mathsf{P}_6] \le q_p\mu_e 2^{-n-k}$, $\Pr[\mathsf{P}_7] \le q_p q_e 2^{-n-k}$, and $\Pr[\mathsf{P}_8] \le q_p q_e 2^{-n-k}$.

- Upper bound on $\Pr[\mathsf{P}_3]$: Fix $i' \in [q_p]$, $i \in [q_e]$, $j \in [\ell_i]$. Then, we have the following system of two equations
$$\mathsf{K} = \hat{\mathsf{K}}^{i'} \oplus \mathsf{N}^i, \quad \mathsf{CS}^i = \hat{\mathsf{X}}^{i'}.$$
Following a similar line of argument as in the case of $\mathtt{P}_1$, we can conclude that the two equations satisfy with probability at most 2^{-n-k}. We have at most $q_p\mu_e$ many i', i, j indices. Thus $\Pr[\mathsf{P}_3] \le q_p\mu_e 2^{-n-k}$. Similarly, $\Pr[\mathsf{P}_5] \le q_p\nu_e 2^{-n-k}$.

- Upper bound on $\Pr[\mathsf{P}_4]$: In this case Δ_0 and K are independent and uniform. So for fixed $i' \in [q_p], i \in [q_e]$, the event $\mathtt{P}_4$ holds with probability at most 2^{-n-k}, whereby $\Pr[\mathsf{P}_4] \le q_p q_e 2^{-n-k}$.

On combining all the above bounds Eq. (3) follows.

B Proof of Eq. (5)

Ideal World Interpolation Probability: In the ideal world, we have

$$\begin{aligned}\Pr[\Lambda_0 = \omega] =& \Pr[\forall i \in [q_p],\ \widetilde{\mathscr{E}}^{\hat{T}^i}_{\hat{K}^i}(\hat{X}^i) = \hat{Y}^i] \times \Pr[\forall i \in [q_e],\ \$(N^i, A^i, M^i) = (C^i, T^i)] \\ &\times \Pr[\forall i \in [q_e],\ \mathsf{U}^i = U^i, \mathsf{V}^i = V^i, \mathsf{X}^i = X^i, \mathsf{Y}^i = Y^i, \Delta^{\mathsf{i}}_{\mathsf{N}} = \Delta^i_N] \\ &\times \Pr[\mathsf{K} = K] \cdot \Pr[\Delta_0 = \Delta_0] \\ =& \Pr[\forall i \in [q_p],\ \widetilde{\mathscr{E}}(\hat{K}^i, \hat{T}^i, \hat{X}^i) = \hat{Y}^i] \times \frac{1}{2^k 2^{n(\sigma_e + \mu_e + 2q_e + 1)}} \qquad (9)\end{aligned}$$

First of all note that in ideal world the construction query responses and primitive query responses are independent. This justifies the first equality. The second equality is justified as follows. First, $\mu_e + q_e$ many blocks are sampled uniformly (and independently) from $\{0,1\}^n$, corresponding to the ciphertext and tag for all q_e queries. Independently, $\sigma_e + q_e$ blocks are sampled uniformly from $\{0,1\}^n$, corresponding to V, and Δ_N values. The key is sampled uniformly from $\{0,1\}^k$ and Δ_0 is sampled uniformly from $\{0,1\}^n$. In total, we have $\sigma_e + \mu_e + 2q_e + 1$ many n-bit samplings and 1 k-bit sampling.

Real World Interpolation Probability: For $a \in \{0,1\}^n$, $r \in \{0,1\}$ and $s \leq \ell + 1$, let $\kappa(a,r,s)$ denote the number of $i \in [q_p]$ such that $\hat{K}^i = a$, and $\hat{T}^i = (r,s)$. Then, we have

$$\begin{aligned}\Pr[\Lambda_1 = \omega] =& \Pr[\forall i \in [q_p],\ \widetilde{\mathscr{E}}_{\hat{K}^i}^{\hat{T}^i}(\hat{X}^i) = \hat{Y}^i] \times \frac{1}{2^k} \times \frac{1}{2^n - \kappa(K,0,1)} \times \prod_{i \in [q_e]} \frac{1}{2^n - \kappa(K_N^i,1,0)} \\ & \times \prod_{i \in [q_e], j \in [\ell_i]} \frac{1}{(2^n - \kappa(K_N^i,1,j))(2^n - \kappa(K_N^i,1,j) - 1)} \\ & \times \prod_{i \in [q_e], j \in [a_i]} \frac{1}{2^n - \kappa(K_N^i,0,j+1)} \times \prod_{i \in [q_e]} \frac{1}{2^n - \kappa(K_N^i,0,0)}. \quad (10)\end{aligned}$$

Finally, we get Eq. (5) on dividing Eq. (10) by Eq. (9).

C Proof of Eq. (7)

From the definition of Bad, we have

$$\begin{aligned}\Pr[\mathtt{Bad}] &= \Pr[\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3 \vee \mathtt{G}_4 \vee \mathtt{G}_5 \vee \mathtt{G}_6 \vee \mathtt{G}_7 \vee \mathtt{G}_8 \vee \mathtt{G}_9 \vee \mathtt{G}_{10}] \\ &\leq \Pr[\mathtt{G}_1] + \Pr[\mathtt{G}_2|\neg\mathtt{G}_1] + \Pr[\mathtt{G}_3] + \Pr[\mathtt{G}_4|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] \\ &\quad + \Pr[\mathtt{G}_5|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] + \Pr[\mathtt{G}_6|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] + \Pr[\mathtt{G}_7|\neg\mathtt{G}_1] + \Pr[\mathtt{G}_8] \\ &\quad + \Pr[\mathtt{G}_9|\neg(\mathtt{G}_1 \vee \mathtt{G}_7 \vee \mathtt{G}_8)] + \Pr[\mathtt{G}_{10}|\neg(\mathtt{G}_1 \vee \mathtt{G}_7 \vee \mathtt{G}_8)]\end{aligned}$$

We analyze all the ten terms in the following manner:

- $\Pr[\mathtt{G}_1]$: Since K is uniformly distributed and there are at most q_p many favorable values for K, we have $\Pr[\mathtt{G}_1] \leq q_p 2^{-k}$.

- $\Pr[\mathtt{G}_2|\neg\mathtt{G}_1]$ and $\Pr[\mathtt{G}_7|\neg\mathtt{G}_1]$: In this case, once we condition on $\{\hat{\mathsf{K}}^{i'} : i' \in [q_p]\}$, we get at least $2^k - q_p$ many choices for K. Further for each such choice K, $\widetilde{\mathscr{E}}_K^{(0,1)}(0)$ is uniformly distributed. Thus, we get
$$\Pr[\mathtt{G}_2|\neg\mathtt{G}_1] \leq \frac{q_p}{2^n(2^k - q_p)}.$$
Using a similar argument for each $i \in [q_e]$, we get $\Pr[\mathtt{G}_7|\neg\mathtt{G}_1] \leq \frac{q_p q_e}{2^n(2^k - q_p)}$.

- $\Pr[\mathtt{G}_3]$ and $\Pr[\mathtt{G}_8]$: Let $\hat{\mathcal{K}}$ denote the set of all indices $i \in [q_p]$ such that $\mathsf{multi}(\hat{\mathsf{K}}^i) \geq 2^{n-1}$. Then $|\hat{\mathcal{K}}| \leq q_p/2^{n-1}$. Since K is uniformly distributed, we have
$$\Pr[\mathtt{G}_3] = \Pr[\mathsf{K}_\mathsf{N}^\star \in \hat{\mathcal{K}}] \leq \frac{q_p}{2^{n-1}2^k}.$$
Similarly, $\Pr[\mathtt{G}_8] \leq \frac{q_p q_e}{2^{n-1}2^k}$.

- $\Pr[\mathtt{G}_4|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)]$: Here conditioned on $\mathsf{K}^\star_\mathsf{N} = \hat{\mathsf{K}}^i$, $\Delta^\star_\mathsf{N}$ is almost uniform due to $\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)$. Specifically, we have

$$\begin{aligned}\Pr[\mathtt{G}_4|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_4)] \leq \sum_{i\in[q_p]} &\Pr[\Delta^\star_\mathsf{N} = \mathsf{T}^\star \oplus \hat{\mathsf{Y}}^i|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3) \wedge (\mathsf{K}^\star_\mathsf{N} = \hat{\mathsf{K}}^i)] \\ &\cdot \Pr[\mathsf{K}^\star_\mathsf{N} = \hat{\mathsf{K}}^i|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] \\ \leq &\frac{q_p}{2^{n-1}(2^k - q_p)}.\end{aligned}$$

Using the same argument, we get

$$\Pr[\mathtt{G}_5|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] = \Pr[\mathtt{G}_6|\neg(\mathtt{G}_1 \vee \mathtt{G}_2 \vee \mathtt{G}_3)] \leq \frac{q_p}{2^{n-1}(2^k - q_p)}.$$

Similarly,

$$\Pr[\mathtt{G}_9|\neg(\mathtt{G}_1 \vee \mathtt{G}_7 \vee \mathtt{G}_8)] \leq \frac{q_p \nu_e}{2^{n-1}(2^k - q_p)}, \quad \Pr[\mathtt{G}_{10}|\neg(\mathtt{G}_1 \vee \mathtt{G}_7 \vee \mathtt{G}_8)] \leq \frac{q_p \mu_e}{2^{n-1}(2^k - q_p)}.$$

Combining all the probabilities we obtain the bound of Eq. (7), assuming $q_p \leq 2^{k-1}$.

D Proof of Eq. (8)

At this point, we know that Λ_1 is good, as $\neg\mathtt{Bad}$ holds. Let $\omega \in \Omega$ be a good transcript. Then, we have

$$\Pr[\mathsf{Forge}|\Lambda_1 = \omega] = \Pr[\widetilde{\mathscr{E}}^{(0,0)}_{\mathsf{K}^\star_\mathsf{N}}(\mathsf{CS}^\star) \oplus \Delta^\star_\mathsf{N} = \mathsf{T}^\star \mid \Lambda_1 = \omega]$$

We do the analysis in two disjoint cases, depending upon the *freshness* of the nonce.

Case 1: $\mathsf{N}^\star$ is fresh, i.e., $\forall i \in [q_e],\ \mathsf{N}^\star \neq \mathsf{N}^i$. In this case, we know that $\mathsf{K}^\star_\mathsf{N} \neq \mathsf{K}^i_\mathsf{N}$ for all $i \in [q_e]$ and $\mathsf{multi}(\mathsf{K}^\star_\mathsf{N}) < 2^{n-1}$ (since $\neg\mathtt{G}_3$ holds). Thus using a similar line of argument as in the analysis of $\mathtt{G}_4$, we obtain

$$\Pr[\mathsf{Forge}|\Lambda_1 = \omega] = \Pr[\widetilde{\mathscr{E}}^{0,0}_{\mathsf{K}^\star_\mathsf{N}}(\mathsf{CS}^\star) = \mathsf{T}^\star \oplus \Delta^\star_\mathsf{N} \mid \Lambda_1 = \omega] \leq \frac{2}{2^n}.$$

Case 2: $\mathsf{N}^\star$ is not fresh, i.e., $\exists i \in [q_e],\ \mathsf{N}^\star = \mathsf{N}^i$. If $\mathsf{T}^\star \neq \mathsf{T}^i$, then the forgery succeeds with at most 2^{1-n} probability (since $\neg\mathtt{Bad}$ holds). Suppose $\mathsf{T}^\star = \mathsf{T}^i$. In this case, $(\mathsf{A}^i, \mathsf{C}^i) \neq (\mathsf{A}^\star, \mathsf{C}^\star)$. Without loss of generality we assume that $\mathsf{A}^i \neq \mathsf{A}^\star$, hence there must be an index $j \in \max\{a_i, a_\star\}$ such that $\mathsf{A}^i_j \neq \mathsf{A}^\star_j$. This trivially holds when $a_i \neq a_\star$, as we can choose $j = \max\{a_i, a_\star\}$. If $a_i = a_\star$, j is chosen to be the smallest index such that $\mathsf{A}^i_j \neq \mathsf{A}^\star_j$. Keeping this index j in mind, we have

$$\begin{aligned}\Pr[\mathsf{Forge}|\Lambda_1 = \omega] &= \Pr[\mathsf{V}^i_\oplus \oplus \mathsf{V}^\star_\oplus = \mathsf{M}^i_\oplus \oplus \mathsf{M}^\star_\oplus|\Lambda_1 = \omega] \\ &\leq \frac{2}{2^n},\end{aligned}$$

where the inequality holds by conditioning on all other indices except j and the fact that $\mathsf{multi}(\mathsf{K}^i) < 2^{n-1}$.

The bound $\Pr[\texttt{Forge}|\Lambda_1 = \omega] \leq 2^{1-n}$ holds for any arbitrary good transcript ω. So, the bound in Eq. (8) follows from case 1 and 2.