LOL: A Highly Flexible Framework for Designing Stream Ciphers

Dengguo Feng[1], Lin Jiao[1], Yonglin Hao[1], Qunxiong Zheng[2], Wenling Wu[3], Wenfeng Qi[2], Lei Zhang[3], Liting Zhang[4], Siwei Sun[5] and Tian Tian[2]

[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
jiaolin_jl@126.com,haoyonglin@yeah.net

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 manufactures. Many modern CPUs are now equipped with extended accumulator instruction sets (such as the SSE/SSE2/SSE4.2/AVX/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][1].

[1]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 [HII+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

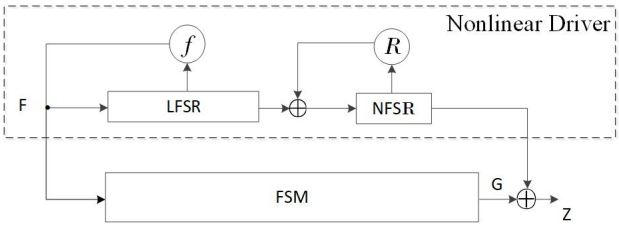

Figure 1: The form of cipher structure

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 $\mathcal{R}$, 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.

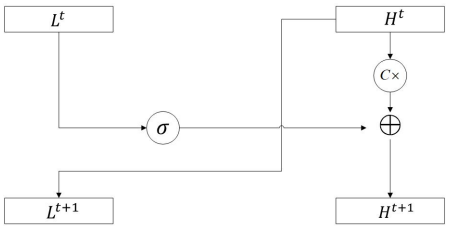

Figure 2: The Feistel-like structure of LFSR updating function

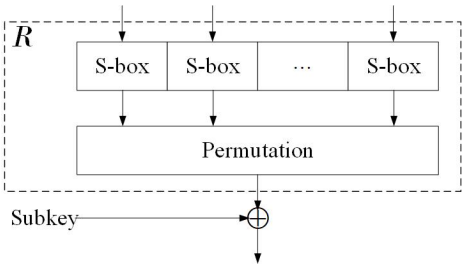

Figure 3: The SP-network of $\mathcal{R}$ function

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 2ℓ ($\ell > 0$) m-bit cells denoted as $a_{2\ell-1}, \ldots, a_0 \in \mathbb{F}_{2^m}$. At step t, the 2ℓ cells are categorized into 2 vectors according to their parities as $L^t = \left(a^t_{2\ell-2}, \cdots, a^t_2, a^t_0\right)$ for even-numbered cells and $H^t = \left(a^t_{2\ell-1}, \cdots, a^t_3, a^t_1\right)$ 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 ℓ irreducible polynomials[2] in $\mathbb{F}_2[x]$ with degree m, $\times$ denotes a cell-to-cell finite field multiplication on $\mathbb{F}_{2^m}$, σ 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:

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

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.

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

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

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

[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:

$$G = \left(\begin{array}{c|c} M_C & I_{\ell m} \\ \hline M_\sigma & \mathbf{0} \end{array}\right).$$

Hereafter, we denote I_i as the i-dimensional unit matrix. M_σ corresponds to the σ. Since σ is a permutation of ℓ m-bit words, M_σ 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

$$M_C = \begin{pmatrix} M_{\alpha_0} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & M_{\alpha_{\ell-1}} \end{pmatrix}.$$

where α_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 $(\alpha_i^{m-1}, \ldots, \alpha_i, 1)$ enabling us to represent arbitrary m-bit cell $a_{2i+1} = (a_{i,m-1} \| \cdots \| a_{i,1} \| a_{i,0})$ with a $\mathbb{F}_{2^m}$ element of the form

$$a_{2i+1} = (a_{i,m-1}, \cdots, a_{i,1}, a_{i,0})(\alpha_i^{m-1}, \ldots, \alpha_i, 1)^T.$$

The definition of M_{α_i} should make sure

$$\begin{aligned} \alpha_i a_{2i+1} \mod g_i(\alpha_i) &= a_{i,m-1}\alpha_i^m + \cdots + a_{i,1}\alpha_i^2 + a_{i,0}\alpha_i \\ &= a_{m-1,i}(b_{i,m-1}\alpha_i^{m-1} + \cdots + b_{i,1}\alpha_i + b_{i,0}) + a_{i,m-2}\alpha_i^{m-1} + \cdots + a_{i,1}\alpha_i^2 + a_{i,0}\alpha_i \\ &= (a_{m-1,i}b_{i,m-1} + a_{i,m-2})\alpha_i^{m-1} + \cdots + (a_{m-1,i}b_{i,1} + a_{i,0})\alpha_i + a_{m-1,i}b_{i,0} \\ &= (a_{i,m-1}, \cdots, a_{i,1}, a_{i,0})M_{\alpha_i}(\alpha_i^{m-1}, \ldots, \alpha_i, 1)^T, \end{aligned}$$

so M_{α_i} is defined as

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

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[3] and Maple[4]. □

For concrete ℓ, 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, σ) can be found after hundreds of iterations. The source codes are attached and will be open source.

Our LFSR design has the following advantages:

[3] https://libntl.org/
[4] https://maple-17.en.softonic.com/

Algorithm 1: Construct a $2\ell m$-bit LFSR (consists of 2ℓ m-bit cells) with a maximum LFSR period of $2^{2\ell m}-1$

Input: The parameters m and ℓ
Output: (C, σ) guranteeing the maximum period of $2^{2\ell m}-1$

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

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. Moreover, it supports executing $\times$ and σ operations synchronously.

Highly Flexible This series of LFSRs have a enriched selection of (C, σ) 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 $2^{|N|/2} \times (2^{2\ell m} - 1)$, 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 $\mathcal{R}$ 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

$$S_i^{t+1} = \mathcal{R}(S_{i-1}^t) \oplus S_i^t, \tag{2}$$

where $1 \le i \le n-1$. Note that all $\mathcal{R}$ functions are not necessarily the same: different $\mathcal{R}$ 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.

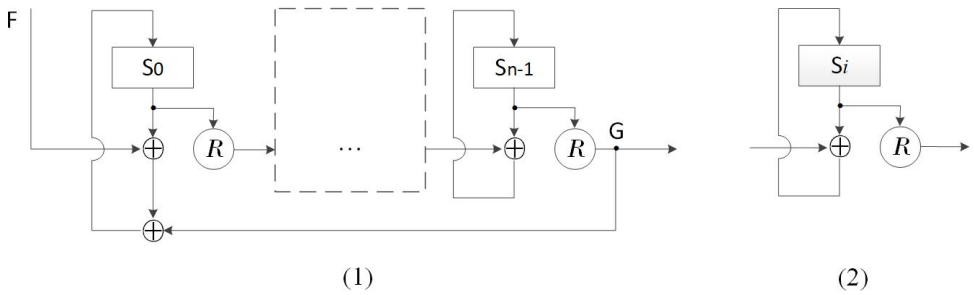

Figure 4: The connector and plugin of the FSM

Specifically, at the tail end of FSM, one $\mathcal{R}$ 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:

$$G^t = \mathcal{R}(S^t_{n-1}),\ S^{t+1}_0 = S^t_0 \oplus F^t \oplus G^t,\ Z^t = G^t \oplus N^t.$$

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 $\mathcal{R}$ 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:

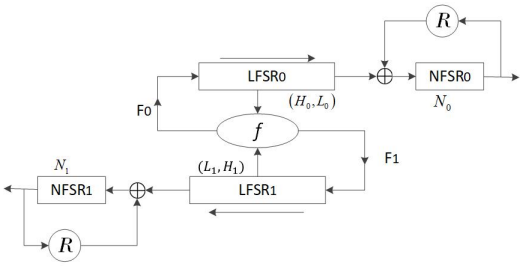

Figure 5: The form of extended nonlinear driver

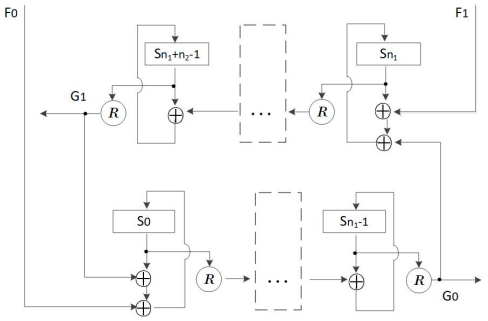

Figure 6: The form of extended FSM

1. To construct an extended nonlinear driver, we recombine the LFSRs from γ, $\gamma \geq 2$ 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 γ 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

$$L = (L_1, L_0),\ H = (H_1, H_0),$$

with a new updating function of

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

All that the designers have to do is selecting proper (C, σ) for the larger (H, L) to keep the maximum period. According to our experiments, such (C, σ) are so abundant that we only need to define a σ for the large C as an expansion of the smaller Cs 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:

$$N^{t+1}_0 = \mathcal{R}(N^t_0) \oplus L^t_0,\ N^{t+1}_1 = \mathcal{R}(N^t_1) \oplus L^t_1.$$

2. To construct an extended FSM with larger throughputs, we can link the connectors from γ single modes end to end in a circle, each feeding into the other. The feedback of the extended LFSR is also divided into γ 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

$$\begin{aligned}
&G_0 = \mathcal{R}(S^t_{n_1-1}),\ G_1 = \mathcal{R}(S^t_{n_1+n_2-1}),\\
&F^t = (F^t_0, F^t_1),\ S^{t+1}_0 = F^t_0 \oplus G^t_1 \oplus S^t_0,\ S^{t+1}_{n_1} = F^t_1 \oplus G^t_0 \oplus S^t_{n_1},\\
&S^{t+1}_i = \mathcal{R}(S^t_{i-1}) \oplus S^t_i,\ i = 1, \ldots, n_1 - 1, n_1 + 1, \ldots, n_1 + n_2 - 1.
\end{aligned}$$

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 γ 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

$$(Z^t_0, Z^t_1) = (G^t_0 \oplus N^t_0, G^t_1 \oplus N^t_1).$$

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. $S = (w_7, \ldots, w_0) = (b_{15}, \ldots, b_0)$.

- $\mathcal{R}$ 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.

$$\mathcal{R}(S) = \mathrm{MC} \circ \mathrm{SR} \circ \mathrm{SB}(S).$$

  The mapping between a 128-bit register state $S = (w_{15}, \ldots, w_0)$ and the 4×4-byte state array of the AES round function is as follows

$$S = \begin{pmatrix} w_0 & w_4 & w_8 & w_{12} \\ w_1 & w_5 & w_9 & w_{13} \\ w_2 & w_6 & w_{10} & w_{14} \\ w_3 & w_7 & w_{11} & w_{15} \end{pmatrix} \tag{3}$$

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 $K = (K_h, K_\ell)$ 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:

$$H = (a_{15}, \ldots, a_3, a_1),\ L = (a_{14}, \ldots, a_2, a_0).$$

Let $\alpha_0, \alpha_1, \ldots, \alpha_7$ be the roots of the irreducible polynomials $g_0(y), g_1(y), \ldots, g_7(y) \in \mathbb{F}_2[y]$ with algebraic degree 16, respectively,

$$\begin{aligned}
g_0(y) =& y^{16} + y^{13} + y^{12} + y^{10} + y^8 + y^7 + y^6 + y^3 + 1,\\
g_1(y) =& y^{16} + y^{15} + y^{12} + y^{10} + y^8 + y^5 + y^3 + y + 1,\\
g_2(y) =& y^{16} + y^{15} + y^{14} + y^{12} + y^{10} + y^7 + y^5 + y^4 + 1,\\
g_3(y) =& y^{16} + y^{14} + y^{11} + y^9 + y^7 + y^5 + y^4 + y^2 + 1,\\
g_4(y) =& y^{16} + y^{15} + y^{13} + y^9 + y^7 + y^4 + 1,\\
g_5(y) =& y^{16} + y^{14} + y^{13} + y^{12} + y^{11} + y^{10} + y^9 + y^7 + y^6 + y^5 + y^3 + y^2 + 1,\\
g_6(y) =& y^{16} + y^{15} + y^{13} + y^9 + y^8 + y^4 + y^3 + y + 1,\\
g_7(y) =& y^{16} + y^{14} + y^{13} + y^{12} + y^{11} + y^{10} + y^7 + y^5 + 1.
\end{aligned} \tag{4}$$

Let $C = (\alpha_7, \ldots, \alpha_1, \alpha_0)$ and define the operation $C\times : \mathbb{F}_{2^{16}}^8 \to \mathbb{F}_{2^{16}}^8$ as

$$(x_7, \ldots, x_1, x_0) \xrightarrow{C\times} (\alpha_7 \cdot x_7, \ldots, \alpha_1 \cdot x_1, \alpha_0 \cdot x_0), \tag{5}$$

where $\alpha_i \cdot x_i$ denotes the multiplication over the finite field $\mathbb{F}_{2^{16}} = \mathbb{F}_2[y]/g_i(y)$ defined by the polynomial $g_i(y)$ in Eq. (4) $(0 \le i \le 7)$. Let the permutation $\sigma : \mathbb{F}_{2^{16}}^8 \to \mathbb{F}_{2^{16}}^8$ be

$$(x_7, \ldots, x_1, x_0) \xrightarrow{\sigma} (x_5, x_0, x_3, x_6, x_4, x_7, x_2, x_1)$$

Then, we can define the LFSR feedback function $f : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \to \mathbb{F}_2^8$ as

$$F = f(X, Y) = (C \times X) \oplus \sigma(Y).$$

With the H, L at step t denoted as $H^t = (a_{15}^t, \ldots, a_3^t, a_1^t)$ and $L^t = (a_{14}^t, \ldots, a_2^t, a_0^t)$, 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

$$F^t = f(H^t, L^t),\ H^{t+1} = F^t,\ L^{t+1} = H^t, \tag{6}$$

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^ts for $t \ge 0$. The initialization phase steps are with $Round_{load} \le t < 0$ where

$$Round_{load} = -12.$$

We demonstrate the two phases as follows:

Keystream generation phase For $t \ge 0$,

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: $S_0^{t+1} = F^t \oplus G^t \oplus S_0^t,\ S_1^{t+1} = \mathcal{R}(S_0^t) \oplus S_1^t,\ S_2^{t+1} = \mathcal{R}(S_1^t) \oplus S_2^t$.

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: $G^t = \mathcal{R}(S_2^t),\ F^t = f(H^t, L^t)$.
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:

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

4. Update the FSM:

$$S_1^{t+1} = \mathcal{R}(S_0^t) \oplus S_1^t,\ S_2^{t+1} = \mathcal{R}(S_1^t) \oplus S_2^t,$$
$$S_0^{t+1} = \begin{cases} F^t \oplus G^t \oplus S_0^t,\ Round_{load} \leq t < -1 \\ F^t \oplus G^t \oplus S_0^t \oplus K_\ell,\ t = -1 \end{cases}$$

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.

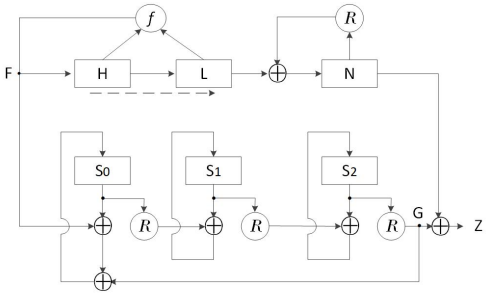

Figure 7: The schematic of LOL-MINI cipher in keystream generation phase

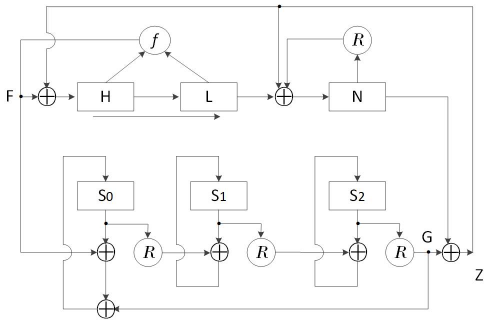

Figure 8: The schematic of LOL-MINI cipher in initialization phase

3.3 Initialization

At the beginning of the initialization $t = Round_{load}$, K and IV are loaded into the FSM as

$$S_0^t = IV, S_1^t = K_h, S_2^t = K_\ell,$$

and the LFSR and NFSR states are set to zero:

$$H^t = L^t = N^t = 0.$$

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 $K = (K_h, K_\ell)$ and $IV = (IV_h, IV_\ell)$, 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) or (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:

$$\begin{aligned} H = &\quad (H_1, H_0) = ((a_{31}, \ldots, a_{17}), (a_{15}, \ldots, a_3, a_1)), \\ L = &\quad (L_1, L_0) = ((a_{30}, \ldots, a_{16}), (a_{14}, \ldots, a_2, a_0)). \end{aligned} \tag{7}$$

Let $\alpha_0, \alpha_1, \ldots, \alpha_{15}$ 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).

$$\begin{aligned} g_8(y) =& y^{16} + y^{15} + y^{14} + y^{10} + y^8 + y^6 + y^4 + y + 1 \\ g_9(y) =& y^{16} + y^{14} + y^{13} + y^{12} + y^{11} + y^{10} + y^8 + y^7 + y^6 + y^2 + 1 \\ g_{10}(y) =& y^{16} + y^{11} + y^{10} + y^8 + y^7 + y + 1 \\ g_{11}(y) =& y^{16} + y^{15} + y^{13} + y^{12} + y^9 + y^7 + y^6 + y^5 + y^3 + y + 1 \\ g_{12}(y) =& y^{16} + y^{15} + y^{14} + y^{12} + y^{10} + y^8 + y^5 + y^3 + y^2 + y + 1 \\ g_{13}(y) =& y^{16} + y^{15} + y^{12} + y^{11} + y^{10} + y^9 + y^8 + y^7 + y^5 + y^4 + y^2 + y + 1 \\ g_{14}(y) =& y^{16} + y^{14} + y^{10} + y^7 + y^6 + y^5 + 1 \\ g_{15}(y) =& y^{16} + y^{15} + y^{14} + y^{13} + y^{12} + y^6 + y^5 + y^3 + 1 \end{aligned} \tag{8}$$

Let $C = (\alpha_{15}, \ldots, \alpha_1, \alpha_0)$ and define the operation $C\times : \mathbb{F}_{2^{16}}^{16} \to \mathbb{F}_{2^{16}}^{16}$ as

$$(x_{15}, \ldots, x_1, x_0) \xrightarrow{C\times} (\alpha_{15} \cdot x_{15}, \ldots, \alpha_1 \cdot x_1, \alpha_0 \cdot x_0), \tag{9}$$

where $\alpha_i \cdot x_i$ denotes the multiplication over the finite field $\mathbb{F}_{2^{16}} = \mathbb{F}_2[y]/g_i(y)$ defined by the polynomial $g_i(y)$ $(0 \le i \le 15)$. Let $C_1 = (\alpha_{15}, \ldots, \alpha_9, \alpha_8), C_0 = (\alpha_7, \ldots, \alpha_1, \alpha_0)$. Obviously, Eq. (9) is a concatenation of $C_1\times$ and $C_0\times$ mappings on $\mathbb{F}_{2^{16}}^8 \to \mathbb{F}_{2^{16}}^8$, 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 $(x_7, \ldots, x_0)$:

$$\text{Eq. (9)} \Leftrightarrow \begin{cases} (x_{15}, \ldots, x_8) \xrightarrow{C_1\times} (\alpha_{15} \cdot x_{15}, \ldots, \alpha_8 \cdot x_8), \\ (x_7, \ldots, x_0) \xrightarrow{C_0\times} (\alpha_7 \cdot x_7, \ldots, \alpha_0 \cdot x_0). \end{cases}$$

It is worth noticing that, according to the definition of $\alpha_0, \ldots, \alpha_7$, 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

reused in the hardware implementations of LOL-MINI and LOL-DOUBLE stream ciphers in some highly compatible application scenarios.

Let the permutation $\sigma : \mathbb{F}_{2^{16}}^{16} \to \mathbb{F}_{2^{16}}^{16}$ be

$$(x_{15}, \ldots, x_0) \xrightarrow{\sigma} (x_{11}, x_6, x_{15}, x_{14}, x_2, x_8, x_0, x_9, x_4, x_7, x_{10}, x_{13}, x_1, x_5, x_{12}, x_3).$$

By defining the feedback function $f : \mathbb{F}_2^{16} \times \mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ as

$$F = (F_1, F_0) = f(X, Y) = (C \times X) \oplus \sigma(Y),$$

and denoting H, L at step t as

$$H^t = (H_1^t, H_0^t) = (a_{31}^t, \ldots, a_3^t, a_1^t),\ L^t = (L_1^t, L_0^t) = (a_{30}^t, \ldots, a_2^t, a_0^t).$$

we can represent the LFSR updating function with the Feistel-like structure in Fig. 2 as follows:

$$(F_1^t, F_0^t) = f(H^t, L^t) = f((H_1^t, H_0^t), (L_1^t, L_0^t)),$$
$$(L_1^{t+1}, L_0^{t+1}) = (H_1^t, H_0^t),\ (H_1^{t+1}, H_0^{t+1}) = (F_1^t, F_0^t)$$

where F_1^t and F_0^t are divided into the feedbacks of each LFSR at step t for the input of each FSM, respectively. As is proved in Section 5.1, the LFSR of LOL-DOUBLE cipher is with a period of $2^{512} - 1$.

For the NFSR part of LOL-DOUBLE cipher, there are two 128-bit NFSRs of N_0 and N_1. Let N_0 and N_1 at step t be N_0^t and N_1^t, 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 $\mathcal{R}$ functions as follows

$$N_0^{t+1} = \mathcal{R}(N_0^t) \oplus L_0^t,\ N_1^{t+1} = \mathcal{R}(N_1^t) \oplus L_1^t.$$

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^ts for $t \geq 0$. The initialization phase steps are 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:

$$G_0^t = \mathcal{R}(S_1^t),\ G_1^t = \mathcal{R}(S_3^t),\ (F_1^t, F_0^t) = f(H^t, L^t) = f((H_1^t, H_0^t), (L_1^t, L_0^t)).$$

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:

$$N_0^{t+1} = \mathcal{R}(N_0^t) \oplus L_0^t,\ N_1^{t+1} = \mathcal{R}(N_1^t) \oplus L_1^t;$$
$$H_0^{t+1} = F_0^t,\ H_1^{t+1} = F_1^t;\ L_0^{t+1} = H_0^t,\ L_1^{t+1} = H_1^t.$$

4. Update the FSM:

$$S_0^{t+1} = F_0^t \oplus G_1^t \oplus S_0^t,\ S_1^{t+1} = \mathcal{R}(S_0^t) \oplus S_1^t,$$
$$S_2^{t+1} = F_1^t \oplus G_0^t \oplus S_2^t,\ S_3^{t+1} = \mathcal{R}(S_2^t) \oplus S_3^t.$$

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:

$$G_0^t = \mathcal{R}(S_1^t),\ G_1^t = \mathcal{R}(S_3^t),\ (F_1^t, F_0^t) = f(H^t, L^t) = f((H_1^t, H_0^t), (L_1^t, L_0^t)).$$

2. Compute the 256-bit keystream block $Z^t = (Z_0^t, Z_1^t)$ as

$$Z_0^t = G_0^t \oplus N_0^t,\ Z_1^t = G_1^t \oplus N_1^t.$$

   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:

$$N_0^{t+1} = \mathcal{R}(N_0^t) \oplus L_0^t \oplus Z_1^t,\ N_1^{t+1} = \mathcal{R}(N_1^t) \oplus L_1^t \oplus Z_0^t,$$
$$L_0^{t+1} = H_0^t,\ L_1^{t+1} = H_1^t,$$
$$H_0^{t+1} = \begin{cases} F_0^t \oplus Z_1^t, & Round_{load} \leq t < -1 \\ F_0^t \oplus Z_1^t \oplus K_\ell, & t = -1 \end{cases}, H_1^{t+1} = \begin{cases} F_1^t \oplus Z_0^t, & Round_{load} \leq t < -1 \\ F_1^t \oplus Z_0^t \oplus K_h, & t = -1 \end{cases}$$

4. Update the FSM:

$$S_0^{t+1} = F_0^t \oplus G_1^t \oplus S_0^t,\ S_1^{t+1} = \mathcal{R}(S_0^t) \oplus S_1^t,$$
$$S_2^{t+1} = F_1^t \oplus G_0^t \oplus S_2^t,\ S_3^{t+1} = \mathcal{R}(S_2^t) \oplus S_3^t,$$

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.

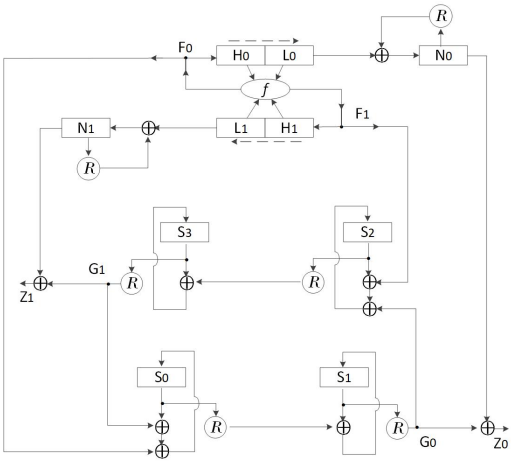

Figure 9: The schematic of LOL-DOUBLE cipher in keystream generation phase

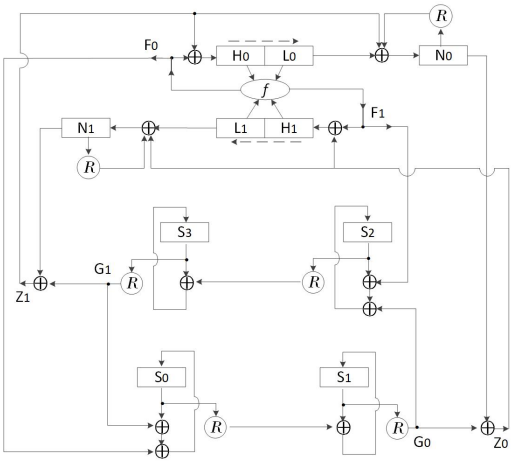

Figure 10: The schematic of LOL-DOUBLE cipher in initialization phase

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

$$S_3^t = K_h, S_2^t = K_\ell,\ S_1^t = IV_h, S_0^t = IV_\ell,$$

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 $Round_{load}$ is also set to -12. It also adopts the FP(1)-mode [HK18] by XORing the master key $K = (K_h, K_\ell)$ 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 256×256, defined as

$$G = \left(\begin{array}{c|c} M_C & I_{128} \\ \hline M_\sigma & \mathbf{0} \end{array}\right).$$

Here M_σ is the shift matrix corresponding to σ permutation and M_c is the diagonal block matrix for the multiplications over the finite field $\mathbb{F}_{2^{16}} = \mathbb{F}_2[y]/g_i(y)$ defined by the polynomial $g_i(y)$ in Eq. (4) $(0 \le i \le 7)$,

$$M_\sigma = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & I_{16} & 0 & 0 \\ 0 & 0 & 0 & I_{16} & 0 & 0 & 0 & 0 \\ I_{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{16} & 0 & 0 & 0 \\ 0 & 0 & I_{16} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I_{16} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_{16} \\ 0 & I_{16} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, M_C = \begin{pmatrix} M_{\alpha_0} & & 0 \\ & \ddots & \\ 0 & & M_{\alpha_7} \end{pmatrix},$$

$$M_{\alpha_i} = \begin{pmatrix} b_{i,15} & b_{i,14} & b_{i,13} & b_{i,12} & b_{i,11} & b_{i,10} & b_{i,9} & b_{i,8} & b_{i,7} & b_{i,6} & b_{i,5} & b_{i,4} & b_{i,3} & b_{i,2} & b_{i,1} & b_{i,0} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The characteristic polynomial of G is found using NTL as

$$\begin{aligned}\eta_{256}(x) = {} & x^{256} + x^{252} + x^{251} + x^{250} + x^{246} + x^{245} + x^{242} + x^{241} + x^{238} + x^{234} + x^{229} + x^{227} + x^{225} + x^{222} + \\ & x^{220} + x^{219} + x^{217} + x^{215} + x^{214} + x^{213} + x^{212} + x^{211} + x^{210} + x^{208} + x^{207} + x^{204} + x^{203} + x^{198} + \\ & x^{196} + x^{194} + x^{191} + x^{186} + x^{185} + x^{184} + x^{183} + x^{182} + x^{179} + x^{178} + x^{177} + x^{175} + x^{170} + x^{168} + \\ & x^{167} + x^{166} + x^{165} + x^{161} + x^{160} + x^{157} + x^{154} + x^{153} + x^{152} + x^{151} + x^{150} + x^{148} + x^{147} + x^{146} + \\ & x^{145} + x^{144} + x^{141} + x^{140} + x^{139} + x^{138} + x^{134} + x^{133} + x^{131} + x^{130} + x^{126} + x^{123} + x^{122} + x^{121} + \\ & x^{119} + x^{118} + x^{116} + x^{115} + x^{113} + x^{110} + x^{109} + x^{108} + x^{107} + x^{104} + x^{103} + x^{102} + x^{101} + x^{100} + \\ & x^{98} + x^{97} + x^{94} + x^{93} + x^{92} + x^{89} + x^{88} + x^{86} + x^{84} + x^{83} + x^{82} + x^{81} + x^{80} + x^{79} + x^{78} + x^{77} + \\ & x^{76} + x^{74} + x^{72} + x^{71} + x^{70} + x^{69} + x^{67} + x^{66} + x^{65} + x^{63} + x^{62} + x^{61} + x^{60} + x^{56} + x^{53} + x^{52} + \\ & x^{51} + x^{50} + x^{48} + x^{47} + x^{44} + x^{42} + x^{39} + x^{38} + x^{37} + x^{34} + x^{32} + x^{31} + x^{30} + x^{29} + x^{28} + x^{26} + \\ & x^{25} + x^{24} + x^{21} + x^{20} + x^{9} + x^{8} + 1.\end{aligned}$$

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 512×512 and rewritten with I_{256}, M_σ and M_C accordingly. Its characteristic polynomial is derived and proved to be primitive in the same manner.

$$\begin{aligned}\eta_{512} = {} & x^{512} + x^{511} + x^{510} + x^{509} + x^{505} + x^{504} + x^{501} + x^{500} + x^{499} + x^{498} + x^{497} + x^{495} + x^{494} + 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^{449} + x^{448} + x^{442} + x^{441} + 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^{388} + x^{386} + x^{381} + \\ & x^{379} + x^{378} + x^{377} + x^{374} + x^{372} + x^{369} + x^{368} + x^{367} + x^{360} + 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^{312} + x^{309} + x^{307} + x^{304} + x^{302} + x^{300} + 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^{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^{173} + x^{172} + x^{171} + x^{167} + x^{166} + x^{162} + x^{160} + x^{159} + x^{157} + \\ & x^{155} + x^{151} + x^{150} + x^{149} + x^{147} + x^{146} + x^{145} + x^{144} + x^{143} + 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^{90} + 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^{60} + 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.\end{aligned}$$

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

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[5]) affected by any bit of the input state after 1 to 5 rounds (1 to 6 rounds), where $768 = 128 \times 6$ ($1280 = 128 \times 10$) indicates that the full diffusion is reached.

Table 1: Diffusion table for LOL-MINI

rounds	H	L	N	S_0	S_1	S_2
1	3	3	33	33	33	97
2	39	69	162	161	417	353
3	202	329	352	544	672	592
4	547	610	**768**	**768**	**768**	**768**
5	**768**	**768**	**768**	**768**	**768**	**768**

Table 2: Diffusion table for LOL-DOUBLE

rounds	H_0	L_0	H_1	L_1	N_0	N_1	S_0	S_1	S_2	S_3
1	3	3	3	3	34	34	33	97	33	97
2	39	70	39	70	195	195	417	417	417	417
3	487	489	486	488	514	514	929	769	929	769
4	969	935	969	935	1088	1088	1184	1088	1184	1088
5	1216	1216	1216	1216	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**
6	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**	**1280**

5.3 Differential attack on intialization

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

[5]For the sake of brevity, the corresponding cryptanalysis results of LOL-DOUBLE are shown in parentheses throughout this section

$z = x \oplus y$. The feasible truncated difference $(\Delta^T x, \Delta^T y, \Delta^T z)$ should lie in the following set $\mathcal{S}$:

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

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

Algorithm 2: Truncated differential model of XOR operations (`xor2Trunc`)

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 $\mathcal{M}.\texttt{var} \leftarrow \Delta^T x, \Delta^T y$ as binaries;
2 Define vector $\vec{x} = (\Delta^T x, \Delta^T y, \Delta^T z)$;
3 Update the model $\mathcal{M}.\texttt{con} \leftarrow A\vec{x} \geq \vec{0}$ where

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

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

can be represented as 16 binary variables as follows

$$\Delta^T X = (\Delta^T x_{15}, \ldots, \Delta^T x_0).$$

Let $Y = \mathcal{R}(X)$. With the knowledge of input truncated difference $\Delta^T x$, the output truncated difference $\Delta^T Y$ can be deduced with MILP modeling technique by calling $(\mathcal{M}, \Delta^T Y) \leftarrow \texttt{aesTruncR}(\mathcal{M}, \Delta^T X)$ as defined in Algorithm 4. Note that the AES SubByte operation does not change the truncated difference so there is $\Delta^T \texttt{SB}(X) = \Delta^T X$.

Algorithm 3: Truncated differential model of AES MixColumn operation (`mcTruncModel`)

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 $\mathcal{M}.\texttt{var} \leftarrow \Delta^T y_0, \ldots, \Delta^T y_3$ as binaries;
2 Declare 1 variable $\mathcal{M}.\texttt{var} \leftarrow \tau$ as binary;
3 $\mathcal{M}.\texttt{con} \leftarrow \sum_{i=0}^{3} (\Delta^T x_i + \Delta^T y_i) \leq 8\tau$;
4 $\mathcal{M}.\texttt{con} \leftarrow \sum_{i=0}^{3} (\Delta^T x_i + \Delta^T y_i) \geq 5\tau$;
5 Return $(\mathcal{M}, \Delta^T y_0, \ldots, \Delta^T y_3)$;

Algorithm 4: Truncated differential model of AES round function $\mathcal{R}$ (`aesTruncModel`)

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:
$\sigma : (0, \ldots 15) \to (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11)$;
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 $\quad (\mathcal{M}, \Delta^T y_{4j+3}, \ldots, \Delta^T y_{4j}) \leftarrow \texttt{mcTruncModel}(\mathcal{M}, \Delta^T w_{4j+3}, \ldots, \Delta^T w_{4j})$;
5 Define vector $\Delta^T Y = (\Delta^T y_{15}, \ldots, \Delta^T y_0)$;
6 Return $(\mathcal{M}, \Delta^T Y)$;

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

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

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

1 Declare 2 variables $\mathcal{M}.\texttt{var} \leftarrow \Delta^T y_0, \Delta^T y_0$ as binaries;
2 Define vector $\vec{x} = (\Delta^T x_0, \Delta^T x_1, \Delta^T y_0, \Delta^T y_1)$;
3 **if** $i \in \{0, 1, 6, 8, 9, 10, 12, 13\}$ **then**
4 Add constraint $\mathcal{M}.\texttt{con} \leftarrow A_1 \vec{x} \geq \vec{0}$ where

$$A_1 = \begin{pmatrix} 1 & -1 & 0 & 1 \\ 1 & 1 & 0 & -1 \\ -1 & 0 & 1 & 1 \\ 1 & 0 & -1 & 1 \end{pmatrix} \tag{10}$$

5 **else**
6 Add constraint $\mathcal{M}.\texttt{con} \leftarrow A_2 \vec{x} \geq \vec{0}$ where

$$A_2 = \begin{pmatrix} 0 & -1 & 1 & 1 \\ 1 & 1 & -1 & 0 \\ -1 & 0 & 1 & 1 \\ 1 & 1 & 0 & -1 \end{pmatrix} \tag{11}$$

7 Return $(\mathcal{M}, \Delta^T y_0, \Delta^T y_1)$;

represented as

$$\Delta^T x = (\Delta^T x_1, \Delta^T x_0), \Delta y = (\Delta^T y_1, \Delta^T y_0)$$

By traversing all 2^{16} Δx values and computing the corresponding Δy'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 $(\mathcal{M}, \Delta^T y_0, \Delta^T y_1) \leftarrow \texttt{alphaTruncModel}(\mathcal{M}, \Delta^T x_0, \Delta^T x_1, i)$. The matrices A_1 in Eq. (10) and A_2 in Eq. (11) are deduced with the H-representation technique in [SHW+14].

Table 3: The feasible truncated difference $(\Delta^T x_0, \Delta^T x_1, \Delta^T y_0, \Delta^T y_1)$'s corresponding to α_i's $(i = 0, \ldots, 15)$ where $y = \alpha_i x$ and $x, y \in \mathbb{F}_2^{16}$

i	$(\Delta^T x_0, \Delta^T x_1, \Delta^T y_0, \Delta^T y_1)$
0, 1, 6, 8, 9, 10, 12, 13	(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)
2, 3, 4, 5, 7, 11, 14, 15	(0,0,0,0), (1,0,1,0), (0,1,1,0), (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)

Since the LOL round functions are composed of $\alpha_i \cdot$, $\oplus$ and $\mathcal{R}$ operations, we are now able to construct MILP model $\mathcal{M}$ 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

$$\begin{aligned} \triangle S_2^{t+1} = \triangle \mathcal{R}(S_1^t) \oplus \triangle S_2^t &= \triangle \mathcal{R}(S_1^t) \oplus \triangle \mathcal{R}(S_1^{t-1}) \oplus S_2^{t-1} \\ &= MC(SR(\triangle SB(S_1^t) \oplus \triangle SB(S_1^{t-1}))) \oplus S_2^{t-1} \end{aligned}$$

so $\Delta^T S_2^{t+1}$ is not only related $\Delta^T(S_1^t)$ but also the truncated difference $\Delta^T(S_1^t) \oplus \Delta^T(S_1^{t-1})$. The inter-round relationships for LOL-MINI and LOL-DOUBLE are shown in Table 4.

Table 4: The inter-round relationships for LOL-MINI and LOL-DOUBLE

i	Inter-round Relationships
LOL-MINI	$\Delta^T S_2^t \oplus \Delta^T N^{t-1} \rightarrow \Delta^T Z^t$, $\Delta^T S_2^t \oplus \Delta^T S_2^{t-1} \rightarrow \Delta^T S_0^{t+1}$, $\Delta^T S_1^t \oplus \Delta^T S_1^{t-1} \rightarrow \Delta^T S_2^{t+1}$, $\Delta^T S_0^t \oplus \Delta^T S_0^{t-1} \rightarrow \Delta^T S_1^{t+1}$
LOL-DOUBLE	$\Delta^T S_3^t \oplus \Delta^T N_1^{t-1} \rightarrow \Delta^T Z_1^t$, $\Delta^T S_1^t \oplus \Delta^T N_0^{t-1} \rightarrow \Delta^T Z_0^t$, $\Delta^T S_3^t \oplus \Delta^T N_0^t \rightarrow \Delta^T N_0^{t+1}$, $\Delta^T S_1^t \oplus \Delta^T N_1^t \rightarrow \Delta^T N_1^{t+1}$, $\Delta^T S_0^t \oplus \Delta^T S_0^{t-1} \rightarrow \Delta^T S_1^{t+1}$, $\Delta^T S_1^t \oplus \Delta^T S_1^{t-1} \rightarrow \Delta^T S_2^{t+1}$, $\Delta^T S_2^t \oplus \Delta^T S_2^{t-1} \rightarrow \Delta^T S_3^{t+1}$, $\Delta^T S_3^t \oplus \Delta^T S_3^{t-1} \rightarrow \Delta^T S_0^{t+1}$

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^{-6} 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.

r	Minimum number of active S-boxes	
	LOL-MINI	LOL-DOUBLE
2	6	6
3	17	21
4	34	52
5	39	59
6	47	-
7	51	-

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 [LIMS21], 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, $\mathcal{R}(IV)$ may represent $\mathcal{R}(IV \oplus c)$ 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,

$$
\begin{aligned}
Z^{-7} =& \mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)))\oplus IV)\oplus\mathcal{R}(IV)\oplus\mathcal{R}(IV)\oplus\mathcal{R}(IV))\\
&\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)))\\
&\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)))\\
&\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{R}(IV))))\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)))\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)))\\
&\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{R}(IV))))\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)\oplus\mathcal{R}(IV))\oplus\mathcal{R}(\mathcal{R}(IV)))\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV))))\oplus\mathcal{R}(\mathcal{R}(\mathcal{R}(IV)))
\end{aligned}
$$

It can be found that IV needs to go through 6 AES rounds for Z^{-7}. Similarly, for LOL-DOUBLE, IV_h and IV_ℓ 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 ≤ 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. [6] 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 $\mathtt{Cor}$, the data complexity of the correlation attacks is evaluated as $\mathtt{Cor}^{-2}$. 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].

[6] 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

$$u = S_2^t, v = S_1^{t-1}, w = S_0^{t-1}, \theta = N^{t-1}$$
$$L = L^{t-1}, H = H^{t-1}, F = F^{t-1}.$$

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

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

For Eq. (12a), we have

$$(\gamma_2 \to \gamma_1) : \gamma_1 \cdot \mathcal{R}(u \oplus \mathcal{R}(v)) = \gamma_2 \cdot (u \oplus \mathcal{R}(v)),$$
$$(\gamma_3 \to \gamma_2) : \gamma_2 \cdot \mathcal{R}(v) = \gamma_3 \cdot v,$$

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:

$$\begin{aligned}\gamma_1 \cdot Z^{t-1} \oplus \gamma_4 \cdot Z^{t} \oplus \gamma_7 \cdot Z^{t+1} \oplus \gamma_{13} \cdot Z^{t+2} = \\ (\gamma_1 \oplus \gamma_5 \oplus \gamma_9 \oplus \gamma_{16}) \cdot \theta \oplus (\gamma_2 \oplus \gamma_6 \oplus \gamma_{10} \oplus \gamma_{17} \oplus \gamma_{23}) \cdot u \oplus (\gamma_3 \oplus \gamma_{11} \oplus \gamma_{18} \oplus \gamma_{20} \oplus \gamma_{24}) \cdot v \\ \oplus (\gamma_7 \oplus \gamma_{14}) \cdot H \oplus (\gamma_4 \oplus \gamma_8 \oplus \gamma_{15}) \cdot L \oplus (\gamma_{12} \oplus \gamma_{19} \oplus \gamma_{21} \oplus \gamma_{22}) \cdot w \oplus (\gamma_{13} \oplus \gamma_{22}) \cdot F.\end{aligned}$$

To construct the linear characteristic $\gamma_1 \cdot Z^{t-1} \oplus \gamma_4 \cdot Z^t \oplus \gamma_7 \cdot Z^{t+1} \oplus \gamma_{13} \cdot Z^{t+2} = 0$, it is also necessary to add constraints to ensure that the coefficients of nonlinear parts u, v, w, θ 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:

$$Z_0^t = v_0 \oplus \mathcal{R}(w_0 \oplus \mathcal{R}(u_0)), \tag{13a}$$
$$Z_1^t = v_1 \oplus \mathcal{R}(w_1 \oplus \mathcal{R}(u_1)), \tag{13b}$$
$$Z_0^{t+1} = L_0 \oplus \mathcal{R}(w_0) \oplus \mathcal{R}(v_0), \tag{13c}$$
$$Z_1^{t+1} = L_1 \oplus \mathcal{R}(w_1) \oplus \mathcal{R}(v_1), \tag{13d}$$
$$Z_0^{t+2} = H_0 \oplus \mathcal{R}(w_0 \oplus \mathcal{R}(w_1 \oplus \mathcal{R}(u_1))) \oplus \mathcal{R}(L_0 \oplus \mathcal{R}(v_0)), \tag{13e}$$
$$Z_1^{t+2} = H_1 \oplus \mathcal{R}(w_1 \oplus \mathcal{R}(w_0 \oplus \mathcal{R}(u_0))) \oplus \mathcal{R}(L_1 \oplus \mathcal{R}(v_1)). \tag{13f}$$

with

$$v_0 = N_0^t, v_1 = N_1^t, u_0 = S_0^t, u_1 = S_2^t, w_0 = S_1^{t+1}, w_1 = S_3^{t+1},$$
$$L_0 = L_0^t, L_1 = L_1^t, H_0 = H_0^t, H_1 = H_1^t, F_0 = F_0^t, F_1 = F_1^t.$$

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 $S_2^{t+1} = \mathcal{R}(S_1^t) \oplus S_2^t$ and $S_2^t = \mathcal{R}(S_1^{t-1}) \oplus S_2^{t-1}$, there is an underlying relationship of

$$S_2^{t+1} = \mathcal{R}(S_1^t) \oplus \mathcal{R}(S_1^{t-1}) \oplus S_2^{t-1}$$

resulting in an additional constraint s.t. the truncated linear mask of S_1^t equal to that of S_1^{t-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^{256}. So there are no effective linear distinguishers or correlation attacks applicable for LOL-MINI and LOL-DOUBLE.

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-DOUBLE. 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 6: LOL-MINI's guess to determine the path

Time	Operation	Variables	Relations	Complexity
t	Guess	N^t		2^{128}
	Determine	S_2^t	$\mathcal{R}(S_2^t) = Z^t \oplus N^t$	
$t+1$	Guess	N^{t+1}		2^{256}
	Determine	L^t	$L^t = N^{t+1} \oplus \mathcal{R}(N^t)$	
	Determine	S_2^{t+1}	$\mathcal{R}(S_2^{t+1}) = Z^{t+1} \oplus N^{t+1}$	
	Determine	S_1^t	$\mathcal{R}(S_1^t) = S_2^{t+1} \oplus S_2^t$	
$t+2$	Guess	N^{t+2}		2^{384}
	Determine	H^t	$H^t = L^{t+1} = \mathcal{R}(N^{t+1}) \oplus N^{t+2}$	
	Determine	S_2^{t+2}	$\mathcal{R}(S_2^{t+2}) = Z^{t+2} \oplus N^{t+2}$	
	Determine	S_1^{t+1}	$\mathcal{R}(S_1^{t+1}) = S_2^{t+1} \oplus S_2^{t+2}$	
	Determine	S_0^t	$\mathcal{R}(S_0^t) = S_1^t \oplus S_1^{t+1}$	

For LOL-DOUBLE, it is required to guess N_0^t, N_0^{t+1}, N_0^{t+2}, N_1^t 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, $\sigma_L : \mathbb{F}_{2^{16}}^{16} \to \mathbb{F}_{2^{16}}^{8}$ is defined according with the permutation σ in LOL-DOUBLE 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 $\sigma_L(L^t) = F_0^t \oplus \lambda_{C_0}(H_0^t)$, the determination of σ_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 $F_0^{t+1} = S_0^{t+2} \oplus \mathcal{R}(S_3^{t+1}) \oplus S_0^{t+1}$, the determination of

Table 7: LOL-DOUBLE's guess to determine the path

Moments	Operations	Variables	Relations	Complexity
t	Guess	N_0^t		2^{128}
	Determine	S_1^t	$\mathcal{R}(S_1^t) = Z_0^t \oplus N_0^t$	
$t+1$	Guess	N_0^{t+1}		2^{256}
	Determine	L_0^t	$L_0^t = N_0^{t+1} \oplus \mathcal{R}(N_0^t)$	
	Determine	S_1^{t+1}	$\mathcal{R}(S_1^{t+1}) = Z_0^{t+1} \oplus N_0^{t+1}$	
	Determine	S_0^t	$\mathcal{R}(S_0^t) = S_1^{t+1} \oplus S_1^t$	
t	Guess	N_1^t		2^{384}
	Determine	S_3^t	$\mathcal{R}(S_3^t) = Z_1^t \oplus N_1^t$	
$t+2$	Guess	N_0^{t+2}		2^{512}
	Determine	H_0^t, L_0^{t+1}	$H_0^t = L_0^{t+1} = \mathcal{R}(N_0^{t+1}) \oplus N_0^{t+2}$	
	Determine	S_1^{t+2}	$\mathcal{R}(S_1^{t+2}) = Z_0^{t+2} \oplus N_0^{t+2}$	
	Determine	S_0^{t+1}	$\mathcal{R}(S_0^{t+1}) = S_1^{t+2} \oplus S_1^{t+1}$	
	Determine	F_0^t	$F_0^t = \mathcal{R}(S_3^t) \oplus S_0^t \oplus S_0^{t+1}$	
	Determine	$L_1^t[2,5,4]$	$\sigma_L(L_1^t, L_0^t) = F_0^t \oplus C_0 \times H_0^t$	
	Reduce	$L_0^t[4,7,1,5,3]$		2^{432}
	Guess	$L_1^t[7,6,3,1,0]$		2^{512}
	Determine	L_1^t		
$t+1$	Determine	N_1^{t+1}	$N_1^{t+1} = \mathcal{R}(N_1^t) \oplus L_1^t$	
	Determine	S_3^{t+1}	$\mathcal{R}(S_3^{t+1}) = Z_1^{t+1} \oplus N_1^{t+1}$	
	Determine	S_2^t	$\mathcal{R}(S_2^t) = S_3^{t+1} \oplus S_3^t$	
$t+3$	Determine	L_0^{t+2}, H_0^{t+1}	$L_0^{t+2} = H_0^{t+1} = F_0^t$	
	Determine	N_0^{t+3}	$N_0^{t+3} = \mathcal{R}(N_0^{t+2}) \oplus L_0^{t+2}$	
	Determine	S_1^{t+3}	$\mathcal{R}(S_1^{t+3}) = Z_0^{t+3} \oplus N_0^{t+3}$	
	Determine	S_1^{t+2}	$\mathcal{R}(S_1^{t+2}) = Z_0^{t+2} \oplus N_0^{t+2}$	
	Determine	S_0^{t+2}	$\mathcal{R}(S_0^{t+2}) = S_1^{t+3} \oplus S_1^{t+2}$	
	Determine	F_0^{t+1}	$F_0^{t+1} = S_0^{t+2} \oplus \mathcal{R}(S_3^{t+1}) \oplus S_0^{t+1}$	
	Determine	$\sigma_L(H^t)$	$F_0^{t+1} = \sigma_L(L^{t+1}) \oplus C_0 \times H_0^{t+1} =$	
		$H_1^t[2,5,4]$	$\sigma_L(H^t) \oplus C_0 \times F_0^t$	
	Reduce	$H_0^t[4,7,1,5,3]$		2^{432}
	Guess	$H_1^t[7,6,3,1,0]$		2^{512}
	Determine	H_1^t		

σ_H can recover part of H^t. Then the components of H_0^t can be reduced, and the remaining unknown components of H_1^t need be guessed to recover the entire H_1^t. Therefore, the total complexity is 2^{512}.

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

$$
\begin{aligned}
&Z^t = G^t \oplus N^t = \mathcal{R}(S_2^t) \oplus N^t,\\
&Z^{t+1} = G^{t+1} \oplus N^{t+1} = \mathcal{R}(\mathcal{R}(S_1^t) \oplus S_2^t) \oplus \mathcal{R}(N^t) \oplus L^t,\\
&Z^{t+2} = G^{t+2} \oplus N^{t+2} = \mathcal{R}(\mathcal{R}(\mathcal{R}(S_0^t) \oplus S_1^t) \oplus \mathcal{R}(S_1^t) \oplus S_2^t) \oplus \mathcal{R}(\mathcal{R}(N^t) \oplus L^t) \oplus H^t.
\end{aligned}
$$

To fit well with the form of the outputs, we can guess the columns and diagonals of the state blocks for Z^t. 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-DOUBLE. Firstly, for attacks to reconstruct the internal state, LOL-MINI and LOL-DOUBLE 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 $P = M = T = N^{1/2} = 2^{640}, D = N/P = 2^{640}$ for LOL-DOUBLE 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-DOUBLE 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-DOUBLE 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-DOUBLE. 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 $(H^0, L^0, N^0, S_0^0, S_1^0, S_2^0)$, 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

$$\begin{aligned}
Y_{S_2^t} &= \mathrm{SB}(S_2^t),\ 0 \le t \le T;\\
Z^t &= \mathrm{MC} \circ \mathrm{SR}(Y_{S_2^t}) \oplus N^t,\ 0 \le t \le T;\\
N^{t+1} &= \mathrm{MC} \circ \mathrm{SR}(Y_{N^t}) \oplus f^{t-1}(H^0, L^0),\ 0 \le t \le T-1;\\
Y_{N^t} &= \mathrm{SB}(N^t),\ 0 \le t \le T-1;\\
S_2^{t+1} &= \mathrm{MC} \circ \mathrm{SR}(Y_{S_1^t}) \oplus S_2^t,\ 0 \le t \le T-1;\\
Y_{S_1^t} &= \mathrm{SB}(S_1^t),\ 0 \le t \le T-1;\\
S_1^{t+1} &= \mathrm{MC} \circ \mathrm{SR}(Y_{S_0^t}) \oplus S_1^t,\ 0 \le t \le T-2;\\
Y_{S_0^t} &= \mathrm{SB}(S_0^t),\ 0 \le t \le T-2;\\
S_0^{t+1} &= f^{t+1}(H^0, L^0) \oplus \mathrm{MC} \circ \mathrm{SR}(Y_{S_2^t}) \oplus S_0^t,\ 0 \le t \le T-3,
\end{aligned} \tag{14}$$

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

$$N^{t+1},\ 0 \le t \le T-1;\ S_0^{t+1},\ 0 \le t \le T-3;\ S_1^{t+1},\ 0 \le t \le T-2;\ S_2^{t+1},\ 0 \le t \le T-1,$$

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

$$Y_{N^t},\ 0 \le t \le T-1;\ Y_{S_0^t},\ 0 \le t \le T-2;\ Y_{S_1^t},\ 0 \le t \le T-1;\ Y_{S_2^t},\ 0 \le t \le T.$$

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

$$\begin{aligned}
Z^t &= \begin{cases} \mathrm{MC} \circ \mathrm{SR}(Y_{S_2^0}) \oplus N^0,\ t = 0 \\ \mathrm{MC} \circ \mathrm{SR}(Y_{S_2^t}) \oplus \mathrm{MC} \circ \mathrm{SR}(Y_{N^{t-1}}) \oplus f^{t-2}(H^0, L^0),\ 1 \le t \le T \end{cases}\\
Y_{N^t} &= \begin{cases} \mathrm{SB}(N^0),\ t = 0 \\ \mathrm{SB}(\mathrm{MC} \circ \mathrm{SR}(Y_{N^{t-1}}) \oplus f^{t-2}(H^0, L^0)),\ 1 \le t \le T-1 \end{cases}\\
Y_{S_2^t} &= \begin{cases} \mathrm{SB}(S_2^0),\ t = 0 \\ \mathrm{SB}\left(S_2^0 \oplus \bigoplus_{i=0}^{t-1} \mathrm{MC} \circ \mathrm{SR}(Y_{S_1^i})\right),\ 1 \le t \le T \end{cases}\\
Y_{S_1^t} &= \begin{cases} \mathrm{SB}(S_1^0),\ t = 0 \\ \mathrm{SB}\left(S_1^0 \oplus \bigoplus_{i=0}^{t-1} \mathrm{MC} \circ \mathrm{SR}(Y_{S_0^i})\right),\ 1 \le t \le T-1 \end{cases}\\
Y_{S_0^t} &= \begin{cases} \mathrm{SB}(S_0^0),\ t = 0 \\ \mathrm{SB}\left(S_0^0 \oplus \bigoplus_{i=0}^{t-1} \left(f^{i+1}(H^0, L^0) \oplus \mathrm{MC} \circ \mathrm{SR}(Y_{S_2^i})\right)\right),\ 1 \le t \le T-2 \end{cases}
\end{aligned} \tag{15}$$

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

$$Y_{N^t},\ 0 \le t \le T-1;\ Y_{S_0^t},\ 0 \le t \le T-2;\ Y_{S_1^t},\ 0 \le t \le T-1;\ Y_{S_2^t},\ 0 \le t \le T.$$

Moreover, $Y_{S_2^{t+1}}$ and Y_{N^t} for $0 \le t \le T-1$ can be substituted by each other according to the linear equations for $Z^t, 0 \le t \le T$, 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

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

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

Cipher	LOL-MINI	LOL-DOUBLE
$\min T$	6	10
Quadratic eqs. m	8832	22816
Variables n	2944	6400
Complexity of XL $\binom{n}{n/\sqrt{m}}^{\omega}$	$2^{357\omega}$	$2^{531\omega}$

† ω is the parameter for solving linear equation system.

Similarly, we can consider the algebraic attack for unknown key, and construct the systems of equations but take account of the feedback of Z. Since LOL-MINI (LOL-DOUBLE) has 12 initialization rounds, and it at least needs 2 (1) keystream block to obtain the equal information entropy as the unknown key, such a system of equations in a much larger number of variables or with higher algebraic degree is even more unlikely to be solved within a time complexity of 2^{256}.

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.

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

†: 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[7]. We also run the same experiments on the SNOW-V stream cipher[8], 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

[7]There are various frameworks such as QueryPerformanceCounter, Supercop etc. Such frameworks can acquire the data directly by calling the codes in App.E. One can also write his own program. We have applied all the methods mentioned above and the results are similar. In our own program, for example, we run the LOL-DOULBE initialization+192-byte-keystream-generation process for 2^20 times and acquire the time consumption 74ms. Then, the throughput value is computed as $8 \times 192/74 \times 2^{20-30} \times 1000 = 20.27$Gbps. Note that the generation of key-iv's is not a subroutine of concrete stream ciphers: the 74ms timing process starts when the 1st key-iv is loaded into the FSM registers so we may generate 2^{20} key-iv's randomly and stored in RAM in advance; or, we can also use a single key-iv but repeat the loading+initialization+keystream-generation process for 2^{20} times: in both ways, we are able to ignore the time consumption on key-iv generations and acquire pure encryption speeds identical to those using off-the-shelf frameworks.

[8]No speed comparison with SNOW-Vi: Firstly, the available codes of SNOW-V are faster than those of SNOW-Vi given in [EMJY21], whose optimal-speed codes for the best speed listed are not presented. Besides, SNOW-Vi is no faster than LOL-MINI/DOUBLE. According to [EMJY21], the speed of SNOW-Vi is 77.04Gbps with 4.2GHz/AVX2 and 92.34Gbps with 3.9GHz/AVX512, which are equivalent to 0.436 and 0.338 cpd (cycle per byte, a parameter unify the effect of CPU frequencies, the smaller the faster). On the contrary, LOL-MINI has 0.206 cpb for AVX2; LOL-DOUBLE has 0.185/0.136 cbp for AVX2/AVX512.

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

[ARM21] ARM. ARM architecture reference manual ARMv8, for ARMv8-a architecture profile. 2021.

[Bab95] Steve Babbage. Improved “exhaustive search” attacks on stream ciphers. 1995.

[BES18] Marko Balogh, Edward Eaton, and Fang Song. Quantum collision-finding in non-uniform random functions. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 467–486. Springer, Heidelberg, 2018.

[BHK+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 216–233. Springer, Heidelberg, August 1999.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *ASIACRYPT*, 2000.

[BSS22] Xavier Bonnetain, André Schrottenloher, and Ferdinand Sibleyras. Beyond quadratic speedups in quantum attacks on symmetric schemes. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 315–344. Springer, Heidelberg, May / June 2022.

[CAE14] CAESAR. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness, 2014.

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 211–240. Springer, Heidelberg, December 2017.

[CP02] Nicolas T. Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[DDW20] Xiaoyang Dong, Bingyou Dong, and Xiaoyun Wang. Quantum attacks on some feistel block ciphers. *Des. Codes Cryptogr.*, 88(6):1179–1203, 2020.

[DSS+20] Xiaoyang Dong, Siwei Sun, Danping Shi, Fei Gao, Xiaoyun Wang, and Lei Hu. Quantum collision attacks on AES-like hashing with low quantum random access memories. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 727–757. Springer, Heidelberg, December 2020.

[dt18] The ZUC design team. The zuc-256 stream cipher. 2018.

[Dwo07a] M. Dworkin. Recommendation for block cipher modes of operation: Galois/-counter mode (cgm) and gmac. 2007.

[Dwo07b] Morris Dworkin. Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. 2007.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symm. Cryptol.*, 2019(3):1–42, 2019.

[EMJY21] Patrik Ekdahl, Alexander Maximov, Thomas Johansson, and Jing Yang. SNOW-Vi : An extreme performance variant of SNOW-V for lower grade cpus. In *WiSec 2021*, pages 261–272. (ACM), 06 2021.

[ENP19] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the linear keystream biases in AEGIS. *IACR Trans. Symm. Cryptol.*, 2019(4):348–368, 2019.

[Fen20] Cheng Feng. Support of 256-bit algorithm in 5G mobile communication system. *Journal of Information Security Reserach*, 6(8):716–721, 2020.

[GK] S. Gueron and M. E. Kounavis. White paper intel carry-less multiplication instruction and its usage for computing the gcm mode.

[Gol97] Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In *EUROCRYPT*, 1997.

[GZ21] Xinxin Gong and Bin Zhang. Comparing large-unit and bitwise linear approximations of SNOW 2.0 and SNOW 3g and related attacks. *IACR Trans. Symm. Cryptol.*, 2021(2):71–103, 2021.

[HII+22] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Mimematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of Rocca and feasibility of its security claim. *IACR Transactions on Symmetric Cryptology*, 2022, Issue 3:123–151, 2022.

[HK97] Shai Halevi and Hugo Krawczyk. MMH: Software message authentication in the Gbit/second rates. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 172–189. Springer, Heidelberg, January 1997.

[HK18] Matthias Hamann and Matthias Krause. On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. *Cryptography and Communications*, 10:959–1012, 2018.

[HK21] Shoichi Hirose and Hidenori Kuwakado. A note on quantum collision resistance of double-block-length compression functions. In Maura B. Paterson, editor, *18th IMA International Conference on Cryptography and Coding*, volume 13129 of *LNCS*, pages 161–175. Springer, Heidelberg, December 2021.

[HS21] Akinori Hosoyamada and Yu Sasaki. Quantum collision attacks on reduced SHA-256 and SHA-512. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 616–646, Virtual Event, August 2021. Springer, Heidelberg.

[ITU17] ITU. Minimum requirements related to technical performance for IMT-2020 radio interface(s). *Mobile, radiodetermination, amateur and related satellite services*, Report ITU-R M.2410-0(11/2017), 2017.

[JKK+19] M. Juntti, R. Kantola, P. Kysti, S. Lavalle, and E. Peltonen. Key drivers and research challenges for 6G ubiquitous wireless intelligence. 2019.

[KLLN16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 207–237. Springer, Heidelberg, August 2016.

[LaL19] Matti Latva-aho and Kari Leppänen. Key drivers and research challenges for 6G ubiquitous wireless intelligence. 2019.

[LIMS21] Fukang Liu, Takanori Isobe, Willi Meier, and Kosei Sakamoto. Weak keys in reduced AEGIS and Tiaoxin. *IACR Trans. Symm. Cryptol.*, 2021(2):104–139, 2021.

[LZ19] Qipeng Liu and Mark Zhandry. On finding quantum multi-collisions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 189–218. Springer, Heidelberg, May 2019.

[MPH92] Michel Mouly, Marie-Bernadette Pautet, and Thomas Haug. *The GSM System for Mobile Communications*. Telecom Publishing, 1992.

[MV04] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, December 2004.

[NIDI19] Boyu Ni, Gembu Ito, Xiaoyang Dong, and Tetsu Iwata. Quantum attacks against type-1 generalized Feistel ciphers and applications to CAST-256. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 433–455. Springer, Heidelberg, December 2019.

[NW06] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for SNOW 2.0. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 144–162. Springer, Heidelberg, March 2006.

[Rue12] Rainer A. Rueppel. Analysis and design of stream ciphers. 2012.

[SAG06] ETSI SAGE. Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2, document 2: SNOW 3G specification, version 1.1, 2006. 2006.

[SAG11] ETSI SAGE. pecification of the 3gpp confidentiality and integrity algorithms 128-EEA3 & 128-EIA3 , document 2: ZUC specification. version 1.6, 2011. *Security architecture and procedures for 5G System*, 3GPP TS 33.501 version 15.2.0 Release 15, 2011.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Heidelberg, December 2014.

[SJZ+22] Zhen Shi, Chenhui Jin, Jiyan Zhang, Ting Cui, Lin Ding, and Yu Jin. A correlation attack on full SNOW-V and SNOW-vi. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 34–56. Springer, Heidelberg, May / June 2022.

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5g. *IACR Trans. Symm. Cryptol.*, 2021(2):1–30, 2021.

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient aes-based encryption scheme for beyond 5g (full version). *IACR Cryptol. ePrint Arch.*, page 116, 2022.

[SLR+15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda AlKhzaimi, and Chao Li. Links among impossible differential, integral and zero correlation linear cryptanalysis. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 95–115. Springer, Heidelberg, August 2015.

[SS22] André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 717–747. Springer, Heidelberg, August 2022.

[TTU16] Ehsan Ebrahimi Targhi, Gelo Noel Tabia, and Dominique Unruh. Quantum collision-resistance of non-uniformly distributed functions. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 79–85. Springer, Heidelberg, 2016.

[UMHA16] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - toward efficient CBC-mode implementation. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 538–558. Springer, Heidelberg, August 2016.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[WP14] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 185–201. Springer, Heidelberg, August 2014.

[ZFZ22] Zhaocun Zhou, Dengguo Feng, and Bin Zhang. Efficient and extensive search for precise linear approximations with high correlations of full SNOW-V. *Des. Codes Cryptogr.*, 90(10):2449–2479, 2022.

[ZXM15] Bin Zhang, Chao Xu, and Willi Meier. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2015.

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 as the final masking M_T for the tag. The subsequent keystream blocks $Z^2, \ldots, Z^{\ell+1}$ are used to encrypt the plaintext blocks $P^1, \ldots, P^\ell$ and generate the ciphertext blocks $C^1, \ldots, C^\ell$, 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].

Security Analysis. Let ℓ 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 $M, M' \twoheadleftarrow \mathbb{F}_2^*$ and $a \in \{0,1\}^\tau$,

$$\Pr[\text{GHASH}(H, M) \oplus \text{GHASH}(H, M') = a | H \twoheadleftarrow \mathbb{F}_2^n] \leq (l+1)2^{-\tau} \tag{16}$$

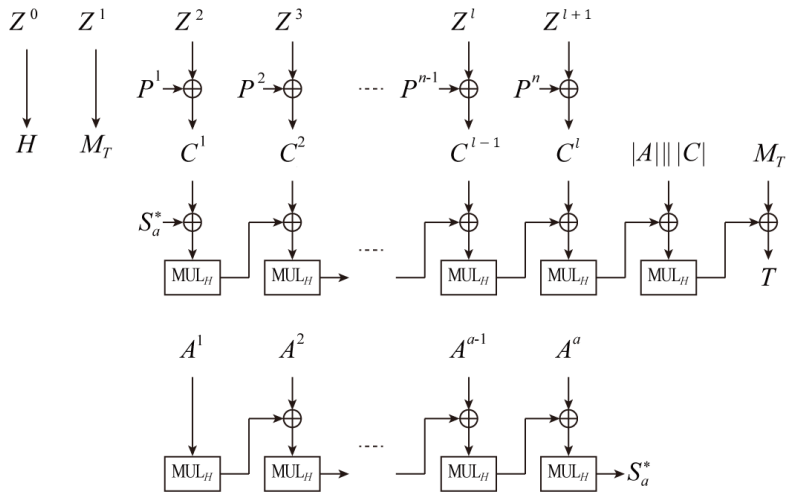

Figure 11: The schematic of GHASH based AEAD process where $\texttt{MUL}_H$ refers to multiplying H over the finite field $\mathbb{F}_{2^n}$

where $a \twoheadleftarrow A$ 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 $\mathcal{A}$ be an adversary that making at most q forward queries of total block length no more than σ. Then*

$$\mathbf{Adv}^{\text{priv}}_{\text{SC}-\text{GCM}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{prf}}_{\text{SC}}(q, \sigma + 2q) + q^2 l / 2^{\tau+1},$$

where $\mathbf{Adv}^{\text{prf}}_{\text{SC}}$ 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 $\mathbf{Adv}^{\text{prf}}_{\text{SC}}(q, \sigma + 2q)$. 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 ϵ-almost XOR universal (AXU). For any $M, M' \twoheadleftarrow \mathbb{F}_2^*$ and $a \in \mathbb{F}_2^\tau$, it should have

$$\Pr[\text{GHASH}(H, M) \oplus \text{GHASH}(H', M') = a | H, H' \twoheadleftarrow \mathbb{F}_2^n] \leq \epsilon,$$

where ϵ 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 $q^2 l / 2^{\tau+1}$. □

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

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

Proof. For the authenticity, McGrew and Viega [MV04] proved that AXU-MAC, which combines an ϵ-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 τ but also the total length of plaintext and associated data ℓ. It is recommended that ℓ 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 $\mathbb{F}_{2^n}$ 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 l, 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 $r, k_1, k_2, \ldots, k_l \xleftarrow{\$} \mathbb{F}_2^n$, 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 $nmh = \bigoplus_{i=1}^{l/2}(m_{2i-1} \oplus k_{2i-1}) \otimes (m_{2i} \oplus k_{2i})$;

2 Compute $Tag = nmh \oplus r$;

Algorithm 7: SC-NMH AEAD mode

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

Output: $C \in \mathbb{F}_2^{|M|}$, $Tag \in \mathbb{F}_2^n$

1 $\mathbf{Pad}(AD) = AD||0^v||[BitLen(AD)]_{64}$, where v is the minimum number that makes the padded AD length is ln and l is even;

2 $\mathbf{Partition}(\mathbf{Pad}(AD)) \rightarrow ad_1, ad_2, \ldots, ad_l$;

3 $\mathbf{Pad}(M) = M||0^u||[BitLen(M)]_{64}$, where u is the minimum number that makes the padded M length is Ln and L is even;

4 $\mathbf{Partition}(\mathbf{Pad}(M)) \rightarrow m_1, m_2, \ldots, m_L$;

5 $s_0, s_0', s_1, s_2, \ldots, s_l, s_{l+1}, s_{l+1}', \ldots, s_{l+L}, s_{l+L}' \leftarrow SC(K, IV_{AEAD})$;

6 $(c_1, c_2, \ldots, c_L) = (m_1, m_2, \ldots, m_L) \oplus (s_{l+1}, s_{l+2}, \ldots, s_{l+L})$;

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 $nmhMAC_M = nmhMAC(s_0', s_{l+1}', s_{l+2}', \ldots, s_{l+L}', m_1, m_2, \ldots, m_L)$;

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 2. Similarly, the security of SC-NMH in Algorithm 7 can also be proved as Proposition 3. As can be seen, the secure bounds of NMH based AEAD is independent to the lengths of plaintexts and associate data.

Proposition 2. *Assuming that $r, k_1, k_2, \cdots k_l \in \mathbb{F}_2^n$ 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 $(m_1^i, m_2^i, \cdots, m_{l_i}^i)$ for $i = 1, 2, \cdots, q$ to nmhMAC, and got the corresponding tags $Tag^1, Tag^2, \cdots, Tag^q$. Now, the adversary tries to make a forgery $(\overline{m_1}, \overline{m_2}, \cdots, \overline{m_{\bar{l}}})$ with $\overline{Tag}$.

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

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

Otherwise, if the adversary tries to challenge nmhMAC with an old key. That is, $\exists j \in [1, q]$, s.t. $(\overline{r}, \overline{k_1}, \overline{k_2}, \cdots \overline{k_{\bar{l}}})$ is a prefix of $(r^j, k_1^j, k_2^j, \cdots k_{l_j}^j)$ or vice versa, we consider in three cases.

1. $\bar{l} = l_j$. Noticing that $(\overline{m_1}, \overline{m_2}, \cdots, \overline{m_{\bar{l}}}, \overline{Tag}) \neq (m_1^j, m_2^j, \cdots, m_{l_j}^j, Tag^j)$, we analyze in the following two subcases.

   (a) $(\overline{m_1}, \overline{m_2}, \cdots, \overline{m_{\bar{l}}}) \neq (m_1^j, m_2^j, \cdots, m_{l_j}^j)$, which means $\exists x \in [1, l_j]$ s.t. $\overline{m_x} \neq m_x^j$. When x is odd, let $a = (\overline{m_x} \oplus k_x^j) \otimes (\overline{m_{x+1}} \oplus k_{x+1}^j) \oplus (m_x^j \oplus k_x^j) \otimes (m_{x+1}^j \oplus k_{x+1}^j)$, and $b = \bigoplus_{i=1}^{\bar{l}/2}(\overline{m_{2i-1}} \oplus k_{2i-1}^j) \otimes (\overline{m_{2i}} \oplus k_{2i}^j) \oplus \bigoplus_{i=1}^{l_j/2}(m_{2i-1}^j \oplus k_{2i-1}^j) \otimes (m_{2i}^j \oplus k_{2i}^j) \oplus a$, we have

   $$\begin{aligned}
   &\Pr[(\overline{m_x} \oplus k_x^j) \otimes (\overline{m_{x+1}} \oplus k_{x+1}^j) \oplus (m_x^j \oplus k_x^j) \otimes (m_{x+1}^j \oplus k_{x+1}^j) = b \oplus \overline{Tag} \oplus Tag^j] \\
   =&\Pr[(\overline{m_x m_{x+1}} \oplus \overline{m_x} k_{x+1}^j \oplus \overline{m_{x+1}} k_x^j) \oplus (m_x^j m_{x+1}^j \oplus m_x^j k_{x+1}^j \oplus m_{x+1}^j k_x^j) = b \oplus \overline{Tag} \oplus Tag^j] \\
   =&\Pr[(\overline{m_x} \oplus m_x^j) k_{x+1}^j = \overline{m_x m_{x+1}} \oplus \overline{m_{x+1}} k_x^j \oplus m_x^j m_{x+1}^j \oplus m_{x+1}^j k_x^j \oplus b \oplus \overline{Tag} \oplus Tag^j] = \frac{1}{2^n}
   \end{aligned}$$

   since each key including k_{x+1}^j is independently-and-uniformly random. When x is even, similar analysis applies.

   (b) $(\overline{m_1}, \overline{m_2}, \cdots, \overline{m_{\bar{l}}}) = (m_1^j, m_2^j, \cdots, m_{l_j}^j)$, but $\overline{Tag} \neq Tag^j$, then by nmhMAC construction, we have $\Pr[0 = \overline{Tag} \oplus Tag^j] = 0$.

2. $\bar{l} > l_j$, let us focus on the last multiplication to compute $\overline{Tag}$. Denote

   $$a_1 = (\overline{m_{\bar{l}-1}} \oplus k_{\bar{l}-1}^j) \otimes (\overline{m_{\bar{l}}} \oplus k_{\bar{l}}^j),$$

   and

   $$a_2 = \bigoplus_{i=1}^{\bar{l}/2}(\overline{m_{2i-1}} \oplus k_{2i-1}^j) \otimes (\overline{m_{2i}} \oplus k_{2i}^j) \oplus a_1,$$

so we have

$$\begin{aligned}
&\Pr[\bigoplus_{i=1}^{\bar{l}/2}(\overline{m_{2i-1}} \oplus k^j_{2i-1}) \otimes (\overline{m_{2i}} \oplus k^j_{2i}) \oplus \overline{r} = \overline{Tag}] \\
=&\Pr[a_1 \oplus a_2 \oplus \overline{r} = \overline{Tag}] \\
=&\Pr[a_1 = a_2 \oplus \overline{r} \oplus \overline{Tag} \oplus \bigoplus_{i=1}^{l_j/2}(m^j_{2i-1} \oplus k^j_{2i-1}) \otimes (m^j_{2i} \oplus k^j_{2i}) \oplus r^j \oplus Tag^j] \\
=&\Pr[a_1 = a_2 \oplus \overline{Tag} \oplus \bigoplus_{i=1}^{l_j/2}(m^j_{2i-1} \oplus k^j_{2i-1}) \otimes (m^j_{2i} \oplus k^j_{2i}) \oplus Tag^j] \leq 3/2^n,
\end{aligned}$$

where $\Pr[\overline{m_{\bar{l}-1}} \oplus k^j_{\bar{l}-1} = 0] = 1/2^n$, $\Pr[\overline{m_{\bar{l}}} \oplus k^j_{\bar{l}} = 0] = 1/2^n$, and $\Pr[a_1 = w] = \frac{1\times(2^n-2)}{(2^n-1)(2^n-1)}$ for any $w \neq 0$ independent of a_1.

3. $\bar{l} < l_j$, by the independent-and-uniform randomness of $k^j_{l_j-1}$ and $k^j_{l_j}$, similar analysis applies.

By the above, we conclude the proof. □

Proposition 3. *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+nmhMAC is upper bounded by,*

$$\begin{aligned}
\mathbf{Adv}^{priv}_{SC-NMH}(q) &\leq \mathbf{Adv}^{prf}_{SC}(q), &(17)\\
\mathbf{Adv}^{auth}_{SC-NMH}(q) &\leq 3/2^n + \mathbf{Adv}^{prf}_{SC}(q+1). &(18)
\end{aligned}$$

Proof. Suppose an adversary $\mathcal{A}$ 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 $s_0, s'_0, s_1, s_2, \cdots, s_l, s_{l+1}, s'_{l+1}, \cdots, s_{l+L}, s'_{l+L}$ from random bits is upper bounded by $\mathbf{Adv}^{prf}_{SC}(q)$ which, for the privacy aspect, proves Proposition 3. For the authentication aspect, suppose $\mathcal{A}$ makes a forgery $(\overline{IV_{AEAD}}, \overline{AD}, \overline{C}, \overline{Tag})$, where the padded $\overline{AD}$ and $\overline{C}$ have $\bar{l}$ and $\overline{L}$ blocks respectively, and we denote $\overline{M}$ as the expected plaintext for $\overline{C}$. Then we consider in two cases.

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

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

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

   (a) $\bar{l} + \overline{L} > l^i + L^i$. By the independent-and-uniform randomness of $s'_{\bar{l}+\overline{L}}$, we have

$$\Pr[nmhMAC_{\overline{M}} = \overline{Tag} \oplus nmhMAC_{\overline{AD}}] = 3/2^n + \mathbf{Adv}^{prf}_{LOL}(q).$$

   (b) $\bar{l} + \overline{L} = l^i + L^i$ and $\bar{l} < l^i$, and $\overline{L} > L^i$. Notice that by the nmhMAC padding rule, we have $\overline{m_{\overline{L}}} \neq m_{L^i}$, then by the randomness of $s'_{\bar{l}+\overline{L}}$, we have

$$\begin{aligned}
&\Pr[nmhMAC_{\overline{M}} = \overline{Tag} \oplus nmhMAC_{\overline{AD}}] \\
=&\Pr[nmhMAC_{\overline{M}} \oplus nmhMAC_{M^i} = \overline{Tag} \oplus nmhMAC_{\overline{AD}} \oplus Tag^i \oplus nmhMAC_{AD^i}] \\
=&3/2^n + \mathbf{Adv}^{prf}_{LOL}(q).
\end{aligned}$$

(c) $\overline{l}+\overline{L} = l^i + L^i$ and $\overline{l} > l^i$, and $\overline{L} < L^i$. Notice that by the nmhMAC padding rule, we have $\overline{m_{\overline{L}}} \neq m_{L^i}$, then by the randomness of $s'_{\overline{l+L}}$, we have

$$\begin{aligned}&\Pr[nmhMAC_{\overline{M}} = \overline{Tag} \oplus nmhMAC_{\overline{AD}}]\\ =&\Pr[nmhMAC_{\overline{M}} \oplus nmhMAC_{M^i} = \overline{Tag} \oplus nmhMAC_{\overline{AD}} \oplus Tag^i \oplus nmhMAC_{AD^i}]\\ =&3/2^n + \mathbf{Adv}^{prf}_{LOL}(q).\end{aligned}$$

(d) $\overline{l} = l^i$ and $\overline{L} = L^i$. In this subcase, we have $(\overline{AD}, \overline{C}) \neq (AD^i, C^i)$, which means $\exists \overline{AD}_{j1} \neq AD^i_{j1}$ for some $j1 \in [1, \overline{l}]$, or $\exists \overline{C}_{j2} \neq C^i_{j2}$ for some $j2 \in [1, \overline{L}]$, or both exist. Then by the independently random keys $s_1, s_2, \cdots, s_{l^i}, s'_{l^i+1}, s'_{l^i+2}, \cdots, s'_{l^i+L^i}$ we have

$$\begin{aligned}&\Pr[nmhMAC_{\overline{M}} = \overline{Tag} \oplus nmhMAC_{\overline{AD}}]\\ =&\Pr[nmhMAC_{\overline{M}} \oplus nmhMAC_{M^i} = \overline{Tag} \oplus nmhMAC_{\overline{AD}} \oplus Tag^i \oplus nmhMAC_{AD^i}]\\ =&3/2^n + \mathbf{Adv}^{prf}_{LOL}(q).\end{aligned}$$

(e) $\overline{l}+\overline{L} < l^i + L^i$. By the independent-and-uniform randomness of $s'_{l^i+L^i}$, we have

$$\begin{aligned}&\Pr[nmhMAC_{\overline{M}} = \overline{Tag} \oplus nmhMAC_{\overline{AD}}]\\ =&\Pr[nmhMAC_{\overline{M}} \oplus nmhMAC_{M^i} = \overline{Tag} \oplus nmhMAC_{\overline{AD}} \oplus Tag^i \oplus nmhMAC_{AD^i}]\\ =&3/2^n + \mathbf{Adv}^{prf}_{LOL}(q).\end{aligned}$$

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.

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

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

†: Implemented with the AVX512 instructions.

Table 12: The software performance (Gbps) of SNOW-V, LOL-MINI and LOL-DOUBLE in NMH based AEAD modes.

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

†: 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

LOL-DOUBLE	Power (W/Hz)	Area (GE)	non-AES(GE)	Throughput(Gbps)
6AES	0.0675	76645.73	29757.97	256
2AES	0.0319	41184.20	25554.95	85.3
1AES	0.0168	29953.02	22138.39	42.67
LOL-MINI	Power (W/Hz)	Area (GE)		Throughput(Gbps)
4AES	0.0442	50502.55	19244.05	128
2AES	0.0260	33730.97	18101.72	64
1AES	0.0146	22416.03	14601.40	32

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. We will explain more as well.

C Galois LFSR in LOL-MINI

The overall Galois LFSR is shown in Fig. 12.

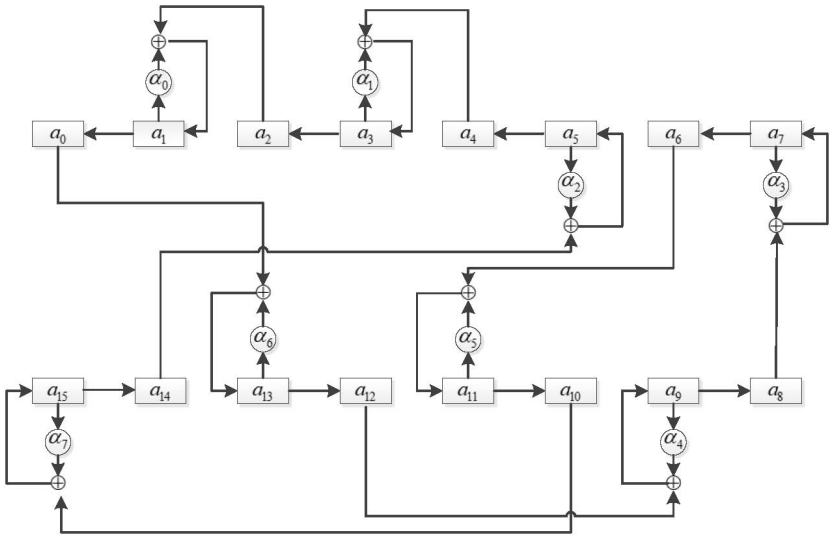

Figure 12: The overall Galois LFSR of LOL-MINI cipher

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_68_df_de_63_49_f5_9b_4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27
```

```
IV: 4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27


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_d6_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_f8_7a_b2_61_c7_37
S0:83_4d_51_b5_45_3f_63_8b_21_47_46_cb_7c_95_52_72
S1:58_5d_d0_cd_eb_63_90_2b_f3_04_b2_0a_1e_57_0d_a3
S2:a5_5e_e4_33_c1_79_bc_d9_7a_9a_3c_c3_99_1e_11_e1
t= -10:
H:e1_11_36_4c_81_12_10_05_c3_e2_f8_bc_c7_20_48_2b
L:0d_b0_80_ac_d2_cf_d6_7a_18_31_9b_19_d1_02_a4_54
N:32_7e_41_99_81_02_e3_2e_5b_1a_ec_a5_b0_57_1a_a8
S0:0c_8f_84_36_75_81_c6_97_99_f7_46_0d_09_d4_dd_6e
S1:40_61_90_b2_b6_c7_97_ee_ea_0d_6e_d1_32_88_8a_54
S2:95_eb_08_2b_f8_59_4d_e6_1b_93_3d_93_9f_70_09_3d
t= -9:
H:09_a2_44_32_48_0c_f1_21_98_91_3e_73_e1_d6_5e_3a
L:e1_11_36_4c_81_12_10_05_c3_e2_f8_bc_c7_20_48_2b
N:59_59_4c_c6_07_74_7f_63_a6_73_40_38_12_21_44_b8
S0:37_53_81_9d_bc_8f_d4_98_5a_7c_94_db_58_55_99_fc
S1:c7_c9_29_15_e5_a9_97_a2_4d_8b_5e_d1_39_4c_4b_6d
S2:f9_c5_ef_5e_31_10_75_55_78_85_01_a0_b4_43_54_08
t=-8:
H:01_10_f1_09_ed_46_c2_c3_cc_47_60_83_73_69_2b_5b
L:09_a2_44_32_48_0c_f1_21_98_91_3e_73_e1_d6_5e_3a
N:b0_a9_ca_1e_b5_f1_35_16_fd_2f_32_a9_b8_6e_00_be
S0:6f_1a_3c_52_56_bd_69_38_30_48_b4_60_39_1d_f6_1f
S1:11_f6_e5_f5_b8_d9_68_78_ec_05_31_e7_8c_8d_b0_99
S2:35_25_55_c6_2e_e3_86_1f_4b_a4_5a_90_ec_cb_70_8d
t= -7:
H:47_a1_29_d5_4c_cd_e7_1b_82_26_9d_f0_1f_af_28_3a
L:01_10_f1_09_ed_46_c2_c3_cc_47_60_83_73_69_2b_5b
N:55_70_2c_55_b2_db_34_c6_ca_e1_8d_84_98_3e_36_04
S0:98_12_df_99_af_81_bb_35_4f_41_1b_39_9e_dc_de_9b
S1:78_d8_26_d0_f6_6a_51_f0_f8_45_e7_fc_6b_d7_57_5c
S2:50_46_b7_24_f2_75_ba_9f_23_5d_e6_4a_28_15_d6_af
t= -6:
H:de_ac_c8_0c_2d_ac_f6_5a_e1_11_9c_1b_f9_c0_84_ac
L:47_a1_29_d5_4c_cd_e7_1b_82_26_9d_f0_1f_af_28_3a
N:31_f3_93_d7_79_11_46_11_85_43_c8_58_df_ad_e8_b7
S0:13_ce_3b_c0_30_f6_79_a9_64_b1_0a_a6_ff_22_6c_33
S1:3d_bd_15_ca_5c_e2_f5_7c_fb_29_71_3f_78_be_54_d4
S2:71_77_81_05_06_d1_90_d1_a6_7f_33_47_03_10_15_22
t= -5:
H:54_fb_53_6e_6c_be_66_67_72_73_7c_0b_2a_24_24_77
L:de_ac_c8_0c_2d_ac_f6_5a_e1_11_9c_1b_f9_c0_84_ac
N:7a_0e_da_84_18_c0_f4_ee_51_4a_1d_74_d1_dc_17_5c
S0:76_c6_fb_79_25_59_59_df_93_81_be_f5_0a_ab_a0_f3
S1:78_d0_48_99_c5_2b_14_2c_86_8c_b2_2b_f9_19_e2_df
S2:fc_ac_0b_62_df_ce_d4_bb_c6_c7_a3_5a_ae_b9_94_f3
t=-4:
H:d8_f2_d9_85_0b_07_02_7c_90_0e_59_04_f4_4e_89_4d
L:54_fb_53_6e_6c_be_66_67_72_73_7c_0b_2a_24_24_77
N:04_f0_5b_37_0e_c6_36_1a_a6_81_cc_d5_aa_29_aa_cc
S0:d4_3a_f8_78_36_9e_af_4d_52_c5_fa_85_2f_39_3e_e2
S1:2b_df_0e_e6_b2_01_ee_a4_06_7b_79_69_fb_7f_d6_df
S2:7e_e3_c4_d4_42_92_22_3f_6d_9c_b4_5f_b2_36_d0_c0
t= -3:
H:8b_e0_92_a3_94_9c_1f_7b_0c_68_dc_2b_41_32_3e_e4
L:d8_f2_d9_85_0b_07_02_7c_90_0e_59_04_f4_4e_89_4d
N:26_28_93_cc_b5_f9_2e_1f_bd_3f_6c_fa_b7_d4_ff_e9
S0:5b_2a_31_ec_ac_c4_86_2c_f8_2c_ea_7b_c4_22_aa_ca
S1:ea_89_39_eb_df_87_b9_a8_77_b1_92_42_fe_04_41_c0
S2:4e_80_e9_c4_3e_7e_24_f6_d5_7b_ab_5f_38_ee_4c_ab
t=-2:
H:15_f4_03_0c_84_2b_cc_e1_18_e2_30_fa_05_38_f1_d1
L:8b_e0_92_a3_94_9c_1f_7b_0c_68_dc_2b_41_32_3e_e4
N:a2_b3_f8_95_73_f5_e8_38_a9_d4_67_7f_34_73_08_b8
S0:68_f6_a1_2c_9d_16_64_d2_5d_f1_b6_7b_76_ce_a4_f2
```

```
S1:47_a1_b6_63_9a_cf_b2_ca_8d_21_13_35_8b_84_3d_9f
S2:02_d8_c0_c5_72_55_3a_58_3d_82_62_e1_a0_a7_d6_85
t= -1:
H:89_71_e4_6c_1f_27_87_b2_ae_ea_2d_9a_f8_6f_2f_ba
L:15_f4_03_0c_84_2b_cc_e1_18_e2_30_fa_05_38_f1_d1
N:be_e5_53_f0_8b_45_ed_a8_27_86_4b_d9_cc_76_f0_97
S0:43_34_bd_d5_f1_c4_0b_58_5a_cf_fc_9e_ba_d2_83_f0
S1:80_22_f3_a5_7f_91_a9_2a_9f_8f_4a_c8_88_02_a4_46
S2:a5_36_d1_04_45_36_6e_60_e3_00_f1_80_7a_f5_d2_85

End of Initialization (t=0):
H:1d_a8_41_ff_0e_70_97_f2_b4_32_fc_69_8e_5a_e8_32
L:89_71_e4_6c_1f_27_87_b2_ae_ea_2d_9a_f8_6f_2f_ba
N:91_ed_66_3f_95_74_2a_00_03_5d_e2_c0_da_76_2f_8d
S0:e0_57_2f_f9_c7_a3_f5_05_7f_0a_50_64_86_a2_eb_e9
S1:ba_3a_17_a7_69_29_4d_7b_42_98_cb_4f_9f_28_6d_88
S2:2a_f2_02_72_a6_46_d3_0a_13_51_ce_72_42_2c_6a_1c

Output Keystream Z's for t=0...15:
ca_7a_6e_4c_c0_01_4d_5b_74_06_c2_4f_65_e1_b5_97
9d_64_c2_18_d0_ef_ba_1f_e1_31_c7_1c_9d_bb_74_ef
06_8b_e0_9f_ec_71_96_2d_7b_2b_ef_a6_1f_b9_c9_25
45_c4_45_b8_43_79_fc_e5_76_eb_5d_eb_91_8e_38_42
92_14_6d_c7_6e_08_0c_6a_38_fc_60_49_e0_5f_90_a4
3b_59_60_f8_e7_f3_8a_cc_8d_6b_87_21_be_b7_d5_eb
19_98_c7_8a_61_96_81_c6_2c_cb_3d_ca_ab_05_e2_d8
a7_d3_87_45_a6_a5_23_9b_33_13_ea_5d_8e_87_6c_31
b7_bd_44_49_77_af_c7_7c_ea_32_9b_82_ef_ee_eb_9f
c0_85_35_b1_82_ee_9b_79_06_9d_fb_0e_9f_0e_d4_15
37_40_62_08_42_51_f3_36_ca_a5_26_61_c6_0d_24_d6
e5_5f_c1_ad_a2_b0_15_cc_e7_4e_2a_a5_6c_d9_06_b4
b5_29_b3_35_8a_84_74_1a_2e_0a_0a_12_b8_26_26_4f
ce_0b_7d_2a_44_2d_2c_97_f6_c5_eb_72_c6_cb_6f_9c
80_db_7c_dd_93_ab_ea_e2_86_9e_93_27_c6_5d_7a_09
e9_9c_b5_75_32_e2_08_45_44_bd_3a_2f_35_9c_d1_ff
```

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

```
LOL-DOUBLE

Key:4e_38_40_08_81_c1_dc_aa_87_68_df_de_63_49_f5_9b_4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27
IV:30_62_1b_29_44_f7_cd_9a_b4_6d_3f_5d_a8_3b_0b_48_8e_fd_d1_19_97_f0_b5_f1_39_76_dd_d2_ad_97_f6_26

t= -12:
H:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_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_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00
N0:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00
N1:00_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
S1:30_62_1b_29_44_f7_cd_9a_b4_6d_3f_5d_a8_3b_0b_48
S2:4e_16_c0_2b_32_93_58_ad_31_19_c4_94_1d_15_85_27
S3:4e_38_40_08_81_c1_dc_aa_87_68_df_de_63_49_f5_9b
t= -11:
H:d5_db_ac_0b_9f_44_4d_ee_5c_e3_0a_2c_e3_fb_04_57_eb_48_24_18_f3_ea_e4_74_4b_83_f8_57_84_0b_94_c6
L:00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00
N0:88_2b_47_7b_90_89_87_17_28_e0_9b_34_e7_68_f7_a5
N1:b6_b8_cf_68_fc_27_2e_8d_3f_80_69_4f_80_98_67_34
S0:65_b5_f5_01_64_1a_51_85_72_f5_25_85_29_9c_62_e0
S1:26_07_8b_ec_2e_55_81_1b_c0_01_52_89_d5_25_f3_70
S2:9b_cd_6c_20_ad_d7_15_43_6d_fa_ce_b8_fe_ee_81_70
S3:43_88_c0_a4_53_0e_0a_d0_9f_59_44_c7_b2_4b_51_cf
t= -10:
H:89_9c_3a_51_b6_8f_32_4c_ac_19_67_eb_a7_eb_46_a2_1c_b0_dc_84_51_df_4e_a9_34_ed_25_9d_1c_57_3b_e2
L:d5_db_ac_0b_9f_44_4d_ee_5c_e3_0a_2c_e3_fb_04_57_eb_48_24_18_f3_ea_e4_74_4b_83_f8_57_84_0b_94_c6
N0:9c_b1_3d_cf_4e_60_74_a1_5e_ba_ab_ee_ab_7a_3f_f4
N1:2e_c9_8e_c5_d6_74_44_9c_5e_fe_ec_52_9c_68_be_ae
S0:cf_bd_e6_ed_c9_e2_31_a1_79_98_69_57_b5_53_3e_36
S1:68_c3_ca_54_34_43_f8_81_37_02_bd_a4_ea_73_dc_07
S2:9a_7a_11_0a_8b_d1_a0_18_e9_03_32_67_be_6d_30_77
S3:91_cd_8d_7d_7d_50_f4_00_a2_db_fd_f2_ec_b1_0b_f1
t= -9:
H:82_63_db_cb_a9_da_48_a3_ed_f2_5e_97_b3_eb_0b_0b_9e_7a_f9_40_3b_1d_e6_59_b6_31_aa_22_0f_e3_ff_9e
L:89_9c_3a_51_b6_8f_32_4c_ac_19_67_eb_a7_eb_46_a2_1c_b0_dc_84_51_df_4e_a9_34_ed_25_9d_1c_57_3b_e2
N0:f2_5b_64_48_7a_37_51_b4_f6_f4_dc_f2_31_b7_99_d2
N1:fb_6e_e6_89_84_49_b6_4f_c2_5c_0b_14_ff_83_9e_da
S0:7f_0e_91_68_24_8b_93_64_91_57_2f_27_26_d8_7f_06
S1:c2_d4_f7_a8_76_ff_57_37_d6_0e_28_ef_8b_21_09_0f
S2:84_a8_f7_0e_6c_6b_9c_1a_5a_4b_c7_1e_a6_fc_04_88
S3:3d_1f_52_f9_13_25_1c_fe_4b_f8_78_ac_9e_c8_42_e3
```

```
t= -8:
H:2d_2b_55_b4_e8_ba_5a_3f_a9_35_ab_43_e2_47_29_ac_d2_22_d4_c3_d6_26_45_d5_1a_57_38_73_2c_b3_74_27
L:82_63_db_cb_a9_da_48_a3_ed_f2_5e_97_b3_eb_0b_0b_9e_7a_f9_40_3b_1d_e6_59_b6_31_aa_22_0f_e3_ff_9e
N0:c8_c2_d0_84_74_69_7a_ff_e9_42_70_5b_cc_f2_2f_2e
N1:f1_d8_70_4b_a4_0c_4a_37_a1_15_fd_90_93_46_9d_14
S0:56_42_a3_22_76_e4_60_fe_49_5c_1c_40_f5_e8_95_fb
S1:e8_af_d6_22_4a_67_2f_c9_1b_51_56_4f_ad_4c_d9_01
S2:5b_d8_c6_f2_fe_e6_97_91_05_8a_b0_af_75_0c_b4_f6
S3:3a_cb_98_5a_51_4d_1b_17_89_a7_d4_d7_ce_d5_cf_01
t= -7:
H:82_18_c5_c9_5e_5a_eb_6c_69_26_90_95_17_4d_c5_b3_79_80_60_c3_33_78_b6_d2_35_41_57_09_b6_f1_91_28
L:2d_2b_55_b4_e8_ba_5a_3f_a9_35_ab_43_e2_47_29_ac_d2_22_d4_c3_d6_26_45_d5_1a_57_38_73_2c_b3_74_27
N0:3e_f9_9e_5b_7b_1e_1a_d0_9a_49_04_b9_98_7d_5b_49
N1:ec_5b_e1_a5_e1_09_28_ce_7a_7c_36_10_04_c2_c4_47
S0:de_1a_b3_aa_e1_90_9c_1b_dd_08_b6_d9_d0_5f_99_c7
S1:56_ca_f7_13_7b_b6_0d_1a_2b_e1_76_fc_30_c5_ff_78
S2:11_02_d3_bf_d4_d5_06_02_85_ee_50_61_ae_b3_5e_6b
S3:24_e5_29_1a_55_22_9b_5c_ca_28_98_5a_13_3c_e8_00
t= -6:
H:48_76_e6_f2_20_14_3e_23_10_f5_9c_d5_12_10_58_ef_eb_c0_dd_20_54_04_46_be_d2_c9_d5_db_ba_76_be_20
L:82_18_c5_c9_5e_5a_eb_6c_69_26_90_95_17_4d_c5_b3_79_80_60_c3_33_78_b6_d2_35_41_57_09_b6_f1_91_28
N0:84_22_3d_64_40_dc_e0_5e_c3_b8_6a_12_86_d3_27_72
N1:d5_18_5e_36_ad_7d_e7_be_e4_7e_29_8b_d2_04_ba_aa
S0:d9_81_8f_2f_54_9d_f2_6b_75_bd_55_12_6e_eb_e3_a0
S1:8c_fb_34_b1_aa_a9_f3_e6_e8_c5_69_3d_75_aa_18_40
S2:67_8d_ab_16_8f_df_22_f1_0f_52_c8_0d_24_de_5d_cd
S3:dc_3d_a1_82_f2_4b_00_e2_2b_71_d4_bd_dd_8b_b6_8a
t= -5:
H:76_35_00_cf_92_8f_93_c9_ff_d3_2a_43_ca_3a_6b_54_19_6b_c6_33_d0_38_7e_29_ee_a7_71_43_b6_75_60_73
L:48_76_e6_f2_20_14_3e_23_10_f5_9c_d5_12_10_58_ef_eb_c0_dd_20_54_04_46_be_d2_c9_d5_db_ba_76_be_20
N0:0a_54_bb_87_2e_1c_89_4c_38_e9_90_53_f0_9b_07_cc
N1:34_4f_41_ab_9f_57_bb_1e_ef_3a_1c_89_db_96_ec_3f
S0:15_f2_17_2a_29_d8_6b_fc_7f_64_0d_da_0a_9a_39_79
S1:98_6b_ad_f2_90_d0_d7_f9_08_27_22_a7_86_55_cc_ef
S2:95_9a_96_bd_5d_8c_51_66_33_39_88_5c_68_37_11_eb
S3:bf_a3_0d_ab_99_28_34_76_eb_d5_ef_cd_66_6d_fc_d3
t= -4:
H:31_2a_57_41_65_7c_1a_42_bd_1f_19_ab_b2_6c_f7_49_92_da_77_f4_b7_a8_47_9f_6a_53_0a_7b_4d_bd_69_9a
L:76_35_00_cf_92_8f_93_c9_ff_d3_2a_43_ca_3a_6b_54_19_6b_c6_33_d0_38_7e_29_ee_a7_71_43_b6_75_60_73
N0:8d_cf_fb_3c_2a_06_00_03_3d_92_3d_bd_df_88_96_58
N1:c0_02_c8_db_97_07_2b_1b_c0_f2_cf_5a_9f_9a_c8_08
S0:b3_67_21_75_01_27_97_7d_fa_0d_1b_28_9c_b1_bc_dc
S1:ac_ee_0c_f3_ac_96_9d_e3_c9_af_6c_66_43_1d_35_c2
S2:ae_e4_7a_7b_16_ec_c2_68_b6_cf_01_a4_2a_c0_e1_6e
S3:39_ab_12_9c_5b_fa_4a_e2_d6_ea_8a_cc_b2_f6_00_03
t= -3:
H:d1_89_b3_de_5e_0b_e8_0a_93_17_88_09_30_00_c8_18_08_dd_8d_e4_63_2d_17_75_a5_21_66_36_6d_40_17_d0
L:31_2a_57_41_65_7c_1a_42_bd_1f_19_ab_b2_6c_f7_49_92_da_77_f4_b7_a8_47_9f_6a_53_0a_7b_4d_bd_69_9a
N0:1e_5b_59_98_80_65_61_e9_69_fa_74_fe_9f_f2_82_bf
N1:80_e0_a1_eb_a5_b0_42_ec_27_85_ff_8b_f3_aa_93_2d
S0:7b_b8_64_4a_f5_0d_ab_13_9f_de_b2_44_6e_6b_63_04
S1:5a_27_45_d6_d9_68_ed_ff_3f_7b_19_d6_6b_00_d4_9e
S2:f2_a2_32_99_62_e1_2a_61_18_4a_b4_10_c5_48_bf_2e
S3:a6_b4_1a_00_68_d8_3b_e9_90_60_d7_bd_69_a5_ca_88
t= -2:
H:f2_9c_3d_0a_d0_78_29_1b_91_e5_f2_9d_7d_e4_ac_3a_5d_a6_2e_2d_3c_51_22_dd_62_46_54_f2_43_0c_89_1a
L:d1_89_b3_de_5e_0b_e8_0a_93_17_88_09_30_00_c8_18_08_dd_8d_e4_63_2d_17_75_a5_21_66_36_6d_40_17_d0
N0:ec_1a_a9_89_dd_c4_c4_de_e8_0f_cb_e8_ed_e5_37_f1
N1:4a_d8_d0_c5_6c_c8_e8_92_22_9b_51_0c_c5_45_b7_c7
S0:a6_fe_eb_8c_6c_ec_cb_22_da_1d_19_3d_de_cd_79_33
S1:0a_01_bf_8c_cf_de_4c_64_39_60_ed_b5_c4_13_0b_ce
S2:1e_65_56_0b_32_fc_62_93_e0_55_32_73_27_5e_91_ab
S3:1e_00_2f_a3_72_5c_c9_b6_5d_ca_24_9a_46_72_60_17
t= -1:
H:32_2d_8d_96_a7_8d_e3_c7_16_38_f4_e3_01_b0_fb_5e_dc_13_e9_09_83_6c_1b_8c_52_1a_7d_eb_38_93_05_82
L:f2_9c_3d_0a_d0_78_29_1b_91_e5_f2_9d_7d_e4_ac_3a_5d_a6_2e_2d_3c_51_22_dd_62_46_54_f2_43_0c_89_1a
N0:ff_7e_e3_3d_10_3b_5c_5e_fc_d0_da_ea_47_0e_e5_24
N1:89_35_4a_4b_e5_05_64_d6_c6_6e_8a_dd_02_15_14_a0
S0:30_35_d2_40_83_48_38_3c_aa_9c_35_da_23_1b_cb_76
S1:14_70_0b_71_9b_3c_9d_93_ce_12_3f_7c_c8_ef_16_de
S2:c0_52_72_14_48_b5_45_8a_1e_62_0d_78_cb_0b_5d_04
S3:58_2c_21_b2_3d_b9_65_64_b3_3a_ff_fb_97_de_fd_74

End of Initialization (t=0):
H:1c_a8_ab_9c_35_1f_d4_d5_f7_f5_fa_30_f8_8d_75_00_e2_68_85_fa_a2_0d_33_f1_36_6c_07_fe_7e_ab_d6_fc
L:32_2d_8d_96_a7_8d_e3_c7_16_38_f4_e3_01_b0_fb_5e_dc_13_e9_09_83_6c_1b_8c_52_1a_7d_eb_38_93_05_82
N0:58_d9_93_d7_0f_a8_82_79_f2_c0_ed_87_bb_0b_00_e2
N1:57_af_98_50_bb_c2_10_b4_fa_d6_0e_b1_17_e0_64_3f
```

```
S0:15_7e_dd_da_f6_d3_37_b6_6b_87_7c_6d_42_b0_8c_0d
S1:0d_ee_a9_70_e4_c8_41_99_ea_1f_e1_6b_4c_e9_eb_59
S2:6d_bc_7a_bd_ec_50_11_ab_92_2f_f2_7c_17_c1_38_bb
S3:0f_23_f4_71_30_47_94_01_3c_c9_80_35_f7_9f_64_86

Output Keystream Z's for t=0...15:
50_41_39_fc_fe_3b_74_49_8a_17_de_e6_58_91_d4_43_3f_e2_c5_11_96_8a_94_a6_8f_96_de_ae_5a_a8_2c_2d
3f_e8_cf_a9_02_e1_59_95_79_fb_00_99_00_69_0d_4a_84_1f_c9_3a_b1_fd_e1_a8_64_d1_42_9c_48_b2_ee_13
e4_bb_97_d3_5a_3d_9b_51_df_9e_2a_72_fe_dc_c5_c9_bb_5f_ce_ca_63_36_dd_89_3d_88_47_2c_95_30_b4_b1
4b_f7_5a_cc_40_ba_64_9f_79_b6_e2_36_37_a0_04_43_af_e3_33_ff_ae_54_e3_c5_45_12_d2_ea_f3_81_1b_63
c6_5c_ce_9d_a5_83_14_32_ed_ba_40_ec_b8_30_10_77_55_54_03_d5_de_f2_61_46_e2_b3_c3_d0_b8_57_6b_15
4d_f7_86_96_ab_64_d4_ca_b1_18_16_f4_8c_87_c7_4b_39_6f_53_a8_57_48_f3_7b_de_83_f7_8b_26_42_90_a5
20_2a_8e_91_c4_6f_2d_c6_6c_21_90_42_02_36_4d_31_d9_ef_5f_53_38_b5_53_eb_d2_81_5a_8e_d1_46_72_6d
e3_74_5d_6f_a0_92_52_01_f4_f1_ea_25_44_d6_62_65_bf_2f_46_2e_d9_c1_34_43_f1_31_90_13_c1_d1_10_0f
8f_53_dc_7c_5a_f3_c5_56_d3_08_45_27_c5_07_66_45_aa_59_1c_f3_1b_79_46_14_49_4b_a7_42_34_13_ce_b8
dc_df_a4_2d_93_5a_9e_01_d4_8e_90_b7_9e_9e_f3_8a_53_7e_9b_af_62_99_c3_bd_00_1f_77_4a_f8_26_07_3a
75_87_c8_7a_c0_bc_72_e8_5c_f4_6f_e3_7f_ed_58_08_db_1f_76_6a_f4_79_db_78_ed_cc_45_dd_e2_03_22_d5
5a_92_ef_35_3e_b7_77_a9_b5_e8_c1_7b_c9_e2_1e_75_ab_83_10_f7_03_dc_c1_b2_e3_2e_5f_a8_12_1a_00_2e
c7_cf_80_4a_f9_39_18_a5_6c_5f_ac_a2_6e_1b_b1_76_8a_91_2e_81_c1_5e_18_ac_df_52_d6_68_13_87_6e_1f
8f_53_53_95_ce_9b_95_68_98_01_c0_bd_37_75_45_6c_a4_24_ae_60_1f_ec_07_15_fa_86_16_ee_90_d5_8a_6f
0c_40_4e_23_4d_c4_99_a1_74_0a_1f_ca_44_b6_dd_ad_02_04_36_22_8d_d9_c7_e6_f8_34_2c_1b_39_1f_a8_a6
b9_56_2a_af_ec_d9_1f_79_82_36_e6_18_d8_a2_d6_fb_58_d0_a0_9d_52_14_42_a3_b2_35_c3_5d_7f_6c_f5_53
```

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 <immintrin.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, 0x7ca1, 0xc553, 0x7dc5, 0xd83, 0xb2eb, 0xd52f, 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, 6, 7);
const __m256i LOLDOUBLE_SHUFFLE1 = _mm256_setr_epi8(0xff, 0xff, 0, 1, 0xff, 0xff, 4, 5, 0xff, 0xff, 0xff, 0xff, 12, 13, 0xff, 0xff, 0xff, 0xff, 8, 9, 0xff, 0xff, 0xff, 0xff, 10, 11, 4, 5, 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);
```

```
30
31      // LFSR Update
32      __m128i mulx = _mm_xor_si128(_mm_slli_epi16(hi, 1),
33        _mm_and_si128(LOLMINI_C, _mm_srai_epi16(hi, 15)));
34      __m128i oddOld = hi;
35      hi = _mm_xor_si128(mulx, _mm_shuffle_epi8(lo, LOLMINI_SIGMA));
36      lo = oddOld;
37
38      //FSM Update
39      state[2] = _mm_aesenc_si128(state[1], state[2]);
40      state[1] = _mm_aesenc_si128(state[0], state[1]);
41      state[0] = _mm_xor_si128(_mm_xor_si128(hi, intermediate), state[0]);
42      return z;
43    }
44    inline void keyiv_setup(const unsigned char* key, const unsigned char* iv)
45    {
46      __m128i mainKey[2];
47      state[0] = _mm_loadu_si128((__m128i*)iv);
48      state[1] = mainKey[0] = _mm_loadu_si128((__m128i*)(key + 16));
49      state[2] = mainKey[1] = _mm_loadu_si128((__m128i*)key);
50      nfsr = hi = lo = ZERO128;
51
52      for (int i = 0; i < 12; ++i)
53      {
54        __m128i output = keystream();
55        nfsr = _mm_xor_si128(nfsr, output);
56        hi = _mm_xor_si128(hi, output);
57      }
58      state[0] = _mm_xor_si128(state[0], mainKey[0]);
59      hi = _mm_xor_si128(hi, mainKey[1]);
60    }
61  };
62
63  //LOL-DOUBLE
64  struct Loldouble
65  {
66    __m256i hi, lo;
67    __m128i state[4];
68    __m128i nfsr[2];
69
70    inline __m256i keystream(void)
71    {
72      // Keystream bits
73      __m128i tmp1 = _mm_aesenc_si128(state[1], ZERO128);
74      __m128i tmp3 = _mm_aesenc_si128(state[3], ZERO128);
75      __m128i z0 = _mm_xor_si128(tmp1, nfsr[0]);
76      __m128i z1 = _mm_xor_si128(tmp3, nfsr[1]);
77
78      //N update
79      nfsr[0] = _mm_aesenc_si128(nfsr[0], _mm256_castsi256_si128(lo));
80      nfsr[1] = _mm_aesenc_si128(nfsr[1], _mm256_extracti128_si256(lo, 1));
81
82      // LFSR Update
83      __m256i mulx = _mm256_xor_si256(_mm256_slli_epi16(hi, 1),
84        _mm256_and_si256(LOLDOUBLE_C, _mm256_srai_epi16(hi, 15)));
85      __m256i oddOld = hi;
86      hi = _mm256_xor_si256(
87        //_mm256_shuffle_epi8(lo, LoldoubleSigma)
88        shuffle256(lo), mulx);
89      lo = oddOld;
90
91      // FSM Update
92      state[1] = _mm_aesenc_si128(state[0], state[1]);
93      state[0] = _mm_xor_si128(
94        _mm_xor_si128(tmp3, state[0]),
95        _mm256_castsi256_si128(hi));
```

```
96      state[3] = _mm_aesenc_si128(state[2], state[3]);
97      state[2] = _mm_xor_si128(
98        _mm_xor_si128(tmp1, state[2]),
99        _mm256_extracti128_si256(hi, 1));
100     return _mm256_set_m128i(z0, z1);
101   }
102   inline void keyiv_setup(const unsigned char* key, const unsigned char* iv)
103   {
104     __m256i mainKey = _mm256_loadu_si256((__m256i*)key);
105     hi = lo = ZERO256;
106     nfsr[0] = nfsr[1] = ZERO128;
107     state[2] = _mm256_castsi256_si128(mainKey);
108     state[3] = _mm256_extracti128_si256(mainKey, 1);
109     state[0] = _mm_load_si128((__m128i*)iv);
110     state[1] = _mm_load_si128((__m128i*)(iv + 16));
111     for (int i = 0; i < 12; ++i)
112     {
113       __m256i output = keystream();
114       hi = _mm256_xor_si256(hi, output);
115       nfsr[0] = _mm_xor_si128(nfsr[0], _mm256_castsi256_si128(output));
116       nfsr[1] = _mm_xor_si128(nfsr[1], _mm256_extracti128_si256(output, 1));
117     }
118     hi = _mm256_xor_si256(hi, mainKey);
119   }
120 };
```

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

```
1  #include <emmintrin.h>
2  #include <immintrin.h>
3  #include <smmintrin.h>
4  #include <wmmintrin.h>
5  const __m256i lol_shu16 = _mm256_set_epi8(23, 22, 13, 12, 31, 30,29, 28, 5,
       4, 17, 16, 1, 0, 19, 18,
6    9, 8, 15, 14,21, 20, 27, 26, 3, 2, 11,10, 25, 24, 7, 6);
7  const __m256i ZERO256 = _mm256_setzero_si256();
8  #define LFSR_DOUBLE(H,L)\
9  { F = _mm256_xor_si256(_mm256_slli_epi16(H, 1),_mm256_and_si256(lol_mul16,
       _mm256_srai_epi16(H, 15)));\
10   F = _mm256_xor_si256( _mm256_permutexvar_epi8(lol_shu16,L),F);\
11   L = H;\
12   H = F;}
13 #define AES_ENC_256(C,M) C= _mm256_aesenc_epi128(M,ZERO256);
14 #define AES_ENC_KEY_256(C,M,R) C= _mm256_aesenc_epi128(M,R);
15
16
17 struct LOL_DOUBLE
18 {
19   __m256i F, H, L, G, S0, S1, Z, N;
20   inline __m256i keystream(void)
21   {
22     AES_ENC_256(G, S1);
23     Z = _mm256_xor_si256(G, N);
24
25     AES_ENC_KEY_256(N, N, L);
26     LFSR_DOUBLE(H, L);
27
28     AES_ENC_KEY_256(S1, S0, S1);
29     S0 = _mm256_xor_si256(S0, _mm256_xor_si256(H, _mm256_permute4x64_epi64(G
       , 0x4e)));
30
31     Z = _mm256_permute4x64_epi64(Z, 0x4e);
32     return Z;
33   }
34
35   inline void keyiv_setup(const unsigned char* key, const unsigned char* iv)
```

```
36   {
37
38     S1 = _mm256_loadu_si256((___m256i*)key);
39     S0 = _mm256_loadu_si256((___m256i*)iv);
40
41     S1 = _mm256_permute2x128_si256(S0, S1, 0x31);
42     S0 = _mm256_permute2x128_si256(S0, S1, 0x20);
43
44     //round 1
45     AES_ENC_256(G, S1);
46     Z = _mm256_permute4x64_epi64(G, 0x4e);
47
48     AES_ENC_KEY_256(N, ZERO256, Z);
49     L = ZERO256;
50     H = Z;
51
52     AES_ENC_KEY_256(S1, S0, S1);
53     S0 = _mm256_xor_si256(S0, Z);
54
55     //round 2-12
56     for (int i = 0; i < 11; i++)
57     {
58       AES_ENC_256(G, S1);
59       Z = _mm256_xor_si256(G, N);
60       Z = _mm256_permute4x64_epi64(Z, 0x4e);
61
62       AES_ENC_KEY_256(N, N, L);
63       N = _mm256_xor_si256(N, Z);
64
65       LFSR_DOUBLE(H, L);
66
67       AES_ENC_KEY_256(S1, S0, S1);
68       S0 = _mm256_xor_si256(S0, _mm256_xor_si256(H, _mm256_permute4x64_epi64
     (G, 0x4e)));
69
70       H = _mm256_xor_si256(H, Z);
71     }
72
73     //reload
74     H = _mm256_xor_si256(H, _mm256_loadu_si256((___m256i*)key));
75   }
76 };
```

Listing 2: Implement LOL-DOUBLE with AVX-512