LOL: A Highly Flexible Framework for Designing Stream Ciphers

Dengguo Feng¹, Lin Jiao¹, Yonglin Hao¹, Qunxiong Zheng², Wenling Wu³, Wenfeng Qi², Lei Zhang³, Litong Zhang⁴, Siwei Sun⁵ and Tian Tian²

¹ State Key Laboratory of Cryptology, Beijing, 100878, China,
² PLA Strategic Support Force Information Engineering University, Zhengzhou, 450001, China
³ Institute of Software, Chinese Academy of Sciences, Beijing, China
⁴ Westone Cryptologic Research Center, Beijing, China
⁵ School of Cryptology, University of Chinese Academy of Sciences, Beijing, China

Abstract. In this paper, we propose LOL, a general framework for designing blockwise stream ciphers, to achieve ultrafast software implementations for the ubiquitous virtual networks in 5G/6G environments and high-security level for post-quantum cryptography. The LOL framework is structurally strong, and all its components as well as the LOL framework itself enjoy high flexibility with various extensions. Following the LOL framework, we propose new stream cipher designs named LOL-MINI and LOL-DOUBLE with the support of the AES-NI and SIMD instructions: the former applies the basic LOL single mode while the latter uses the extended parallel-dual mode. Both LOL-MINI and LOL-DOUBLE support 256-bit key length and, according to our thorough evaluations, have 256-bit security margins against all existing cryptanalysis methods including differential, linear, integral, etc. The software performances of LOL-MINI and LOL-DOUBLE can reach 89 Gbps and 135 Gbps. In addition to pure encryptions, the LOL-MINI and LOL-DOUBLE stream ciphers can also be applied in a stream-cipher-then-MAC strategy to make an AEAD scheme.

Keywords: Stream Cipher · 5G/6G Mobile System · Fast Software Implementation

1 Introduction

Stream ciphers have always been playing important roles in various communication systems protecting integrity and confidentiality. Examples can be seen in the A5/1 stream cipher in GSM [MPH92] and the SNOW3G [SAG06], ZUC-128 [SAG11] in UMTS and LTE (or usually referred to as 4G). The 5G mobile communication system is getting widely used all over the world. In comparison with 4G, the 5G system has a much faster speed with the data rates over 20 Gbps for downlinks and 10 Gbps for uplinks [ITU17]. Such a high speed stimulates the wide use of software-defined networks where all network nodes are virtual and stream ciphers as well as other primitives are implemented on general CPUs rather than dedicated hardware devices, raising higher software speed requirements. From the evolution of mobile communications, a new mobile generation appears approximately every ten years, and research for beyond-5G or 6G has already started. In 2019, 6Genesisi project published the first white paper of 6G [LaL19] raising several requirements for the 6G system including the data transmission speeds over 100 Gbps, up to 1 Tbps, etc. On the other hand, the latest studies in quantum computation reveal that quantum computers are not only a threat to traditional public-key schemes based on discrete logarithms and RSA: it can improve the cryptanalysis results on symmetric-key primitives as well (just
name some as [TTU16, KLLN16, CNS17, DDW20, BES18, NIDI19, LZ19, DSS+20, HS21, HK21, BSS22, SS22]). Therefore, both the software efficiency and the security of stream ciphers as well as other symmetric-key primitives are faced with severe challenges.

Along with the increasing requirement in speeds and securities, there is also development in software implementation technologies. In the past few decades, the idea of Single Instruction Multiple Data (SIMD) has been widely accepted by CPUs manufacturers. Many modern CPUs are now equipped with extended accumulator instruction sets (such as the SSE/SSE2/SSE4.2/AZX/AVX2/AVX512 for Intel and NEON for ARM CPUs etc.) so that several repeated basic operations (such as modular add/minus, XOR, OR, etc.) can be implemented in parallel with only 1 accumulator instruction. Furthermore, with the Advanced Encryption Standard (AES) block cipher being widely used, the AES round function, i.e., a sequential call of a Subbyte (SB), a ShiftRow (SR), a MixColumn (MC) and an AddRoundKey (AK), has also been adopted as a basic operation and can be accomplished with 1 AES-NI instruction supported by most of the modern CPUs. Some SoCs for mobile devices are also equipped with an instruction set for AES [ARM21], and more and more SoCs will support the instruction by the time when 6G system is realized. The latest AVX512 instruction of Intel even supports running 4 AES round function calls in parallel. Therefore, the AES round function has become a handy building block for designing new cryptographic primitives. A typical example is the authentication encryption design called AEGIS [WP14]: it applies the AES round function in parallel for high performances and mounted to the final portfolio of the CAESAR competition [CAE14].

Faced with the higher efficiency and security challenges, the 3GPP standardization organization propose the requirement of 5G stream cipher standards including a speed over 20 Gbps and secure margins of 256 bits [Fen20]. The 4G standards SNOW3G and ZUC-128 do not satisfy the 256-bit secure margin requirement and the designers are urged to submit 256-bit key versions. As a response, the ZUC design team proposed ZUC-256 [dt18] in 2018, which is hardware-oriented and shares the same structure with its predecessor ZUC-128 for compatibility reasons. After that, the SNOW design team gives the software-oriented stream cipher SNOW-V [EJMY19] along with its performance-improved variant SNOW-Vi [EMJY21], which are referred to as SNOW-V/Vi hereafter for short. In 2022, Sakamoto et al. propose another primitive: the authenticated-encryption scheme (AEAD) Rocca [SLN+21] supporting 256-bit keys for encryptions and 128-bit tags for authentications, which is the first dedicated to 6G systems. Same with AEGIS, both Rocca and SNOW-V/Vi adopt the parallel AES round functions called as the basic building block, which enables them to have extremely high software performances: the SNOW-V encryption speed can reach 58 Gbps and that of Rocca is higher than 100 Gbps catering for not only 5G but also 6G requirements [JKK+19].

The parallel AES round function calls provide high software performances on modern CPUs but do not guarantee the security of AES-based primitives. Some cryptanalysis results indicate that the aforementioned AEGIS, Rocca and SNOW-V/Vi have some structural and theoretical security issues. The simple output function of AEGIS results in linear biases in keystream bits making AEGIS vulnerable to linear distinguishers [WP14, ENP19]. It also result in weak key recovery attacks mounting to half of the total initialization rounds [LIMS21]. On full SNOW-V/Vi, there are fast correlation attacks (FCA) recovering the internal states within $2^{256}$ secure bounds [SJZ+22, ZFZ22]. For Rocca, there is a full key-recovery attack utilizing encryption-decryption oracles for finding nonce-repeated pairs with the differential properties of AES S-box and the meet-in-the-middle techniques against AES for low-complexity state recoveries in a single-key and nonce-respecting setting [HII+22].

\footnote{The designers of Rocca then provide a modified version [SLN+22] that key feedforward is added in initialization, which turns the key-recovery attack [HII+22] into a state recovery attack.}
Motivations. Plenty of lessons can be learnt from the development of SNOW-V/Vi, Rocca and AEGIS. From SNOW-V/Vi, we can see that the direct LFSR-FSM connection can easily result in high linear correlations leaving it easy to conduct FCAs. Such a phenomenon has already been detected and utilized in the FCAs on SNOW 2.0 and SNOW 3G [NW06, ZXM15, GZ21]. From Rocca and AEGIS, we find that the differential properties and effective cryptanalysis techniques on AES can be even more effective on designs with parallel AES round function calls. Therefore, we need to pay special attention to the secure design of the overall structure, adding non-AES components properly, such as NFSRs, to resist these attacks. Meanwhile, comparing Rocca and SNOW-V/Vi, the much faster speed of Rocca than SNOW-V/Vi indicates that the non-AES components should be better designed to improve the overall software efficiency of stream ciphers. On the other hand, Rocca tries to design an AEAD scheme directly while SNOW-V/Vi are simply conventional stream cipher designs suitable for the existing GCM-like scheme [Dwo07b] for authentications. The attack on Rocca in [HHH+22] is based on repeated nonce pairs acquired from querying decryption oracles, which cannot be applied to the well-developed GMAC-based AEAD mode of SNOW-V/Vi. Thus, directly designing a secure AEAD may be even faster but more challenging than the relatively mature stream-cipher-then-GMAC strategy. Besides, to achieve such a dramatically fast encryption/decryption speed for a pure software environment in beyond-5G or 6G mobile system, the support from AES round function with SIMD instructions has become a must for high software performances on modern CPUs before more basic instructions appear.

Contribution. In this paper, we provide a general framework for designing blockwise stream ciphers as well as construction methods of its basic components suitable for SIMD implementations. We name our framework as LOL (League of Legends), which is extendable from a basic single mode with various extension methods for plenty of flexible designs. The LOL framework structure is a combination of a nonlinear driver on finite fields and an FSM with memories, avoiding the direct LFSR-FSM-connection weakness in SNOW-V/Vi. All the underlying components of the LOL framework also enjoy high flexibility and are equipped with specific extension methods catering to different needs. Especially, the LFSR in LOL suits perfectly for the SIMD instructions enabling high efficiencies, resourceful parameter selections, and maximum periods. The updating functions of the nonlinear driver and the FSM in the LOL framework are both constructed on SP-network, which makes LOL stream ciphers directly inherit some security properties from structural conclusions and, more effectively and favorably be evaluated with definitive security bounds by automatic analysis methods compared with previous stream ciphers. Following such LOL framework, we provide 2 concrete stream cipher designs namely LOL-MINI of the LOL single mode, and LOL-DOUBLE of extended parallel-dual mode. Both designs support the 256-bit key length and have software encryption speeds well over 20 Gbps which are 89 Gbps for LOL-MINI and 135 Gbps for LOL-DOUBLE. From the security aspect, LOL-MINI and LOL-DOUBLE take the first attempt to load the key and IV bits to FSM rather than NLFSRs enabling us to give provable secure bounds against differential attacks. Besides, our thorough evaluations further prove that the LOL-MINI and LOL-DOUBLE stream ciphers can provide 256-bit secure margins against various other cryptanalysis methods including FCAs, integral, guess-and-determine, etc. To sum up, the LOL-MINI and LOL-DOUBLE provide 256-bit secure margins and high software performances satisfying the 3GPP 5G requirements, even targeting 6G systems. In addition to pure encryptions, LOL stream ciphers can also be applied in AEAD schemes using the stream-cipher-then-MAC strategy where MAC tags are generated with standard authentication modes such as GCM[Dwo07a] and NMH[HK97] etc. With provable length-independent securities [MV04], such AEAD schemes may support mutable tag lengths.
We apply LOL-MINI and LOL-DOUBLE to 128-bit-tag GCM and NMH and evaluate their software performances. It turns out that, LOL-MINI enjoys a higher GCM speed of 43.42 Gbps while LOL-DOUBLE is more suitable for NMH reaching 47.05 Gbps. On the contrary, the SNOW-V-GCM and SNOW-V-NMH speeds are 28.23 and 21.98 Gbps on the same platform.

It is also noticeable that, although our current designs are software-oriented, we have also considered hardware compatibility from the perspective of circuit reuse in the architecture and component extensions, because the reusability of hardware components is important to reducing the area and cost of ASICs.

Outline. This paper is organized as follows. In Section 2, we give a description to the LOL framework. Then, we detail the LOL-MINI and LOL-DOUBLE stream cipher designs in Section 3 and Section 4 respectively. After that, we evaluate the security of LOL-MINI and LOL-DOUBLE against various cryptanalysis methods in Section 5. The software performance is evaluated in Section 6 before we finally conclude the paper in Section 7. The AEAD mode applications and the hardware implementations are given in supplementary materials as Appendix A and Appendix B respectively.

2 The LOL Framework

The LOL framework is a combination of a nonlinear driver on finite fields and a Finite State Machine (FSM) with memories shown in Fig. 1. The nonlinear driver is in the form of a linear feedback shift register (LFSR) in series with a nonlinear feedback shift register (NFSR) shown in the dotted box of Fig. 1.

![Figure 1: The form of cipher structure](image)

The following notations are used hereafter.

- Denote the state of LFSR by \((H,L)\), the state of NFSR by \(N\), the state of FSM by \(S\), the keystream block by \(Z\).

- The subscript corresponds to the state number and the superscript corresponds to the step.

- Denote the state updating function of LFSR and NFSR (FSM) by \(f\) and \(R\), respectively.

- Define two intermediate variables: the feedback of LFSR as \(F\) and the output of FSM as \(G\).

2.1 Nonlinear Driver

The LOL nonlinear driver takes the concatenated structure of an LFSR and an NFSR, which can benefit from both long periods and nonlinearities. As components of the
nonlinear driver, we have selected LFSRs and NFSRs with the following features.

Figure 2: The Feistel-like structure of LFSR updating function

Figure 3: The SP-network of $R$ function

**LFSR.** We first present the description of the LFSR design.

1. The LFSR is of Galois type and defined on the finite field $\mathbb{F}_{2^m}$.

2. The LFSR consists of an even number $2\ell$ ($\ell > 0$) $m$-bit cells denoted as $a_{2\ell-1}, \ldots, a_0 \in \mathbb{F}_{2^m}$. At step $t$, the $2\ell$ cells are categorized into 2 vectors according to their parities as $L^t = (a_{2\ell-2}, \ldots, a_2, a_0)$ for even-numbered cells and $H^t = (a_{2\ell-1}, \ldots, a_3, a_4)$ for the odd-numbered ones.

3. The updating function of LFSR utilizes a Feistel-like structure shown in Fig. 2. The feedback function of LFSR at step $t$ is

$$F^t = f(H^t, L^t) = (C \times H^t) \oplus \sigma(L^t),$$

where $C$ denotes a vector consisting of the roots of $\ell$ irreducible polynomials$^2$ in $\mathbb{F}_2[x]$ with degree $m$, $\times$ denotes a cell-to-cell finite field multiplication on $\mathbb{F}_{2^m}$, $\sigma$ denotes a cell-wise permutation. Here, XOR integrates different expansion fields by the polynomial basis correspondence on $\mathbb{F}_{2^m}$. Then the LFSR is updated as follows:

$$\begin{cases}
H^{t+1} = F^t = (C \times H^t) \oplus \sigma(L^t), \\
L^{t+1} = H^t.
\end{cases}$$  \hspace{1cm} (1)

4. The LFSR updates half of the state ($H$) according to the feedback as shown above and outputs half of the state ($L$) as the LFSR sequence at each step for high throughputs. Specifically for the suitability of SIMD implementation, it uses a 256-bit register as a benchmark and outputs at least 128 bits at each step.

We now discuss how to prove the maximum period of LFSR (m-sequence): an essential property to resist Berlekamp-Massey attacks and correlation attacks [Rue12]. The core idea is to (1) derive the LFSR generating matrix according to its updating function; (2) verify that the corresponding characteristic polynomial is primitive.

**Problem 1.** Given the specific irreducible polynomials of $C \times$ and vector permutations $\sigma$, how to prove that the LFSR defined in Eq. (1) has a maximum period of $2^{2\ell m} - 1$?

We first derive the generator matrix of LFSR. According to Eq. (1), the generator matrix can be derived as follows:

$$(H^{t+1}, L^{t+1}) = (H^t, L^t) \cdot G$$

$^2$These irreducible polynomials can be the same or different.
where the generator matrix $G$ consists of $4 \ell m \times \ell m$ blocks as follows:

$$G = \begin{pmatrix} M_C & I_{\ell m} \\ M_G & 0 \end{pmatrix}.$$  

Hereafter, we denote $I_i$ as the $i$-dimensional unit matrix. $M_{\sigma}$ corresponds to the $\sigma$. Since $\sigma$ is a permutation of $\ell$ $m$-bit words, $M_{\sigma}$ consists of $\ell \times \ell$ $m$-dimensional blocks: the block entries at positions $(\sigma(i), i)$ ($i = 0, \ldots, \ell - 1$) equal to $I_m$ while others are 0. $M_C$ corresponds to $C \times$. It is a diagonal block matrix of the form

$$M_C = \begin{pmatrix} M_{a_0} & 0 & \cdots \\ & & & \\ 0 & & & M_{a_{\ell-1}} \end{pmatrix},$$

where $a_i$ is the root of the $m$-degree irreducible polynomial $g_i(y)$ of the following form ($i = 0, \ldots, \ell - 1$):

$$g_i(y) = y^m + b_{i,m-1} \cdot y^{m-1} + \cdots + b_{i,1} \cdot y + b_{i,0} \in \mathbb{F}_2[y].$$

$g_i(y)$ defines the finite field $\mathbb{F}_{2^m} = \mathbb{F}_2[y]/(g_i(y))$ with basis $(\alpha_i^{m-1}, \ldots, a_i, 1)$ enabling us to represent arbitrary $m$-bit cell $a_{2i+1} = (a_{i,m-1} \parallel \cdots \parallel a_1, a_0)$ with a $\mathbb{F}_{2^m}$ element of the form

$$a_{2i+1} = (a_{i,m-1}, \ldots, a_1, a_0)(\alpha_i^{m-1}, \ldots, a_i, 1)^T.$$  

The definition of $M_{a_i}$ should make sure

$$a_0 a_{2i+1} = a_{i,m-1} a_{i}^m + \cdots + a_i a_i + a_{i,0} a_i,$$

$$= a_{i,m-1}(b_{i,m-1} a_{i}^{m-1} + \cdots + b_{i,1} a_{i} + b_{i,0}) + a_{i,m-2} a_{i}^{m-1} + \cdots + a_{i,1} a_i^2 + a_{i,0} a_i,$$

$$= (a_{i,m-1}, b_{i,m-1} + a_{i,m-2}) a_{i}^{m-1} + \cdots + (a_{i,m-1}, b_{i,1} + a_{i,0}) a_i + a_{m-1,i} b_{i,0},$$

$$= (a_{i,m-1}, \cdots, a_1, a_0) M_{a_i}(\alpha_i^{m-1}, \ldots, a_i, 1)^T,$$

so $M_{a_i}$ is defined as

$$M_{a_i} = \begin{pmatrix} b_{i,m-1} & b_{i,m-2} & \cdots & b_{i,1} & b_{i,0} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}.$$  

The characteristic polynomial of $G$ is then deduced as $\eta(x) = \det(xI_{2\ell m} - G)$ and the maximum period can be guaranteed when $\eta(x)$ is primitive: the primitivity of $\eta(x)$ can be verified with off-the-shelf mathematic softwares such as NTL and Maple.

For concrete $\ell, m$ settings, we can randomly generate $(C \times, \sigma)$’s and identify the ones guaranteeing maximum period $2^{2\ell m} - 1$ with the above methods. The whole process is summarized in Algorithm 1. Such an algorithm can be quite efficient in practice. We implement Algorithm 1 with NTL finding that in the cases of $m = 16, \ell = 8$ or $m = 16, \ell = 16$, the feasible $(C, \sigma)$ can be found after hundreds of iterations. The source codes are attached and will be open source.

Our LFSR design has the following advantages:

\footnote{https://libntl.org/}

\footnote{https://maple-17.en.softonic.com/}
Algorithm 1: Construct a $2^\ell m$-bit LFSR (consists of $2\ell$ m-bit cells) with a maximum LFSR period of $2^{2\ell m} - 1$

**Input:** The parameters $m$ and $\ell$

**Output:** $(C, \sigma)$ guaranteeing the maximum period of $2^{2\ell m} - 1$

1. Define the flag $f = 0$;
2. while $f = 0$ do
3. Define $C$ with $\ell$ randomly generated $m$-degree irreducible polynomials $g_0, \ldots, g_{\ell - 1}$;
4. Define $\sigma$ as a randomly chosen $\ell$-degree permutation over $[0, \ell - 1]$;
5. According to $(C, \sigma)$, deduce the generator matrix $G$ along with its characteristic polynomial $\eta(x)$;
6. If $\eta(x)$ is primitive, set $f = 1$;
7. Return $(C, \sigma)$.

**Software Friendly** The LFSR applies to software-oriented parallel implementations. It updates the cells in parallel by multiplications over multiple expansion fields generated by several irreducible polynomials with the same degree, which supports all the components of $C \times H^t$ to implement by the left-shift-then-XOR operations in parallel on platforms supporting AVX. This design strategy avoids the computations of both $a \times x$ and $a^{-1} \times x$ over the same field as that in SNOW-V, which cannot be implemented together for different shift directions over the same field. Moreover, it supports executing $\times$ and $\sigma$ operations synchronously.

**Highly Flexible** This series of LFSRs have a enriched selection of $(C, \sigma)$ resources with a maximum periods. Previous stream ciphers usually used Fibonacci LFSRs that update each cell sequentially by the same feedback function. Such designs usually result in lower parallelization and efficiency since they cannot fully cultivate the power of SIMD-supported modern CPUs. To solve this problem, some parallelized extension of large-bit words but also in the same sequentially update mode has been proposed, such as the LFSR in SNOW-V/Vi. However, the available tapping selection ranges for constructing this kind of LFSRs are not abundant. In our design, the design space is enlarged by choosing Galois LFSR with a permutation-based connection and allowing to define finite fields with different irreducible polynomials. Thus, it supports the hardware-friendly expansion of the LOL family with plenty of circuit reuse strategies.

In summary, our newly proposed LFSR designing method has perfectly addressed the requirements of large-bit width, high parallelism, high speed, and high throughput for software implementation while taking into account the maximum period for security, and providing a resourceful solution.

**NFSR.** The description of the NFSR design is as follows:

1. The NFSR consists of one register $N$. Specifically for the suitability of SIMD implementation, it uses a 128-bit register as a benchmark.

2. The updating function of NFSR utilizes an SP-network function $\mathcal{R}$ shown in Fig. 3, which generates the NFSR state at the next step by operating on the entire NFSR directly and taking the output of LFSR as the subkey. The original NFSR state is used as the output. The updating function of NFSR at step $t$ is

$$N^{t+1} = \mathcal{R}(N^t) \oplus L^t.$$
The flexibility of NFSR lies in the definition of its SP-network updating function and its length that can be modified according to different efficiency/security requirements.

For the period of the whole nonlinear driver, the state equality \((H^t || L^t, N^t) = (H^{t+T} || L^{t+T}, N^{t+T})\) can only happen when \(N^t = N^{t+T}\) with birthday collision as well as the LFSR reaches an integral period. Thus the average period of the concatenated LFSR and NFSR is about \(2^{\lceil N/2 \rceil} \times (2^{2m} - 1)\), at least not less than the LFSR period. The large period of the NLFSR and the randomness of the nonlinear round function work together to ensure the unpredictability of the keystream.

2.2 FSM

The FSM utilizes parallel SP-network functions inspired by [WP14] to quickly implement diffusion and confusion through two types of components: connectors and plugins.

1. The FSM consists of \(n\) memory cells \((S_0, S_1, \ldots, S_{n-1})\).
2. The connector is at both ends of the FSM for connection with the nonlinear driver and further extensions, whose specific structure is shown in Fig. 4 (1).
3. The plugins are in the middle of FSM (i.e., the dotted box of Fig. 4(1)) and can be added or removed according to the needs of security and efficiency. The specific structure is shown in Fig. 4 (2).
4. The \(R\) functions used in the FSM are also with SP-network shown in Fig. 3, which take adjacent memory cells of the FSM as inputs and subkeys as follows

\[
S_{i+1}^t = R(S_{i-1}^t) \oplus S_i^t,
\]

where \(1 \leq i \leq n - 1\). Note that all \(R\) functions are not necessarily the same: different \(R\) definitions can be used simultaneously in one LOL framework depending on the application requirements equipping the designers with plenty of freedom. Specifically for extremely high software speeds, the whole Eq. (2) can be set as a standard AES round function that can be accomplished with a single SIMD instruction on many modern CPUs like Intel and AMD (AES-NI), and so far as to some SoCs for mobile devices.

The nonlinear driver and FSM are linked by the connector of FSM as shown in Fig. 1, where the feedback of LFSR is also the input of the connector, and the output of NFSR is combined with the output of the FSM to generate the keystream blocks.

![Figure 4: The connector and plugin of the FSM](image)

Specifically, at the tail end of FSM, one \(R\) function operates on the last memory cell \((S_{n-1})\) and generates an intermediate variable \((G)\), denoted as the output of the FSM.
The keystream block \((Z)\) is generated by XORing \(G\) with the output of NFSR \((N)\). At the head end of FSM, the first memory cell \((S_0)\) updates itself by XORed with the feedback of LFSR \((F)\) and the intermediate variable \((G)\), shown as follows:

\[ G^t = R(S_{n-1}^t), \quad S_0^{t+1} = S_0^t \oplus F^t \oplus G^t, \quad Z^t = G^t \oplus N^t. \]

Such a nonlinear driver connected with an FSM makes up a single mode of the LOL framework.

### 2.3 Expansion Method

In Section 2.1 and Section 2.2, we have seen plenty of flexibilities in the underlying components of the LOL framework such as the LNFSR size, the \(R\) function definitions, the number of plugins in FSM, etc. Furthermore, the overall structure of the LOL framework is also expandable. For example, in order to increase the encryption throughput, we can expand the basic single mode above to a parallel extended mode using the following expansion methods:

1. To construct an extended nonlinear driver, we recombine the LFSRs from \(\gamma\), \(\gamma \geq 2\) single modes as a whole and redefine the updating function in Eq. (1) to maintain the Feistel-like structure with a maximum period. The output of the extended LFSR is of \(\gamma\) times the length in the single mode, which is divided and fed to each NFSR in the single mode. The NFSR update process remains unchanged. Example can be seen in Fig. 5 where 2 single mode nonlinear drivers \((H_0, L_0, N_0)\) and \((H_1, L_1, N_1)\) are reorganized by redefining the extended LFSR \((H, L)\) composed of

\[ L = (L_1, L_0), \quad H = (H_1, H_0), \]

with a new updating function of

\[ F^t = f(H^t, L^t) = (C \times H^t) \oplus \sigma(L^t), \quad H^{t+1} = F^t, \quad L^{t+1} = H^t. \]

All that the designers have to do is selecting proper \((C, \sigma)\) for the larger \((H, L)\) to keep the maximum period. According to our experiments, such \((C, \sigma)\) are so abundant that we only need to define a \(\sigma\) for the large \(C\) as an expansion of the smaller \(C\)s used in \((H_1, L_1)\) or \((H_0, L_0)\). It will be illustrated by the following LOL cipher instantiations in detail. The outputs of the extended LFSR \(L = (L_1, L_0)\) are divided and fed to the original NFSRs \(N_0\) and \(N_1\) naturally:

\[ N_0^{t+1} = R(N_0^t) \oplus L_0^t, \quad N_1^{t+1} = R(N_1^t) \oplus L_1^t. \]
2. To construct an extended FSM with larger throughputs, we can link the connectors from \( \gamma \) single modes end to end in a circle, each feeding into the other. The feedback of the extended LFSR is also divided into \( \gamma \) parts and serves as the inputs of the connectors. The updates of the plugins in the FSM are unchanged. The output of each FSM in the single mode remains and still XORed with the output of the corresponding NFSR, which are concatenated together to generate the keystream block. An example composed of the FSMs from 2 single mode can be seen in Fig. 6. They are connected and updated by

\[
G_0 = R(S_{n_1-1}^t), \quad G_1 = R(S_{n_1+n_2-1}^t),
\]

\[
F^t = (F_{0t}, F_{1t}), \quad S_{0t+1}^t = F_{0t}^t \oplus G_{1t}^t \oplus S_{0t}^t, \quad S_{1t+1}^t = F_{1t}^t \oplus G_{0t}^t \oplus S_{n_1}^t,
\]

\[
S_{i+1}^t = R(S_{i-1}^t) \oplus S_{i}^t, \quad i = 1, \ldots, n_1 - 1, n_1 + 1, \ldots, n_1 + n_2 - 1.
\]

3. The extended nonlinear driver and FSM are then linked in the same form as Fig. 1 with the taps for \( F, G \) and \( N \) corresponding to each single mode so as to generate \( \gamma \) times the length of keystream blocks and provide higher encryption speeds. For the extension of 2 single mode, the output keystream block is generated by

\[
(Z_{0t}, Z_{1t}) = (G_{0t}^t \oplus N_{0t}^t, G_{1t}^t \oplus N_{1t}^t).
\]

The parallel expanded mode above can be regarded as a parallel combination of several single mode structures. The specific combination of the nonlinear driver in Fig. 5 and the FSM in Fig. 6 is named specifically as the “parallel-dual mode” since it is a combination of 2 single modes. There are also other diverse extended modes enriching the LOL framework catering to various efficiency and security demands.

### 2.4 Design Rationale

As can be seen, the LOL framework adapts from the classical source-filter, generator-with-memory stream cipher models. It integrates the underlying nonlinear function with the classical SP-network of block ciphers, and connects the blockwise components with XOR operations.

The security evaluations of LOL stream ciphers can be carried out easily because the off-the-shelf security proofs and cryptanalysis techniques of the underlying components can be directly applied to LOL primitives. With S-box being the unique nonlinear operation, the LOL framework is more friendly to automatic cryptanalysis tools than many other stream ciphers such as the SNOW family (with the modular adds) and the Grain family (bit oriented). With drawing secure bounds against most cryptanalysis methods equivalent to lower bounding the number of (active) S-boxes involved, the LOL stream cipher designers can easily determine the crucial parameters such as initialization round number, the key/IV placement etc. As to component selection, the LOL framework assembles the advantages of existing designs such as SNOW, Grain, and AEGIS, while avoiding their disadvantages: the nonlinear driver avoids the high linear correlation of SNOW; the large-memory FSM avoids the dedicated FCA on Grain; the LNFSR-FSM combinations avoids the linear distinguishing attacks on AEGIS. Besides, the blockwise structure and non-AES components design strategy make LOL ciphers highly flexible and parallelable, resulting in software efficiency on modern CPUs.

To sum up, the LOL framework provides with highly flexible components, software efficient structure and various expansions. It also enjoys hospitality to cryptanalysis for drawing secure bounds as well. In other words, LOL framework is friendly to not only ordinary users but also stream cipher designers as well.
2.5 Instantiations

In the remainder of this paper, we present 2 stream cipher instances following the LOL framework. The 2 stream ciphers are named LOL-MINI and LOL-DOUBLE respectively: the former takes the LOL single mode while the latter follows the parallel-dual mode. We first present some universal preliminaries here.

- According to our experiments, 16-bit cells usually make LFSRs with higher efficiencies in comparison with the 32/64-bit counterparts. So we select $m = 16$.

- The ciphers use 128-bit registers as storage units in the description.

- State correspondence: a 128-bit register state $S$ can be divided into eight 16-bit words $w_7, \ldots, w_0$ where $w_0$ is the least significant word, or into sixteen 8-bit bytes $b_{15}, \ldots, b_0$ where $b_0$ is the least significant byte, i.e. $S = (w_7, \ldots, w_0) = (b_{15}, \ldots, b_0)$.

- $R$ functions are all specified as a compound of AES SubByte (SB), ShiftRow (SR), and MixColumn (MC) operations (equivalent to the AES round function without AddRoundKey (ARK)), i.e.

$$R(S) = MC \circ SR \circ SB(S).$$

The mapping between a 128-bit register state $S = (w_15, \ldots, w_0)$ and the $4 \times 4$-byte state array of the AES round function is as follows

$$S = \begin{pmatrix} w_0 & w_4 & w_8 & w_{12} \\ w_1 & w_5 & w_9 & w_{13} \\ w_2 & w_6 & w_{10} & w_{14} \\ w_3 & w_7 & w_{11} & w_{15} \end{pmatrix}$$

(3)

3 LOL-MINI Cipher

LOL-MINI cipher adopts the single mode of LOL framework, which supports a 256-bit key and a 128-bit initialization vector (IV). We denote the 256-bit master key as $K = (K_h, K_l)$ and the 128-bit IV as $IV$. The internal state of LOL-MINI cipher consists of six 128-bit registers: 2 for the LFSR denoted as $(H, L)$, 1 for the NFSR $N$ and 3 for the FSM $(S_0, S_1, S_2)$ (corresponding a connector with 1 plugin or equivalently $n = 3$ for Fig. 4). LOL-MINI cipher outputs a 128-bit keystream block at each step.

3.1 Nonlinear Driver

The LFSR of LOL-MINI cipher consists of 16 16-bit cells denoted as $(a_{15}, \ldots, a_1, a_0)$. The 16 cells are stored in the two 128-bit registers $(H, L)$ according to the parities of their subscripts as follows:

$$H = (a_{15}, \ldots, a_3, a_1), \quad L = (a_{14}, \ldots, a_2, a_0).$$
Let \( \alpha_0, \alpha_1, \ldots, \alpha_7 \) be the roots of the irreducible polynomials \( g_0(y), g_1(y), \ldots, g_7(y) \in \F_2[y] \) with algebraic degree 16, respectively,
\[
\begin{align*}
g_0(y) &= y^{16} + y^{13} + y^{12} + y^{10} + y^8 + y^7 + y^6 + y^3 + 1, \\
g_1(y) &= y^{16} + y^{15} + y^{12} + y^{10} + y^8 + y^5 + y^3 + y + 1, \\
g_2(y) &= y^{16} + y^{15} + y^{14} + y^{12} + y^{10} + y^7 + y^5 + y^4 + 1, \\
g_3(y) &= y^{16} + y^{14} + y^{11} + y^9 + y^7 + y^4 + y^2 + 1, \\
g_4(y) &= y^{16} + y^{15} + y^{13} + y^9 + y^7 + y^4 + 1, \\
g_5(y) &= y^{16} + y^{14} + y^{13} + y^{12} + y^{10} + y^9 + y^7 + y^6 + y^3 + y^2 + 1, \\
g_6(y) &= y^{16} + y^{15} + y^{13} + y^9 + y^8 + y^4 + y^3 + y + 1, \\
g_7(y) &= y^{16} + y^{14} + y^{13} + y^{12} + y^{11} + y^{10} + y^7 + y^5 + 1.
\end{align*}
\]

Let \( C = (\alpha_7, \ldots, \alpha_1, \alpha_0) \) and define the operation \( C \times : \F_{2^{16}}^8 \rightarrow \F_{2^{16}}^8 \) as
\[
(x_7, \ldots, x_1, x_0) \overset{C \times}{\mapsto} (\alpha_7 \cdot x_7, \ldots, \alpha_1 \cdot x_1, \alpha_0 \cdot x_0),
\]
where \( \alpha_i \cdot x_i \) denotes the multiplication over the finite field \( \F_{2^{16}} = \F_2[y]/g_i(y) \) defined by the polynomial \( g_i(y) \) in Eq. (4) \((0 \leq i \leq 7)\).

Let the permutation \( \sigma : \F_{2^{16}}^8 \rightarrow \F_{2^{16}}^8 \) be defined by
\[
(x_7, \ldots, x_1, x_0) \overset{\sigma}{\mapsto} (x_5, x_0, x_3, x_6, x_4, x_7, x_2, x_1)
\]
Then, we can define the LFSR feedback function \( f : \F_2^2 \times \F_2^2 \rightarrow \F_2^2 \) as
\[
F = f(X, Y) = (C \times X) \oplus \sigma(Y).
\]

With the \( H, L \) at step \( t \) denoted as \( H^t = (a_1^{t+1}, \ldots, a_5^{t+1}, a_1^t) \) and \( L^t = (a_1^t, \ldots, a_2^t, a_0^t) \), the LFSR updating function of LOL-MINI at step \( t \) can be defined as Eq. (6), which is identical to the Feistel-like structure in Fig. 2
\[
F^t = f(H^t, L^t), \quad H^{t+1} = F^t, \quad L^{t+1} = H^t,
\]

The 128-bit \( F^t \) represents the feedback of LFSR at step \( t \). As is proved in Section 5.1, the LFSR of LOL-MINI cipher is with a period of \( 2^{256} - 1 \). The overall structure of such Galois LFSR is shown in Fig. 12 in Appendix C.

For the NFSR of LOL-MINI cipher, let \( N \) at step \( t \) be \( N^t \), which maps to the state array of the AES round function as Eq. (3). The NFSR is update by \( \mathcal{R} \) as follows
\[
N^{t+1} = \mathcal{R}(N^t) \oplus L^t.
\]

### 3.2 Updating Function

The LOL-MINI updating functions of the initialization and the keystream generation phases are slightly different. The keystream generation phase starts from the \( t = 0 \) step and outputs the 128-bit keystream blocks \( Z^t \)’s for \( t \geq 0 \). The initialization phase steps are with \( \text{Round}_{\text{load}} \leq t < 0 \) where \( \text{Round}_{\text{load}} = -12 \).

We demonstrate the two phases as follows:

**Keystream generation phase** For \( t \geq 0 \),

1. Compute the intermediate 128-bit variables: \( G^t = \mathcal{R}(S_2^t) \), \( F^t = f(H^t, L^t) \).
2. Compute and output the 128-bit keystream block: \( Z^t = G^t \oplus N^t \).
3. Update the nonlinear driver: \( N^{t+1} = \mathcal{R}(N^t) \oplus L^t \), \( H^{t+1} = F^t \), \( L^{t+1} = H^t \).
4. Update the FSM: \( S_1^{t+1} = F^t \oplus G^t \oplus S_0^t \), \( S_2^{t+1} = \mathcal{R}(S_0^t) \oplus S_1^t \), \( S_2^{t+1} = \mathcal{R}(S_1^t) \oplus S_2^t \).

The schematic of LOL-MINI cipher in the keystream generation phase is shown in Fig. 7.
Initialization phase For $Round_{load} \leq t < 0$,

1. Compute the intermediate 128-bit variables: $G^t = R(S^t_2)$, $F^t = f(H^t, L^t)$.

2. Compute the 128-bit keystream block $Z^t = G^t \oplus N^t$ not for output but for feeding back to the nonlinear driver.

3. Update the nonlinear driver with $Z^t$ block fed into both the LFSR and the NFSR:

   $$L^{t+1} = H^t, \quad H^{t+1} = \begin{cases} F^t \oplus Z^t, & Round_{load} \leq t < -1 \\ F^t \oplus Z^t \oplus K_h, & t = -1 \end{cases}$$
   $$N^{t+1} = R(N^t) \oplus L^t \oplus Z^t,$$

4. Update the FSM:

   $$S^{t+1}_1 = R(S^t_0) \oplus S^t_1, \quad S^{t+1}_2 = R(S^t_1) \oplus S^t_2,$$
   $$S^{t+1}_0 = \begin{cases} F^t \oplus G^t \oplus S^t_0, & Round_{load} \leq t < -1 \\ F^t \oplus G^t \oplus S^t_0 \oplus K_l, & t = -1 \end{cases}$$

As can be seen, the 256-bit master key is reloaded to the internal states at the final initialization $t = -1$ step: the higher 128-bit $K_h$ is injected to the LFSR state $H$ and the lower $K_l$ to the FSM state $S_0$. The schematic of LOL-MINI cipher in the initialization phase is shown in Fig. 8.

LOL-MINI consists of 12 initialization rounds so there is a $Round_{load} = -12$ setting. At the final initialization round ($t = -1$), LOL-MINI adopts the FP-(1) mode [HK18] by XORing the master key to the internal states of $H$ and $S_0$ again.

Same with the existing stream cipher designs, we also limit the keystream length to a maximum of $2^{64}$ for a single pair of key and IV, and each key may be used with a maximum of $2^{64}$ different IVs for the sake of security.
4 LOL-DOUBLE Cipher

LOL-DOUBLE cipher supports a 256-bit key and a 256-bit IV, denoted as $K = (K_0, K_1)$ and $IV = (IV_0, IV_1)$, respectively. The internal state of LOL-DOUBLE cipher consists of ten 128-bit registers in total. Here, two 256-bit LFSRs of $(H_0, L_0)$ and $(H_1, L_1)$ jointly form the extended 512-bit LFSR $(H, L)$ with a period of $2^{512} - 1$. There are two 128-bit NFSRs of $N_0$ and $N_1$. The two FSMs, each having two 128-bit registers of $(S_0, S_1)$, $(H_1, L_1, N_1, S_2, S_3)$ form the circular in Fig. 6 (equivalent to the $n_1 = n_2 = 2$ setting with no plugins). As can be seen, LOL-DOUBLE follows exactly the parallel-dual mode of the LOL framework: the combinations $(H_0, L_0, N_0, S_0, S_1)$, and $(H_1, L_1, N_1, S_2, S_3)$ form 2 standard LOL single modes accordingly; the 256-bit output keystream block of LOL-DOUBLE is the concatenation of the two 128-bit output keystream blocks from the 2 LOL single modes $(Z_0, Z_1)$.

4.1 Nonlinear Driver

The LFSR of LOL-DOUBLE cipher consists of 32 16-bit cells denoted as $(a_{31}, \ldots, a_0)$. The 32 cells are stored in two 256-bit registers $(H, L)$ (or equivalently 4 128-bit registers $H_1, H_0, L_1, L_0$) according to the parities of their subscripts as follows:

$$H = (H_1, H_0) = ((a_{31}, \ldots, a_{17}), (a_{15}, \ldots, a_3, a_1)),$$

$$L = (L_1, L_0) = ((a_{30}, \ldots, a_{16}), (a_{14}, \ldots, a_2, a_0)).$$

Let $a_0, a_1, \ldots, a_{15}$ be the roots of the sixteen irreducible polynomials $g_0(y), g_1(y), \ldots, g_{15}(y) \in \mathbb{F}_2[x]$ with algebraic degree 16 respectively, where $g_0(y), g_1(y), \ldots, g_7(y)$ are the same with those in Eq. (4) and $g_8(y), g_9(y), \ldots, g_{15}(y)$ are defined as Eq. (8).

$$g_0(y) = y^{16} + y^{15} + y^{14} + y^{10} + y^8 + y^6 + y^4 + y + 1$$

$$g_1(y) = y^{16} + y^{14} + y^{13} + y^{12} + y^{11} + y^{10} + y^8 + y^7 + y^6 + y^4 + y^2 + 1$$

$$g_{10}(y) = y^{16} + y^{11} + y^{10} + y^8 + y^7 + y + 1$$

$$g_{11}(y) = y^{16} + y^{15} + y^{13} + y^{12} + y^9 + y^7 + y^6 + y^5 + y^3 + y + 1$$

$$g_{12}(y) = y^{16} + y^{15} + y^{14} + y^{12} + y^{10} + y^8 + y^5 + y^3 + y^2 + y + 1$$

$$g_{13}(y) = y^{16} + y^{15} + y^{12} + y^{11} + y^{10} + y^9 + y^8 + y^7 + y^6 + y^4 + y^2 + y + 1$$

$$g_{14}(y) = y^{16} + y^{14} + y^{10} + y^9 + y^6 + y^5 + 1$$

$$g_{15}(y) = y^{16} + y^{15} + y^{14} + y^{12} + y^6 + y^5 + y^3 + 1$$

Let $C = (a_{15}, \ldots, a_1, a_0)$ and define the operation $C \times : \mathbb{F}_{2^{16}}^* \rightarrow \mathbb{F}_{2^{16}}^*$ as

$$(x_{15}, \ldots, x_1, x_0) \xrightarrow{C \times} (a_{15} \cdot x_{15}, \ldots, a_1 \cdot x_1, a_0 \cdot x_0),$$

where $a_i \cdot x_i$ denotes the multiplication over the finite field $\mathbb{F}_{2^{16}} = \mathbb{F}_2[x]/g_i(y)$ defined by the polynomial $g_i(y)$ ($0 \leq i \leq 15$). Let $C_1 = (a_{15}, \ldots, a_9, a_8), C_0 = (a_7, \ldots, a_1, a_0)$. Obviously, Eq. (9) is a concatenation of $C_1 \times$ and $C_0 \times$ mappings on $\mathbb{F}_{2^{16}}^* \rightarrow \mathbb{F}_{2^{16}}^*$, where $C_1 \times$ acts on the high 128 bits $(x_{15}, \ldots, x_8)$ and $C_0 \times$ acts on the low 128 bits $(x_7, \ldots, x_0)$:

$$Eq. (9) \Leftrightarrow \begin{cases} (x_{15}, \ldots, x_8) \xrightarrow{C_1 \times} (a_{15} \cdot x_{15}, \ldots, a_8 \cdot x_8), \\ (x_7, \ldots, x_0) \xrightarrow{C_0 \times} (a_7 \cdot x_7, \ldots, a_0 \cdot x_0). \end{cases}$$

It is worth noticing that, according to the definition of $a_0, \ldots, a_7$, the $C_0 \times$ is identical to the definition of $C \times$ of LOL-MINI in Eq. (5) indicating that the $C \times$ circuits can be
We demonstrate the two phases as follows:

Round with a period of $F$ we can represent the LFSR updating function with the Feistel-like structure in Fig. 2 as

The schematic of LOL-DOUBLE in the keystream generation phase is shown in Fig. 9.

Keystream generation phase

For each mapping to the state array of the AES round function as Eq. (3), and taking the output of each corresponding LFSR as the subkey. The NFSR part is updated by two $R$ functions as follows:

$N_{0}^{t+1} = R(N_{0}^{t}) \oplus L_{0}^{t}$, $N_{1}^{t+1} = R(N_{1}^{t}) \oplus L_{1}^{t}$.

4.2 Updating Function

The LOL-DOUBLE updating functions of the initialization and the keystream generation phases are slightly different. The keystream generation phase starts from the $t = 0$ step and outputs the 256-bit keystream blocks $Z^{t}$'s for $t \geq 0$. The initialization phase stops with Round$_{load} \leq t < 0$ where $Round_{load} = -12$.

We demonstrate the two phases as follows:

**Keystream generation phase** For $t \geq 0$,

1. Compute the intermediate 128-bit variables:

   $G_{0}^{t} = R(S_{0}^{t})$, $G_{1}^{t} = R(S_{1}^{t})$, $(F_{0}^{t}, F_{0}^{t}) = f(H_{0}^{t}, L_{0}^{t})$.

2. Compute and output the 256-bit keystream block $Z^{t} = (Z_{0}^{t}, Z_{1}^{t})$ as

   $Z_{0}^{t} = G_{0}^{t} \oplus N_{0}^{t}$, $Z_{1}^{t} = G_{1}^{t} \oplus N_{1}^{t}$.

   Note that $Z_{0}$ is placed at the higher and $Z_{1}$ at the lower 128 bits.

3. Update the nonlinear driver:

   $N_{0}^{t+1} = R(N_{0}^{t}) \oplus L_{0}^{t}$, $N_{1}^{t+1} = R(N_{1}^{t}) \oplus L_{1}^{t}$;

   $H_{0}^{t+1} = F_{0}^{t}$, $H_{1}^{t+1} = F_{1}^{t}$, $L_{0}^{t+1} = H_{0}^{t}$, $L_{1}^{t+1} = H_{1}^{t}$.

4. Update the FSM:

   $S_{0}^{t+1} = F_{0}^{t} \oplus G_{1}^{t} \oplus S_{0}^{t}$, $S_{1}^{t+1} = R(S_{0}^{t}) \oplus S_{1}^{t}$,

   $S_{2}^{t+1} = F_{1}^{t} \oplus G_{0}^{t} \oplus S_{2}^{t}$, $S_{3}^{t+1} = R(S_{2}^{t}) \oplus S_{3}^{t}$.

The schematic of LOL-DOUBLE in the keystream generation phase is shown in Fig. 9.
Initialization phase  For $Round_{load} \leq t < 0$, 

1. Compute the intermediate 128-bit variables:

   \[ G'_0 = \mathcal{R}(S'_0), \quad G'_1 = \mathcal{R}(S'_1), \quad (F'_1, F'_0) = f(H'_1, L'_1) = f((H'_0, L'_0), (L'_1, L'_0)). \]

2. Compute the 256-bit keystream block $Z^t = (Z^t_0, Z^t_1)$ as

   \[ Z^t_0 = G'_0 \oplus N^t_0, \quad Z^t_1 = G'_1 \oplus N^t_1. \]

   Such $Z^t$ is not output but fed back to the nonlinear driver.

3. Update the nonlinear driver with the 256-bit $Z^t$ XORed into the feedbacks of both the LFSR and NFSR:

   \[ N^t+1_0 = R(N^t_0) \oplus L^t_0 \oplus Z^t_1, \quad N^t+1_1 = R(N^t_1) \oplus L^t_1 \oplus Z^t_0, \]
   \[ H^t+1_0 = \begin{cases} F^t_0 \oplus Z^t_1, & Round_{load} \leq t < -1 \\ F^t_0 \oplus Z^t_1 \oplus K_\ell, & t = -1 \end{cases}, \quad H^t+1_1 = \begin{cases} F^t_1 \oplus Z^t_0, & Round_{load} \leq t < -1 \\ F^t_1 \oplus Z^t_0 \oplus K_h, & t = -1 \end{cases} \]

4. Update the FSM:

   \[ S^t+1_0 = F^t_0 \oplus G'_1 \oplus S^t_0, \quad S^t+1_1 = R(S^t_0) \oplus S^t_1, \]
   \[ S^t+1_2 = F^t_1 \oplus G'_0 \oplus S^t_2, \quad S^t+1_3 = R(S^t_2) \oplus S^t_3. \]

As can be seen, the 256-bit master key is XORed into the $H$ register of the LFSR at the final initialization $t = -1$ step. The schematic of LOL-DOUBLE cipher in the initialization phase is shown in Fig. 10.

![Figure 9: The schematic of LOL-DOUBLE cipher in keystream generation phase](image)

![Figure 10: The schematic of LOL-DOUBLE cipher in initialization phase](image)

### 4.3 Initialization

The key and IV of LOL-DOUBLE are applied in a manner similar to LOL-MINI. At the beginning of the initialization $t = Round_{load}$, the key and IV are loaded into the FSM by assigning

\[ S'_3 = K_h, S'_2 = K_\ell, S'_1 = IV_h, S'_0 = IV_\ell, \]
and the LFSR and NFSR states are set to zero:

\[ H_0 = L_0 = H_1 = L_1 = N_0 = N_1 = 0. \]

LOL-DOUBLE also has 12 initialization rounds so Round\textsuperscript{load} is also set to \(-12\). It also adopts the FP\textsuperscript{(1)}-mode \([HK18]\) by XORing the master key \(K = (K_h, K_l)\) to the states of \(H\) at the final initialization round. Besides, there are limitations that the maximum keystream length of a key-IV pair should be \(2^{64}\) and each key should be used with a maximum of \(2^{64}\) different IVs.

5 Security Analysis

The security has always been the main concern of newly designed stream ciphers, we evaluate the resistance of LOL-MINI and LOL-DOUBLE against various cryptanalysis methods so as to prove their 256-bit security margins.

5.1 Maximum period proof of the LFSR

For LOL framework. The proving method that the LFSR defined in the LOL framework has a maximum period based on specific irreducible polynomials and vector permutation is given in Section 2.1.

For LOL-MINI and LOL-DOUBLE. For the LFSR in LOL-MINI, the generator matrix \(G\) is of size \(256 \times 256\), defined as

\[
G = \begin{pmatrix} M_c & I_{128} \\
M_\sigma & 0
\end{pmatrix}.
\]

Here \(M_\sigma\) is the shift matrix corresponding to \(\sigma\) permutation and \(M_c\) is the diagonal block matrix for the multiplications over the finite field \(\mathbb{F}_{2^8} = \mathbb{F}_2[y]/g_i(y)\) defined by the polynomial \(g_i(y)\) in Eq. (4) (\(0 \leq i \leq 7\)),

\[
M_\sigma = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & I_{16} & 0 & 0 \\
0 & 0 & 0 & I_{16} & 0 & 0 & 0 & 0 \\
I_{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & I_{16} & 0 & 0 & 0 \\
0 & I_{16} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & I_{16} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & I_{16} & 0 \\
0 & I_{16} & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix},
\]

\[
M_c = \begin{pmatrix}
M_{\alpha_0} & & & & & & & 0 \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
0 & & & & & & & M_{\alpha_7}
\end{pmatrix},
\]

\[
M_{\alpha_i} = \begin{pmatrix}
b_{i,15} & b_{i,14} & b_{i,13} & b_{i,12} & b_{i,11} & b_{i,10} & b_{i,9} & b_{i,8} & b_{i,7} & b_{i,6} & b_{i,5} & b_{i,4} & b_{i,3} & b_{i,2} & b_{i,1} & b_{i,0} \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}.
\]
The characteristic polynomial of $G$ is found using NTL as

$$\eta_{256}(x) = x^{256} + x^{252} + x^{251} + x^{250} + x^{246} + x^{245} + x^{242} + x^{241} + x^{238} + x^{234} + x^{229} + x^{227} + x^{225} + x^{222} + x^{220} + x^{219} + x^{217} + x^{215} + x^{214} + x^{213} + x^{212} + x^{211} + x^{210} + x^{208} + x^{207} + x^{204} + x^{203} + x^{198} + x^{196} + x^{194} + x^{193} + x^{186} + x^{185} + x^{184} + x^{183} + x^{182} + x^{179} + x^{178} + x^{177} + x^{175} + x^{170} + x^{168} + x^{167} + x^{166} + x^{165} + x^{161} + x^{157} + x^{154} + x^{153} + x^{152} + x^{151} + x^{150} + x^{148} + x^{147} + x^{146} + x^{145} + x^{144} + x^{143} + x^{140} + x^{139} + x^{138} + x^{134} + x^{133} + x^{131} + x^{130} + x^{126} + x^{123} + x^{122} + x^{121} + x^{119} + x^{118} + x^{116} + x^{115} + x^{113} + x^{110} + x^{109} + x^{108} + x^{107} + x^{104} + x^{103} + x^{102} + x^{101} + x^{100} + x^{98} + x^{97} + x^{94} + x^{93} + x^{92} + x^{89} + x^{88} + x^{86} + x^{84} + x^{83} + x^{82} + x^{81} + x^{79} + x^{78} + x^{77} + x^{76} + x^{74} + x^{72} + x^{71} + x^{70} + x^{69} + x^{67} + x^{66} + x^{65} + x^{63} + x^{62} + x^{61} + x^{60} + x^{56} + x^{53} + x^{52} + x^{51} + x^{50} + x^{48} + x^{47} + x^{44} + x^{42} + x^{39} + x^{38} + x^{37} + x^{34} + x^{32} + x^{31} + x^{30} + x^{29} + x^{28} + x^{26} + x^{25} + x^{24} + x^{21} + x^{20} + x^{9} + x^{8} + 1.$$  

Then we can verify it primitive using Maple, which proves the maximum period $2^{256} - 1$ of the LFSR.

For LOL-DOUBLE, the generator matrix $G$ of the LFSR is expanded to the size $512 \times 512$ and rewritten with $I_{512}, M_G$ and $M_C$ accordingly. Its characteristic polynomial is derived and proved to be primitive in the same manner.

$$\eta_{512} = x^{512} + x^{511} + x^{510} + x^{509} + x^{505} + x^{504} + x^{503} + x^{500} + x^{499} + x^{498} + x^{497} + x^{496} + x^{495} + x^{494} + x^{493} + x^{492} + x^{488} + x^{486} + x^{483} + x^{482} + x^{481} + x^{479} + x^{477} + x^{476} + x^{475} + x^{474} + x^{472} + x^{471} + x^{470} + x^{466} + x^{464} + x^{461} + x^{460} + x^{457} + x^{455} + x^{454} + x^{452} + x^{448} + x^{448} + x^{444} + x^{440} + x^{439} + x^{437} + x^{435} + x^{432} + x^{429} + x^{426} + x^{425} + x^{424} + x^{423} + x^{422} + x^{421} + x^{420} + x^{419} + x^{417} + x^{416} + x^{414} + x^{411} + x^{410} + x^{408} + x^{407} + x^{405} + x^{403} + x^{400} + x^{396} + x^{393} + x^{390} + x^{389} + x^{386} + x^{384} + x^{379} + x^{378} + x^{377} + x^{374} + x^{372} + x^{369} + x^{368} + x^{367} + x^{366} + x^{358} + x^{357} + x^{355} + x^{354} + x^{349} + x^{346} + x^{344} + x^{343} + x^{341} + x^{338} + x^{337} + x^{336} + x^{335} + x^{334} + x^{333} + x^{331} + x^{330} + x^{328} + x^{325} + x^{324} + x^{322} + x^{319} + x^{318} + x^{316} + x^{315} + x^{314} + x^{309} + x^{307} + x^{304} + x^{302} + x^{301} + x^{297} + x^{295} + x^{293} + x^{292} + x^{291} + x^{290} + x^{285} + x^{284} + x^{283} + x^{280} + x^{276} + x^{268} + x^{267} + x^{263} + x^{258} + x^{257} + x^{255} + x^{254} + x^{252} + x^{249} + x^{248} + x^{245} + x^{242} + x^{241} + x^{240} + x^{236} + x^{234} + x^{232} + x^{231} + x^{230} + x^{227} + x^{226} + x^{224} + x^{219} + x^{210} + x^{209} + x^{208} + x^{207} + x^{206} + x^{205} + x^{203} + x^{200} + x^{197} + x^{196} + x^{193} + x^{188} + x^{187} + x^{186} + x^{184} + x^{180} + x^{175} + x^{172} + x^{171} + x^{167} + x^{166} + x^{162} + x^{160} + x^{159} + x^{157} + x^{155} + x^{154} + x^{150} + x^{149} + x^{147} + x^{145} + x^{142} + x^{141} + x^{137} + x^{134} + x^{133} + x^{127} + x^{126} + x^{125} + x^{124} + x^{123} + x^{118} + x^{116} + x^{115} + x^{112} + x^{111} + x^{109} + x^{108} + x^{103} + x^{102} + x^{101} + x^{100} + x^{99} + x^{98} + x^{95} + x^{94} + x^{92} + x^{91} + x^{86} + x^{84} + x^{81} + x^{80} + x^{79} + x^{78} + x^{72} + x^{69} + x^{68} + x^{65} + x^{63} + x^{62} + x^{61} + x^{58} + x^{57} + x^{55} + x^{54} + x^{52} + x^{51} + x^{49} + x^{46} + x^{41} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{31} + x^{30} + x^{29} + x^{27} + x^{25} + x^{23} + x^{22} + x^{20} + x^{18} + x^{17} + x^{16} + 1.$$  

Then we can verify it primitive using Maple. Thus the LFSR of LOL-DOUBLE has the maximum period $2^{512} - 1$.

### 5.2 Full diffusion round analysis

**For LOL framework.** The number of full diffusion rounds in the initialization of a stream cipher is to evaluate how many initialization iterations each state bit of the input goes through at least to affect all the state bits. For the LOL framework, the number of full diffusion rounds is related to the specific parameters and components selection and the loading state of the key/IV. Here, we propose to load the key/IV into the rapidly diffusing FSM instead of the conventional LFSR for the first time, making it possible to achieve a smaller number of full diffusion rounds.

**For LOL-MINI and LOL-DOUBLE.** For LOL-MINI, the internal state $N, S_0, S_1, S_2$ reach full diffusion after 4 rounds and $H, L$ reach full diffusion after 5 rounds. For LOL-DOUBLE,
the internal state $N_0, N_1, S_0, S_1, S_2, S_3$ reach full diffusion after 5 rounds and $H_0, L_0, H_1, L_1$ reach full diffusion after 6 rounds. Thus we load the key and IV into the FSM rather than the LFSR as current stream ciphers used to do in LOL-MINI and LOL-DOUBLE since the LFSR needs 1 more round than the FSM to reach full diffusion. Table 1 (Table 2) shows the total number of output state bits in LOL-MINI (LOL-DOUBLE\textsuperscript{3}) affected by any bit of the input state after 1 to 5 rounds (1 to 6 rounds), where 768 = 128 × 6 (1280 = 128 × 10) indicates that the full diffusion is reached.

Table 1: Diffusion table for LOL-MINI

<table>
<thead>
<tr>
<th>rounds</th>
<th>$H$</th>
<th>$L$</th>
<th>$N$</th>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>97</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>69</td>
<td>162</td>
<td>161</td>
<td>417</td>
<td>353</td>
</tr>
<tr>
<td>3</td>
<td>202</td>
<td>329</td>
<td>352</td>
<td>544</td>
<td>672</td>
<td>592</td>
</tr>
<tr>
<td>4</td>
<td>547</td>
<td>610</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
</tr>
<tr>
<td>5</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
</tr>
</tbody>
</table>

Table 2: Diffusion table for LOL-DOUBLE

<table>
<thead>
<tr>
<th>rounds</th>
<th>$H_0$</th>
<th>$L_0$</th>
<th>$H_1$</th>
<th>$L_1$</th>
<th>$N_0$</th>
<th>$N_1$</th>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>34</td>
<td>34</td>
<td>33</td>
<td>33</td>
<td>97</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>195</td>
<td>195</td>
<td>417</td>
<td>417</td>
<td>417</td>
<td>417</td>
</tr>
<tr>
<td>3</td>
<td>487</td>
<td>489</td>
<td>486</td>
<td>488</td>
<td>514</td>
<td>514</td>
<td>929</td>
<td>769</td>
<td>929</td>
<td>769</td>
</tr>
<tr>
<td>4</td>
<td>969</td>
<td>935</td>
<td>969</td>
<td>935</td>
<td>1088</td>
<td>1088</td>
<td>1184</td>
<td>1088</td>
<td>1184</td>
<td>1088</td>
</tr>
<tr>
<td>5</td>
<td>1216</td>
<td>1216</td>
<td>1216</td>
<td>1216</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
</tr>
<tr>
<td>6</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
<td>1280</td>
</tr>
</tbody>
</table>

5.3 Differential attack on initialization

For LOL framework. For the initialization phase, we consider differential attacks under the related-key chosen-IV model where differences can be injected in both key and IV bits. For the LOL framework, the ability to resist differential analysis is related to the specific parameters and the maximum differential probability of the S-boxes used in the SP-network round function. For any specific set, we can use the byte-based MILP modeling technique and search for the truncated differential characteristics with the fewest active S-boxes so as to draw upper bounds to the differential propagation probabilities. The truncated differential propagation rules as well as the corresponding MILP modeling techniques for XOR and the SP-network round functions are extensively studied and can be used in our cryptanalysis directly. The truncated differential propagation rules of the LFSR can be deduced with the H-representation technique in [SHW+14].

For LOL-MINI and LOL-DOUBLE. For the byte pair $(x, x') \in \mathbb{F}_2^8 \times \mathbb{F}_2^8$, we denote its XOR difference as $\Delta x = x \oplus x'$ and the truncated difference is a binary variable $\Delta^T x \in \mathbb{F}_2$ is defined as:

$$
\Delta^T x = \begin{cases} 
0 & \Delta x = 0 \\
1 & \Delta x \neq 0 
\end{cases}
$$

The truncated differential propagation rules and the corresponding MILP modeling techniques for XOR and the AES round functions $\mathcal{R}$ are general. Let $(x, y, z) \in \mathbb{F}_2^8$ satisfy

\textsuperscript{3}For the sake of brevity, the corresponding cryptanalysis results of LOL-DOUBLE are shown in parentheses throughout this section.
z = x ⊕ y. The feasible truncated difference $(\Delta^T x, \Delta^T y, \Delta^T z)$ should lie in the following set $S$:

$$S = \{(0,0,0), (1,0,0), (0,1,1), (1,1,0), (1,1,1)\}.$$  

So the truncated differential propagation rule of such XOR operations can be captured with the MILP model $\mathcal{M}$ by calling $(\mathcal{M}, \Delta^T z) \leftarrow \text{xor2Trunc}(\mathcal{M}, \Delta^T x, \Delta^T y)$ defined in Algorithm 2. A 128-bit state $X$ can be divided into 16 bytes so its truncated difference

\begin{algorithm}[h]
\caption{Truncated differential model of XOR operations (xor2Trunc)}
\begin{algorithmic}[1]
\Input{Initial model $\mathcal{M}$ and 2 binary variables $\Delta^T x, \Delta^T y$}
\Output{Updated model $\mathcal{M}$ and a binary variable $\Delta^T z$}
1 Declare 2 variables $\mathcal{M}.\text{var} \leftarrow \Delta^T x, \Delta^T y$ as binaries; 
2 Define vector $\vec{x} = (\Delta^T x, \Delta^T y, \Delta^T z)$;
3 Update the model $\mathcal{M}.\text{con} \leftarrow A\vec{x} \geq 0$ where

$$A = \begin{pmatrix}
1 & 1 & -1 \\
1 & -1 & 1 \\
-1 & 1 & 1
\end{pmatrix}.$$ 

\Return $(\mathcal{M}, \Delta^T z)$;
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{Truncated differential model of AES MixColumn operation (mcTruncModel)}
\begin{algorithmic}[1]
\Input{Initial model $\mathcal{M}$ and a vector of 4 binary variables $\Delta^T x_0, \ldots, \Delta^T x_3$}
\Output{Updated model $\mathcal{M}$, a vector of 4 binary variables $\Delta^T y_0, \ldots, \Delta^T y_3$}
1 Declare 4 variables $\mathcal{M}.\text{var} \leftarrow \Delta^T y_0, \ldots, \Delta^T y_3$ as binaries; 
2 Declare 1 variable $\mathcal{M}.\text{var} \leftarrow \tau$ as binary; 
3 $\mathcal{M}.\text{con} \leftarrow \sum_{i=0}^{3}(\Delta^T x_i + \Delta^T y_i) \leq 8\tau$;
4 $\mathcal{M}.\text{con} \leftarrow \sum_{j=0}^{3}(\Delta^T x_i + \Delta^T y_j) \geq 5\tau$;
5 \Return $(\mathcal{M}, \Delta^T y_0, \ldots, \Delta^T y_3)$;
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[h]
\caption{Truncated differential model of AES round function $\mathcal{R}$ (aesTruncModel)}
\begin{algorithmic}[1]
\Input{Initial model $\mathcal{M}$ and a vector of 16 binary variables $\Delta^T x = (\Delta^T x_{15}, \ldots, \Delta^T x_0)$}
\Output{Updated model $\mathcal{M}$, a vector of 16 binary variables $\Delta^T y = (\Delta^T y_{15}, \ldots, \Delta^T y_0)$}
1 Define permutation over integers: $\sigma : \{0,\ldots,15\} \rightarrow \{0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11\}$; 
2 Define vector $\Delta^T W = (\Delta^T w_{15}, \ldots, \Delta^T w_0) = (\Delta^T x_{\sigma(15)}, \ldots, \Delta^T x_{\sigma(0)})$;
3 for $j = 0, \ldots, 3$ do 
4 \ $(\mathcal{M}, \Delta^T y_{j+3}, \ldots, \Delta^T y_{j}) \leftarrow \text{mcTruncModel}(\mathcal{M}, \Delta^T w_{j+3}, \ldots, \Delta^T w_{j})$;
5 Define vector $\Delta^T Y = (\Delta^T y_{15}, \ldots, \Delta^T y_0)$;
6 \Return $(\mathcal{M}, \Delta^T Y)$;
\end{algorithmic}
\end{algorithm}

We consider the $C \times H$ operation in the LOL LFSR updating functions. For $(x, y) \in \mathbb{F}_2^{16} \times \mathbb{F}_2^{16}$ satisfying $y = \alpha_i x$ where $i = 0, \ldots, 15$, the truncated differences can be
The inter-round relationships for LOL-MINI and LOL-DOUBLE are shown in Table 4.

Table 3: The feasible truncated difference relationships among states in different rounds. For example, there is a relationship between rounds. In addition to the constraints of round functions, the parallel problem is able to construct MILP model. The MILP model can be constructed by calling Algorithm 5 defined as seen, the truncated differential propagation rules of $\alpha_i$’s can be acquired by calling Algorithm 5 defined as $\alpha_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$.

Table 3: The feasible truncated difference relationships among states in different rounds. For example, there is a relationship $\Delta T x = (\Delta T x_0, \Delta T x_1, \Delta T x_2, \Delta T x_3)$, $\Delta T y = (\Delta T y_0, \Delta T y_1, \Delta T y_2)$.

Algorithm 5: Truncated differential model of $y = \alpha_i x$ (alphaTruncModel)

\begin{algorithm}
\textbf{Input:} Initial model $M$, 2 binary variables $(\Delta T x_0, \Delta T x_1)$, integer $i \in \{0, \ldots, 15\}$
\textbf{Output:} Updated model $M$, 2 binary variables $(\Delta T y_0, \Delta T y_1)$

1. Declare 2 variables $M$.var $\leftarrow (\Delta T y_0, \Delta T y_1)$ as binaries;
2. Define vector $\bar{x} = (\Delta T x_0, \Delta T x_1, \Delta T y_0, \Delta T y_1)$;
3. if $i \in \{0, 1, 6, 8, 9, 10, 12, 13\}$ then
   4. Add constraint $M$.con $\leftarrow A_1 \bar{x} \geq 0$ where
      \[
      A_1 = \begin{pmatrix}
      1 & -1 & 0 & 1 \\
      1 & 1 & 0 & -1 \\
      -1 & 0 & 1 & 1 \\
      1 & 0 & -1 & 1 \\
      \end{pmatrix}
      \] (10)
5. else
6. Add constraint $M$.con $\leftarrow A_2 \bar{x} \geq 0$ where
   \[
   A_2 = \begin{pmatrix}
   0 & -1 & 1 & 1 \\
   1 & 1 & -1 & 0 \\
   -1 & 0 & 1 & 1 \\
   1 & 1 & 0 & -1 \\
   \end{pmatrix}
   \] (11)
7. Return $(M, \Delta T y_0, \Delta T y_1)$;

represented as $\Delta T x = (\Delta T x_0, \Delta T x_1, \Delta T x_2, \Delta T x_3)$, $\Delta y = (\Delta T y_0, \Delta T y_1, \Delta T y_2)$. By traversing all $2^{16}$ $\Delta x$ values and computing the corresponding $\Delta y$’s, we are able to acquire the feasible truncated difference $(\Delta T x_0, \Delta T x_1, \Delta T y_0, \Delta T y_1)$’s in Table 3. As can be seen, the truncated differential propagation rules of $y = \alpha_i x$ corresponding to $i = 0, \ldots, 15$ are slightly different. The MILP model can be constructed by calling Algorithm 5 defined as $(M, \Delta T y_0, \Delta T y_1) \leftarrow \alpha i \text{TruncModel}(M, \Delta T x_0, \Delta T x_1, i)$. The matrices $A_1$ in Eq. (10) and $A_2$ in Eq. (11) are deduced with the H-representation technique in [SHW+14].

<table>
<thead>
<tr>
<th>$i$</th>
<th>$(\Delta T x_0, \Delta T x_1, \Delta T y_0, \Delta T y_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 6, 8, 9, 10, 12, 13</td>
<td>$(0,0,0,0), (1,0,1,0), (0,1,1,0), (1,0,1,0), (0,1,0,1), (1,1,0,1), (1,0,1,1), (0,1,1,1)$</td>
</tr>
<tr>
<td>2, 3, 4, 5, 7, 11, 14, 15</td>
<td>$(0,0,0,0), (1,0,1,0), (1,1,1,0), (1,0,0,1), (0,1,0,1), (1,1,0,1), (1,0,1,1), (0,1,1,1), (1,1,1,1)$</td>
</tr>
</tbody>
</table>

Since the LOL round functions are composed of $\alpha_i$, $\oplus$ and $\mathcal{R}$ operations, we are now able to construct MILP model $M$ capturing the truncated differential path covering $r$ LOL rounds. In addition to the constraints of round functions, the parallel $\mathcal{R}$ calls also indicate relationships among states in different rounds. For example, there is a relationship $\Delta S_2^{i+1} = \Delta \mathcal{R}(S_1^i) \oplus \Delta S_2^i = \Delta \mathcal{R}(S_1^i) \oplus \Delta \mathcal{R}(S_1^{i-1}) \oplus \Delta S_2^{i-1}$ $\mathcal{M}(SR(\Delta \mathcal{R}(S_1^i) \oplus \Delta \mathcal{R}(S_1^{i-1})))) \oplus \Delta S_2^{i-1}$ so $\Delta T S_2^{i+1}$ is not only related $\Delta T (S_1^i)$ but also the truncated difference $\Delta T (S_1^i) \oplus \Delta T (S_1^{i-1})$. The inter-round relationships for LOL-MINI and LOL-DOUBLE are shown in Table 4.
write the expressions of the internal states and keystream outputs of LOL stream ciphers in
22 LOL Framework
Z
on the key and the constant part of the initial state. In addition, when the state word
the expressions, we omit the constants and only focus on how the IV evolves as the state is
initialization has already gone through
proved secure against integral attack as long as the first keystream output after the
integral distinguisher for
SP-network function iterations to integral attacks. That is, if there is no key-independent
of initialized rounds is enough resist integral attacks by considering the resistant of the
For LOL framework.
As the LOL framework is composed of blockwise states, we can
5.4 Integral attack

With the round function modeling and inter-round relationships, we are able to lower bound the number of active S-boxes for r-round LOL-MINI/LOL-DOUBLE. The result is shown in Table 5 for LOL-MINI (LOL-DOUBLE) indicating that there is no differential characteristic with probability exceeding $2^{-256}$ after 6 rounds (4 rounds) of the initialization since the maximum differential probability of the S-box in AES is $2^{-4}$ and there are at least 47 (52) active S-boxes. That is, LOL-MINI and LOL-DOUBLE ciphers resist the related-key chosen IV attacks.

Table 5: The active S-box lower bounds for r initialization rounds.

<table>
<thead>
<tr>
<th>r</th>
<th>Minimum number of active S-boxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>51</td>
</tr>
</tbody>
</table>

5.4 Integral attack

For LOL framework. As the LOL framework is composed of blockwise states, we can
write the expressions of the internal states and keystream outputs of LOL stream ciphers in
terms of the initial state that loads the key/IV. Then we can evaluate whether the number of
initialized rounds is enough resist integral attacks by considering the resistant of the
SP-network function iterations to integral attacks. That is, if there is no key-independent
integral distinguisher for $n$ iterated SP-network function calls, the LOL cipher can be
proved secure against integral attack as long as the first keystream output after the
initialization has already gone through $n$ SP-network function iterations.

For LOL-MINI and LOL-DOUBLE. As [LIM521], we express the keystream blocks in
the initialization phase of LOL-MINI in terms of the key/IV. For simplicity, when writing
the expressions, we omit the constants and only focus on how the IV evolves as the state is
updated. Consequently, $R(IV)$ may represent $R(IV \oplus c)$ where $c$ is a constant depending
on the key and the constant part of the initial state. In addition, when the state word
does not depend on the IV, it is simply written as 0. Then,

$$Z^{-7} = R(R(R(R(R(R(IV)))) \oplus IV) \oplus R(IV) \oplus R(IV) \oplus R(IV) \oplus R(IV))$$
$$\oplus R(R(IV) \oplus R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$
$$\oplus R(R(R(IV))) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$
$$\oplus R(R(R(IV))) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$

As [LIM521], we express the keystream blocks in

$$Z^{-7} = R(R(R(R(R(R(IIV)))) \oplus IV) \oplus R(IV) \oplus R(IV) \oplus R(IV) \oplus R(IV) \oplus R(IV))$$
$$\oplus R(R(IV) \oplus R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$
$$\oplus R(R(R(R(IV))) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$
$$\oplus R(R(R(R(IV))) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV)) \oplus R(R(IV) \oplus R(IV))$$
It can be found that $IV$ needs to go through 6 AES rounds for $Z^{-7}$. Similarly, for LOL-DOUBLE, $IV_h$ and $IV_\ell$ need to go through at least 5 AES rounds for the keystream block $Z^{-7} = (Z_0^{-7}, Z_1^{-7})$.

Since both LOL-MINI and LOL-DOUBLE provide 256-bit security, it is necessary to evaluate the case when the IV traverses all the possible values under the same 256-bit key. Moreover, the most efficient integral attack result for round-reduced AES: there is no integral distinguishers on 5 or more AES rounds [SLR+15]. According to our evaluations, it is impossible to the IV from going through $\leq 5$ AES rounds for $Z^{-7}$ making it impossible to conduct integral attacks. To sum up, 12 initialization rounds for LOL-MINI and LOL-DOUBLE are secure against integral attacks.

5.5 Slide attacks

For LOL framework. The idea of slide attacks is to form the same state at a shift of steps for different key/IV pairs, which can be detected by similar keystreams and further recover some key information. LOL ciphers can be secure against slide attacks by adding key feedforward in initialization, or using different update functions for initialization and key generation.

For LOL-MINI and LOL-DOUBLE. For LOL-MINI and LOL-DOUBLE, such sliding properties would be difficult to find, due to the update of large blocks at each step and the feedback of the keystream block in the initialization, and further the use of FP(1)-mode in the initialization.

5.6 Correlation attack

For LOL framework. The main idea of correlation attacks is to search for a linear approximation with high correlation between the keystream and the state of LFSR and construct the check equations from it, then use the check equations to recover the state of LFSR and further recover the key. In terms of the overall structure of LOL framework, the combination of nonlinear drivers and FSMs with memories avoids the direct LFSR-FSM connection in the SNOW stream cipher family so as to eliminate high linear correlations utilized in FCAs. For specific parameters of LOL ciphers, we can evaluate the upper bounds of the linear correlations between the keystream and the LFSR bits. For correlation $Cor$, the data complexity of the correlation attacks is evaluated as $Cor^{-2}$. Such linear correlations can be upper bounded with the number of active S-boxes of the SP-network functions in the truncated linear approximations, which can be evaluated with the MILP model based truncated linear characteristic searching technique.

For LOL-MINI and LOL-DOUBLE. Similar with the situation of differential attacks, the propagation rules of truncated linear properties of AES round functions, as well as the corresponding MILP modeling techniques, are extensively studied and can be applied directly. The truncated linear masks for 16-bit cells, regarded as a concatenation of 2 bytes, can be represented with 2 binary variables in the MILP model. The truncated linear propagation rules over finite fields can be described with linear constraints in the same way with its differential counterpart in Section 5.3 using the H-representation technique in [SHW+14].

\[\text{\footnotesize The there is also FCAs on nonlinear-driver based Grain-like stream ciphers but such attacks are not applicable to the LOL framework due to the large memory of FSM.}\]
For LOL-MINI, at least 4 keystream blocks are necessary for constructing a linear approximation. For simplicity, we make the following symbolic substitution
\[ u = S^t_{2}, v = S^{t-1}_{1}, w = S^{t-1}_{0}, \theta = N^{t-1} \]
\[ L = L^{t-1}, H = H^{t-1}, F = F^{t-1}. \]

Then, the 4 consecutive keystream blocks can be represented as follows:

\[ Z^{t-1} = \theta \oplus R(u \oplus R(v)), \] (12a)
\[ Z^{t} = L \oplus R(\theta) \oplus R(u), \] (12b)
\[ Z^{t+1} = H \oplus R(L \oplus R(\theta)) \oplus R(u \oplus R(v \oplus R(w))), \] (12c)
\[ Z^{t+2} = F \oplus R(H \oplus R(\theta)) \oplus R(v \oplus R(w)) \oplus R(w \oplus R(u \oplus R(v) \oplus F)). \] (12d)

For Eq. (12a), we have
\[ (\gamma_2 \rightarrow \gamma_1) : \gamma_1 \cdot R(u \oplus R(v)) = \gamma_2 \cdot (u \oplus R(v)), \]
\[ (\gamma_3 \rightarrow \gamma_2) : \gamma_2 \cdot R(v) = \gamma_3 \cdot v, \]

and derive
\[ \gamma_1 \cdot Z^{t-1} = \gamma_1 \cdot \theta \oplus \gamma_2 \cdot u \oplus \gamma_3 \cdot v. \]

We have the linear approximations derived from other equations in Eq. (12) likewise resulting in the linear relationship between LFSR and Z bits as follows:
\[ \gamma_1 \cdot Z^{t} \oplus \gamma_4 \cdot Z^{t+1} \oplus \gamma_7 \cdot Z^{t+1} \oplus \gamma_13 \cdot Z^{t+2} = \]
\[ (\gamma_1 \oplus \gamma_7 \oplus \gamma_4 \oplus \gamma_10 \cdot \theta \oplus (\gamma_2 \oplus \gamma_7 \oplus \gamma_6 \oplus \gamma_10 \oplus \gamma_17 \oplus \gamma_23) \cdot u \oplus (\gamma_5 \oplus \gamma_3 \oplus \gamma_11 \oplus \gamma_12 \oplus \gamma_27) \cdot v) \]
\[ \oplus (\gamma_4 \oplus \gamma_7 \oplus \gamma_14) \cdot H \oplus (\gamma_4 \oplus \gamma_8 \oplus \gamma_{15}) \cdot L \oplus (\gamma_1 \oplus \gamma_9 \oplus \gamma_{23} \oplus \gamma_{27} \oplus \gamma_{22} \oplus \gamma_{21}) \cdot w \oplus (\gamma_4 \oplus \gamma_5 \oplus \gamma_22) \cdot F. \]

To construct the linear characteristic \[ \gamma_1 \cdot Z^{t-1} \oplus \gamma_4 \cdot Z^{t} \oplus \gamma_7 \cdot Z^{t+1} \oplus \gamma_13 \cdot Z^{t+2} = 0, \]
we also need to add constraints to ensure that the coefficients of nonlinear parts \( u, v, w, \theta \) on the right side of the above equation are zero, which make the objective function of the MILP model. The linear inequality constraints describing the linear propagations are easily obtained according to the MILP model of the AES round function.

Similarly, it requires at least 3 keystream blocks to construct the linear approximation for LOL-DOUBLE. We represent the 3 consecutive keystream blocks as follows:

\[ Z^{t}_0 = v_0 \oplus R(u_0 \oplus R(u_0)), \] (13a)
\[ Z^{t}_1 = v_1 \oplus R(u_1 \oplus R(u_1)), \] (13b)
\[ Z^{t+1}_0 = L_0 \oplus R(u_0 \oplus R(u_0)), \] (13c)
\[ Z^{t+1}_1 = L_1 \oplus R(u_1 \oplus R(u_1)), \] (13d)
\[ Z^{t+2}_0 = H_0 \oplus R(u_0 \oplus R(u_1 \oplus R(u_1))) \oplus R(L_0 \oplus R(u_0)), \] (13e)
\[ Z^{t+2}_1 = H_1 \oplus R(u_1 \oplus R(u_1 \oplus R(u_0))) \oplus R(L_1 \oplus R(u_1)), \] (13f)

with
\[ v_0 = N^{t}_0, v_1 = N^{t}_1, u_0 = S^{t}_0, u_1 = S^{t+1}_2, w_0 = S^{t+1}_3, \]
\[ L_0 = L^{t}_0, L_1 = L^{t}_1, H_0 = H^{t}_0, H_1 = H^{t}_1, F_0 = F^{t}_0, F_1 = F^{t}_1. \]

Besides, we should also consider the state timing relationship for modeling parallel AES round functions in the FSM as we have done in the differential analysis. For example, since \( S^{t+1}_2 = R(S^{t}_1) \oplus S^{t}_1 \) and \( S^{t}_2 = R(S^{t-1}_1) \oplus S^{t-1}_2 \), there is an underlying relationship of
\[ S^{t+1}_2 = R(S^{t}_1) \oplus R(S^{t-1}_1) \oplus S^{t-1}_2 \]
resulting in an additional constraint s.t. the truncated linear mask of \( S^{t}_1 \) is equal to that of \( S^{t-1}_1 \). This property is used directly in [ENP19] for finding good truncated linear masks of AEGIS.
According to our analysis, the truncated linear masks for 4 (3) LOL-MINI (LOL-
DOUBLE) keystream blocks have at least 80 (94) active S-boxes. Considering the linear
properties of AES S-boxes, no linear distinguisher or FCA can be conducted within a
complexity of $2^{56}$. So there are no effective linear distinguishers or correlation attacks
applicable for LOL-MINI and LOL-Duo.

### 5.7 Guess-and-determine attack

**For LOL framework.** Guess-and-determine attacks act as a common tool to achieve state
recovery in the keystream generation phase. According to the blockwise structure of LOL
framework, the attackers can determine the minimum number of consecutive keystream
blocks needed to form the guess-and-determine attack by which information about the
whole internal states are all involved, and number of keystream blocks needed to filter
the unique candidate. Then due to the full diffusion property of SP-network function,
the attackers need to guess some complete blocks. Combined with the large block of
LOL stream ciphers, the complexity of the byte-based guess-and-determine attack can be
bounded.

**For LOL-MINI and LOL-Duo.** A keystream block of LOL-MINI (LOL-
DOUBLE) at each step only involves 2 (4) state units. As the internal state consists of 6 (10) units,
the attackers at least need to consider 3 (3) consecutive steps in order to involve the
information about the whole internal states.

We first consider the attack based on 128-bit blocks. It is required to guess $N_t, N_{t+1}, N_{t+2}$ according to $Z_t, Z_{t+1}, Z_{t+2}$ with a time complexity of $2^{384}$ on LOL-
MINI and another 3 keystream blocks are needed for filtering. The determined guess-
determination path is shown in Table 6.

<table>
<thead>
<tr>
<th>Time</th>
<th>Operation</th>
<th>Variables</th>
<th>Relations</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>Guess</td>
<td>$N_t$</td>
<td>$\mathcal{R}(S_2^t) = Z_t \oplus N_t$</td>
<td>$2^{128}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_2^t$</td>
<td>$\mathcal{R}(S_2^t) = Z_t \oplus N_t$</td>
<td>$2^{256}$</td>
</tr>
<tr>
<td>$t+1$</td>
<td>Guess</td>
<td>$N_{t+1}$</td>
<td>$L^t = N_{t+1} \oplus \mathcal{R}(N_t)$</td>
<td>$2^{256}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$L^t$</td>
<td>$L^t = N_{t+1} \oplus \mathcal{R}(N_t)$</td>
<td>$2^{384}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_2^{t+1}$</td>
<td>$\mathcal{R}(S_2^{t+1}) = Z_{t+1} \oplus N_{t+1}$</td>
<td>$2^{384}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^{t+1}$</td>
<td>$\mathcal{R}(S_1^{t+1}) = S_2^{t+1} \oplus S_2^{t+1}$</td>
<td>$2^{384}$</td>
</tr>
</tbody>
</table>

For LOL-Duo, it is required to guess $N_0, N_0^{t+1}, N_0^{t+2}, N_1$ according to $Z_0^t, Z_1^t,$
$Z_0^{t+1}, Z_1^{t+1}, Z_0^{t+2}, Z_1^{t+2}$ with a time complexity of $2^{512}$ and another 4 keystream blocks
are needed for filtering. The guessing path is shown in Table 6. Here, $\sigma_L : \mathbb{F}_{2^{16}}^2 \rightarrow \mathbb{F}_{2^{16}}^8$ is
defined according with the permutation $\sigma$ in LOL-Duo as follows

$$(x_{15}, \ldots, x_0) \xrightarrow{\sigma_L} (x_4, x_7, x_{10}, x_{13}, x_1, x_5, x_{12}, x_3).$$

Thus, for $\sigma_L(L^t) = F_0^t \oplus \lambda_c (H_0^t)$, the determination of $\sigma_L$ can recover part of $L^t$. Then the
components of $L_0^t$ can be reduced, and the remaining unknown components of $L_1^t$ need be
guessed to recover the entire $L_1^t$. The complexity first decreases and then increases, while
has not changed overall. Similarly, for $F_0^{t+1} = S_0^{t+2} \oplus \mathcal{R}(S_2^{t+1}) \oplus S_2^{t+1}$, the determination of
The analysis and security claim are similar for LOL-DOUBLE.

To fit well with the form of the outputs, we can guess the columns and diagonals of the state blocks for each. For such reasons, we believe the time complexity of a guess-and-determine attack on LOL-MINI cannot be lower than $2^{256}$.

Then we consider the guess-and-determine attack based on bytes. For LOL-MINI, considering 3 consecutive keystream blocks, we have

$Z' = G^t \oplus N^t = \mathcal{R}(S_1^t) \oplus N^t,$

$Z^{t+1} = G^{t+1} \oplus N^{t+1} = \mathcal{R}(\mathcal{R}(S_1^t) \oplus S_2^t) \oplus \mathcal{R}(N^t) \oplus L^t,$

$Z^{t+2} = G^{t+2} \oplus N^{t+2} = \mathcal{R}(\mathcal{R}(S_0^t) \oplus S_3^t) \oplus \mathcal{R}(S_1^t) \oplus S_2^t) \oplus \mathcal{R}(\mathcal{R}(N^t) \oplus L^t) \oplus H^t$.

<table>
<thead>
<tr>
<th>Moments</th>
<th>Operations</th>
<th>Variables</th>
<th>Relations</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>Guess</td>
<td>$N_0^t$</td>
<td>$\mathcal{R}(S_1^t) = Z_0^t \oplus N_0^t$</td>
<td>$2^{128}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^t$</td>
<td></td>
<td>$2^{512}$</td>
</tr>
<tr>
<td>$t + 1$</td>
<td>Guess</td>
<td>$N_0^{t+1}$</td>
<td>$L_0^t = N_0^{t+1} \oplus \mathcal{R}(N_0^t)$</td>
<td>$2^{516}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$L_0^t$</td>
<td>$\mathcal{R}(S_1^{t+1}) = Z_0^{t+1} \oplus N_0^{t+1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^{t+1}$</td>
<td>$\mathcal{R}(S_0^t) = S_1^{t+1} \oplus S_1^t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t + 2$</td>
<td>Guess</td>
<td>$N_0^{t+2}$</td>
<td>$H_0^t = L_0^{t+1} = \mathcal{R}(N_0^{t+1}) \oplus N_0^{t+2}$</td>
<td>$2^{512}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$L_0^t$</td>
<td>$\mathcal{R}(S_1^{t+2}) = Z_0^{t+2} \oplus N_0^{t+2}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^{t+2}$</td>
<td>$\mathcal{R}(S_0^{t+1}) = S_1^{t+2} \oplus S_1^{t+1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^{t+1}$</td>
<td>$F_0^t = \mathcal{R}(S_0^t) \oplus S_1^t \oplus S_0^{t+1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^t$</td>
<td>$\sigma_L(L_1^t, L_0^t) = F_0^t \oplus C_0 \times H_0^t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$L_1^t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t + 1$</td>
<td>Determine</td>
<td>$N_1^{t+1}$</td>
<td>$N_1^{t+1} = \mathcal{R}(N_1^t) \oplus L_1^t$</td>
<td>$2^{512}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_3^t$</td>
<td>$\mathcal{R}(S_3^t) = Z_3^t \oplus N_1^t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_2^t$</td>
<td>$\mathcal{R}(S_2^t) = S_1^t \oplus S_3^t$</td>
<td></td>
</tr>
<tr>
<td>$t + 2$</td>
<td>Determine</td>
<td>$L_1^{t+2}$</td>
<td>$H_1^t = L_0^{t+1} \oplus F_0^t$</td>
<td>$2^{512}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$N_0^{t+3}$</td>
<td>$N_0^{t+3} = \mathcal{R}(N_0^{t+2}) \oplus N_0^{t+3}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^{t+3}$</td>
<td>$\mathcal{R}(S_1^{t+3}) = Z_0^{t+3} \oplus N_0^{t+3}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^{t+2}$</td>
<td>$\mathcal{R}(S_0^{t+2}) = S_1^{t+3} \oplus S_1^{t+2}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^{t+1}$</td>
<td>$F_0^{t+1} = S_0^{t+2} \oplus \mathcal{R}(S_1^{t+3}) \oplus S_0^{t+1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_0^t$</td>
<td>$\sigma_L(H_1^t)$</td>
<td>$\sigma_L(L_1^{t+1}) \oplus C_0 \times H_0^t = H_1^t[2, 5, 4]$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$S_1^t$</td>
<td>$F_0^{t+1} = \sigma_L(L_1^{t+1}) \oplus C_0 \times H_0^t = H_1^t[2, 5, 4]$</td>
<td>$2^{512}$</td>
</tr>
<tr>
<td></td>
<td>Determine</td>
<td>$L_1^t$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\sigma_H$ can recover part of $H^t$. Then the components of $H^t$ can be reduced, and the remaining unknown components of $H^t$ need be guessed to recover the entire $H^t$. Therefore, the total complexity is $2^{512}$.

For LOL-MINI, considering 3 consecutive keystream blocks, we have

$Z' = G^t \oplus N^t = \mathcal{R}(S_1^t) \oplus N^t,$

$Z^{t+1} = G^{t+1} \oplus N^{t+1} = \mathcal{R}(\mathcal{R}(S_1^t) \oplus S_2^t) \oplus \mathcal{R}(N^t) \oplus L^t,$

$Z^{t+2} = G^{t+2} \oplus N^{t+2} = \mathcal{R}(\mathcal{R}(S_0^t) \oplus S_3^t) \oplus \mathcal{R}(S_1^t) \oplus S_2^t \oplus \mathcal{R}(\mathcal{R}(N^t) \oplus L^t) \oplus H^t$.

To fit well with the form of the outputs, we can guess the columns and diagonals of the state blocks for $Z'$. However, for $Z^{t+1}$ and $Z^{t+2}$, two or more AES rounds are involved, which implies that the attackers at least need to guess a complete 128-bit block due to the full diffusion property of AES round function for each. For such reasons, we believe the time complexity of a guess-and-determine attack on LOL-MINI cannot be lower than $2^{256}$. The analysis and security claim are similar for LOL-DOUBLE.
5.8 Time/Memory/Data tradeoff attacks

For LOL framework. The TMD-TO attacks have two phases. In preprocessing phase, the mapping table from different secret keys or internal states to keystreams is computed and stored with time complexity $P$ and memory $M$. In the real-time phase, attackers have intercepted $D$ keystreams and search them in the table with time complexity $T$, expecting to get some matches and further recover the corresponding input. To balance the parameters, the most popular tradeoffs are Babbage-Golic [Bab95, Gol97] with $TM = N, P = M, T \leq D$ and Biryukov-Shamir [BS00] with $MT^2D^2 = N^2, P = N/D, T \leq D^2$, where $N$ is the input space.

For attacks to reconstruct the internal state, LOL ciphers will be immune if and only if it satisfies the rule that the internal state size should be at least twice the secret key size. Moreover, for the application of FP(1)-mode in the initialization phase, the attacks cannot benefit from reconstructing the internal state to recover the key. For attacks to recover the secret key directly, i.e., the attackers pre-compute some mappings from different key/IV pairs to generated keystream segments, there will exist a trivial attack but unrealistic to achieve in practice if the IV size is smaller than the key size. Therefore, the security bound against TMD-TO attacks of LOL ciphers can be given by the specific sizes of parameters such as state, key and IV.

For LOL-MINI and LOL-DDOUBLE. Firstly, for attacks to reconstruct the internal state, LOL-MINI and LOL-DDOUBLE are immune since they both satisfy the rule that the internal state size of 768 bits and 1280 bits should be at least twice the secret key size of 256 bits. Moreover, according to Babbage-Golic’s tradeoffs, the optimal choice of attack parameters are: $P = M = T = N^{1/2} = 2^{384}, D = N/P = 2^{384}$ for LOL-MINI with $N = 2^{768}$, and $P = M = T = N^{1/2} = 2^{649}, D = N/P = 2^{649}$ for LOL-DDOUBLE with $N = 2^{768}$. In addition, we use FP(1)-mode in the initialization phase, the attacks cannot benefit from reconstructing the internal state to recover the key.

Secondly, for attacks to recover the secret key directly, there is a trivial attack but unrealistic to achieve in practice on LOL-MINI, since the IV size of 128 bits is smaller than the key size of 256 bits, while it is resisted by LOL-DDOUBLE since the IV size of 256 bits is equal to the key size of 256 bits. Moreover, according to Biryukov-Shamir’s tradeoffs, the search space includes the IVs and the key of the cipher, and the optimal choice of attack parameters are: $P = M = T = D = N^{1/2} = 2^{192}$ for LOL-MINI with $N = 2^{384}$, and $P = M = T = D = N^{1/2} = 2^{256}$ for LOL-DDOUBLE with $N = 2^{512}$. 

5.9 Algebraic attacks

For LOL framework. In an algebraic attack the attacker derives a number of nonlinear equations in either unknown key bits or unknown state bits and solves the system of equations. For LOL ciphers, an algebraic representation of the update function can be given as follows: the S-boxes in the SP-network functions can be expressed as a set of nonlinear boolean functions by computing the annihilating implicit equations of the S-box, and the other operations can be expressed as a set of linear boolean functions by redefining some variables. According to the specific parameters, the attackers can judge whether the algebraic representation of an LOL cipher is solvable by the growth of the number of equations and variables with time, and the estimation of the algebraic degree.

For LOL-MINI and LOL-DDOUBLE. It is well-known that each AES S-box can be completely defined by a system of 23 linearly independent bi-affine quadratic equations between the input bits and the output bits in 80 terms, which contains 64 quadratic terms and 16 linear terms, and there are no more such equations [CP02].
We consider the algebraic attack for unknown state. For LOL-MINI, the initial variables are \((H^0, L^0, N^0, S^0_0, S^0_1, S^0_2)\), and the update function can be rewritten as 5 blocks of linear equations, each block has 128 bit-equations, and 4 blocks of quadratic equations, each block has \(23 \times 16 = 368\) bit-equations.

\[
Y_{S^2_2} = \text{SB}(S^2_0), \quad 0 \leq t \leq T; \\
Z^t = \text{MC} \circ \text{SR}(Y_{S^2_2}) \oplus N^0, \quad 0 \leq t \leq T; \\
N^{t+1} = \text{MC} \circ \text{SR}(Y_{N^t}) \oplus f^{t-1}(H^0, L^0), \quad 0 \leq t \leq T - 1; \\
Y_{N^t} = \text{SB}(N^t), \quad 0 \leq t \leq T - 1; \\
S^2_2^{t+1} = \text{MC} \circ \text{SR}(Y_{S^2_2}) \oplus S^2_0, \quad 0 \leq t \leq T - 1; \\
Y_{S^1_2} = \text{SB}(S^2_1), \quad 0 \leq t \leq T - 1; \\
S^1_2^{t+1} = \text{MC} \circ \text{SR}(Y_{S^1_2}) \oplus S^1_1, \quad 0 \leq t \leq T - 2; \\
Y_{S^0_2} = \text{SB}(S^2_0), \quad 0 \leq t \leq T - 2; \\
S^0_2^{t+1} = f^{t+1}(H^0, L^0) \oplus \text{MC} \circ \text{SR}(Y_{S^2_2}) \oplus S^0_0, \quad 0 \leq t \leq T - 3,
\]

where \(f^{t-1}(H^0, L^0) = L^0\) and \(f^0(H^0, L^0) = H^0\), by introducing variables of

\[
N^{t+1}, \quad 0 \leq t \leq T - 1; \quad S^0_2^{t+1}, \quad 0 \leq t \leq T - 3; \quad S^1_2^{t+1}, \quad 0 \leq t \leq T - 2; \quad S^2_2^{t+1}, \quad 0 \leq t \leq T - 1,
\]

and extra intermediate variables to represent the outputs of the S-box

\[
Y_{N^t}, \quad 0 \leq t \leq T - 1; \quad Y_{S^2_2}, \quad 0 \leq t \leq T - 2; \quad Y_{S^1_2}, \quad 0 \leq t \leq T - 1; \quad Y_{S^0_2}, \quad 0 \leq t \leq T.
\]

It also can be rewritten as the following equations by eliminating four blocks of intermediate linear equations

\[
Z^t = \begin{cases} 
\text{MC} \circ \text{SR}(Y_{S^2_2}) \oplus N^0, & t = 0 \\
\text{MC} \circ \text{SR}(Y_{S^2_2}) \oplus \text{MC} \circ \text{SR}(Y_{N^{t-1}}) \oplus f^{t-2}(H^0, L^0), & 1 \leq t \leq T 
\end{cases}
\]

\[
Y_{N^t} = \begin{cases} 
\text{SB}(N^0), & t = 0 \\
\text{SB}(\text{MC} \circ \text{SR}(Y_{N^{t-1}}) \oplus f^{t-2}(H^0, L^0)), & 1 \leq t \leq T - 1 
\end{cases}
\]

\[
Y_{S^2_2} = \begin{cases} 
\text{SB}(S^2_0), & t = 0 \\
\text{SB} \left( S^2_0 \oplus \bigoplus_{i=0}^{t-1} \text{MC} \circ \text{SR}(Y_{S^1_2}) \right), & 1 \leq t \leq T 
\end{cases}
\]

\[
Y_{S^1_2} = \begin{cases} 
\text{SB}(S^1_0), & t = 0 \\
\text{SB} \left( S^1_0 \oplus \bigoplus_{i=0}^{t-1} \text{MC} \circ \text{SR}(Y_{S^1_2}) \right), & 1 \leq t \leq T - 1 
\end{cases}
\]

\[
Y_{S^0_2} = \begin{cases} 
\text{SB}(S^0_0), & t = 0 \\
\text{SB} \left( S^0_0 \oplus \bigoplus_{i=0}^{t-1} \left( f^{t+1}(H^0, L^0) \oplus \text{MC} \circ \text{SR}(Y_{S^2_2}) \right) \right), & 1 \leq t \leq T - 2 
\end{cases}
\]

and only introduce the intermediate variables to represent the outputs of the S-box

\[
Y_{N^t}, \quad 0 \leq t \leq T - 1; \quad Y_{S^2_2}, \quad 0 \leq t \leq T - 2; \quad Y_{S^1_2}, \quad 0 \leq t \leq T - 1; \quad Y_{S^0_2}, \quad 0 \leq t \leq T.
\]

Moreover, \(Y_{S^2_2^{t+1}}\) and \(Y_{N^t}\) for \(0 \leq t \leq T - 1\) can be substituted by each other according to the linear equations for \(Z^t, 0 \leq t \leq T\), and then the above equation system becomes pure quadratic with large number of terms. The equations are derived similarly for LOL-DOUBLE.

For each of LOL-MINI and LOL-DOUBLE in the keystream generation phase, we can construct a system of equations like Eq. (14), and compute the total number of terms and the number of equations, listed in Table 8. We can see that the number of variables is always larger than the number of equations, thus LOL-MINI and LOL-DOUBLE are immune to algebraic attacks with linearization.
Table 8: The equation system of algebraic attack with linearization on LOL-MINI and LOL-DOUBLE

<table>
<thead>
<tr>
<th>Cipher</th>
<th>LOL-MINI</th>
<th>LOL-DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear eqs.</td>
<td>$128 \times (5T - 2)$</td>
<td>$128 \times 8T$</td>
</tr>
<tr>
<td>Quadratic eqs.</td>
<td>$368 \times 4T$</td>
<td>$368 \times (6T + 2)$</td>
</tr>
<tr>
<td>Equations</td>
<td>$2112T - 256$</td>
<td>$3232T + 736$</td>
</tr>
<tr>
<td>Linear terms</td>
<td>$128 \times (8T + 3)$</td>
<td>$128 \times (12T + 10)$</td>
</tr>
<tr>
<td>Quadratic terms</td>
<td>$(64 \times 16) \times 4T$</td>
<td>$(64 \times 16) \times (6T + 2)$</td>
</tr>
<tr>
<td>Terms</td>
<td>$5120T + 384$</td>
<td>$7680T + 3328$</td>
</tr>
</tbody>
</table>

We can also construct a system of equations for each of LOL-MINI and LOL-DOUBLE in the keystream generation phase like Eq. (15). Moreover, to obtain the equal information entropy as the initial unknown state, the minimum $t$ for LOL-MINI and LOL-DOUBLE are 6 and 10 respectively. The total number of variables and the number of quadratic equations are listed in Table 9. Then the methods for solving a system of multivariate quadratic equations such as XL algorithm are calling for them, while such a system of equations in a large number of variables can not be solved within a time complexity of $2^{256}$.

Table 9: The equation system of algebraic attack with MQ on LOL-MINI and LOL-DOUBLE

<table>
<thead>
<tr>
<th>Cipher</th>
<th>LOL-MINI</th>
<th>LOL-DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min T$</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Quadratic eqs.</td>
<td>8832</td>
<td>22816</td>
</tr>
<tr>
<td>Variables $n$</td>
<td>2944</td>
<td>6400</td>
</tr>
<tr>
<td>Complexity of XL</td>
<td>$(\frac{n}{\sqrt{m}})^{\omega}$</td>
<td>$2^{357\omega}$</td>
</tr>
</tbody>
</table>

† $\omega$ is the parameter for solving linear equation system.

6 Software Implementation and Performance

In this section, we evaluate the software performance of LOL-MINI and LOL-DOUBLE implemented in C++ (Visual Studio 2022) utilizing AVX2/AVX512/AES-NI/PCLMULQDQ intrinsics on a laptop with Intel i7-11800H CPU 2.30GHz with Turbo Boost up to 4.6GHz. All experiments are carried out in a single process/thread manner. We perform a key/IV setup procedure before each encryption process with various lengths of the input plaintext. The software performance is measured in Gbps as stream ciphers used to do. Test vectors and reference implementations are given in Appendix D, and Appendix E. This section is written with Intel intrinsics notation, but similar implementations can also be made on other CPUs, e.g. AMD and ARM.

Encryption/Decryption Speeds. Unlike the block ciphers where each message block is encrypted by a constant number of round function calls, stream ciphers usually start
Table 10: The software performance (Gbps) of AEGIS, SNOW-V, Rocca, LOL-MINI and LOL-DOUBLE in the pure encryption mode. Note that the LOL-DOUBLE are implemented in both AVX2 and AVX512 manners while others only use the AVX2 instructions. The shortest message length reaching 20Gbps are represented in a **bold** format.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>AEGIS256</th>
<th>SNOW-V</th>
<th>Rocca</th>
<th>LOL-MINI</th>
<th>LOL-DOUBLE</th>
<th>LOL-DOUBLE†</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>12.58</td>
<td>5.35</td>
<td>12.47</td>
<td>11.22</td>
<td>6.77</td>
<td>9.36</td>
</tr>
<tr>
<td>64</td>
<td>19.94</td>
<td>9.61</td>
<td>23.20</td>
<td>19.58</td>
<td>13.04</td>
<td>17.37</td>
</tr>
<tr>
<td>96</td>
<td>24.70</td>
<td>13.31</td>
<td>31.25</td>
<td>24.34</td>
<td>18.16</td>
<td>23.52</td>
</tr>
<tr>
<td>128</td>
<td>29.30</td>
<td>16.14</td>
<td>40.26</td>
<td>30.88</td>
<td><strong>22.55</strong></td>
<td>29.47</td>
</tr>
<tr>
<td>160</td>
<td>34.17</td>
<td>19.14</td>
<td>47.25</td>
<td>36.53</td>
<td>26.85</td>
<td>34.88</td>
</tr>
<tr>
<td>192</td>
<td>38.34</td>
<td><strong>21.54</strong></td>
<td>54.78</td>
<td>40.55</td>
<td>31.27</td>
<td>38.86</td>
</tr>
<tr>
<td>224</td>
<td>41.90</td>
<td>23.62</td>
<td>60.84</td>
<td>44.56</td>
<td>34.57</td>
<td>43.41</td>
</tr>
<tr>
<td>256</td>
<td>45.75</td>
<td>25.47</td>
<td>66.82</td>
<td>45.20</td>
<td>37.08</td>
<td>47.93</td>
</tr>
<tr>
<td>1024</td>
<td>70.77</td>
<td>40.73</td>
<td>109.51</td>
<td>71.22</td>
<td>68.30</td>
<td>92.40</td>
</tr>
<tr>
<td>2048</td>
<td>79.54</td>
<td>45.44</td>
<td>129.83</td>
<td>79.62</td>
<td>82.39</td>
<td>110.54</td>
</tr>
<tr>
<td>4096</td>
<td>84.96</td>
<td>49.52</td>
<td>141.78</td>
<td>85.01</td>
<td>91.43</td>
<td>122.42</td>
</tr>
<tr>
<td>8192</td>
<td>87.83</td>
<td>51.39</td>
<td>147.85</td>
<td>87.84</td>
<td>96.71</td>
<td>129.52</td>
</tr>
<tr>
<td>16384</td>
<td>89.38</td>
<td>52.37</td>
<td>152.04</td>
<td>89.42</td>
<td>99.59</td>
<td>135.00</td>
</tr>
</tbody>
</table>

†: Implemented with the AVX512 instructions.

with a constant-round initialization phase for a thoroughly diffused internal state followed by iteratively calls of a keystream generation function for producing pseudo-random bits for encryptions and internal state updates simultaneously. Therefore, when we use short messages, the software performance reflects mainly the efficiency of the initialization phase; for long messages, on the contrary, the performance is determined mainly by the keystream generation function. Since the software performance of stream ciphers are more sensitive to the message lengths than block ciphers, we evaluate the software performances using message lengths ranging from 32 to 16384 bytes. We also run the same experiments on the SNOW-V stream cipher, the pure encryption mode of AEGIS and Rocca on the same platform using exactly the source codes from the origins [EJMY19, WP14, SLN+21] for fair comparisons. The data are shown in Table 10.

As can be seen, the encryption/decryption speeds of LOL-MINI and LOL-DOUBLE can reach 89 Gbps and 99 Gbps with the AVX2 instructions respectively. When using AVX512 instructions, the speed of LOL-DOUBLE can reach 135 Gbps catering for not only 5G but also 6G requirements. The LOL-MINI and AEGIS share a similar efficiency (LOL-MINI is only slightly faster than AEGIS on short messages), slower than LOL-DOUBLE but much faster than SNOW-V. Rocca still enjoys the highest speed, but with different security target, such as 128-bit security against distinguishing. LOL-MINI and LOL-DOUBLE indicate that the LNFSR+AES structure can still be competitive with pure AES based primitives. In addition, considering the security issues listed in the introduction and the diversity of cipher designs (not only relying on the AES encryption round function), LOL-MINI and LOL-DOUBLE are more competitive. It is also noticeable that the AEGIS performance on our platform is much better than the original 56 Gbps in [WP14] indicating that the latest CPUs has made progress in supporting AES-NI instructions and such improvements can be fully utilized by the frequent usage of the parallel AES operations. SNOW-V, on the contrary, share the same speed with the origin reflecting the necessity of optimal non-AES
components designs. Moreover, combined with standard GCM and NMH modes, LOL-MINI and LOL-DOUBLE stream ciphers can also be applied to AEAD schemes generating 128-bit tags at speeds over 40 Gbps, which is much better than that of the similar AEAD application of SNOW-V [EJMY19], please refer to Appendix A for more information.

7 Conclusions

In this paper, we propose LOL framework for designing extendable, software efficient stream ciphers with high parallelism and flexibility. It is also friendly to automatic cryptanalysis tools for conducting security evaluations. Especially, we provide 2 concrete stream cipher designs namely LOL-MINI and LOL-DOUBLE, which support 256-bit key and have software performances of 89 Gbps and 135 Gbps, satisfying the speed and security requirements in beyond 5G/6G mobile system. Combined with standard GCM and NMH modes, LOL-MINI and LOL-DOUBLE stream ciphers can also be applied to AEAD schemes generating 128-bit tags at speeds over 40 Gbps, much faster than the 28 Gbps SNOW-V-GCM counterpart. We expect to see more designs adopting the LOL framework in the future.

References


A Applications of LOL-MINI and LOL-DOUBLE in AEAD Schemes

In [EJMY19], the SNOW-V stream cipher is combined with the GHASH based Galois Counter Mode (GCM) to make an AEAD scheme called SNOW-V-GCM. Such GHASH, as well as other universal hash based MACs, enjoys high speeds and concrete security proofs. Naturally, combined with universal hash function based MACs, LOL-MINI and LOL-DOUBLE stream ciphers can also be applied in AEAD schemes providing both confidentiality and integrity protections. We consider 2 AEAD modes based on universal hash functions GHASH and NMH respectively. The descriptions of the 2 AEAD modes are provided in Appendix A.1 and Appendix A.2. The software performance evaluations are given in Appendix A.3 for 128-bit tag versions.

A.1 GHASH based AEAD (GCM)

In the GHASH based AEAD modes, the keystream bits of stream ciphers are divided into \( n \)-bit blocks denoted as \( Z^0, Z^1, \ldots \). The universal hash function GHASH applies multiplication over finite field \( \mathbb{F}_{2^n} \). The general process of the GHASH based AEAD mode is shown in Fig. 11. As can be seen, the \( Z^0, Z^1, \ldots \), are \( n \)-bit keystream blocks generated by the stream cipher. The 1st keystream block \( Z^0 \) is used as the key \( H \) of GHASH and the 2nd keystream block \( Z^1 \) is used as the final masking \( M_T \) for the tag. The subsequent keystream blocks \( Z^2, \ldots, Z^{l+1} \) are used to encrypt the plaintext blocks \( P^1, \ldots, P^l \) and generate the ciphertext blocks \( C^1, \ldots, C^l \), feeding into GHASH for computing the final tag \( T \): such tag can also be truncated to \( \tau \leq n \) bits according to different needs. More details on message/associate data padding rules as well as other restrictions can be found in [Dwo07b].

![Figure 11](image-url)  
Figure 11: The schematic of GHASH based AEAD process where \( \text{MUL}_H \) refers to multiplying \( H \) over the finite field \( \mathbb{F}_{2^n} \).

Security Analysis. Let \( \ell \) be the maximum block length in queries, it is proved in [MV04] that the GHASH on \( \mathbb{F}_{2^n} \) is \((l+1)2^{-\tau}\) almost XOR universal. In other words, let \( H \) denote the GHASH key, the following Eq. (16) holds for any \( M, M' \in \mathbb{F}_{2^n}^* \) and \( a \in \{0, 1\}^\tau \),

\[
\text{Pr}[\text{GHASH}(H, M) \oplus \text{GHASH}(H, M') = a|H \leftarrow \mathbb{F}_{2^n}^*] \leq (l+1)2^{-\tau}
\]  
(16)
where \( a \leftarrow A \) means select an element \( a \) from set \( A \) uniformly at random. Let \( SC \) be arbitrary stream cipher generating the keystream blocks \( Z^0, Z^1, \ldots \), it can be prove that the security of \( SC \)-GCM in Fig. 11 relies on the randomness of \( SC \): Theorem 1 for encryption and Theorem 2 for authentication.

**Theorem 1. (Privacy)** Let \( A \) be an adversary that making at most \( q \) forward queries of total block length no more than \( \sigma \). Then

\[
\text{Adv}_{SC-GCM}^{\text{priv}}(A) \leq \text{Adv}_{SC}^{\text{prf}}(q, \sigma + 2q) + q^2 \ell/2^{\tau+1},
\]

where \( \text{Adv}_{SC}^{\text{prf}} \) is the adversary’s advantage in distinguishing the \( SC \) stream cipher from a random function.

**Proof.** The advantage of distinguishing the ciphertexts of \( SC \)-GCM from random bits is bounded by \( \text{Adv}_{SC}^{\text{prf}}(q, \sigma + 2q) \). The authentication tags also affect the privacy of \( SC \)-GCM. The tag is equal to the output of the GHASH function XOR the mask \( M_T \) generated by LOL. Note that the GHASH function is a hash function that satisfies \( \epsilon \)-almost XOR (AXU). For any \( M, M' \leftarrow F_2^n \) and \( a \in F_2^\tau \), it should have

\[
\Pr[\text{GHASH}(H, M) \oplus \text{GHASH}(H', M') = a | H, H' \leftarrow F_2^n] \leq \epsilon,
\]

where \( \epsilon \) is taken as \((l + 1)2^{-\tau}, \) following Eq. (16). For a random function, the probability that any two tags satisfy a given difference is \( 1/2^\tau \). Therefore, the distinguishing advantage of tags in any two queries is \( l/2^\tau \). For an adversary that making at most \( q \) queries, the distinguishing advantage should be bounded with \( q^2 \ell/2^{\tau+1} \).

**Theorem 2. (Authenticity)** Let \( A \) be an adversary that making at most \( q \) forward queries of total block length no more than \( \sigma \), and at most \( q' \) backward queries of total block length no more than \( \sigma' \). Then

\[
\text{Adv}_{SC-GCM}^{\text{auth}}(A) \leq \text{Adv}_{SC}^{\text{prf}}(q + q', \sigma + 2q + 2q') + q'(l + 1)/2^\tau.
\]

**Proof.** For the authenticity, McGrew and Viega [MV04] proved that AXU-MAC, which combines an \( \epsilon \)-almost XOR universal function and a random function, is a strong MAC. And for an adversary that making at most \( q' \) backward queries, the forgery advantage against AXU-MAC is no greater than \( q' \epsilon \). The result is obtained by substituting \( \epsilon = (l + 1)2^{-\tau} \) into \( q' \epsilon \) and combining with the advantage of distinguishing the \( SC \) against a random function.

As can be seen from Theorem 1 and Theorem 2, the secure bounds of GHASH based AEAD schemes is determined by not only the tag length \( \tau \) but also the total length of plaintext and associated data \( \ell \). It is recommended that \( \ell \) should be limited to \( 2^{n/2} \) blocks per key.

### A.2 NMH base AEAD

NMH (Nonlinear Modular Hashing) is a family of universal hash function [WC81] that was originally analyzed by Mark Wegman and Larry Carter, and formally introduced by Shai Halevi and Hugo Krawczyk with their proposed MMH (Multilinear Modular Hashing) [HK97]. Compared with MMH, NMH uses dot-product to reduce the number of multiplications, which has benefited several cryptographic designs, e.g. UMAC [BHK’99].

Similar with GHASH, for generating \( n \)-bit tag, NMH is also defined over finite field \( F_{2^n} \), where \( \oplus \) and \( \otimes \) stand for addition and multiplication respectively. The general process for NMH to generate \( n \)-bit MAC tag is described as Algorithm 6. As can be seen, for each message \((m_1, m_2, \cdots, m_l)\) with an even block length \( \ell \), nmhMAC takes \( \ell + 1 \) keystream blocks \((r, k_1, \ldots, k_l)\) and outputs an \( n \)-bit tag. It is required that different keystream
blocks should be used for different messages. For arbitrary stream cipher SC, the NMH based AEAD mode SC-NMH is derived naturally as Algorithm 7. As can be seen in Step 5 of Algorithm 7, the stream cipher SC is first initialized with key K and IV IV before generating the keystream blocks $s_0, s'_0, \ldots$.

**Algorithm 6: nmhMAC**

**Input:** Keystream blocks $r, k_1, k_2, \ldots, k_l \in \mathbb{F}_2^n$, message blocks $m_1, m_2, \ldots m_l \in \{0,1\}^n$, where $l$ is even

**Output:** MAC tag $Tag \in \mathbb{F}_2^n$

1. $nmh = \bigoplus_{i=1}^{l/2} (m_{2i-1} \oplus k_{2i-1}) \oplus (m_{2i} \oplus k_{2i})$;
2. Compute $Tag = nmh \oplus r$;

**Algorithm 7: SC-NMH AEAD mode**

**Input:** The associate data $AD \in \mathbb{F}_2^n$, $M \in \mathbb{F}_2^n$, the key and IV for initialize SC denoted as $(K, IV_{AEAD})$

**Output:** $C \in \mathbb{F}_2^{\lfloor M/2 \rfloor}, \text{Tag} \in \mathbb{F}_2^n$

1. $Pad(AD) = AD||0^n||\lfloor \text{BitLen}(AD) \rfloor_{64}$, where $v$ is the minimum number that makes the padded AD length is $ln$ and $l$ is even;
2. $\text{Partition}(Pad(AD)) \rightarrow ad_1, ad_2, \ldots, ad_l$;
3. $Pad(M) = M||0^n||\lfloor \text{BitLen}(M) \rfloor_{64}$, where $u$ is the minimum number that makes the padded $M$ length is $Ln$ and $L$ is even;
4. $\text{Partition}(Pad(M)) \rightarrow m_1, m_2, \ldots, m_L$;
5. $s_0, s_1, s_2, \ldots, s_l, s_{i+1}, s_{i+1}, \ldots, s_{i+L}, s_{i+L} \leftarrow SC(K, IV_{AEAD})$;
6. $(c_1, c_2, \ldots, c_L) = (m_1, m_2, \ldots, m_L) \oplus (s_{i+1}, s_{i+2}, \ldots, s_{i+L})$;
7. $C = MSB_m(c_1, c_2, \ldots, c_L)$;
8. $nmhMAC_{AD} = nmhMAC(s_0, s_1, s_2, \ldots, s_l, ad_1, ad_2, \ldots, ad_l)$;
9. $nmhMAC_M = nmhMAC(s_0, s_{i+1}, s_{i+2}, \ldots, s_{i+L}, m_1, m_2, \ldots, m_L)$;
10. $Tag = nmhMAC_{AD} \oplus nmhMAC_M$;

**Security Analysis.** The security of NMH MAC is extensively studied in [HK97] and can be summarized as the following Proposition 1. Similarly, the security of SC-NMH in Algorithm 7 can also be proved as Proposition 2. As can be seen, the secure bounds of NMH based AEAD is independent to the lengths of plaintexts and associate data.

**Proposition 1.** Assuming that $r, k_1, k_2, \ldots, k_l \in \mathbb{F}_2^n$ of Algorithm 6 are independently-and-uniformly random keys for each new message, nmhMAC is a secure MAC. That is, for any adversary trying to forge against nmhMAC, its success probability is no more than $3/2^n$.

**Proof.** For any adversary trying to forge against nmhMAC, suppose it has made $q$ queries $(m_1^j, m_2^j, \ldots, m_l^j)$ for $i = 1, 2, \ldots, q$ to nmhMAC, and got the corresponding tags $Tag_1^j, Tag_2^j, \ldots, Tag_q^j$. Now, the adversary tries to make a forgery $(\tilde{m}_1, \tilde{m}_2, \ldots, \tilde{m}_l)$ with $Tag$.

If it tries to challenge nmhMAC with a new key $(r^i, k_1^i, k_2^i, \ldots, k_l^i) \notin \{(r^i, k_1, k_2, \ldots, k_l), i \in [1, q]\}$, then by nmhMAC construction and the randomness of $(r, k_1, k_2, \ldots, k_l)$, we get

$$Pr[\bigoplus_{i=1}^{l/2} (m_{2i-1} \oplus k_{2i-1}) \oplus (\tilde{m}_{2i} \oplus k_{2i}) \oplus r = Tag] = \frac{1}{2^n}.$$ 

Otherwise, if the adversary tries to challenge nmhMAC with an old key. That is, $\exists j \in [1, q]$, s.t. $(r, k_1, k_2, \ldots, k_l)$ is a prefix of $(r^i, k_1^i, k_2^i, \ldots, k_l^i)$ or vice versa, we consider in three cases.
1. \( l = l_j \). Noticing that \((\overline{m_1}, \overline{m_2}, \cdots, \overline{m_l}, \overline{Tag}) \neq (m_1', m_2', \cdots, m_l', Tag')\), we analyze in the following two subcases.

(a) \((\overline{m_1}, \overline{m_2}, \cdots, \overline{m_l}) \neq (m_1', m_2', \cdots, m_l')\), which means \( \exists x \in [1, l_j] \) s.t. \( \overline{m_x} \neq m_x' \). When \( x \) is odd, let \( a = (\overline{m_x} \oplus k_x') \oplus (m_x' \oplus k_x') \oplus (m_x'' \oplus k_x'') \), and \( b = \bigoplus_{i=1}^{l_j/2} (\overline{m_{2i-1}} \oplus k_{2i-1}) \oplus (\overline{m_{2i}} \oplus k_{2i}) \oplus \bigoplus_{i=1}^{l_j/2} (m_{2i-1}' \oplus k_{2i-1}') \oplus (m_{2i}' \oplus k_{2i}') \oplus a \), we have

\[
\Pr[(\overline{m_x} \oplus k_x') \oplus (m_x' \oplus k_x') \oplus (m_x'' \oplus k_x'') = b \oplus \overline{Tag} \oplus Tag'] = \Pr[(\overline{m_x} \oplus k_x') \oplus (m_x' \oplus k_x') \oplus (m_x'' \oplus k_x'') = b \oplus \overline{Tag} \oplus Tag'] = \Pr[(\overline{m_x} \oplus m_x') k_x' = \overline{m_x} \oplus m_x' k_x' \oplus b \oplus \overline{Tag} \oplus Tag'] = \frac{1}{2^n}
\]

since each key including \( k_x' \) is independently-and-uniformly random. When \( x \) is even, similar analysis applies.

(b) \((\overline{m_1}, \overline{m_2}, \cdots, \overline{m_l}) = (m_1', m_2', \cdots, m_l')\), but \( \overline{Tag} \neq Tag' \), then by \texttt{nmhMAC} construction, we have \( \Pr[a = Tag \oplus Tag'] = 0 \).

2. \( l > l_j \), let us focus on the last multiplication to compute \( Tag \). Denote

\[
a_1 = (\overline{m_{l-1}} \oplus k_{l-1}') \oplus (\overline{m_l} \oplus k_l')
\]

and

\[
a_2 = \bigoplus_{i=1}^{l_j/2} (\overline{m_{2i-1}} \oplus k_{2i-1}) \oplus (\overline{m_{2i}} \oplus k_{2i}) \oplus a_1,
\]

so we have

\[
\Pr[\bigoplus_{i=1}^{l_j/2} (\overline{m_{2i-1}} \oplus k_{2i-1}) \oplus (\overline{m_{2i}} \oplus k_{2i}) \oplus Tag = \overline{Tag}]
\]

\[
= \Pr[a_1 \oplus a_2 \oplus Tag = \overline{Tag}]
\]

\[
= \Pr[a_1 \oplus \overline{Tag} \oplus \bigoplus_{i=1}^{l_j/2} (m_{2i-1}' \oplus k_{2i-1}') \oplus (m_{2i}' \oplus k_{2i}') \oplus \overline{Tag'}] \leq 3/2^n,
\]

where \( \Pr[\overline{m_{l-1}} \oplus k_{l-1}' = 0] = 1/2^n \), \( \Pr[\overline{m_l} \oplus k_l' = 0] = 1/2^n \), and \( \Pr[a_1 = w] = \frac{1}{2^n} \) for any \( w \neq 0 \) independent of \( a_1 \).

3. \( l < l_j \), by the independent-and-uniform randomness of \( k_{l-1}' \) and \( k_l' \), similar analysis applies.

By the above, we conclude the proof.

\[ \square \]

**Proposition 2.** Assuming that \( SC \) is a secure stream cipher, \( SC-NMH \) in Algorithm 7 is a secure AEAD. That is, for any adversary that never repeats IVs, its advantage against LOL+\texttt{nmhMAC} is upper bounded by,

\[
\text{Adv}_{SC-NMH}^{\text{prev}}(q) \leq \text{Adv}_{SC}^{\text{prf}}(q), \quad (17)
\]

\[
\text{Adv}_{SC-NMH}^{\text{auth}}(q) \leq 3/2^n + \text{Adv}_{SC}^{\text{prf}}(q+1). \quad (18)
\]
Proof. Suppose an adversary $A$ has made $q$ queries to SC-NMH, getting the corresponding ciphertexts and tags, and it never repeats $IV_{AEAD}$, then its advantage to distinguish the SC outputs $s_0, s_1, s_2, s_{t+1}, s_{t+1}, \cdots, s_{t+L}, s_{t+L}$ from random bits is upper bounded by $\text{Adv}_{SC}(q)$ which, for the privacy aspect, proves Proposition 2. For the authentication aspect, suppose $A$ makes a forgery $(IV_{AEAD}, AD, C, Tag)$, where the padded $AD$ and $C$ have $l$ and $l'$ blocks respectively, and we denote $M$ as the expected plaintext for $C$. Then we consider in two cases.

1. $IV_{AEAD}$ is a new IV. That is, $IV_{AEAD} \neq IV_{AEAD}^i$ for $i \in [1, q]$. The by the prf security of SC, we have

$$\Pr[\text{the forgery is valid}] \leq 1/2^n + \text{Adv}_{LOL}^{prf}(q + 1).$$

2. $\exists i \in [1, q]$ s.t. $IV_{AEAD} = IV_{AEAD}^i$. Then we consider in the following five subcases.

(a) $l + l' > s_i + l'$. By the independent-and-uniform randomness of $s_i'$, we have

$$\Pr[\text{matterMAC}_M = Tag \oplus \text{nmhMAC}_{AD}] = 3/2^n + \text{Adv}_{LOL}^{prf}(q).$$

(b) $l + l' = s_i + l'$ and $l < l'$, and $l > l'$. Notice that by the nmhMAC padding rule, we have $\overline{M} \neq m_L$, then by the randomness of $s_i' + e$, we have

$$\Pr[\text{matterMAC}_M = Tag \oplus \text{nmhMAC}_{AD}] = \Pr[\text{matterMAC}_M \oplus \text{nmhMAC}_{M_L} = Tag \oplus \text{nmhMAC}_{AD} \oplus Tag' \oplus \text{nmhMAC}_{AD}] = 3/2^n + \text{Adv}_{LOL}^{prf}(q).$$

(c) $l + l' = s_i + l'$ and $l < l'$, and $l < l'$. Notice that by the nmhMAC padding rule, we have $\overline{M} \neq m_L$, then by the randomness of $s_i' + e$, we have

$$\Pr[\text{matterMAC}_M = Tag \oplus \text{nmhMAC}_{AD}] = \Pr[\text{matterMAC}_M \oplus \text{nmhMAC}_{M_L} = Tag \oplus \text{nmhMAC}_{AD} \oplus Tag' \oplus \text{nmhMAC}_{AD}] = 3/2^n + \text{Adv}_{LOL}^{prf}(q).$$

(d) $l = l'$ and $l = l'$. In this subcase, we have $(AD, C) \neq (AD', C')$, which means $3AD_{j,1} \neq AD_{j,1}'$ for some $j, 1 \in [1, l]$; or $3C_{j,2} \neq C_{j,2}'$ for some $j, 2 \in [1, l]$, or both exist. Then by the independently random keys $s_1, s_2, \cdots, s_l, s_{l+1}, s_{l+2}, \cdots, s_{l+L}$, we have

$$\Pr[\text{matterMAC}_M = Tag \oplus \text{nmhMAC}_{AD}] = \Pr[\text{matterMAC}_M \oplus \text{nmhMAC}_{M_L} = Tag \oplus \text{nmhMAC}_{AD} \oplus Tag' \oplus \text{nmhMAC}_{AD}] = 3/2^n + \text{Adv}_{LOL}^{prf}(q).$$

(e) $l + l' < s_i + l'$. By the independent-and-uniform randomness of $s_i' + e$, we have

$$\Pr[\text{matterMAC}_M = Tag \oplus \text{nmhMAC}_{AD}] = \Pr[\text{matterMAC}_M \oplus \text{nmhMAC}_{M_L} = Tag \oplus \text{nmhMAC}_{AD} \oplus Tag' \oplus \text{nmhMAC}_{AD}] = 3/2^n + \text{Adv}_{LOL}^{prf}(q).$$

By the above, we conclude the proof.
A.3 Software Performance of AEAD Mode

As can be seen in Appendix A.1 and Appendix A.2, both GHASH and NMH are using operations over finite field $\mathbb{F}_{2^n}$ for generating $n$ bit tags. Such $n$ can be arbitrary positive integer. In modern CPUs, there is a specific support for 64-bit carry-less multiplications which largely accelerates the $\otimes$ operation over $\mathbb{F}_{2^{128}}$ [GK]. Therefore, both the standard AES-GCM and the SNOW-V GCM in [EJMY19] generate 128-bit tags with the finite field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/g(x)$ defined by 128-degree low-hamming-weight irreducible polynomial

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

Using the same field, we apply stream ciphers SNOW-V, LOL-MINI and LOL-DOUBLE to standard 128-bit GHASH and NMH based AEAD schemes for software performance evaluations listing the results in Table 11 and Table 12 respectively. As can be seen, both LOL-MINI and LOL-DOUBLE enjoy higher speeds than SNOW-V, either in GCM or NMH modes. LOL-MINI has advantage over LOL-DOUBLE in GCM mode. LOL-DOUBLE enjoys the highest speed in NMH mode when AVX512 instructions are available. The lower speed of LOL-DOUBLE results from the different lengths between output blocks and AEAD blocks: each 256-bit LOL-DOUBLE output block has to be split into 2 128-bit blocks before taking part in AEAD operations while the 128-bit output blocks of LOL-MINI can directly be used in AEAD schemes.

Table 11: The software performance (Gbps) of SNOW-V, LOL-MINI and LOL-DOUBLE in GHASH based AEAD modes.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>SNOW-V</th>
<th>LOL-MINI</th>
<th>LOL-DOUBLE</th>
<th>LOL-DOUBLE$^\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3.14</td>
<td>5.08</td>
<td>3.71</td>
<td>4.03</td>
</tr>
<tr>
<td>64</td>
<td>5.95</td>
<td>9.70</td>
<td>7.26</td>
<td>8.22</td>
</tr>
<tr>
<td>96</td>
<td>7.60</td>
<td>11.97</td>
<td>9.11</td>
<td>9.92</td>
</tr>
<tr>
<td>128</td>
<td>9.91</td>
<td>15.88</td>
<td>12.03</td>
<td>13.75</td>
</tr>
<tr>
<td>160</td>
<td>10.76</td>
<td>16.69</td>
<td>12.94</td>
<td>14.36</td>
</tr>
<tr>
<td>192</td>
<td>12.67</td>
<td>20.05</td>
<td>15.55</td>
<td>17.72</td>
</tr>
<tr>
<td>224</td>
<td>13.09</td>
<td>20.25</td>
<td>15.90</td>
<td>17.73</td>
</tr>
<tr>
<td>256</td>
<td>14.72</td>
<td>23.05</td>
<td>18.18</td>
<td>20.69</td>
</tr>
<tr>
<td>1024</td>
<td>23.11</td>
<td>36.05</td>
<td>29.77</td>
<td>33.38</td>
</tr>
<tr>
<td>2048</td>
<td>25.59</td>
<td>39.71</td>
<td>33.19</td>
<td>37.15</td>
</tr>
<tr>
<td>4096</td>
<td>27.03</td>
<td>41.71</td>
<td>35.25</td>
<td>39.39</td>
</tr>
<tr>
<td>8192</td>
<td>27.81</td>
<td>42.90</td>
<td>36.39</td>
<td>40.59</td>
</tr>
<tr>
<td>16384</td>
<td>28.23</td>
<td>43.42</td>
<td>36.96</td>
<td>41.21</td>
</tr>
</tbody>
</table>

$^\dagger$: Implemented with the AVX512 instructions.

It is also noticeable that the high speed of generating 128-bit GCM/NMH tags relies on the support of 64-bit carry-less multiplications of current CPUs. When 128-or-more-bit carry-less multiplications are supported by future CPUs, the large output block of LOL-DOUBLE will become an advantage for generating longer GHASH/NMH tags. For example, for 256-bit tags, we may use finite field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/h(x)$ defined with the low-hamming-weight irreducible polynomial $h(x)$ as follows.

$$h(x) = x^{256} + x^{10} + x^5 + x^2 + 1$$

So the current drawbacks of LOL-DOUBLE may become advantages in the future.
Table 12: The software performance (Gbps) of SNOW-V, LOL-MINI and LOL-DOUBLE in NMH based AEAD modes.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>SNOW-V</th>
<th>LOL-MINI</th>
<th>LOL-DOUBLE</th>
<th>LOL-DOUBLE†</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3.13</td>
<td>5.59</td>
<td>4.44</td>
<td>5.29</td>
</tr>
<tr>
<td>64</td>
<td>5.45</td>
<td>9.63</td>
<td>7.75</td>
<td>9.81</td>
</tr>
<tr>
<td>96</td>
<td>7.30</td>
<td>12.81</td>
<td>10.50</td>
<td>13.37</td>
</tr>
<tr>
<td>128</td>
<td>8.75</td>
<td>15.27</td>
<td>12.68</td>
<td>16.17</td>
</tr>
<tr>
<td>160</td>
<td>10.02</td>
<td>17.33</td>
<td>14.48</td>
<td>18.77</td>
</tr>
<tr>
<td>192</td>
<td>10.97</td>
<td>18.96</td>
<td>15.95</td>
<td>20.87</td>
</tr>
<tr>
<td>224</td>
<td>11.94</td>
<td>20.36</td>
<td>17.24</td>
<td>22.78</td>
</tr>
<tr>
<td>256</td>
<td>12.62</td>
<td>21.56</td>
<td>18.24</td>
<td>24.29</td>
</tr>
<tr>
<td>1024</td>
<td>18.69</td>
<td>30.83</td>
<td>27.45</td>
<td>38.28</td>
</tr>
<tr>
<td>2048</td>
<td>20.29</td>
<td>33.49</td>
<td>30.01</td>
<td>42.51</td>
</tr>
<tr>
<td>4096</td>
<td>21.55</td>
<td>34.52</td>
<td>31.44</td>
<td>45.01</td>
</tr>
<tr>
<td>8192</td>
<td>21.73</td>
<td>35.55</td>
<td>32.22</td>
<td>46.35</td>
</tr>
<tr>
<td>16384</td>
<td>21.98</td>
<td>35.48</td>
<td>33.16</td>
<td>47.06</td>
</tr>
</tbody>
</table>

†: Implemented with the AVX512 instructions.

B Hardware Implementation Aspects

We implement LOL-MINI and LOL-DOUBLE with the Verilog hardware description language and report the resource consumptions using the SM2C10LL shown in Table 13.

Table 13: The hardware speed evaluations for LOL ciphers

<table>
<thead>
<tr>
<th>LOL-DOUBLE</th>
<th>Power (W/Hz)</th>
<th>Area (GE)</th>
<th>non-AES(GE)</th>
<th>Throughput(Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6AES</td>
<td>0.0675</td>
<td>76645.73</td>
<td>29757.97</td>
<td>256</td>
</tr>
<tr>
<td>2AES</td>
<td>0.0319</td>
<td>41184.20</td>
<td>25554.95</td>
<td>85.3</td>
</tr>
<tr>
<td>1AES</td>
<td>0.0168</td>
<td>29953.02</td>
<td>22138.39</td>
<td>42.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOL-MINI</th>
<th>Power (W/Hz)</th>
<th>Area (GE)</th>
<th>Throughput(Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4AES</td>
<td>0.0442</td>
<td>50502.55</td>
<td>19244.05</td>
</tr>
<tr>
<td>2AES</td>
<td>0.0200</td>
<td>33730.97</td>
<td>18101.72</td>
</tr>
<tr>
<td>1AES</td>
<td>0.0146</td>
<td>22416.83</td>
<td>14601.40</td>
</tr>
</tbody>
</table>

For the main differences when targeting the hardware, LOL Framework supports the hardware-friendly expansion of the LOL family with plenty of circuit reuse strategies.

C Galois LFSR in LOL-MINI

The overall Galois LFSR is shown in Fig. 12.
D Test vectors

The 8-bit bytes are separated with _ symbols. The 128/256-bit states are represented from the most significant byte to the least significant one from left to right.

A test vector for LOL-MINI is presented as follows.

**LOL-MINI**

Key: 4e_38_40_08_81_c1_dc_aa_87_f5_9b_4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27

IV: 8a_fd_d1_19_97_f0_b5_f1_39_76_dd_2d_ad_97_f6_26

\[ t = -12: \]

\[ H:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00 \]

\[ L:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00 \]

\[ N:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00 \]

\[ S0:8e_fd_d1_19_97_f0_b5_f1_39_76_dd_d2_ad_97_f6_26 \]

\[ S1:4e_38_40_08_81_c1_dc_aa_87_68_df_de_63_49_f5_9b \]

\[ S2:4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27 \]

\[ t = -11: \]

\[ H:0d_b0_80_ac_d2_cf_06_7a_18_31_9b_19_d1_02_a4_54 \]

\[ L:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00 \]

\[ N:6e_d3_e3_cf_b1_ac_b5_19_7b_52_f4_7a_b2_61_c7_37 \]

\[ S0:83_4d_51_b5_45_3f_63_86_21_47_66_cb_7c_95_52_72 \]

\[ S1:58_5d_4d_cd_ab_63_90_2b_13_04_b2_0a_1e_57_0d_a3 \]

\[ S2:a5_5e_44_33_c1_79_bc_69_7a_9a_3c_c3_99_16_e1 \]

\[ t = -10: \]

\[ H:e1_11_36_4c_09_12_10_05_c3_e2_f8_bc_c7_20_48_2b \]

\[ L:0d_b0_80_ac_d2_cf_06_7a_18_31_9b_19_d1_02_a4_54 \]

\[ N:32_7e_41_99_81_02_e3_2e_bb_1a_ec_e5_b0_57_1a_a8 \]

\[ S0:0c_8f_84_36_75_81_c6_97_99_f7_46_0d_09_44_6e_68 \]

\[ S1:40_61_90_b2_b6_c7_97_ee_ea_0d_b6_01_32_88_8a_54 \]

\[ S2:95_4b_08_2b_f8_59_4d_4e_1b_02_34_93_93_97_70_09_3d \]

\[ t = -9: \]

\[ H:09_a2_44_32_48_0c_f1_21_98_91_3a_73_e1_6e_5e_3a \]

\[ L:e1_11_36_4c_81_12_10_05_c3_e2_f8_bc_c7_20_48_2a \]

\[ N:59_59_4c_c6_07_74_7f_63_a6_73_40_38_12_21_44_88 \]

\[ S0:37_53_81_9d_bc_8f_44_98_5a_7c_94_6b_58_55_99_7c \]

\[ S1:c7_c9_29_15_e5_a9_97_a2_4d_b6_5e_01_32_88_8a_54 \]

\[ S2:f9_c5_ef_5e_31_10_75_55_7b_85_01_0a_b4_43_45_08 \]

\[ t = 8: \]

\[ H:01_10_f1_09_4e_4d_c3_c3_cc_47_60_83_73_69_2b_5b \]

\[ L:09_a2_44_32_48_0c_f1_21_98_91_3a_73_e1_6e_5e_3a \]

\[ N:ba_0a_ca_1e_bf_f1_35_16_fd_2f_32_a5_8b_6e_00_be \]

\[ S0:61_1a_3c_32_56_b6_fd_49_38_30_48_b4_60_39_1d_16_1f \]

\[ S1:11_f6_a5_f5_b8_49_68_7b_ec_06_31_e7_bc_8d_03_99 \]

\[ S2:35_2b_5c_2d_c6_8e_8d_1f_4b_4a_5a_90_ec_cb_70_8d \]
A test vector for LOL-DOUBLE is presented as follows.
F. Author et al. 45

\begin{verbatim}
#include

__m128i LOLMINI_C = _mm_setr_epi16(0x35c9 , 0x952b , 0xd4b1 , 0x4ab5 , 0xa291 , 0x7eed , 0xa31b , 0x7ca1) ;

...
\end{verbatim}

\begin{verbatim}
\begin{verbatim}

L:76_35_00_cf_92_8f_93_c9_ff_63_2a_4a_3a_6b_54_19_6b_c6_33_d0_38_7e_29_ea_a7_71_43_b6_75_60_73
00:8d_fc_fb_3c_2a_06_00_03_3d_92_b4_bd_d8_88_96_58
11:c0_02_c8_db_97_07_2b_1b_c0_02_f2_5a_9a_c8_08
00:67_21_75_01_27_77_7a_04_1b_28_8c_5c_6c_b0_5c
01:3c_ae_0c_f3_ac_96_9e_63_0f_6f_6c_63_14_63_c3
02:ae_4e_7a_7b_16_ec_c2_6b_6c_cf_01_a2_c0_c1_6e
03:39_ab_12_9c_5b_ff_a4_6a_0a_8a_cc_b2_f8_00_03

H:31_89_b3_de_0e_08_a9_93_17_88_09_30_08_c8_18_08_dd_84_63_23_17_55_a1_21_66_36_60_17_00
L:32_4a_49_7f_7c_1a_42_bd_1f_19_ab_b2_6c_f7_49_92_da_77_74_f7_b8_47_9f_a6_53_0a_7b_4d_b0_69_9a
00:79_ba_98_08_65_61_69_6a_7a_74_6e_5e_9f_22_bf
01:00_e0_a1_ab_5b_02_42_ac_27_85_ff_8b_fb_3a_93_2d
02:7b_b8_64_4a_5f_ad_13_9f_de_b2_44_66_6b_63_04
03:5a_27_45_6d_69_68_dd_1f_3f_fb_19_6d_00_04_8e
02:ss_2a_32_99_02_01_a2_61_1a_4a_84_10_c5_48_bf_2e
03:ss_a6_64_1a_00_68_db_3b_9b_60_0d_7d_db_69_a5_c8_8a

\begin{verbatim}
Output Keystream Z's for t=0...15:
S3:0f_23_f4_71_30_47_94_01_3c_c9_80_35_f7_9f_64_86
S2:6d_bc_7a_bd_ec_50_11_ab_92_2f_f2_7c_17_c1_38_bb
S1:0d_ee_a9_70_e4_c8_41_99_ea_1f_e1_6b_4c_e9_e0_59
S0:15_7e_dd_da_63_37_b6_6b_87_7c_6d_42_b0_8c_0d
N1:57_af_98_50_bb_c2_10_b4_fa_d6_0e_b1_17_e0_64_3f
N0:58_d9_93_d7_0f_a8_82_79_f2_c0_0d_87_bb_0b_00_e2

End of Initialization (t=0):
S0:30_35_d2_40_83_48_38_3c_aa_9c_35_da_23_b1_cb_76
N1:89_35_4a_ab_05_64_c6_c6_6a_6e_d0_22_15_4a
N0:30_20_58_43_38_33_ca_9c_35_da_23_b1_cb_76
H:32_2d_8d_96_a7_8d_63_c7_16_38_f4_03_01_fb_5e_dc_13_e9_09_83_6c_1b_8c_52_1a_7d_ab_38_93_05_82
00:ff_7e_03_3d_10_3b_5c_se_f0_dd_0a_ee_47_0e_65_24
01:89_35_4a_ab_05_64_c6_c6_6a_6e_d0_22_15_4a
02:30_20_58_43_38_33_ca_9c_35_da_23_b1_cb_76
14_10_0b_71_9c_3c_9d_0e_12_0f_7c_9f_0f_f6_1e_6f
02:c0_52_72_14_48_b5_4a_8e_62_0d_7b_0b_08_5d_04
03:58_2c_21_2b_3d_6b_65_6e_f3_fb_97_9e_fd_17_74

\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
Output Keystream Z's for t=0...15:
50_41_39_fc_fe_3b_74_49_8a_17_de_e6_58_91_4d_33_f2_e1_c5_11_6b_84_8f_96_0e_de_5a_8a_2c_2d
00:48_92_93_d0_01_f3_6a_73_5b_2c_f3_30_f7_13_0b_f4_3e_03_1b_f5_6e_6a_6b_7b_f5_f6_0a_02_f0_03_f1
01:57_af_98_50_bb_c2_10_b4_fa_d6_0e_b1_17_e0_64_3f
02:15_7e_da_da_63_37_b6_6b_87_7c_6d_42_b0_8c_0d
01:0d_ea_a9_70_c8_41_99_ee_1f_e1_6b_4c_e9_e0_59
02:6d_bc_7a_ad_ac_50_11_ab_92_2f_f2_7c_17_c1_38_bb
03:0f_23_f4_71_30_47_94_01_3c_c9_80_35_f7_9f_64_86

End of Initialization (t=0):
50_41_39_fc_fe_3b_74_49_8a_17_de_e6_58_91_4d_33_f2_e1_c5_11_6b_84_8f_96_0e_de_5a_8a_2c_2d

\end{verbatim}
\end{verbatim}
\end{verbatim}

E Reference implementations

Similar to [EJMY19], we present the reference implementation of LOL-MINI and LOL-DOUBLE with AVX2 as Listing 1. The LOL-DOUBLE implementation with AVX512 is given as Listing 2.

```
#include "imintrin.h"

const __m128i LOLMINI_C = _mm_setr_epi16(0x35c9, 0x952b, 0xd4b1, 0x4ab5, 0xa291, 0x7eed, 0xa31b, 0x7ca1);
```
const __m128i LOLMINI_SIGMA = _mm_setr_epi8(2, 3, 4, 5, 14, 15, 8, 9, 12, 13, 6, 7, 0, 1, 10, 11);

const __m128i ZERO128 = _mm_setzero_si128();

const __m256i LOLDouble_C = _mm256_setr_epi16(0x35c9, 0x952b, 0xd4b1, 0x4ab5, 0xa291, 0x7eed, 0xa31b, 0xc553, 0x7dc5, 0xd83, 0xbb2eb, 0x5d2f, 0x9fb7, 0x44e1, 0xf069);

const __m256i LOLDouble_Shuffle0 = _mm256_setr_epi8(6, 7, 0xff, 0xff, 10, 11, 2, 3, 0xff, 0xff, 0xff, 0xff, 14, 15, 8, 9, 2, 3, 0xff, 0xff, 0, 1, 0xff, 0xff, 12, 13, 14, 15, 0xff, 0xff, 0, 1, 0xff, 0xff, 0xff, 0xff, 12, 13, 0xff, 0xff, 0xff, 0xff, 0xff, 0, 1, 0xff, 0xff, 8, 9, 0xff, 0xff, 0xff, 0xff, 10, 11, 4, 5, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff);

#define shuffle256 (x) _mm256_xor_si256(_mm256_shuffle_epi8((x) , LOLDouble_Shuffle0) , _mm256_permute4x64_epi64(_mm256_shuffle_epi8((x) , LOLDouble_Shuffle1) , 0x4e))

const __m256i ZERO256 = _mm256_setzero_si256();

//LOL-MINI

struct Lolmini {
    __m128i hi, lo, nfsr, state[3];

    inline __m128i keystream(void) {
        //Keystream Bits
        __m128i intermediate = _mm_aesenc_si128(state[2], ZERO128);
        __m128i z = _mm_xor_si128(intermediate, nfsr);

        //N Update
        nfsr = _mm_aesenc_si128(nfsr, lo);

        //LFSR Update
        __m128i mulx = _mm_xor_si128(_mm_shuffle_epi8(hi, 1), _mm_and_si128(LOLMINI_C, _mm_srai_epi16(hi, 15)));
        __m128i oddOld = hi;
        hi = _mm_xor_si128(hi, mulx, _mm_shuffle_epi8(lo, LOLMINI_SIGMA));
        lo = oddOld;

        //FSM Update
        state[2] = _mm_aesenc_si128(state[1], state[2]);
        state[1] = _mm_aesenc_si128(state[0], state[1]);
        state[0] = _mm_xor_si128(state[0], state[1], state[2]);

        return z;
    }

    inline void keyiv_setup(const unsigned char* key, const unsigned char* iv) {
        __m128i mainKey[2];
        state[0] = _mm_loadu_si128((__m128i*)iv);
        state[1] = mainKey[0] = _mm_loadu_si128((__m128i*)(key + 16));
        state[2] = mainKey[1] = _mm_loadu_si128((__m128i*)key);
        nfsr = hi = lo = ZERO128;

        for (int i = 0; i < 12; ++i) {
            __m128i output = keystream();
            nfsr = _mm_xor_si128(nfsr, output);
            hi = _mm_xor_si128(hi, output);
        }
        state[0] = _mm_xor_si128(state[0], mainKey[0]);
        hi = _mm_xor_si128(hi, mainKey[1]);
    }
};
// LOL-DOUBLE

struct Loldouble
{
    __m256i hi, lo;
    __m128i state[4];
    __m128i nfsr[2];

    inline __m256i keystream(void)
    {
        // Keystream bits
        __m128i tmp1 = _mm_aesenc_si128(state[1], ZERO128);
        __m128i tmp3 = _mm_aesenc_si128(state[3], ZERO128);
        __m128i z0 = _mm_xor_si128(tmp1, nfsr[0]);
        __m128i z1 = _mm_xor_si128(tmp3, nfsr[1]);

        // N update
        nfsr[0] = _mm_aesenc_si128(nfsr[0], _mm256_castsi256_si128(lo));
        nfsr[1] = _mm_aesenc_si128(nfsr[1], _mm256_extracti128_si256(hi, 1));

        // LFSR Update
        __m256i mulx = _mm256_xor_si256(_mm256_slli_epi16(hi, 1),
                                      _mm256_and_si256(LOLDOUBLE_C, _mm256_srai_epi16(hi, 15)));
        __m256i oddOld = hi;
        hi = _mm256_xor_si256(shuffle256(lo), mulx);
        lo = oddOld;

        // FSM Update
        state[1] = _mm_aesenc_si128(state[1], state[1]);
        state[0] = _mm_xor_si128(_mm_xor_si128(tmp1, state[0]),
                                  _mm256_castsi256_si128(hi));
        state[3] = _mm_aesenc_si128(state[3], state[3]);
        state[2] = _mm_xor_si128(_mm_xor_si128(tmp3, state[2]),
                                  _mm256_extracti128_si256(hi, 1));
        return _mm256_set_m128i(z0, z1);
    }

    inline void keyiv_setup(const unsigned char* key, const unsigned char* iv)
    {
        __m256i mainKey = _mm256_loadu_si256((__m256i*)key);
        hi = lo = ZERO256;
        nfsr[0] = nfsr[1] = ZERO128;
        state[2] = _mm256_castsi256_si128(mainKey);
        state[3] = _mm256_extracti128_si256(mainKey, 1);
        state[0] = _mm_load_si128((__m128i*)iv);
        state[1] = _mm_load_si128((__m128i*)(iv + 16));
        for (int i = 0; i < 12; ++i)
        {
            __m256i output = keystream();
            hi = _mm256_xor_si256(hi, output);
            nfsr[0] = _mm_xor_si128(nfsr[0], _mm256_castsi256_si128(output));
            nfsr[1] = _mm_xor_si128(nfsr[1], _mm256_extracti128_si256(output, 1));
            hi = _mm256_xor_si256(hi, mainKey);
        }
    }
};

Listing 1: Implement LOL-MINI and LOL-DOUBLE with AVX2

#include <xmmintrin.h>
#include <immintrin.h>
#include <smmintrin.h>
#include <wmmintrin.h>
const __m256i lol_shu16 = _mm256_set_epi8(23, 22, 13, 12, 31, 30, 29, 28, 5, 4, 17, 16, 1, 0, 19, 18, 9, 8, 15, 14, 21, 20, 27, 26, 3, 2, 11, 10, 25, 24, 7, 6);
const __m256i ZERO256 = _mm256_setzero_si256();

#define LFSR_DOUBLE(H, L)
{ F = _mm256_xor_si256(_mm256_slli_epi16(H, 1), _mm256_and_si256(lol_mul16, _mm256_srai_epi16(H, 15)));
 F = _mm256_xor_si256(_mm256_permutexvar_epi8(lol_shu16, L), F);
 L = H;
 H = F;}

#define AES_ENC_256(C, M) C = _mm256_aesenc_epi128(M, ZERO256);
#define AES_ENC_KEY_256(C, M, R) C = _mm256_aesenc_key128(M, R);

struct LOL_DOUBLE
{
__m256i F, H, L, G, S0, S1, Z, N;
inline __m256i keystream(void)
{
 AES_ENC_256(G, S1);
 Z = _mm256_xor_si256(G, N);
 AES_ENC_KEY_256(N, N, L);
 LFSR_DOUBLE(H, L);
 AES_ENC_KEY_256(S1, S0, S1);
 S0 = _mm256_xor_si256(S0, _mm256_xor_si256(H, _mm256_permute4x64_epi64(G, 0x4e)));
 Z = _mm256_permute4x64_epi64(Z, 0x4e);
 return Z;
}

inline void keyiv_setup(const unsigned char* key, const unsigned char* iv)
{
 S1 = _mm256_loadu_si256((__m256i*)key);
 S0 = _mm256_loadu_si256((__m256i*)iv);
 S1 = _mm256_permute2x128_si256(S0, S1, 0x31);
 S0 = _mm256_permute2x128_si256(S0, S1, 0x20);

//round 1
 AES_ENC_256(G, S1);
 Z = _mm256_permute4x64_epi64(G, 0x4e);
 AES_ENC_KEY_256(N, ZERO256, Z);
 L = ZERO256;
 H = Z;
 AES_ENC_KEY_256(S1, S0, S1);
 S0 = _mm256_xor_si256(S0, Z);

//round 2-12
for(int i = 0; i < 11; i++)
{
 AES_ENC_256(G, S1);
 Z = _mm256_xor_si256(G, N);
 Z = _mm256_permute4x64_epi64(Z, 0x4e);
 AES_ENC_KEY_256(N, N, L);
 N = _mm256_xor_si256(N, Z);
 LFSR_DOUBLE(H, L);
 AES_ENC_KEY_256(S1, S0, S1);
S0 = _mm256_xor_si256(S0, _mm256_xor_si256(H, _mm256_permute4x64_epi64(G, 0x4e)))
H = _mm256_xor_si256(H, Z);

// reload
H = _mm256_xor_si256(H, _mm256_loadu_si256((__m256i *)key));

Listing 2: Implement LOL-DOUBLE with AVX-512