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 $E_K(0^n)$
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 \(\gamma^1L\).

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 is 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\(^{\text{cae, Wu16}}\). 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\(^{\text{IM18, Poe18, IIMP19}}\). Its worth noting that OCB2 was included in ISO/IEC 19772:2009\(^{\text{ISO}}\) and forgeries are found a decade later. On a same note, practical forgeries are found for Round 1 submission SNEIKEN v1\(^{\text{Saa19}}\) by exploiting 1 round iterative differential\(^{\text{Per19, Kha19}}\).

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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Forgery type</th>
<th># Enc. queries</th>
<th># Dec. queries</th>
<th>Success prob.</th>
<th># blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limdolen-128</td>
<td>associated data-only</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>ciphertext-only</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 4</td>
</tr>
<tr>
<td></td>
<td>associated data and ciphertext</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 1</td>
</tr>
<tr>
<td>Limdolen-256</td>
<td>associated data-only</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>ciphertext-only</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 4</td>
</tr>
<tr>
<td></td>
<td>associated data and ciphertext</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>≥ 1</td>
</tr>
<tr>
<td>HERN</td>
<td>associated data-only</td>
<td>(2^n \text{ (} 1 \leq n \leq 63\text{)})</td>
<td>1</td>
<td>(2^n)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>ciphertext-only</td>
<td>(2^n \text{ (} 1 \leq n \leq 31\text{)})</td>
<td>(2^n)</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>associated data and ciphertext</td>
<td>(2^n \text{ (} 1 \leq n \leq 63\text{)})</td>
<td>1</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

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\(^{\text{RS07}}\) 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>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \odot Y, X \oplus Y, X</td>
<td>Y, X</td>
</tr>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>${0,1}^n$</td>
<td>bitstring with length at least $n$</td>
</tr>
<tr>
<td>$X \triangleleft {0,1}^n$</td>
<td>random $n$ bitstring drawn from ${0,1}^n$</td>
</tr>
<tr>
<td>$1^n, 0^n$</td>
<td>length $n$ bitstring with all 1’s, 0’s</td>
</tr>
<tr>
<td>$X^n$</td>
<td>$n$ repetitions of bitstring $X$</td>
</tr>
<tr>
<td>$\ll i, \lll i$</td>
<td>Left shift (left cyclic shift) by $i$ bits</td>
</tr>
<tr>
<td>$(X_0, \cdots, X_{l-1}) \triangleleft \ll X$</td>
<td>$n$-bit block parsing of $X$ where $</td>
</tr>
<tr>
<td>$x_0, \cdots, x_{</td>
<td>X</td>
</tr>
<tr>
<td>$X[i]$</td>
<td>$i$-th byte of $X$ starting from left</td>
</tr>
<tr>
<td>$K, N, T (k_i, n_i, t_i)$</td>
<td>key, nonce and tag (in bits)</td>
</tr>
<tr>
<td>$AD, M, C (ad_i, m_i, c_i)$</td>
<td>associated data, plaintext and ciphertext (in bits)</td>
</tr>
</tbody>
</table>
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](image)

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., $(X_0, \ldots, X_{l-1}) \leftarrow AD||M$. If $|X_{l-1}| = n$, then a single byte is XORed to the last byte of $X_{l-1}$. This pad_byte equals $0x00$ ($0x80$) 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 $\text{aeadK}$ where $\text{aeadK} = \text{Limdolen-BC-n}(K,N)$. Next, three $n$-bit masks given by

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

$$\alpha_x = \text{LB}(\alpha)$$

$$\alpha_{\text{inv}x} = \text{RB}(\alpha)$$

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

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

![Fig. 2: Tag generation phase of Limdolen-n](image)

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.
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.

### 3.1 Adversarial Model

Fig. 3: Encryption phase of Limdolen-\( n \)

\begin{algorithm}
\begin{enumerate}
\item \textbf{function} \( \text{tag\_generation}(K, N, AD, M) \):
\item \( T \leftarrow 0^n \)
\item \( \text{aeadK} \leftarrow \text{Limdolen-BC-n}(K, N) \)
\item \( \alpha \leftarrow \text{Limdolen-BC-n}(\text{aeadK}, 0^n) \)
\item \( \text{alpha}_x \leftarrow \text{LB}(\alpha) \)
\item \( \text{alpha\_inv}_x \leftarrow \text{RB}(\alpha) \)
\item \( (X_0, \ldots, X_{l-1}) \leftarrow AD||M \)
\item \( \text{blockToggle} = 1 \)
\item \textbf{for} \( i = 0 \) to \( l-1 \) \textbf{do}:
\item \quad \textbf{if} \( i = l-1 \) \textbf{do}:
\item \quad \quad \( T \leftarrow T \oplus \text{addPaddingMarker}(X_i) \)
\item \quad \else \textbf{if} \( \text{blockToggle} = 1 \) \textbf{do}:
\item \quad \quad \( T \leftarrow T \oplus \text{Limdolen-BC-n}(\text{aeadK}, X_i \oplus \text{alpha}_x) \)
\item \quad \else \textbf{if} \( \text{blockToggle} = 0 \) \textbf{do}:
\item \quad \quad \( T \leftarrow T \oplus \text{Limdolen-BC-n}(\text{aeadK}, X_i \oplus \text{alpha}_x) \)
\item \quad \text{blockToggle} = \text{blockToggle} \oplus 1 \)
\item \( T \leftarrow T \oplus \text{alpha\_inv}_x \)
\item \( T \leftarrow \text{Limdolen-BC-n}(\text{aeadK}, T) \)
\item \textbf{return} \( T \)
\end{enumerate}
\end{algorithm}

\begin{algorithm}
\begin{enumerate}
\item \textbf{function} \( \text{addPaddingMarker}(X) \):
\item \textbf{if} \( |AD| = 0 \) : \textbf{return} \( 0^n \)
\item \textbf{else}:
\item \quad \text{pad\_byte} = 0x0c0
\item \quad \textbf{else}:
\item \quad \quad \text{pad\_byte} = 0x80
\item \quad \textbf{if} \( |X_{l-1}| = n \) :
\item \quad \quad \( X_{l-1}[|n| - 1] \leftarrow X_{l-1}[|n| - 1] \oplus \text{pad\_byte} \)
\item \quad \else:
\item \quad \quad \( u \) : \# bytes of \( X_{l-1} \)
\item \quad \quad \( X_{l-1}[u] = \text{pad\_byte} \)
\item \quad \textbf{for} \( i = u + 1 \) to \( n - 1 \) \textbf{do}:
\item \quad \quad \( X_{l-1}[i] = 0x00 \)
\item \quad \textbf{return} \( X_{l-1} \)
\end{enumerate}
\end{algorithm}

\begin{algorithm}
\begin{enumerate}
\item \textbf{function} \( \text{LB}(\alpha) \):
\item \textbf{for} \( i = 0 \) to \( \frac{n}{8} - 1 \) \textbf{do}:
\item \quad \( \alpha[i] \leftarrow \alpha[i] \ll 1 \)
\item \textbf{return} \( \alpha \)
\end{enumerate}
\end{algorithm}

\begin{algorithm}
\begin{enumerate}
\item \textbf{function} \( \text{RB}(\alpha) \):
\item \textbf{for} \( i = 0 \) to \( \frac{n}{8} - 1 \) \textbf{do}:
\item \quad \( \alpha[i] \leftarrow \alpha[i] \gg 1 \)
\item \textbf{return} \( \alpha \)
\end{enumerate}
\end{algorithm}
Table 3: Security claims of Limdolen in bits [Meh19]

<table>
<thead>
<tr>
<th>Goal</th>
<th>Limdolen-128</th>
<th>Limdolen-256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality of plaintext</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Integrity of plaintext</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Integrity of associated data</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Data limit (in blocks)</td>
<td>$2^{64}$</td>
<td>$2^{128}$</td>
</tr>
</tbody>
</table>

symbol ⊥ 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: “A forges” by changing $AD$ only
- ciphertext-only: “A forges” by changing $C$ only
- associated data and ciphertext: “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 $K \leftarrow \{0,1\}^n$ be fixed. Let $N \leftarrow \{0,1\}^n$, $AD_0 \leftarrow \{0,1\}^n$, $M = \epsilon$ and $(\epsilon, T)$ be the corresponding ciphertext and tag pair. Then for a positive integer $i \geq 1$ and $AD_0' \leftarrow \{0,1\}^n$, $AD_1' \leftarrow \{0,1\}^n$ and $AD' = (AD_0'\|AD_1'\|AD_0'\|AD_1')\|AD_0$, 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

$$T = \text{Limdolen-BC-}n(\text{aead}K, \alpha_{\text{inv}}'x \oplus \text{addPaddingMarker}(AD_0))$$

$$T' = \text{Limdolen-BC-}n(\text{aead}K, \delta_c' \oplus \alpha_{\text{inv}}'x \oplus \text{addPaddingMarker}(AD_0)),$$

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

**Corollary 1.** To construct forgery for arbitrary number of blocks, we only need to ensure that the XOR sum $\delta_c$ (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 $E^n_K(\cdot)$. 
3.3.1 Associated data-only forgery Let $u \geq 1$ and $i \geq 1$ be two positive integers. Fix $K \leftarrow \{0,1\}^n$. We construct forgery as follows.

**Step 1** Let $N \leftarrow \{0,1\}^n$, $AD \leftarrow \{0,1\}^{u\times n}$, $(AD_0, \cdots, AD_{u-1}) \leftarrow AD$ and $M = \epsilon$. Encrypt $(N, AD, M)$ and observe $(C, T)$.

**Step 2** Let $X, Y \leftarrow \{0,1\}^n$ and $W = X || Y || X || Y$.

**Step 3** Forge with $(N, AD', C, T)$ where $AD' = AD_0 || \cdots || AD_{u-2} || W' || AD_{u-1}$.

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 $\delta_c$ and $\delta'_c$, which are given by

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

If $u - 1$ is even then

$$\delta'_c = \bigoplus_{i \bmod 2 = 0}^{i < u - 1} \mathcal{E}^n_{\text{aead}K}(AD_i \oplus \alpha) \bigoplus_{i \bmod 2 = 1}^{i < u - 1} \mathcal{E}^n_{\text{aead}K}(AD_i \oplus \text{alpha}_x)$$

$$+ 2i \bigoplus_{i \bmod 2 = 0}^{i < u - 1} (\mathcal{E}^n_{\text{aead}K}(X \oplus \alpha) \oplus \mathcal{E}^n_{\text{aead}K}(Y \oplus \text{alpha}_x))$$

$$= \delta_c \oplus 0^n \implies T' = T.$$
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 an integer \( u \geq 4 \) and \( K \overset{\$}{\leftarrow} \{0,1\}^n \). 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 \( \pi \) and \( \psi \) which permutates the set \( S_e \) and \( S_o \), respectively. Assume that \( \pi \) and \( \psi \) are not identity permutations simultaneously. We now construct forgery as follows.

**Step 1** Let \( N \overset{n}{\leftarrow} \{0,1\}^n \), \( AD = \epsilon \), \( M \overset{n}{\leftarrow} \{0,1\}^{u\times n} \) and \( (M_0,\cdots,M_{u-1}) \overset{n}{\leftarrow} M \). Encrypt \((N,AD,M)\) and observe \((C,T)\).

**Step 2** Let \((C_0,\cdots,C_{u-2},C_{u-1}) \overset{n}{\leftarrow} C\) and compute \( Z_i = M_i \oplus C_i \) for \( i = 0,\cdots,u-2 \).

**Step 3** Forge with \((N,AD,C',T)\) where

\[
C' = Z_0 \oplus M_{\pi(0)} \parallel Z_1 \oplus M_{\psi(0)} \parallel \cdots \parallel Z_{u-1} \oplus M_{\psi(1)} \parallel C_{u-1}.
\]

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

\[
M_{\pi(0)} \parallel M_{\psi(0)} \parallel \cdots \parallel M_{\psi(1)} \parallel C_{u-1}
\]

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 \( M' = M_{\pi(0)} \parallel M_{\psi(0)} \parallel \cdots \parallel M_{\pi(1)} \parallel M_{\psi(1)} \parallel \cdots \parallel M_{\psi(u-1)} \) 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 \( \alpha \) and \( \text{alpha}_x \) 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

\[
\delta'_c = \bigoplus_{\pi(i),i \in S_e} E^n_{\text{eadK}}(M_{\pi(i)} \oplus \alpha) \bigoplus_{\psi(i),i \in S_o} E^n_{\text{eadK}}(M_{\psi(i)} \oplus \text{alpha}_x)
\]

\[
= \bigoplus_{i \in S_e} E^n_{\text{eadK}}(M_i \oplus \alpha) \bigoplus_{i \in S_o} E^n_{\text{eadK}}(M_i \oplus \text{alpha}_x)
\]

\[
= \delta_c \implies T' = T.
\]

**Remark 3.** If \( \pi \) and \( \psi \) both are identity permutations then \( C' = C \implies \) the decryption query is not valid. The number of valid forgeries then equals \( \left\lfloor \frac{u}{2} \right\rfloor \cdot \left\lfloor \frac{u-1}{2} \right\rfloor - 1 \). 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 considered 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 proceeds as follows.

**Step 1** Let $N \leftarrow \{0, 1\}^n$. Encrypt $(N, AD, M)$ and observe $(C, T)$.

**Step 2** Compute the keystream bytes $Z[i] = M[i] \oplus C[i]$ for $i = 0, \ldots, v - 1$.

**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](image-url)

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

<table>
<thead>
<tr>
<th>Input data</th>
<th>associated data-only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td>000102030405060708090A0B0C0D0E0F</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>6B22729F7CE9F8E1ED6B86365BF23B</td>
</tr>
<tr>
<td><strong>AD</strong></td>
<td>BE0A1CDB4142106B5F2B5BC8911E75E</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>Empty string</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>Empty string</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>EF4F6DE8694CAAB285D3B43C3645D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input data</th>
<th>ciphertext-only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td>000102030405060708090A0B0C0D0E0F</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>92C2A61831DCDE2EF3DEB606DF03DD0A</td>
</tr>
<tr>
<td><strong>AD</strong></td>
<td>Empty string</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>ACCC9952DBB1C0C6F8A10DE6D64F483A</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>Empty string</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>EF4F6DE8694CAAB285D3B43C3645D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input data</th>
<th>associated data and ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td>000102030405060708090A0B0C0D0E0F</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>2B2CC65156A6ACF4D3B1CCE3E69F43C934</td>
</tr>
<tr>
<td><strong>AD</strong></td>
<td>0C558F14C1E88FED</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>60D1B7E5BA6D6C62</td>
</tr>
<tr>
<td><strong>CT</strong></td>
<td>93C65CBEF3839D</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>C248D7D75062DE6163AFC13CADEBC55B</td>
</tr>
</tbody>
</table>

### 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 $H_{if\text{\_step}}$ (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

<table>
<thead>
<tr>
<th>Input data</th>
<th>associated data-only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td></td>
</tr>
<tr>
<td>F1C7D9C204EBFC44EAB6AC2072CF2E8ADB83B564AC2936452ED69BB9</td>
<td></td>
</tr>
<tr>
<td><strong>N</strong></td>
<td></td>
</tr>
<tr>
<td>52A7C83F392D526069F66D23E481E29394847678D3F3546C5C042F07C6B</td>
<td></td>
</tr>
<tr>
<td><strong>AD</strong></td>
<td></td>
</tr>
<tr>
<td>Empty string</td>
<td></td>
</tr>
<tr>
<td><strong>M</strong></td>
<td></td>
</tr>
<tr>
<td>Empty string</td>
<td></td>
</tr>
<tr>
<td><strong>T</strong></td>
<td></td>
</tr>
<tr>
<td>031A478E12BD15F488B7781D652000F801DA248F8662E886735DC91C8C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input data</th>
<th>ciphertext-only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td></td>
</tr>
<tr>
<td>0102030405060708090A0BCDDEF0101111214145161781911A1B1C1</td>
<td></td>
</tr>
<tr>
<td><strong>N</strong></td>
<td></td>
</tr>
<tr>
<td>8196CF5D26A4D3728EC8D8B2CA5CA01EF7394366A2A98A09EA6CE9FBF3CCAAB5</td>
<td></td>
</tr>
<tr>
<td><strong>AD</strong></td>
<td></td>
</tr>
<tr>
<td>Empty string</td>
<td></td>
</tr>
<tr>
<td><strong>M</strong></td>
<td></td>
</tr>
<tr>
<td>30C149B58F94DC688879CB971F4691972E4CF834030C2D12EDB9CBB7FB2520C</td>
<td></td>
</tr>
<tr>
<td><strong>T</strong></td>
<td></td>
</tr>
<tr>
<td>5D8346A12E48902E831412E33E7C9E6964955FFBD82D5655E141</td>
<td></td>
</tr>
</tbody>
</table>

- Process $N = n_0, n_1, \ldots, n_{127}$. At each step, one bit of $N$ is used to update the state, i.e., $H_{\text{if step}}(n_i)$, for $i = 0, \ldots, 127$.
- Process $AD = a_{d0}, a_{d1}, \ldots, a_{d_{adlen}-1}$. At each step, one bit of $AD$ is used to update the state, i.e., $H_{\text{if step}}(a_i)$, for $i = 0, \ldots, adlen - 1$.
- Run the $H_{\text{if step}}$ for 512 steps with zero-stream, i.e., $H_{\text{if step}}(0)$, for $i = 0, \ldots, 511$.

4.1.2 Processing plaintext The plaintext $M = m_0, m_1, \ldots, m_{mlen-1}$ is used to update the state bit-by-bit and the corresponding ciphertext bit is generated using the function $H_{\text{enc step}}(\cdot)$ (see Algorithm 2).

- $C \leftarrow \epsilon$
- $c_i \leftarrow H_{\text{enc step}}(m_i)$, $C \leftarrow C \cup c_i$, for $i = 0, \ldots, mlen - 1$

4.1.3 Finalization After processing all the plaintext bits, the $H_{\text{if step}}$ runs for 512 times with zero input, and then the tag is generated, i.e., $H_{\text{if step}}(0)$, for $i = 0, \ldots, 511$. 

12
Algorithm 2 Core components of HERN

1: function \texttt{H\_core\_step}: \\
2: \hspace{1em} a \leftarrow \texttt{SB}(s_{030}, s_{029}, s_{132}, s_{124}, s_{231}, s_{24}, s_{315}, s_{314}) \\
3: \hspace{1em} b \leftarrow \texttt{SB}'(s_{030}, s_{029}, s_{132}, s_{124}, s_{231}, s_{24}, s_{315}, s_{314}) \\
4: \hspace{1em} f^0 \leftarrow s^0_0 \oplus s^0_{31} \oplus s^0_{22} \oplus s^1_{13} \\
5: \hspace{1em} f^1 \leftarrow s^1_0 \oplus s^1_{28} \oplus s^1_{30} \oplus s^1_7 \\
6: \hspace{1em} f^2 \leftarrow s^2_0 \oplus s^2_{22} \oplus s^2_{27} \oplus s^2_{26} \\
7: \hspace{1em} f^3 \leftarrow s^3_0 \oplus s^3_8 \oplus s^3_{19} \oplus s^3_{31} \\
8: \hspace{1em} s^i \leftarrow s^i \ll 1, \text{ for } i = 0, 1, 2, 3 \\
9: \hspace{1em} s^i_{63} \leftarrow f^i, \text{ for } i = 0, 1, 2, 3 \\
10: \text{ function } \texttt{SB}(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3): \\
11: \hspace{1em} \text{ return } 1 \oplus x_0 y_0 \oplus x_1 y_1 \oplus x_2 y_2 \oplus x_3 y_3 \\
12: \text{ function } \texttt{SB}'(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3): \\
13: \hspace{1em} \text{ return } x_0 y_2 \oplus y_0 y_3 \oplus x_1 x_3 \oplus y_1 x_2 \\
14: \text{ function } \texttt{Adda}: \\
15: \hspace{1em} s^0_{63} \leftarrow s^0_{63} \oplus a \\
16: \hspace{1em} s^2_{63} \leftarrow s^2_{63} \oplus a \\
17: \text{ function } \texttt{Addb}: \\
18: \hspace{1em} s^3_{63} \leftarrow s^3_{63} \oplus b \\
19: \text{ function } \texttt{H\_if\_step}(x): \\
20: \hspace{1em} \texttt{H\_core\_step} \\
21: \hspace{1em} a \leftarrow a \oplus x \\
22: \hspace{1em} \texttt{Adda} \\
23: \hspace{1em} \texttt{Addb} \\
24: \text{ function } \texttt{H\_enc\_step}(m): \\
25: \hspace{1em} \texttt{H\_core\_step} \\
26: \hspace{1em} a \leftarrow a \oplus m \\
27: \hspace{1em} \texttt{Adda} \\
28: \hspace{1em} c \leftarrow b \oplus m \\
29: \hspace{1em} \text{ return } c \\

- \ T \leftarrow \epsilon \\
- \ t_i \leftarrow \texttt{H\_enc\_step}(0), \ T \leftarrow T || t_i, \text{ for } i = 0, \ldots, mlen - 1 \\
- \ \text{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 $K \leftarrow \{0,1\}^{128}$ be fixed. To construct the forgery we proceed as follows.

**Step 1** Let $N \leftarrow \{0,1\}^{128}$, $AD \leftarrow \{0,1\}^*$ 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 $Q \triangleq (N, AD, M, C, T)$.

**Step 3** For each $i = 0$ to $2^n - 1$, decrypt $(N', AD', C', T')$ where

- $N' = N$
- $AD' = AD\|0^n$
- $C' = \epsilon$
- $T' = T \ll n \ | \ (i_0\|\cdots\|i_{n-1})$, and $(i_0, \cdots, i_{n-1}) \leftarrow i$.

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

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 $Q' \triangleq (N, AD', \epsilon)$. 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 $T' = T \ll n \ | \ \Delta$ where $\Delta$ 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, $H_{\text{if step}(\cdot)}$ runs for 1024 times with zero input. For $Q'$, since $AD' = AD\|0^n$ and $M' = \epsilon$, $H_{\text{if step}(\cdot)}$ 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 $H_{\text{if step}(0)}$ 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, $T' = T \ll n \ | \ \Delta$.
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 $K \leftarrow \{0,1\}^{128}$ be fixed. We construct forgery as follows.

Step 1 Let $N \leftarrow \{0,1\}^{128}$, $AD \leftarrow \{0,1\}^*$, $M \leftarrow \{0,1\}^{2n}$. 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

$N' = N$

$AD' = AD$

$C' = c_0||c_{|M|−n−1}$

$T' = 0^n|T \gg n$.

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 $Q : \defeq (N, AD, M)$ satisfy Step 2 with output as $(C, T)$. Let $AD' = AD$, $M' = m_0||\cdots||m_{|M|−n−1}$ and $Q' : \defeq (N, AD', M')$. Then $T' = 0^n|T \gg n$ 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 $m'_i = m_i \iff c'_i = c_i$, 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:

$t_i = b_{1152+|AD|+|M|+i}$

$t'_i = b'_{1152+|AD|+|M|−n+i}$

Now, the last $n$ bits of both $M$ and $C$ being zero $\iff S'_{1152+|AD|+|M|−n} = S'_{1152+|AD|+|M|−n}$. So, given $b_{1152+|AD|+|M|−n}, \cdots, b_{1152+|AD|+|M|−1}$ are all zero, then $T' = 0^n|T \gg n$. □
**Attack complexities.** Step 2 requires $2^n$ encryption queries (on average), while to satisfy 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 $K \leftarrow \mathbb{Z}_2^{128}$ be fixed. The forgery then proceed as follows.

**Step 1** Let $N \leftarrow \{0, 1\}^{128}$, $AD \leftarrow \{0, 1\}^*$, $M = 0^n$. 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

$$N' = N$$
$$AD' = AD \parallel 0^n$$
$$C' = \epsilon$$
$$T' = T,$$

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 $(\alpha, \alpha x)$ 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

<table>
<thead>
<tr>
<th>Input data</th>
<th>associated data-only</th>
<th>ciphertext-only</th>
<th>associated data and ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>000102030405060708090A0B0C0D0E0F</td>
<td>000102030405060708090A0B0CD0E0F</td>
<td>000102030405060708090A0B0CD0E0F</td>
</tr>
<tr>
<td>$N$</td>
<td>D8A4ADC965EECE563305CC01A53C928</td>
<td>D8A4ADC965EECE563305CC01A53C928</td>
<td>D8A4ADC965EECE563305CC01A53C928</td>
</tr>
<tr>
<td>$AD$</td>
<td>CA5F</td>
<td>CA5F00</td>
<td>CA5F00</td>
</tr>
<tr>
<td>$M$</td>
<td>Empty string</td>
<td>Empty string</td>
<td>Empty string</td>
</tr>
<tr>
<td>$CT$</td>
<td>Empty string</td>
<td>Empty string</td>
<td>Empty string</td>
</tr>
<tr>
<td>$T$</td>
<td>0FC40BF26954B37993E9C56C6C49ACAB6</td>
<td>0FC40BF26954B37993E9C56C6C49ACAB6</td>
<td>0FC40BF26954B37993E9C56C6C49ACAB6</td>
</tr>
</tbody>
</table>

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**


