Practical Forgery Attacks on Limdolen and HERN

Raghvendra Rohit and Guang Gong

Department of Electrical and Computer Engineering, University of Waterloo,
Waterloo, Ontario, N2L 3G1, CANADA.
{rsrohit, ggong}@uwaterloo.ca

Abstract. In this paper, we investigate the security of Limdolen and HERN which are Round 1 submissions of the ongoing NIST Lightweight Cryptography Standardization Project. We show that some non-conservative design choices made by the designers solely to achieve a lightweight design lead to practical forgery attacks. In particular, we create associated data-only, ciphertext-only and associated data and ciphertext forgeries which require a feasible number of forging attempts.

Limdolen employs a tweaked PMAC based construction to offer authenticated encryption functionality. It has two variants, Limdolen-128 and Limdolen-256 with key sizes 128 and 256 bits, respectively. The designers claim 128(256)-bit integrity security for Limdolen-128(256). Our main observation is that it uses a sequence of period 2 consisting of only two distinct secret masks. This structural flaw attributes to a successful forgery (all three types) with probability 1 after observing the output of a single encryption query. While, HERN is a 128-bit authenticated encryption scheme whose high level design is inspired from the CAESAR finalist Acorn. We show a message modification strategy by appending/removing a sequence of consecutive '0' bits. Accordingly, we can construct associated data-only, ciphertext-only and associated data and ciphertext forgery with the success rate of 2^{-1}, 2^{-1} and 1 after 2, 4 and 2 encryption queries, respectively.

Overall, our attacks defeat the claim of 128(256) and 128-bit integrity security of Limdolen-128(256) and HERN, respectively. We have experimentally verified the correctness of our attacks with the reference implementations. Notably, these are the first cryptanalytic results on both algorithms. Consequently, our results are expected to help in further understanding of similar designs.

Keywords. NIST lightweight cryptography standardization project · AEAD · Limdolen · HERN · Forgery

1 Introduction

The Internet of Things (IoT), sensor networks, distributed control systems and cyber physical systems are the most pre-eminent buzzwords these days. They have applications ranging from smart locks to wearable technology to home automation and healthcare. Typically, they operate in constrained environments and require reasonable efficiency with low implementation cost and sufficient security. The current standardized cryptographic primitives are designed for desktop and server environments, and many of them do not fit into the resource requirements of constrained devices. As a result, National Institute of Standards and Technology (NIST) initiated a lightweight cryptography project in 2013 and published the call for submissions of lightweight Authenticaed Encryption with Associated Data (AEAD) algorithms and hash functions, in August 2018 [NIS19]. In total, NIST received 57 submissions and 56 out of them were announced as the Round 1 candidates in April 2019. Two of such submissions are Limdolen [Meh19] and HERN [YSMW19].

Limdolen is a family of lightweight AEAD algorithms with key sizes 128 and 256 bits. At a high level, it adopts a Parallelizable Message Authentication Code (PMAC) [BR02] mode to compute tag and then use counter mode of encryption to generate the ciphertext. The XOR value of tag and nonce serve as the initial counter. However, compared to PMAC where random and indistinguishable secret masks[1] are used, Limdolen-128/(256) utilizes two distinct 128(256)-bit secret masks only. The designers state that

[1] masks derived from PMAC key where PMAC key equals $\mathcal{E}_K(0^n)$

"Due to Limdolen's target of constrained environments, rather than a series of calculations, we will alternate between $i = 0$ and $i = 1$, the two most common values of i in $\gamma^i L$."

Moreover, during the tag computation phase, *the associated data and message are first combined together to form a single input and then the padding procedure ie executed.* Based on the design choices and security proofs of PMAC and counter mode of encryption[2], the designers claim 128(256)-bit integrity security for Limdolen-128(256).

On the other hand, HERN is a 128-bit authenticated encryption scheme and adopts a stream cipher style construction similar to the CAESAR finalist Acorn [cae, Wu16]. The state size is 256 bits and at each clock cycle, 4 nonlinear bits are feedback to the state (except during ciphertext and tag generation phase). *After processing the associated data, the state is updated 512 times by adding '0' bit stream to the feedback bits. A similar procedure is applied after plaintext processing.* Accordingly, they claim that HERN achieves 128-bit integrity security.

Analyzing the security of NIST LWC Round 1 submissions with respect to forgery attacks is crucial before they are standardized and put in practice. A strong motivation is the recent forgery and plaintext recovery attacks on OCB2 [IM18, Poe18, IIMP19]. Its worth noting that OCB2 was included in ISO/IEC 19772:2009 [ISO] and forgeries are found a decade later. On a same note, practical forgeries are found for Round 1 submission SNEIKEN v1 [Saa19] by exploiting 1 round iterative differential [Per19, Kha19].

In this work, we investigate the security of Limdolen and HERN with reference to associated data-only, ciphertext-only and associated data and ciphertext forgeries in the nonce-respecting scenario. Table 1 presents a summary of our forgery attacks.

Table 1: Summary of forgery attacks on Limdolen and HERN. '$-$' denotes that input could be either empty or non-empty.

Algorithm	**Forgery type**	# Enc. queries	# Dec. queries	Success prob.	# blocks
Limdolen-128	associated data-only	1	1	1	≥ 1
	ciphertext-only	1	1	1	≥ 4
	associated data and ciphertext	1	1	1	≥ 1
Limdolen-256	associated data-only	1	1	1	≥ 1
	ciphertext-only	1	1	1	≥ 4
	associated data and ciphertext	1	1	1	≥ 1
HERN	associated data-only	2^n $(1 \leq n \leq 63)$	2^n	1	$-$
	ciphertext-only	2^{2n} $(1 \leq n \leq 31)$	2^n	1	$-$
	associated data and ciphertext	2^n $(1 \leq n \leq 63)$	1	1	$-$

Our contributions. We present the practical forgery attacks on Limdolen and HERN in the nonce-respecting setting. Our attacks exploit the structural flaws in the underlying design of these algorithms. Thus, our contributions are summarized as follows.

[2] they consider it as SIV mode [RS07] in the reference document

- We exploit the period 2 secret masks of Limdolen-128/256 and show that the XOR sum value before the last block cipher call is always a constant even if we add/remove/permutate blocks arbitrary number of times.
- For both variants of Limdolen, we show the general construction of associated data-only, ciphertext-only and associated data and ciphertext forgeries which have a successful probability of 1 after observing the output of a single encryption query. While, after one query, the designers claim the success probability of 2^{-128} and 2^{-256} for Limdolen-128 and Limdolen-256, respectively.
- By modifying input data with a sequence of consecutive zero bits, we show that HERN can not distinguish between associated data and plaintext processing phases. This observation is independent of number of rounds.
- For HERN, we create associated data-only, ciphertext-only and associated data and ciphertext forgery with the success rate of 2^{-1}, 2^{-1} and 1 after 2, 4 and 2 encryption queries, respectively. For the same number of queries, designers claim the success rate of 2^{-127}, 2^{-126} and 2^{-127}, respectively. We present a generalized version of our attack, i.e., for $1 \leq n \leq 63$ ($1 \leq n' \leq 31$) the success rate of forgeries are 2^{-n}, $2^{-n'}$ and 1 after 2^n, $2^{2n'}$ and 2^n encryption queries, respectively.
- To validate our theory, we have experimentally verified the correctness with the reference implementations. We have also provided examples for each type of forgery.

Organization of the paper. The rest of the paper is organised as follows. A brief description of Limdolen is provided in Section 2. In Section 3, we present the details of forgery attacks on Limdolen along with the experimental results. Section 4 and 5 present the specifications and forgery attacks on HERN, respectively. Finally, the paper is concluded in Section 6.

We conclude this section by defining the notations used throughout the paper in Table 2.

Table 2: Notations

Notation	Description
$X \odot Y, X \oplus Y, X\|Y, X\|\|Y$	bitwise AND, XOR, OR and concatenation of X and Y
$\|X\|$	length of X in bits
$\{0,1\}^{\geq n}$	bitstring with length at least n
$X \xleftarrow{\$} \{0,1\}^n$	random n bitstring drawn from $\{0,1\}^n$
1^n, 0^n	length n bitstring with all 1's, 0's
X^n	n repetitions of bitstring X
$\ll i$, $\lll i$	Left shift (left cyclic shift) by i bits
$(X_0, \cdots, X_{l-1}) \xleftarrow{n} X$	n-bit block parsing of X where $\|X_i\|= n$ for $0 \leq i \leq l-2$ and $1 \leq \|X_{l-1}\|\leq n$
$x_0, \cdots, x_{\|X\|-1}$	bit representation of X
$X[i]$	i-th byte of X starting from left
K, N, T (k_i, n_i, t_i)	key, nonce and tag (in bits)
AD, M, C (ad_i, m_i, c_i)	associated data, plaintext and ciphertext (in bits)

2 Specifications of Limdolen

Limdolen is a family of lightweight AEAD algorithms with key sizes 128 and 256 bits. We denote an instance of Limdolen by Limdolen-n and its corresponding underlying block cipher by Limdolen-BC-n where $n \in \{128, 256\}$. In this section, we first give a brief overview of Limdolen-BC-n and Limdolen-n. We then list the security goals claimed by the designers.

2.1 Description of Limdolen Block Cipher

The block cipher Limdolen-BC-n takes as input an n-bit key K, n-bit plaintext P and outputs an n-bit ciphertext after iterating the round function RF-n for $r = 16$ times. The round function consists of bitwise XOR, bitwise AND and L^{i} operations. The L^{i} operation takes 4 bytes as input and then performs left cyclic shift by i bits on each byte.

A pictorial depiction of RF-n is shown in Figure 1. Note that the round key is obtained by simply XORing the master key with constants, i.e., $RK_i = K \oplus const_i$. We omit the description of constants as our attacks are independent of them and refer the reader to [Meh19] for more details.

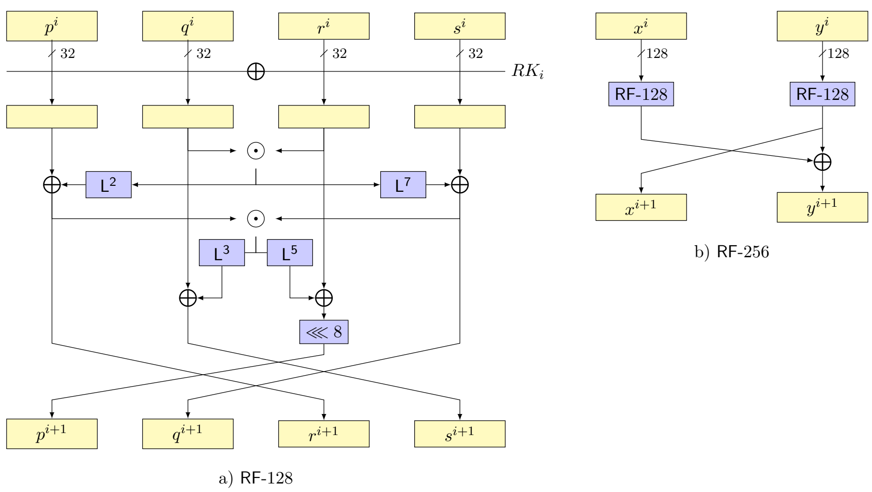

Fig. 1: Round functions of Limdolen block ciphers

2.2 Description of Limdolen AEAD

Limdolen adopts a tweaked PMAC [BR02] based construction to provide AEAD functionality. It has two variants Limdolen-n, $n \in \{128, 256\}$. For both the variants, the size of key, nonce and tag are equal to n bits. A high level overview of Limdolen-n is illustrated in Algorithm 1 and the individual phases are described below.

2.2.1 Padding The associated data AD and the message M are first concatenated together to form a single input message. It is then divided into chunks of n-bit blocks, i.e., $(X_0, \cdots, X_{l-1}) \xleftarrow{n} AD||M$. If $|X_{l-1}|= n$, then a single byte is XORed to the last byte of X_{l-1}. This pad_byte equals `0xC0` (`0x80`) depending on whether the length of associated data is zero (non-zero). In case the number of bytes of X_{l-1} is less than $n/8$, first a pad_byte is appended to X_{l-1}, followed by adding zero bytes until the block length becomes n. This procedure is denoted by addPaddingMarker$(\cdot)$ in Algorithm 1.

Remark 1. The padding rule described above follows the Limdolen's specification document (cf. Page 9 [Meh19]). However, in the reference implementation the pad_byte is always XORed to the last byte of X_{l-1}. Here, we emphasize that our attacks are independent of location of this byte.

2.2.2 Tag generation The tag computation of Limdolen-n is almost similar to PMAC [BR02] and is shown in Figure 2. First the PMAC key is derived by encrypting nonce with the master key. We denote it by aeadK where aeadK $=$ Limdolen-BC-$n(K, N)$. Next, three n-bit masks given by

$$\alpha = \text{Limdolen-BC-}n(\text{aeadK}, 0^n)$$
$$\text{alpha\_x} = \text{LB}(\alpha)$$
$$\text{alpha\_inv\_x} = \text{RB}(\alpha)$$

are computed where the function LB(α) (resp. RB(α)) rotates each byte of α left (resp. right) by 1. Each n-bit block X_i (except the last block) is XORed alternately with α or alpha_x which is then encrypted with Limdolen-BC-n using aeadK as the key. At each iteration, the output is XORed to δ_c which acts as a checksum. The tag is then given by

$$T = \text{Limdolen-BC-}n(\text{aeadK}, \delta_c \oplus \text{alpha\_inv\dot{}x} \oplus \text{addPaddingMarker}(X_{l-1})).$$

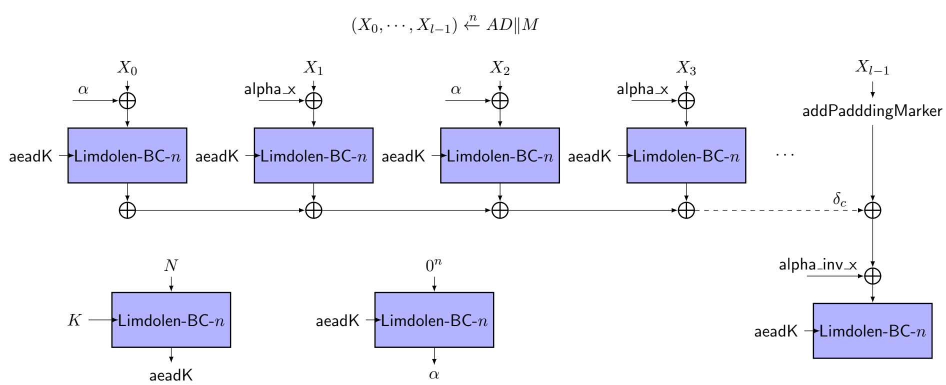

Fig. 2: Tag generation phase of Limdolen-n

2.2.3 Encryption The encryption is similar to the counter-mode of operation. The XOR value of nonce and tag is used as the intial counter. This phase is shown in Figure 3.

The decryption is similar to encryption and hence the details are omitted.

2.3 Security Claims

The security claims of Limdolen in the nonce-respecting setting are summarized in Table 3.

3 Forgery Attacks on Limdolen

In this section, we present the details of forgery attacks on both variants of Limdolen. First, we give a brief overview of the adversarial model and the main idea of our attack. Next, we show the construction of associated data-only, ciphertext-only and associated data and ciphertext forgeries that require a single encryption query and one forging attempt for successful verification. Finally, we provide the experimental results.

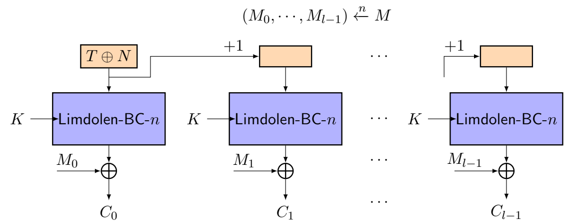

Fig. 3: Encryption phase of Limdolen-n

Algorithm 1 Limdolen-n AEAD

```
function tag_generation(K, N, AD, M):
    T ← 0^n
    aeadK ← Limdolen-BC-n(K, N)
    α ← Limdolen-BC-n(aeadK, 0^n)
    alpha_x ← LB(α)
    alpha_inv_x ← RB(α)
    (X_0, ···, X_{l-1}) ←^n AD‖M
     blockToggle = 1

     for i = 0 to l − 1 do:
        if i == l − 1 do:
            T ← T ⊕ addPaddingMarker(X_i)
        else if blockToggle = 1 do:
           T ← T ⊕ Limdolen-BC-n(aeadK, X_i ⊕ α)
        else if blockToggle = 0 do:
           T ← T ⊕ Limdolen-BC-n(aeadK, X_i ⊕ alpha_x)
        blockToggle = blockToggle ⊕ 1

     T ← T ⊕ alpha_inv_x
     T ← Limdolen-BC-n(aeadK, T)
     return T

function encryption(K, N, AD, M):
    T = tag_generation(K, N, AD, M)
    C ← ϵ
    ctr = T ⊕ N
    (M_0, ···, M_{l-1}) ←^n M
     for i = 0 to l − 2 do:
        C ← C‖Limdolen-BC-n(K, ctr) ⊕ M_i
        ctr = ctr + 1 mod 2^n
     Z = Limdolen-BC-n(K, ctr)
     (i_0, ···, i_{|M_{l-1}|-1}) ←^1 M_{l-1}
     C ← C‖z_0 ⊕ i_0‖···‖z_{|M_{l-1}|-1} ⊕ i_{|M_{l-1}|-1}
     return C, T
```

```
function addPaddingMarker(X):
    if |AD| = 0 :
        pad_byte = 0xC0
    else :
        pad_byte = 0x80

    if |X_{l-1}| == n :
        X_{l-1}[n/8 − 1] ← X_{l-1}[n/8 − 1] ⊕ pad_byte
    else :
        u : # bytes of X_{l-1}
        X_{l-1}[u] = pad_byte
        for i = u + 1 to n/8 − 1 do:
            X_{l-1}[i] = 0x00

    return X_{l-1}

function LB(α):
    for i = 0 to n/8 − 1 do:
        α[i] ← α[i] ≪ 1
    return α

function RB(α):
    for i = 0 to n/8 − 1 do:
        α[i] ← α[i] ≫ 1
    return α
```

3.1 Adversarial Model

We assume that the adversary $\mathcal{A}$ is nonce-respecting, which means it never makes two queries to the encryption oracle with the same nonce. Nevertheless, $\mathcal{A}$ is allowed to repeat nonces in decryption queries. We say that **"$\mathcal{A}$ forges"** if decryption oracle ever returns a plaintext other than error

Table 3: Security claims of Limdolen in bits [Meh19]

Goal	Limdolen-128	Limdolen-256
Confidentiality of plaintext	128	256
Integrity of plaintext	128	256
Integrity of associated data	128	256
Data limit (in blocks)	2^{64}	2^{128}

symbol $\perp$ on input of (N, AD, C, T) where (C, T) has never been output by encryption oracle on input of a query (N, AD, M) for some AD and M [Rog02].

In the sequel, we classify three types of forgeries based on the input modification.

- associated data-only: **"$\mathcal{A}$ forges"** by changing AD only
- ciphertext-only: **"$\mathcal{A}$ forges"** by changing C only
- associated data and ciphertext: **"$\mathcal{A}$ forges"** by changing both AD and C.

3.2 Core Idea of Forgery

For simplicity, we explain the idea for a single complete block of associated data which is given in Lemma 1.

Lemma 1. *Let $K \xleftarrow{\$} \{0,1\}^n$ be fixed. Let $N \xleftarrow{\$} \{0,1\}^n$, $AD_0 \xleftarrow{\$} \{0,1\}^n$, $M = \epsilon$ and (ϵ, T) be the corresponding ciphertext and tag pair. Then for a positive integer $i \geq 1$ and $AD'_0 \xleftarrow{\$} \{0,1\}^n$, $AD'_1 \xleftarrow{\$} \{0,1\}^n$ and $AD' = (AD'_0 \| AD'_1 \| AD'_0 \| AD'_1)^i \| AD_0$, we have $C' = \epsilon$ and $T' = T$.*

Proof. Since $M' = M = \epsilon \implies C' = C = \epsilon$. We now look at the tag generation of AD and AD'. The respective tags are given by

$$T = \text{Limdolen-BC-}n(\text{aeadK}, \text{alpha\_inv\^{}x} \oplus \text{addPaddingMarker}(AD_0))$$
$$T' = \text{Limdolen-BC-}n(\text{aeadK}, \delta'_c \oplus \text{alpha\_inv\^{}x} \oplus \text{addPaddingMarker}(AD_0)),$$

where $\delta'_c = 0^n$ (see Figure 4 for $i = 1$ case). Thus $T' = T$. $\square$

Corollary 1. *To construct forgery for arbitrary number of blocks, we only need to ensure that the XOR sum δ_c (see Figure 2) before the last call of block cipher is a constant.*

Remark 2. Lemma 1 trivially holds for partial last block.

3.3 Basic Forgery

We describe the basic minimal example of the forgery attack against Limdolen-n. We assume that blocks are complete and the number of blocks is at least 1. From now onwards, we refer Limdolen-BC-n with key K by $\mathcal{E}_K^n(\cdot)$.

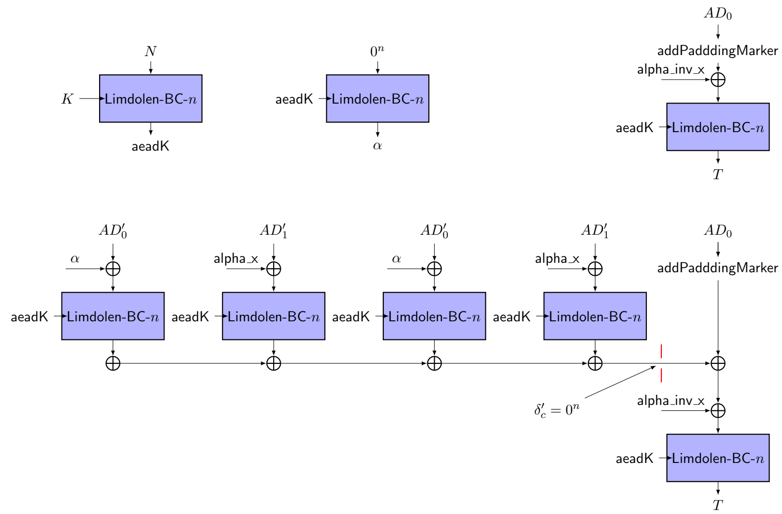

Fig. 4: Limdolen forgery for a single AD block

3.3.1 Associated data-only forgery Let $u \geq 1$ and $i \geq 1$ be two positive integers. Fix $K \xleftarrow{\$} \{0,1\}^n$. We construct forgery as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$, $AD \leftarrow \{0,1\}^{u \times n}$, $(AD_0, \cdots, AD_{u-1}) \xleftarrow{n} AD$ and $M = \epsilon$. Encrypt (N, AD, M) and observe (C, T).
Step 2 Let $X, Y \xleftarrow{\$} \{0,1\}^n$ and $W = X \| Y \| X \| Y$.
Step 3 Forge with (N, AD', C, T) where

$$AD' = AD_0 \| \cdots \| AD_{u-2} \| W^i \| AD_{u-1}.$$

Note that $AD' \neq AD \implies$ the decryption query is valid. This will pass the verification with probability 1 and returns empty plaintext as the output. To see why this forgery works, consider the values of δ_c and δ'_c, which are given by

$$\delta_c = \bigoplus_{i \bmod 2=0}^{i<u-1} \mathcal{E}^n_{\mathsf{aeadK}}(AD_i \oplus \alpha) \bigoplus_{i \bmod 2=1}^{i<u-1} \mathcal{E}^n_{\mathsf{aeadK}}(AD_i \oplus \mathsf{alpha\_x})$$

If $u - 1$ is even then

$$\begin{aligned}\delta'_c = &\bigoplus_{i \bmod 2=0}^{i<u-1} \mathcal{E}^n_{\mathsf{aeadK}}(AD_i \oplus \alpha) \bigoplus_{i \bmod 2=1}^{i<u-1} \mathcal{E}^n_{\mathsf{aeadK}}(AD_i \oplus \mathsf{alpha\_x}) \\ &2i \bigoplus (\mathcal{E}^n_{\mathsf{aeadK}}(X \oplus \alpha) \oplus \mathcal{E}^n_{\mathsf{aeadK}}(Y \oplus \mathsf{alpha\_x})) \\ = &\ \delta_c \oplus 0^n \implies T' = T.\end{aligned}$$

Similarly, if u is odd then $\delta'_c = \delta_c \oplus 0^n$ and $T' = T$. The only difference is that masks α and alpha_x are interchanged.

Some observations on associated data-only forgery.

1. The converse also holds true, i.e., given $AD = AD_0 \| \cdots \| AD_{u-2} \| W^i \| AD_{u-1}$, the modified associated data of the form $AD_0 \| \cdots \| AD_{u-2} \| W^l \| AD_{u-1}$ will give the same tag for all l satisfying $1 \leq l < i$.

2. The forgery is independent of whether the last block is a partial AD/M block or consists of both AD and M bytes.
3. We can modify AD in a number of ways. For instance, the following modification also results in a successful forgery.

$$AD' = \begin{cases} X\|Y\|AD_0\|\cdots\|AD_{u-2}\|X\|Y\|AD_{u-1} & \text{if } u \text{ is odd,} \\ Y\|X\|AD_0\|\cdots\|AD_{u-2}\|X\|Y\|AD_{u-1} & \text{o.w.} \end{cases}$$

3.3.2 Ciphertext-only forgery Fix a integer $u \geq 4$ and $K \xleftarrow{\$} \{0,1\}^n$. Let $S_e = \{0, 2, \cdots, \}$ and $S_o = \{1, 3, \cdots, \}$ be the set of even and odd integers less than $u-1$. Consider two permutations π and ψ which permutates the set S_e and S_o, respectively. Assume that π and ψ are not identity permutations simultaneously. We now construct forgery as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$, $AD = \epsilon$, $M \xleftarrow{n} \{0,1\}^{u\times n}$ and $(M_0, \cdots, M_{u-1}) \xleftarrow{n} M$. Encrypt (N, AD, M) and observe (C, T).
Step 2 Let $(C_0, \cdots, C_{u-2}, C_{u-1}) \xleftarrow{n} C$ and compute $Z_i = M_i \oplus C_i$ for $i = 0, \cdots, u-2$.
Step 3 Forge with (N, AD, C', T) where

$$C' = Z_0 \oplus M_{\pi(0)}\|Z_1 \oplus M_{\psi(0)}\|Z_2 \oplus M_{\pi(1)}\|Z_3 \oplus M_{\psi(1)}\|\cdots\|C_{l-1}.$$

We have $C' \neq C \implies$ the decryption query is valid. This will always pass the verification and returns

$$M_{\pi(0)}\|M_{\psi(0)}\|M_{\pi(1)}\|M_{\psi(1)}\|\cdots\|M_{l-1}$$

as the output.

To see the correctness of this forgery, we look at the decryption of (N, AD, C', T). First note that ciphertext computation is done via counter mode of operation (see Figure 3). Since the counter $T \oplus N$ is same for both encryption and decryption queries, then $M' = M_{\pi(0)}\|M_{\psi(0)}\|M_{\pi(1)}\|M_{\psi(1)}\|\cdots\|M_{l-1}$ is obtained (not released yet). Next, to see if the tags of M' and M are same it is enough to show that $\delta'_c = \delta_c$. This follows trivially as the masking value is α and alpha_x for each element in S_e and S_o, respectively. So, permutating these sets individually will not change the XOR sum value. Formally, we have

$$\begin{aligned}\delta'_c &= \bigoplus_{\pi(i), i\in S_e} \mathcal{E}^n_{\mathsf{aeadK}}(M_{\pi(i)} \oplus \alpha) \bigoplus_{\psi(i), i\in S_o} \mathcal{E}^n_{\mathsf{aeadK}}(M_{\psi(i)} \oplus \mathsf{alpha\_x}) \\ &= \bigoplus_{i\in S_e} \mathcal{E}^n_{\mathsf{aeadK}}(M_i \oplus \alpha) \bigoplus_{i\in S_o} \mathcal{E}^n_{\mathsf{aeadK}}(M_i \oplus \mathsf{alpha\_x}) \\ &= \delta_c \implies T' = T.\end{aligned}$$

Remark 3. If π and ψ both are identity permutations then $C' = C \implies$ the decryption query is not valid. The number of valid forgeries then equals $\lceil \frac{u}{2} \rceil \lceil \frac{u-1}{2} \rceil - 1$. Furthermore, these are independent of the length of the last message block.

Remark 4. Associated Data and Ciphertext Forgery is a direct application of associated data-only and ciphertext-only forgeries.

3.4 Forgeries Associated with Last Block

Until now, we have consider the cases where the last block is not modified. To forge the last block, all the previous blocks before it must contain AD bytes. Assume there is only 1 block and it consists of u bytes of AD and v bytes of M such that $u + v \leq n/8$. The forgery then proceed as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$. Encrypt (N, AD, M) and observe (C, T).
Step 2 Compute the keystream bytes $Z[i] = M[i] \oplus C[i]$ for $i = 0, \cdots, v-1$
Step 3 For $1 \leq l \leq v$, forge with (N, AD', C, T) where $AD' = AD\|M[0]\|M[l-1]$ and

$$C' = \begin{cases} \epsilon & \text{if } l = v, \\ Z[0] \oplus M[l]\|\cdots\|Z[v-l-1] \oplus M[v-1] & \text{o.w.} \end{cases}$$

We have $AD' \neq AD$ and $C' \neq C$. Thus, the decryption query is valid and will pass the verification with probability 1 as $AD'\|M' = AD\|M$. The output is $M' = M[l]\|\cdots\|M[v-1]$. Further note that this is a special case of associated data and ciphertext forgery.

Remark 5. The above forgery incorporates both cases of Remark 1 whether pad_byte is XORed to the last byte of block or it is appended after AD and M bytes in case of $u + v < n/8$.

3.5 Experimental Verification

We have verified the attacks using the reference implementation of Limdolen[Meh19]. In Tables 4 and 5, we list the examples of forgeries for Limdolen-128 and Limdolen-256, respectively.

4 Specifications of HERN

HERN adopts a stream cipher based construction similar to the CAESAR finalist Acorn [Wu16]. The state consists of four 64-bit registers which are updated in an LFSR based style by feeding the two nonlinear bits a and b to the registers. A pictorial representation of HERN state update function is shown in Figure 5 and the individual core components are illustrated in Algorithm 2.

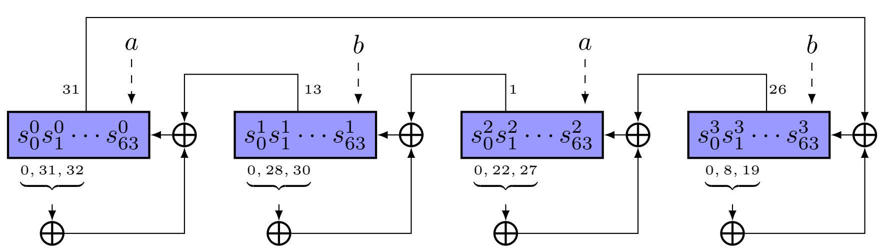

Fig. 5: Schematic of HERN state update function

4.1 Description of HERN AEAD

The HERN AEAD algorithm takes as input a 128-bit key K, 128-bit nonce N, *adlen* bits associated data AD, *mlen* bits plaintext M and outputs a *mlen* bits ciphertext C and 128-bit authentication tag T. The encryption consists of 3 phases, namely 1) Initialization, 2) Processing plaintext and 3) Finalization, which are described as follows.

Table 4: Examples of forgeries for Limdolen-128

Input data	**associated data-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	6B22729F7CEA8F9E1EDFB968365BF23B	6B22729F7CEA8F9E1EDFB968365BF23B
AD	BE0A1CDB4142106B5F2BB5BC8911E75E	A5687AF34938ED433536D8AB281FED78 5D1808F6DDD8D60B23EE9E0E061A5B93 A5687AF34938ED433536D8AB281FED78 5D1808F6DDD8D60B23EE9E0E061A5B93 BE0A1CDB4142106B5F2BB5BC8911E75E
M	Empty string	Empty string
C	Empty string	Empty string
T	EF4F60E08694CABB285D3841C433645D	EF4F60E08694CABB285D3841C433645D
Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	92C2A61831DCDE2EF3DB6060DF03DD0A	92C2A61831DCDE2EF3DB6060DF03DD0A
AD	Empty string	Empty string
M	ACCC9952DBB1CC0C8FA8106D463F483A BF23441F82A4BC61D2BF42AF6E4C1F1A 19B86CF46A3800F9E01066264FAF600E D2A42D5449E9B51BA9F8CB1744EA315D	19B86CF46A3800F9E01066264FAF600E BF23441F82A4BC61D2BF42AF6E4C1F1A ACCC9952DBB1CC0C8FA8106D463F483A D2A42D5449E9B51BA9F8CB1744EA315D
C	07AC6C25FAF2BA41F3B808502BA15F66 13237F247E2777389835C8C5B88BC655 E5EB9286DF5EE3FB8140B3588BC18C11 FBF38906197E5B6E069E50E4D8FABF45	B2D899834B7B76B49C007E1B22317752 13237F247E2777389835C8C5B88BC655 509F67206ED72F0EEEF8C5138251A425 FBF38906197E5B6E069E50E4D8FABF45
T	EDFDDE9B652A0FB16A7BFF22FD3B44D8	EDFDDE9B652A0FB16A7BFF22FD3B44D8
Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	2B2CC56156A6ACF4D3B1CCE369F4C934	2B2CC56156A6ACF4D3B1CCE369F4C934
AD	0C558F14C1E88FED	0C558F14C1E88FED60D1B7E5BA6EDC
M	60D1B7E5BA6EDC62	62
CT	93C6C56CBBF3B39D	91
T	C248D7D75062DE6163AFC13CADEBC55B	C248D7D75062DE6163AFC13CADEBC55B

4.1.1 Initialization The initialization consists of loading the key K and constants into the state and processing the nonce N, associated data AD and running H_if_step (see Algorithm 2) for 512 steps with zero input.

– Load the state with K and constants. We refer the reader to [YSMW19] for more details as this part is irrelevant for our attack.

Table 5: Examples of forgeries for Limdolen-256

Input data	**associated data-only**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	F1C79DD92DA67B984480270726EAB7568B4F1AA10C3BB0B525549E4239265B99	F1C79DD92DA67B984480270726EAB7568B4F1AA10C3BB0B525549E4239265B99
AD	5DA7FC78E3F3692D526069F6DD622EA81E2929484787D3F4354C5CC42DF07CE6	9A0F11FDF7A50B9B8F7C4CF1EB76932DF7E3ED26188C255317E18DE9E9BF6EAB E8B5B01D38A75A30F02DBE8517460F2E3C09E0E4CB2327B4CF63D2795F7DEC65 9A0F11FDF7A50B9B8F7C4CF1EB76932DF7E3ED26188C255317E18DE9E9BF6EAB E8B5B01D38A75A30F02DBE8517460F2E3C09E0E4CB2327B4CF63D2795F7DEC65 5DA7FC78E3F3692D526069F6DD622EA81E2929484787D3F4354C5CC42DF07CE6
M	Empty string	Empty string
C	Empty string	Empty string
T	301A471671BDF1CFAE68714DE61562000F8012DA449F8562E58B7635DC819CAC	301A471671BDF1CFAE68714DE61562000F8012DA449F8562E58B7635DC819CAC

Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	8196CF5D26A4D3728EC8D8B2CA5CA01EF7394366A2A98A09EA6CE9FBF3CCAAB5	8196CF5D26A4D3728EC8D8B2CA5CA01EF7394366A2A98A09EA6CE9FBF3CCAAB5
AD	Empty string	Empty string
M	9EEE67E185CE4A27D8F49C630FA67BF978E7BB6106B714F90FE08CB9CA425A68 30C149B58F94DC688879CB971F4691972E4CF834030C2D12EDB9CBB7FB25202C E769E176FDBEDE8537A91D56F0AEED1EFAE552FEF17F10DE38DC963401B660E8 1F415F1DFF3DA236E7BF8CD76D79F5685E476650C6762EFE52C432547A923C9A	E769E176FDBEDE8537A91D56F0AEED1EFAE552FEF17F10DE38DC963401B660E8 30C149B58F94DC688879CB971F4691972E4CF834030C2D12EDB9CBB7FB25202C 9EEE67E185CE4A27D8F49C630FA67BF978E7BB6106B714F90FE08CB9CA425A68 1F415F1DFF3DA236E7BF8CD76D79F5685E476650C6762EFE52C432547A923C9A
C	DF35A5881ADE06A920E381ADC2DE31A12E33E72C969EE55F35BF7DE2955FE1A1 4462C84E15647050EFDFC01B37FEBC0A0AC1EE3E02BED877CC233A9C2FE38900 2086D28CD3FF11D08F27CFE769BE4C914806A3DAE1676EFC7CC3135A508CA7E3 9CEE6811416763C0AA2A012395D883F5C2C9FC12EDDBCB509381739F0A9738EA	A6B2231F62AE920BCFBE00983DD6A746AC310EB36156E1780283676F5EABDB21 4462C84E15647050EFDFC01B37FEBC0A0AC1EE3E02BED877CC233A9C2FE38900 5901541BAB8F8572607A4ED296B6DA76CA044A4516AF6ADB4BFF09D79B789D63 9CEE6811416763C0AA2A012395D883F5C2C9FC12EDDBCB509381739F0A9738EA
T	3B4230CF23BB7D7E413E13451E8B899856A45A9C7ECB77FF32F257C7BD8780DA	3B4230CF23BB7D7E413E13451E8B899856A45A9C7ECB77FF32F257C7BD8780DA

Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	7C5734DCCA90853A2959276055D75ABDD4A0AD9BA48B4A845BD99D935FFDA78F	7C5734DCCA90853A2959276055D75ABDD4A0AD9BA48B4A845BD99D935FFDA78F
AD	CF84ACF34B794508DA221B691F332141	CF84ACF34B794508DA221B691F3321412F7C1F507F95FDA0E177B57A66C6C2
M	2F7C1F507F95FDA0E177B57A66C6C2F0	F0
C	8341876562C3BF87B49A155858082690	5C
T	5D8D4291C38C8FC922D7B697E873860593FD26971E590710D30A1F348A41E665	5D8D4291C38C8FC922D7B697E873860593FD26971E590710D30A1F348A41E665

– Process $N = n_0, n_1, \ldots, n_{127}$. At each step, one bit of N is used to update the state, i.e, $\mathsf{H\_if\_step}(n_i)$, for $i = 0, \cdots, 127$.
– Process $AD = ad_0, ad_1, \ldots, ad_{adlen-1}$. At each step, one bit of AD is used to update the state, i.e, $\mathsf{H\_if\_step}(ad_i)$, for $i = 0, \cdots, adlen - 1$.
– Run the H_if_step for 512 steps with zero-stream, i.e., $\mathsf{H\_if\_step}(0)$, for $i = 0, \cdots, 511$.

4.1.2 Processing plaintext

The plaintext $M = m_0, m_1, \cdots, m_{mlen-1}$ is used to update the state bit-by-bit and the corresponding ciphertext bit is generated using the function $\mathsf{H\_enc\_step}(\cdot)$ (see Algorithm 2).

– $C \leftarrow \epsilon$
– $c_i \leftarrow \mathsf{H\_enc\_step}(m_i)$, $C \leftarrow C \| c_i$, for $i = 0, \cdots, mlen - 1$

4.1.3 Finalization

After processing all the plaintext bits, the H_if_step runs for 512 times with zero input, and then the tag is generated.

– $\mathsf{H\_if\_step}(0)$, for $i = 0, \cdots, 511$.

Algorithm 2 Core components of HERN

function H_core_step:
  $a \leftarrow \mathsf{SB}(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14})$
  $b \leftarrow \mathsf{SB'}(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14}) \oplus s^0_{32}$
  $f^0 \leftarrow s^0_0 \oplus s^0_{31} \oplus s^0_{32} \oplus s^1_{13}$
  $f^1 \leftarrow s^1_0 \oplus s^1_{28} \oplus s^1_{30} \oplus s^2_1$
  $f^2 \leftarrow s^2_0 \oplus s^2_{22} \oplus s^2_{27} \oplus s^3_{26}$
  $f^3 \leftarrow s^3_0 \oplus s^3_8 \oplus s^3_{19} \oplus s^0_{31}$
  $s^i \leftarrow s^i \lll 1$, for $i = 0, 1, 2, 3$
  $s^i_{63} \leftarrow f^i$, for $i = 0, 1, 2, 3$

function SB$(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3)$:
  return $1 \oplus x_0y_0 \oplus x_1y_1 \oplus x_2y_2 \oplus x_3y_3$

function SB'$(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3)$:
  return $x_0y_2 \oplus y_0y_3 \oplus x_1x_3 \oplus y_1x_2$

function Adda:
  $s^0_{63} \leftarrow s^0_{63} \oplus a$
  $s^2_{63} \leftarrow s^2_{63} \oplus a$

function Addb:
  $s^1_{63} \leftarrow s^1_{63} \oplus b$
  $s^3_{63} \leftarrow s^3_{63} \oplus b$

function H_if_step(x):
  H_core_step
  $a \leftarrow a \oplus x$
  Adda
  Addb

function H_enc_step(m):
  H_core_step
  $a \leftarrow a \oplus m$
  Adda
  $c \leftarrow b \oplus m$
  return c

– $T \leftarrow \epsilon$
– $t_i \leftarrow$ H_enc_step(0), $T \leftarrow T \| t_i$, for $i = 0, \cdots, mlen - 1$
– return (C, T)

The decryption procedure is identical to encryption and hence the details are omitted.

4.2 Security Claims

The designers state that *"HERN is designed to have confidentiality of the plaintexts under adaptive chosen-plaintext attacks and the integrity of the ciphertexts under adaptive forgery attacks."* Considering the nonce-respecting setting and a data limit of 2^{64} bits (i.e., $adlen + mlen \leq 2^{64}$), they claim 128-bit security for confidentiality and integrity.

5 Forgery Attacks on HERN

In this section, we provide the details of forgery attacks on HERN. In particular, we show that a message can be modified by appending or removing a sequence of consecutive '0' bits of length n. Moreover, we show that the best success rate of forgery is achieved for $n = 1$ case.

5.1 Basic Forgery

The adversarial model is similar to Subsection 3.1. In the following, we explain the minimal example of our forgery attack against HERN. For the description of forgeries, we let S_i, a_i, b_i denote the state of HERN and two nonlinear bits a and b at the beginning of i-th round.

5.1.1 Associated data-only forgery Let $1 \leq n \leq 63$ and $K \xleftarrow{\$} \{0,1\}^{128}$ be fixed. To construct the forgery we proceed as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^{128}$, $AD \xleftarrow{\$} \{0,1\}^{\star}$ and $M = \epsilon$. Encrypt (N, AD, M) and observe (C, T).
Step 2 Repeat Step 1 until we obtain a tag whose first n bits are all zero. Define this query as $Q :\stackrel{\text{def}}{=} (N, AD, M, C, T)$.
Step 3 For each $i = 0$ to $2^n - 1$, decrypt (N', AD', C', T') where

$$\begin{aligned} N' &= N \\ AD' &= AD \| 0^n \\ C' &= \epsilon \\ T' &= T \lll n \mid (i_0 \| \cdots \| i_{n-1}), \text{ and } (i_0, \cdots, i_{n-1}) \xleftarrow{1} i. \end{aligned}$$

If the verification succeeds with output as an empty plaintext, we stop.

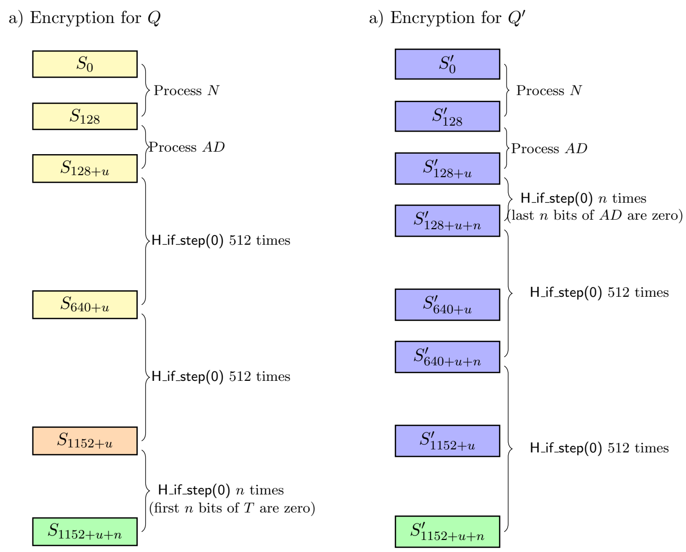

Fig. 6: Associated data-only forgery of HERN

The decryption queries are valid as $AD' \neq AD$ and $T' \neq T$. To see why such a query work, consider the encryption of Q and $Q' \stackrel{\text{def}}{=} (N, AD', \epsilon)$. This is illustrated in Lemma 2 (also shown in Figure 6).

Lemma 2. *Let Q and Q' be defined as above and $|AD| = u$. Then $T' = T \lll n \mid \Delta$ where Δ is an n-bit string.*

Proof. After processing 128 bits of nonce and first u bits of AD, the states are same, i.e., $S_{128+u} = S'_{128+u}$. For query Q, as M is empty, $\mathsf{H\_if\_step}(\cdot)$ runs for 1024 times with zero input. For Q', since $AD' = AD \| 0^n$ and $M' = \epsilon$, $\mathsf{H\_if\_step}(\cdot)$ is iterated for $n + 1024$ times with zero bit. The tag generation phase for Q and Q' starts from S_{1152+u} and $S'_{1152+u+n}$, respectively.

Note that the first n bits of T are zero and they are not added to the state. This is equivalent to the fact that $\mathsf{H\_if\_step}(0)$ runs for another n times starting from round $1152+u$. Hence, $S_{1152+u+n} = S'_{1152+u+n} \implies$ the last $128 - n$ bits of T are the same as the first $128 - n$ bits of T'. Since the states are unknown, the last n bits of T' has to be guessed. Thus, $T' = T \lll n \mid \Delta$. □

Attack complexities. On average, step 2 requires 2^n encryption queries while step 3 needs 2^n decryption queries. Thus, for $1 \leq n \leq 63$, the success rate of forgery is 2^{-n}. For $n = 1$ the success rate is 2^{-1} after querying encryption oracle 2 times. This clearly violates the designers claim that success rate of forgery is 2^{-127} after two encryption queries.

Some observations on associated data-only forgery.

1. The designers imposed a data limit of 2^{64} bits before a re-keying is done. In order to satisfy this constraint, we restrict the values of n in the range $1, \cdots, 63$. However, this is just a theoretical reasoning and we do not need so many queries especially when we can construct forgery for $n = 1$ case.
2. The forgery still works if we change 512 to some other number. Hence, it is independent of the number of rounds.

5.1.2 Ciphertext-only forgery Let $1 \leq n \leq 31$ and $K \xleftarrow{\$} \{0,1\}^{128}$ be fixed. We construct forgery as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^{128}$, $AD \xleftarrow{\$} \{0,1\}^{\star}$, $M \leftarrow \{0,1\}^{\geq 1}$. Encrypt $(N, AD, M \| 0^n)$ and observe (C, T).
Step 2 Repeat Step 1 until a ciphertext whose last n bits are zero is obtained. Denote this query by (N, AD, M, C, T).
Step 3 Decrypt (N', AD', C', T') where

$$\begin{aligned} N' &= N \\ AD' &= AD \\ C' &= c_0 \| \cdots \| c_{|M|-n-1} \\ T' &= 0^n | T \gg n. \end{aligned}$$

Step 4 If verification fails, repeat Step 2 and Step 3.

We have $C' \neq C$ as the lengths are different and $T' \neq T$. Thus, each query in step 3 is a valid decryption query. Upon successful verification, only first $|M|-n$ bits of M are returned. A formal proof of correctness of decryption query is given in Lemma 3.

Lemma 3. *Let $Q \overset{\text{def}}{:=} (N, AD, M)$ satisy Step 2 with output as (C, T). Let $AD' = AD$, $M' = m_0 \| \cdots \| m_{|M|-n-1}$ and $Q' \overset{\text{def}}{:=} (N, AD', M')$. Then $T' = 0^n \mid T \gg n$ iff the n nonlinear bits $b_{1152+|AD|+|M|-n}, \cdots, b_{1152+|AD|+|M|-1}$ are all zero.*

Proof. We have $AD' = AD$ and $m'_i = m_i \implies c'_i = c_i$, for $0 \leq i \leq |M|-n-1$. Therefore, $S_{1152+u+|M|-n} = S'_{1152+u+|M|-n}$. However, the tag generation phase for Q starts from $S_{640+u+|M|}$, and for Q' it starts from $S'_{640+u+|M|-n}$. The corresponding tag bits are given by:

$$\begin{aligned} t_i &= b_{1152+|AD|+|M|+i} \\ t'_i &= b'_{1152+|AD|+|M|-n+i}. \end{aligned}$$

Now, the last n bits of both M and C being zero $\implies S_{1152+|AD|+|M|-n} = S'_{1152+|AD|+|M|-n}$. So, given $b_{1152+|AD|+|M|-n}, \cdots, b_{1152+|AD|+|M|-1}$ are all zero, then $T' = 0^n \mid T \gg n$. □

Attack complexities. Step 2 requires 2^n encryption queries (on average), while to satisy both Step 2 and Step 3 simultaneously, 2^{2n} encryption queries (on average) are needed. Thus, for $1 \leq n \leq 31$, the success rate of forgery is 2^{-n} after observing output of 2^{2n} encryption queries. The value of n is chosen to satisfy the data limit restriction of 2^{64} bits.

Remark 6. Similar to associated data-only forgery, the best success rate is achieved for $n = 1$ case which is 2^{-1} after 4 encryption queries.

5.1.3 Associated data and ciphertext forgery

Let $1 \leq n \leq 63$ and $K \xleftarrow{\$} \{0,1\}^{128}$ be fixed. The forgery then proceed as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^{128}$, $AD \xleftarrow{\$} \{0,1\}^{\star}$, $M = 0^n$. Encrypt (N, AD, M) and observe (C, T).
Step 2 Repeat step 1 until we obtain $C = 0^n$. Denote this query by (N, AD, M, C, T).
Step 3 Forge with (N', AD', C', T') where

$$\begin{aligned} N' &= N \\ AD' &= AD \| 0^n \\ C' &= \epsilon \\ T' &= T, \end{aligned}$$

which will always be successful (with empty message as an output) as the states after $640 + |AD| + n$ rounds are same. The proof is similar to Lemma 2 and 3, and hence omitted.

Attack complexities. Step 2 requires 2^n encryption queries on average, while step 3 requires only a single decryption query. Thus, for $1 \leq n \leq 63$, the success rate of forgery is 1.

5.2 Experimental Verification

We have verified the attacks using the reference implementation of HERN [YSMW19]. In Table 6, we list the examples for $n = 8$.

6 Concluding Remarks

We have demonstrated a series of practical forgery attacks on Limdolen and HERN in the nonce-respecting scenario. Our attacks defeat the designer's claim of 128(256) and 128-bit integrity security of Limdolen-128(256) and HERN, respectively.

For both variants of Limdolen, we have shown the constructions of associated data-only, ciphertext-only and associated data and ciphertext forgeries which require a single encryption and a single decryption query, and have a successful probability of 1. The crux of our forgery attacks lie in Lemma 1 and the observation that only a sequence of period 2 consisting of (α, alpha_x) is used for masking. Moreover, we have found a discrepancy for the padding in the specification document and reference implementation (see Remark 1). However, the presented attacks are independent of this inconsistency. To resist our attacks, the period 2 masking sequence has to be replaced by a sequence with unpredictable properties.

For HERN, we have found that associated data and message processing phases are not distinguishable. As a result one can modify a message by appending or removing a sequence of zero bits. Accordingly, we have presented round independent associated data-only, ciphertext-only and associated data and ciphertext forgeries with the success rate of 1 after 2(2), 4(2) and 2(1)

Table 6: Examples of forgeries for HERN

Input data	associated data-only	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	D8A4ADC965EECE56330E5CC01A53C928	D8A4ADC965EECE56330E5CC01A53C928
AD	CA5F	CA5F00
M	Empty string	Empty string
CT	Empty string	Empty string
T	00FC40BF26954B37993E9C56C6C49ACA	FC40BF26954B37993E9C56C6C49ACAB6
Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	3E1327BCC61246AC87901E0922C1A354	3E1327BCC61246AC87901E0922C1A354
AD	9524	9524
M	8500	85
CT	0D00	0D
T	8472B9D92F6AAC22CE3F188CC13D711C	008472B9D92F6AAC22CE3F188CC13D71
Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	7B8A185D3B33E4F906E02F291BEF6C06	7B8A185D3B33E4F906E02F291BEF6C06
AD	4328	432800
M	00	Empty string
CT	00	Empty string
T	A72C78D89FAD7A7D785EF13AB2EC085B	A72C78D89FAD7A7D785EF13AB2EC085B

encryption(decryption) queries, respectively. A simple fix to resist our attack is to complement a state bit (except the last bit of each register) after $640 + |AD|$ and $640 + |AD| + |M|$ clock cycles.

Acknowledgement

This work is supported by NSERC.

References

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 384–397. Springer, 2002.

[cae] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar.html.

[IIMP19] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality. Cryptology ePrint Archive, Report 2019/311, 2019. https://eprint.iacr.org/2019/311.

[IM18] Akiko Inoue and Kazuhiko Minematsu. Cryptanalysis of OCB2. Cryptology ePrint Archive, Report 2018/1040, 2018. https://eprint.iacr.org/2018/1040.

[ISO] ISO: Information Technology - Security techniques - Authenticated encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772 (2009).

[Kha19] Mustafa Khairallah. Forgery Attack on SNEIKEN. 2019. https://eprint.iacr.org/2019/408.

[Meh19] Carl E. Meher. Limdolen: A Lightweight Authenticated Encryption Algorithm. NIST LWC Round 1 Submssion. 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/Limdolen-Spec.pdf.

[NIS19] NIST lightweight cryptography standardization process. https://csrc.nist.gov/projects/lightweight-cryptography, accessed 31 May 2019.

[Per19] Leo Perrin. Probability 1 Iterated Differential in the SNEIK Permutation. Cryptology ePrint Archive, Report 2019/374, 2019. https://eprint.iacr.org/2019/374.

[Poe18] Bertram Poettering. Breaking the confidentiality of OCB2. Cryptology ePrint Archive, Report 2018/1087, 2018. https://eprint.iacr.org/2018/1087.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107. ACM, 2002.

[RS07] Phillip Rogaway and Thomas Shrimpton. The SIV mode of operation for deterministic authenticated-encryption (key wrap) and misuse-resistant nonce-based authenticated-encryption. *Aug*, 20:3, 2007.

[Saa19] Markku Juhani O. Saarinen. SNEIKEN and SNEIKHA authenticated encryption and cryptographic hashing. NIST LWC Round 1 Submission, 2019. Available online at https://github.com/pqshield/sneik.

[Wu16] Hongjun Wu. ACORN: a lightweight authenticated cipher (v3). 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[YSMW19] Dingfeng Ye, Danping Shi, Yuan Ma, and Peng Wang. HERN and HERON: Lightweight AEAD and Hash Constructions based on Thin Sponge (v1). 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/.