Sablier v1

Designers: Bin Zhang, Zhenqing Shi, Chao Xu, Yuan Yao, Zhenqi Li
Submitter: Bin Zhang, Zhenqing Shi, Chao Xu, Yuan Yao, Zhenqi Li
Trusted Computing and Information Assurance Lab,
Institute of Software, Chinese Academy of Sciences
{zhangbin,shizhenqing,xuchao,yaoyuan,lizhenqi}@tca.iscas.ac.cn

15th-March-2014

Contents

List of Figures

List of Tables

Introduction

Sablier v1 is a hardware-efficient stream cipher with bulit-in authentication. Unlike the traditional LFSR-based stream ciphers and the usual nonlinear/linear shift registers combined structure in Grain and Trivium, Sablier adopts a new internal structure to generate the keystream from a 80-bit key and a 80-bit IV. Only bitwise *xor*, bitwise logical *and* and bitwise intra-word rotation are used in Sablier v1. It can be efficiently implemented in constrained hardware environments and the encryption speed is expected to be 16 times faster than Trivium in hardware. Compared to Grain-128a, the authentication mechanism has the feature that the authentication process will not slow down the encryption process by carefully leak extraction from the internal states. So far, no attack faster than exhaustive key search has been identified.

Chapter 1

Specification

In this section, we describe the stream cipher Sablier v1.

1.1 Parameters

Sablier v1[1] has three parameters: key length, nonce length and tag length. The parameter space is as follows. The key length is 10 bytes, the nonce length is 10 bytes and the tag length is 4-byte. From a 80-bit key and a 80-bit initialization vector, it generates keystream with length up to 2^{64} bits.

1.2 Recommended Parameter Sets

Primary recommended parameter set `Sablierv1`: 10-byte (80-bit) key, 10-byte (80-bit) nonce, 4-byte (32-bit) tag.

1.3 Authenticated Encryption

The inputs to authenticated encryption are a plaintext P, associated data A, a public message number N, i.e., IV, and a key K. The maximum number of bytes in P is at least $65536 = 2^{16}$ bytes. The maximum number of bytes in A is at least 65536 bytes. The number of bytes in N is the nonce length. The number of bytes in K is the key length. There is no secret message number, i.e., the secret message number is empty.

The output of the authenticated encryption is (C, T), where C is an unauthenticated ciphertext C and a tag T. The unauthenticated ciphertext C is obtained by the usual binary additive encryption using the keystream generated by Sablier, shown in Fig.1.1. The total ciphertext length is the number of bytes in P plus the tag length.

The following operations are used in the description.

[1]We use Sablier to denote Sablier v1 hereafter.

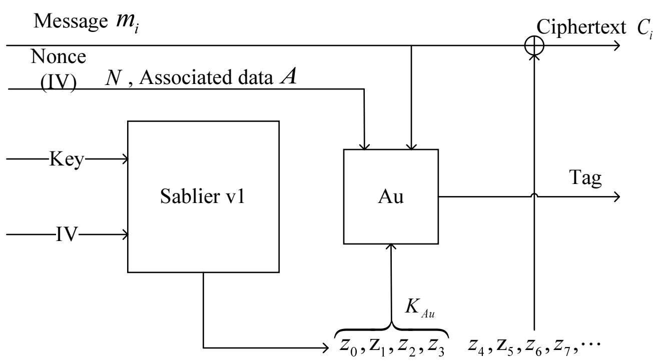

Figure 1.1: The authenticated encryption setting of Sablier

$\oplus$	the 16-bit bitwise exclusive or operator.
$\cdot$	the 16-bit logical and operator.
$\ggg$	the 32-bit right rotation operator. $x \ggg n$ means x being rotated to the right over n bit positions.
$\|$	concatenation.
$\mathbf{1}$	the 16-bit all one vector.
$a[i]$	the i-th least significant bit of a 16-bit word a.

Five registers L_i $(i = 1, \ldots, 5)$ are used in Sablier. The key and the initialization vector of Sablier are denoted by K and IV, respectively. We denote the generated keystream by z.

L_1, L_5:	the two largest registers, each with four 16-bit words, namely $L_{1,i}, L_{5,i}$ with $1 \leq i \leq 4$.
L_2, L_4:	the two second largest registers, each with two 16-bit words, namely $L_{2,i}, L_{4,i}$ with $1 \leq i \leq 2$.
L_3:	the smallest register, consists of one 16-bit word.
K:	the 80-bit key of Sablier, composed of 10 bytes, namely K_i with $0 \leq i \leq 9$.
IV:	the 80-bit initialization vector of Sablier, composed of 10 bytes, namely IV_i with $0 \leq i \leq 9$.
z:	the keystream generated by Sablier. The 16-bit output of the ith step is denoted by z_i. Then $z = z_0, z_1, z_2, \cdots$.

1.4 The Algorithm

Now we describe the stream cipher Sablier, whose structure is depicted in Fig.1.2. In Fig.1.2, the lane is a 16-bit word and χ is just the nonlinear function in Keccak, restricted to 16-bit word. The operation of Sablier can be imaged as the mixing of the sand in a sandglass, or as the shaking of the cocktail in a shaker in the bar. We first mix or shake the container for a large number of times, which is just the initialization phase of Sablier. After that, something will be emitted after each mixing or shaking step as keystream, described in Section 1.4.2.

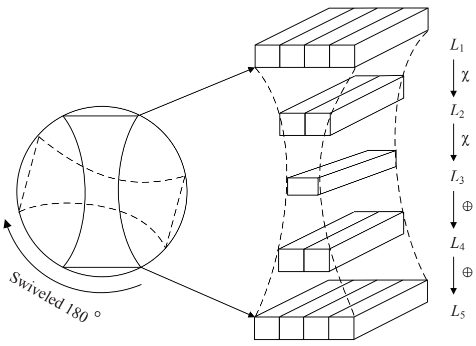

Figure 1.2: The structure of Sablier

1.4.1 Initialization Process (Key/IV Setup)

The initialization process of Sablier consists of loading the key and IV into the registers L_i $(1 \le i \le 5)$ and running the cipher 64 rounds without generating the output.

1. Key and IV loading.

$$\begin{aligned}
\begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \\ L_5 \end{bmatrix} &= \begin{bmatrix} L_{1,1}, L_{1,2}, L_{1,3}, L_{1,4} \\ L_{2,1}, L_{2,2} \\ L_3 \\ L_{4,1}, L_{4,2} \\ L_{5,1}, L_{5.2}, L_{5,3}, L_{5,4} \end{bmatrix} \\
&= \begin{bmatrix} K_0 \parallel IV_0, K_1 \parallel IV_1, K_2 \parallel IV_2, K_3 \parallel IV_3 \\ K_4 \parallel IV_4, K_5 \parallel IV_5 \\ IV_6 \parallel K_3 \oplus IV_7 \\ K_6 \oplus IV_0 \parallel K_6 \oplus IV_1, K_7 \oplus IV_2 \parallel K_7 \oplus IV_3 \\ K_8 \oplus IV_4 \parallel IV_8, K_9 \oplus IV_5 \parallel IV_8, K_8 \oplus IV_6 \parallel IV_9, K_9 \oplus IV_7 \parallel IV_9 \end{bmatrix}
\end{aligned}$$

2. State updating. Run the cipher 128 steps (64 rounds) without generating the output, as shown below.

```
1 : For $0 \le i \le 127$ do
2 :     $L_5 \leftarrow (L_{5,1} \oplus L_{5,2} \oplus L_{5,3} \oplus L_{4,2}, L_{5,1} \oplus L_{5,2}, L_{5,3} \oplus L_{5,4},$
                                        $L_{5,2} \oplus L_{5,3} \oplus L_{5,4} \oplus L_{4,1})$
3 :     $L_4 \leftarrow (L_{4,1} \oplus L_3 \parallel L_{4,2} \oplus L_3) \ggg 5$
4 :     $L_3 \leftarrow L_3 \oplus ((L_{2,1} \oplus \mathbf{1}) \cdot L_{2,2}) \oplus \mathrm{RC}(i)$
5 :     $L_2 \leftarrow (L_{2,1} \oplus ((L_{1,1} \oplus \mathbf{1}) \cdot L_{1,2}), L_{2,2} \oplus ((L_{1,3} \oplus \mathbf{1}) \cdot L_{1,4}))$
6 :     $(L_1, L_2, L_3, L_4, L_5) \leftarrow (L_5, L_4, L_3, L_2, L_1),$
7 : end for
```

where $\mathrm{RC}(i)$ is defined as follows. As in Keccak, $\mathrm{rc}[t] \in \mathrm{GF}(2)$ $(t \ge 0)$ are defined as the output of a binary LFSR:

$$\mathrm{rc}[t] = \left(x^t \bmod x^8 + x^6 + x^5 + x^4 + 1\right) \bmod x \text{ in } \mathrm{GF}(2)[x].$$

Let $\mathrm{RC}(2k)[15] = \mathrm{rc}[4k]$ for $0 \le k \le 63$ and all other values of $\mathrm{RC}(2k)[s]$ $(s \ne 15)$ are zero;

$$\mathrm{RC}(2k+1)[s] = \begin{cases} \mathrm{rc}[4k+1], & \text{if } s = 7 \\ \mathrm{rc}[4k+2], & \text{if } s = 3 \\ \mathrm{rc}[4k+3], & \text{if } s = 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

for $0 \le k \le 63$.

1.4.2 The Keystream Generation Algorithm

After initialization, the cipher outputs a 16-bit word at each step, as shown below. Here each unit is a 16-bit word.

```
The keystream generation
$t = 0$;
repeat until enough keystream bits are generated.
{
  The lower half round :
  1 : $L_5 \leftarrow (L_{5,1} \oplus L_{5,2} \oplus L_{5,3} \oplus L_{4,2}, L_{5,1} \oplus L_{5,2}, L_{5,3} \oplus L_{5,4},$
                                    $L_{5,2} \oplus L_{5,3} \oplus L_{5,4} \oplus L_{4,1})$
  2 : $L_4 \leftarrow (L_{4,1} \oplus L_3 \parallel L_{4,2} \oplus L_3) \ggg 5$
  3 : $L_3 \leftarrow L_3 \oplus ((L_{2,1} \oplus \mathbf{1}) \cdot L_{2,2}) \oplus C_1$
  4 : $L_2 \leftarrow (L_{2,1} \oplus ((L_{1,1} \oplus \mathbf{1}) \cdot L_{1,2}), L_{2,2} \oplus ((L_{1,3} \oplus \mathbf{1}) \cdot L_{1,4}))$
  5 : $(L_1, L_2, L_3, L_4, L_5) \leftarrow (L_5, L_4, L_3, L_2, L_1)$
  The upper half round :
  6 : $L_5 \leftarrow (L_{5,1} \oplus L_{5,2} \oplus L_{5,3} \oplus L_{4,2}, L_{5,1} \oplus L_{5,2}, L_{5,3} \oplus L_{5,4},$
                                    $L_{5,2} \oplus L_{5,3} \oplus L_{5,4} \oplus L_{4,1})$
```

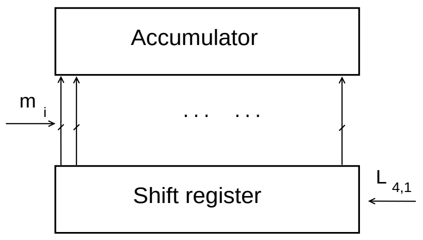

Figure 1.3: The authentication mechanism

```
    7 : $L_4 \leftarrow (L_{4,1} \oplus L_3 \parallel L_{4,2} \oplus L_3) \ggg 5$
    8 : $L_3 \leftarrow L_3 \oplus ((L_{2,1} \oplus \mathbf{1}) \cdot L_{2,2}) \oplus C_2$
    9 : $L_2 \leftarrow (L_{2,1} \oplus ((L_{1,1} \oplus \mathbf{1}) \cdot L_{1,2}), L_{2,2} \oplus ((L_{1,3} \oplus \mathbf{1}) \cdot L_{1,4}))$
    10 : $(L_1, L_2, L_3, L_4, L_5) \leftarrow (L_5, L_4, L_3, L_2, L_1)$
    **Output the keystream** :
    11 : $z_t = L_{2,2} \oplus L_3 \oplus L_{5,3}$
    12 : $t = t + 1$;
}
end-repeat
```

where $C_1 = 0x1735$ and $C_2 = 0x9cb6$ are the hexadecimal expression of the second ten decimals of π.

1.4.3 Authentication

Sablier supports two different modes of operation: with and without authentication. If $IV_0[0] = 1$, then the authentication is activated; if $IV_0[0] = 0$, there is no authentication. The authentication mechanism in our cipher is similar to the one used in Grain-128a [4], but we made some modifications to make it more efficient without a penalty in security. The masking sequence used for authentication is different from those used for encryption, which makes the whole mechanism more efficient, i.e., the authentication will not slow down the encryption process. Here we adopt the leak extraction in [3] to constitute the masking sequence.

Assume that the length of the plaintext P is l_1, denoted by $P = (p_0, p_1, ..., p_{l_1-1})$ and the length of the associated data A is l_2, denoted by $A = (a_1, a_2, ..., a_{l_2-1})$. The whole message used in the authentication is $M = A \parallel IV \parallel P \parallel 1$ of length $l = l_1 + l_2 + 80 + 1$ bits, denoted by $m_1, m_2, ..., m_{l-1}, m_l$. Note that $m_l = 1$ is the padding, which can ensure that M and $M||0$ have different tags. This padding is crucial for the security of the authentication.

Two registers called the accumulator and the shift register are used in Fig.1.3, each of which is 32-bit. The content of the accumulator at time i is denoted by $(r_i^0, r_i^1, ..., r_i^{31})$. The content of the shift register is denoted by

$(s_i, s_{i+1}, ..., s_{i+31})$. The blocks of the masking sequence used for authentication are denoted by $L_{4,1}^4, L_{4,1}^5, ...$ respectively[2], where $L_{4,1}$ is just the state register of 16-bit length in Sablier. The initialization phase of the accumulator is $r_0^i = z_0[i]$, $r_0^{i+16} = z_1[i]$, $0 \leq i \leq 15$ and the initialization phase of the shift register is $s_i = z_2[i]$, $s_{i+16} = z_3[i]$, $0 \leq i \leq 15$, which means that the first 4 keystream blocks are used to initialize the accumulator and the shift register, respectively. Then the accumulator is updated as $r_{i+1}^j = r_i^j \oplus m_i s_{i+j}$, where $0 \leq j \leq 31$ and $0 \leq i \leq l$. The shift register is updated as $s_{i+32} = L_{4,1}^{4+i/16}[i \bmod 16]$.

The final content of the accumulator $r_{l+1}^0, r_{l+1}^1, ..., r_{l+1}^{31}$ is the *tag* which can be used for authentication. The tag bit is denoted by $t_i = a_{l+1}^i$, $0 \leq i \leq 31$. Please refer to Fig.1.3 for a graphical representation of the authentication mechanism. We also support the use of shorter tags of length w $(1 \leq w \leq 31)$, defined as $t_i^{(w)} = t_{32-w+i}$, $0 \leq i \leq w-1$. $t_i^{(w)}$ means the right-most part of the tag in Fig.1.3.

[2]The superscript of the state register represents the time instant in the encryption process hereafter.

Chapter 2

Security Goals

Security Goal	`Sablierv1` bits of secuirty
confidentiality for the plaintext	80
integrity for the plaintext	32
integrity for the associated data	32
integrity for the public message number	32

There is no secret message number in Sablier. The public message number is a nonce, i.e., the IV. The cipher does not promise any integrity or confidentiality if the legitimate key holder uses the same nonce (IV) to encrypt two different (plaintext, associated data) pairs under the same key.

The numbers in the table are actually on different scales: 2^{32} is the expected number of online forgery attempts for a successful forgery, while 2^{80} is the expected number of key guesses to find the secret key. Any successful forgery or successful key guess should be assumed to completely compromise confidentiality and integrity of all messages.

The table also assumes that the legitimate key holder does not approach 2^{64} message bits encrypted under a single key.

Chapter 3

Security Analysis

In this section, we will analyze the security of Sablier with respect to several attacks.

3.1 Period and Time/Memory/Data Tradeoffs

The 208-bit state of Sablier ensures that the period of the keystream is large enough for any practical applications, but the exact value of the keystream period of Sablier is difficult to predict in theory. The average period of the keystream is estimated to be larger than 2^{104}, if we assume that the invertible state updating function of Sablier is random. Besides, the 208-bit size internal state also eliminates the threat of the known form of the time/memory/data tradeoff attacks with respect to 80-bit secuirty.

3.2 Linear Distinguishing Attacks

Here we use the linear sequential circuit approximation (LSCA) method [9] to evaluate the strength of Sablier against linear distinguishing attacks.

3.2.1 Basic Linear Sequential Circuit Approximation

Golić has shown that for a binary keystream generator with M bits of memory whose initial state is chosen uniformly at random, there exists a linear function of at most $M+1$ consecutive output bits which is an unbalanced function of the initial state variables. An effective method is also developed for the linear model determination based on linear sequential circuit approximation of autonomous finite state machines. Some linear function of consecutive output bits exhibits an unbalance to which the adversary can apply the standard chi-square frequency statistical test. The test is successful if and only if the length of the sequence is chosen to be inversely proportional to the square of the correlation coefficient. If the key length is k-bit, the statistical weakness is effective if and only if the correlation coefficient is greater than $2^{-k/2}$. Next, we use Golić's method to extract the linear sequential circuit approximation of Sablier step by step.

In [9], the output function is a boolean function in the linear sequential circuit approximation method, while our algorithm is 16-bit word-oriented, so we introduce a output mask ω to transform the vectorial output function into a boolean function. Let $S_t = (s_{t,1}, s_{t,2}, \cdots, s_{t,208})^T$ be the state vector at time t, and $\omega \cdot z_t$ be the bitwise output sequence after the inner product between the output mask ω and z_t. Now Sablier can be regarded as an autonomous finite state machine defined by

$$\begin{aligned} S_t &= F(S_{t-1}), \ t \geq 1 \\ \omega \cdot z_t &= f_\omega(S_t), \quad t \geq 1 \end{aligned}$$

We first decomposes the output boolean function and each of the boolean functions in the next-state function of the keystream generator into the sum of linear functions and an unbalanced boolean function. Thus we have

$$\begin{aligned} S_t &= AS_{t-1} + \Delta(S_{t-1}), \ t \geq 1 \\ \omega \cdot z_t &= B_\omega S_t, \qquad\qquad\quad t \geq 1 \end{aligned} \tag{1}$$

where S_t is a 208×1 column vector, A is an 208×208 matrix and B_ω is a 1×208 row vector and Δ is a 208×1 noise vector. In order to acquire the parameters of the above the equations, we study the update function in details. We denote the state at time t in 16-bit word as

$$(L^t_{1,1}, L^t_{1,2}, L^t_{1,3}, L^t_{1,4}, L^t_{2,1}, L^t_{2,2}, L^t_3, L^t_{4,1}, L^t_{4,2}, L^t_{5,1}, L^t_{5,2}, L^t_{5,3}, L^t_{5,4})^T.$$

After the state updating, the new state at time $t+1$ is as follow.

$$\begin{bmatrix} L^{t+1}_{1,1} \\ L^{t+1}_{1,2} \\ L^{t+1}_{1,3} \\ L^{t+1}_{1,4} \\ L^{t+1}_{2,1} \\ L^{t+1}_{2,2} \\ L^{t+1}_{3} \\ L^{t+1}_{4,1} \\ L^{t+1}_{4,2} \\ L^{t+1}_{5,1} \\ L^{t+1}_{5,2} \\ L^{t+1}_{5,3} \\ L^{t+1}_{5,4} \end{bmatrix} = \begin{bmatrix} L^t_{1,1} \oplus L^t_{1,2} \oplus L^t_{1,3} \oplus L^t_{1,4} \oplus L^t_{2,2} \\ L^t_{1,1} \oplus L^t_{1,2} \\ L^t_{1,3} \oplus L^t_{1,4} \\ L^t_{1,3} \oplus L^t_{1,4} \oplus L^t_{2,1} \\ (L^t_{1,2} \oplus L^t_{2,1} \oplus L^t_{2,2} \oplus L^t_3) \gg 5 \oplus (L^t_{1,4} \oplus L^t_3) \ll 11 \\ (L^t_{1,2} \oplus L^t_{2,1} \oplus L^t_{2,2} \oplus L^t_3) \ll 11 \oplus (L^t_{1,4} \oplus L^t_3) \gg 5 \\ L^t_3 \oplus L^t_{2,2} \oplus (L^t_{4,2} \oplus L^t_3) \gg 5 \oplus (L^t_{4,1} \oplus L^t_3) \ll 11 \\ (L^t_{4,2} \oplus L^t_3) \ll 11 \oplus (L^t_{4,1} \oplus L^t_3) \gg 5 \\ L^t_{4,1} \oplus L^t_{5,2} \oplus (L^t_{4,2} \oplus L^t_3) \gg 5 \oplus (L^t_{4,1} \oplus L^t_3) \ll 11 \\ L^t_{4,2} \oplus L^t_{5,1} \oplus L^t_{5,2} \oplus L^t_{5,3} \\ L^t_{5,1} \oplus L^t_{5,2} \\ L^t_{5,3} \oplus L^t_{5,4} \\ L^t_{4,1} \oplus L^t_{5,2} \oplus L^t_{5,3} \oplus L^t_{5,4} \end{bmatrix} \oplus$$

$$\begin{bmatrix} L_{1,3}^t \cdot L_{1,4}^t \\ 0 \\ 0 \\ L_{1,1}^t \cdot L_{1,2}^t \\ (L_{1,1}^t \cdot L_{1,2}^t \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus C_1) \gg 5 \oplus (L_{1,3}^t \cdot L_{1,4}^t \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus C_1) \ll 11 \\ (L_{1,1}^t \cdot L_{1,2}^t \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus C_1) \ll 11 \oplus (L_{1,3}^t \cdot L_{1,4}^t \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus C_1) \gg 5 \\ L_3^{t+1} \\ (L_{5,1}^t \oplus L_{5,2}^t) \cdot (L_{4,2}^t \oplus L_{5,3}^t) \\ (L_{5,3}^t \oplus L_{5,4}^t) \cdot (L_{4,1}^t \oplus L_{5,2}^t) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

where $L_3^{t+1} = C_1 \oplus C_2 \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus ((L_{4,2}^t \oplus L_3^t) \gg 5 \oplus (L_{4,1}^t \oplus L_3^t) \ll 11) \cdot ((L_{4,2}^t \oplus L_3^t) \ll 11 \oplus (L_{4,1}^t \oplus L_3^t) \gg 5)$. Thus the matrices A and B_ω are as follows.[1]

$$A = \begin{bmatrix} e_1 \oplus e_{17} \oplus e_{33} \oplus e_{49} \oplus e_{81} \\ e_2 \oplus e_{18} \oplus e_{34} \oplus e_{50} \oplus e_{82} \\ e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51} \oplus e_{83} \\ \cdot \\ \cdot \\ \cdot \\ e_{128} \oplus e_{176} \oplus e_{192} \oplus e_{208} \end{bmatrix}$$

$$B_\omega = \omega \cdot \begin{bmatrix} e_{81} \\ e_{82} \\ \cdot \\ \cdot \\ \cdot \\ e_{96} \end{bmatrix} \oplus \omega \cdot \begin{bmatrix} e_{97} \\ e_{98} \\ \cdot \\ \cdot \\ \cdot \\ e_{112} \end{bmatrix} \oplus \omega \cdot \begin{bmatrix} e_{177} \\ e_{178} \\ \cdot \\ \cdot \\ \cdot \\ e_{192} \end{bmatrix}.$$

Using the decomposition in Eq.(1), it then follows that S_t satisfies the following expressions.

$$S_t = A^t S_0 + \sum_{l=0}^{t-1} A^l \Delta_{t-l}, \quad t \geq 1$$

The degree of the minimal polynomial of A is $m = 208$, which is denoted by

[1] e_i denotes the ith row of the 208×208 identity matrix.

$\varphi(x) = \sum_{k=0}^{m} \phi_k x^k$, precisely

$$\begin{aligned}\varphi(x) =& x^{208} + x^{192} + x^{188} + x^{187} + x^{186} + x^{185} + x^{184} + x^{183} + x^{182} + \\ & x^{181} + x^{178} + x^{177} + x^{175} + x^{174} + x^{170} + x^{168} + x^{165} + x^{163} + \\ & x^{159} + x^{158} + x^{157} + x^{156} + x^{155} + x^{152} + x^{150} + x^{149} + x^{146} + \\ & x^{144} + x^{143} + x^{140} + x^{139} + x^{137} + x^{134} + x^{131} + x^{129} + x^{128} + \\ & x^{125} + x^{124} + x^{123} + x^{122} + x^{121} + x^{120} + x^{113} + x^{110} + x^{106} + \\ & x^{105} + x^{103} + x^{100} + x^{95} + x^{94} + x^{90} + x^{89} + x^{88} + x^{87} + x^{85} + \\ & x^{84} + x^{82} + x^{81} + x^{80} + x^{79} + x^{78} + x^{76} + x^{74} + x^{73} + x^{72} + \\ & x^{70} + x^{68} + x^{66} + x^{64} + x^{63} + x^{60} + x^{59} + x^{52} + x^{45} + x^{42} + x^{41} + \\ & x^{34} + x^{33} + x^{32} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{20} + x^{19} + 1\end{aligned}$$

Since $\sum_{k=0}^{m} \varphi_k A^k = 0$, it follows that

$$\begin{aligned}\sum_{k=0}^{m} \varphi_k S_{t+k} &= \sum_{k=0}^{m} \varphi_k (A^{t+k} S_0 + \sum_{l=0}^{t+k-1} A^l \Delta_{t+k-l}) \\ &= \sum_{k=0}^{m} \varphi_k A^{t+k} S_0 + \sum_{k=0}^{m} \varphi_k \sum_{l=0}^{t+k-1} A^l \Delta_{t+k-l} \\ &= \sum_{k=0}^{m} \varphi_k \sum_{l=0}^{t+k-1} A^l \Delta_{t+k-l} \\ &= \sum_{\tau=0}^{m} \sum_{r=0}^{m-\tau} \varphi_{r+\tau} A^r \Delta_{t+\tau}\end{aligned}$$

Multiplying both the most right and the most left sides of the above equation by B_ω, we got

$$\sum_{k=0}^{m} \varphi_k \omega \cdot z_{t+k} = \varphi_k B_\omega S_{t+k} = \sum_{\tau=0}^{m} \sum_{r=0}^{m-\tau} \varphi_{\tau+r B_\omega A^r \Delta_{t+\tau}} \tag{3}$$

For brevity, let $C_\tau = \sum_{r}^{m-\tau} \varphi_{r+\tau} B_\omega A^r$. Thus, with A and B_ω, we can easily compute the vectors C_τ for $\tau = 0, 1, \cdots, 208$. Note that there are some independence among the noise components of vector Δ_t in (2), i.e., the non-balanced parts of L_{21}^{t+1} and L_{22}^{t+1} do not satisfy the independent assumption, which results in the smaller correlation coefficients. Since we just want to get the lower bound of (3), the noise vector is transmitted into

$$\Delta_t = \begin{bmatrix} L_{1,3}^t \cdot L_{1,4}^t \\ 0 \\ 0 \\ L_{1,1}^t \cdot L_{1,2}^t \\ 0 \\ 0 \\ L_3^{t+1} \\ (L_{5,1}^t \oplus L_{5,2}^t) \cdot (L_{4,2}^t \oplus L_{5,3}^t) \\ (L_{5,3}^t \oplus L_{5,4}^t) \cdot (L_{4,1}^t \oplus L_{5,2}^t) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where $L_3^{t+1} = C_1 \oplus C_2 \oplus L_{2,1}^t \cdot L_{2,2}^t \oplus ((L_{4,2}^t \oplus L_3^t) \gg 5 \oplus (L_{4,1}^t \oplus L_3^t) \ll 11) \cdot ((L_{4,2}^t \oplus L_3^t) \ll 11 \oplus (L_{4,1}^t \oplus L_3^t) \gg 5)$. There are three types nonlinear expressions here, i.e., $x_1x_2, (x_1+x_3)(x_2+x_3), (x_1+x_2)(x_3+x_4)$, which have the same correlation coefficients equal to $1/2$. Then we have the correlation coefficients vector for each bit of state S_{t+1}, i.e.,

$$\begin{bmatrix} (\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2}) \\ (0, 0, \cdots, 0) \\ (0, 0, \cdots, 0) \\ (\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2}) \\ (0, 0, \cdots, 0) \\ (0, 0, \cdots, 0) \\ (\frac{1}{4}, \frac{1}{4}, \cdots, \frac{1}{4}) \\ (\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2}) \\ (\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2}) \\ (0, 0, \cdots, 0) \\ (0, 0, \cdots, 0) \\ (0, 0, \cdots, 0) \\ (0, 0, \cdots, 0) \end{bmatrix}. \qquad (4)$$

In order to compute the correlation coefficient of $\sum_{k=0}^{208} \varphi_k \omega \cdot z_{t+k}$, we just count the number of 1s in C_τ for $\tau = 0, 1, \cdots, 208$ corresponding to the non-zero components of (4). In our experiments, we have searched all the output masks $\omega \in \mathrm{GF}(2^{16})$ and found that when $\omega = 0x4008$, we have the best correlation coefficient 2^{-9493}.

3.2.2 Linear Equation with Greater Correlation Coefficient

Given a linear equation of consecutive output bits of the form in (3), linear equations with greater correlation coefficients may be found using the generating

Table 3.1: A sample result of differential cryptanalysis

rounds	t	p	rounds	t	p
1	206	0	13	118	2^{-704}
2	204	0	14	114	2^{-672}
3	205	0	15	101	2^{-571}
4	206	0	16	86	2^{-461}
5	202	0	17	73	2^{-370}
6	201	0	18	62	2^{-297}
7	197	0	19	48	2^{-211}
8	186	0	20	27	2^{-96}
9	179	0	21	18	2^{-54}
10	163	0	22	2	0.188382
11	147	2^{-950}	23	6	0.000543607
12	136	2^{-854}	24	1	0.368679

function method. For the minimal polynomial $\varphi(x)$, which has the hamming weight 87, we need to find the low weight multiple polynomial $h(x) = f(x)\varphi(x)$. Therefore, we should multiply both sides of (3) by an appropriate polynomial $f(x)$ to obtain a new linear approximation. We have made a thorough search over all the polynomials $f(x)$ with non-zero constant term and with the degree up to 29. The maximum correlation coefficient, among all those polynomials, is smaller then the original correlation coefficient. In the experiments, we found that the lowest weight of $h(x)$ is 75, but the total correlation is smaller than 2^{-10000}. In summary, it is safe to conclude that Sablier is extremely strong against linear distinguishing attacks.

3.3 Differential Cryptanalysis

In order to investigate the immunity of Sablier against differential attacks, we introduce a single bit difference at each internal state position and try to trace the propagation of this difference. We gather the difference biases after several number of initialization rounds and try to distinguish it from the purely random case. We statistically test whether a output difference bit is uniformly distributed or not, i.e., the null hypothesis H_0 is that the considered bit is indeed uniformly distributed and the alternative hypothesis H_1 is the opposite. Table 3.1 is a sample result when $n = 2^{10}$ and when the difference is introduced at $(L_{1,1})[0]$. Precisely, let n be the number of samples, m be the number of times a random output bit is 1 and θ be the event that $m < m_l$ or $m > m_h$ for the given thresholds m_l and m_h. If H_0 is true, then $p_0 = 1 - \sum_{i=m_l}^{m_h} \binom{n}{i}(\frac{1}{2})^n$, thus the expected occurrences of θ during 208, which is the size of Sablier's state, experiments is $t_0 = 208 \cdot p_0$. Thus, m_l and m_h can be determined by the relation $t_0 \leq 1$, i.e., only one output difference bit would be considered as non-random on average when H_0 is true. Assume that θ occurs t times during the experiments and $p = \sum_{i=t}^{208} \binom{n}{t} p_0^t (1-p_0)^{208-t} \leq 0.05$, then we detect a small probability event and conclude that Sablier is non-random.

Our experiments show that the probability p is never smaller than 0.05 after 23 rounds of initialization for any position of the internal state. In other words, Sablier is non-distinguishable with the purely random case after 23 rounds of initialization with respect to the single bit differential cryptanalysis.

3.4 Cube Attacks and Variants

Cube attacks, formally introduced by Dinur and Shamir [6], is a generic key extraction technique exploiting the simple algebraic structure of some output bits after a reasonable size of cube summation. The success of cube attacks highly depends on the sparsity of the superpoly. Accordingly, we estimate the degree, the referenced variables and the monomial density for each keystream bit and each internal state bit, and take them as the immunity indices against cube type attacks [1, 6, 7]. Since a fully symbolic calculation of the algebraic normal form (ANF) is impossible after a few number of rounds, here we ignore the possible cancellations between the monomials. Denote the three quantity by $\deg(t), \mathrm{ref}(t)$ and $\mathrm{den}(t)$ for a monomial term t respectively, then $\mathrm{ref}(t_1 \cdot t_2)$, $\deg(t_1 \cdot t_2)$ and $\mathrm{den}(t_1 \cdot t_2)$ are estimated as $\mathrm{ref}(t_1) \cup \mathrm{ref}(t_2)$, $\min(|\mathrm{ref}(t_1 t_2)|, \deg(t_1) + \deg(t_2))$ and $\mathrm{den}(t_1)\mathrm{den}(t_2)$ respectively and $\mathrm{ref}(t_1 + t_2)$, $\deg(t_1 + t_2)$ and $\mathrm{den}(t_1 + t_2)$ are estimated as $\mathrm{ref}(t_1) \cup \mathrm{ref}(t_2)$, $\min(|\mathrm{ref}(t_1 t_2)|, \max(\deg(t_1), \deg(t_2)))$ and $\mathrm{den}(t_1) + \mathrm{den}(t_2)$ respectively.

Our experiments show that the degree and the referenced variables of each keystream bit reach their maximums after 16 rounds of initialization, the referenced variables of each internal state bit reach the maximum after 20 rounds of initialization and the degree of each internal state bit reaches the maximums after 21 rounds of initialization. The monomial density of each keystream bit reaches the maximum value which can be represented by a 64-bit double floating-point variable after 9 rounds, whereas the number is 12 for each internal state bit. Thus, we conclude that the full 64-round Sablier is secure against the current form of cube type attacks.

3.5 Guess and Determine Attacks

First note that in Sablier, if the cipher is run from a certain state for 4 successive rounds, we can get 5 keystream words, namely

$$z_i = L_{2,2}^i \oplus L_3^i \oplus L_{5,3}^i, \quad \text{for } \ i = t, t+1, t+2, t+3, t+4.$$

Then we can recover all the state words with a complexity of 2^{160} following the process shown in Table 3.2, where 0.5 represents a half round. In the guess-and-determine path in Table 3.2, we have considered the effect of the redundant equations, i.e., the equations which will filter out some wrong candidates and thus reduce the complexity in the following stages, and the information leakage

Table 3.2: A guess-and-determine path

guessed	restored	complexity
$L_{1,1}^{t}, L_{4,1}^{t+2}, L_{5,3}^{t+3}, L_{1,2}^{t+2}, L_{5,2}^{t+2}$	$L_{5,4}^{t+3}, L_{5,3}^{t+4}$	2^{80}
$L_{1,3}^{t}$	$L_{4,2}^{t+0.5}$	2^{96}
$L_{1,4}^{t}$	$L_{2,1}^{t}, L_{1,3}^{t+1}$	2^{112}
$L_{1,2}^{t+1}$	$L_{1,2}^{t}, L_{4,1}^{t+0.5}, L_{1,1}^{t+1}, L_{1,4}^{t+1}, L_{1,3}^{t+2}$	2^{128}
$L_{5,2}^{t+1}$	$L_{5,1}^{t+1}, L_{2,1}^{t+0.5}, L_{4,1}^{t+1}$	2^{144}
$L_{2,1}^{t+0.5} L_{2,2}^{t+0.5}$	$L_{5,3}^{t+1}$	2^{152}
$L_{5,3}^{t+1} L_{5,4}^{t+1}$	$L_{5,4}^{t+1}, L_{3}^{t+0.5}, L_{2,1}^{t+1}, L_{2,2}^{t+1}, L_{3}^{t+1}, L_{4,2}^{t+1}, L_{4,2}^{t+4}, L_{5,1}^{t+2}, L_{5,3}^{t+2}, L_{5,4}^{t+2}, L_{2,1}^{t+1.5}, L_{2,2}^{t+1.5}, L_{3}^{t+1.5}, L_{4,1}^{t+1.5}, L_{4,2}^{t+1.5}, L_{1,1}^{t+2}, L_{1,4}^{t+2}, L_{2,1}^{t+2}, L_{2,2}^{t+2}, L_{3}^{t+2}, L_{4,2}^{t+2}, L_{5,1}^{t+3}, L_{5,2}^{t+3}, L_{2,1}^{t+2.5}, L_{2,2}^{t+2.5}, L_{3}^{t+2.5}, L_{4,1}^{t+2.5}, L_{4,2}^{t+2.5}, L_{1,1}^{t+3}, L_{1,2}^{t+3}, L_{1,3}^{t+3}, L_{1,4}^{t+3}, L_{2,1}^{t+3}, L_{2,2}^{t+3}, L_{3}^{t+3}, L_{4,1}^{t+3}, L_{4,2}^{t+3}, L_{4,1}^{t+4}, L_{5,1}^{t+4}, L_{5,2}^{t+4}, L_{5,4}^{t+4}, L_{2,1}^{t+3.5}, L_{2,2}^{t+3.5}, L_{3}^{t+3.5}, L_{4,1}^{t+3.5}, L_{4,2}^{t+3.5}, L_{1,1}^{t+4}, L_{1,2}^{t+4}, L_{1,3}^{t+4}, L_{1,4}^{t+4}, L_{2,1}^{t+4}, L_{2,2}^{t+4}, L_{3}^{t+4}$	2^{160}
$L_{2,2}^{t+0.5}$	$L_{2,2}^{t}, L_{3}^{t}, L_{5,3}^{t}, L_{5,1}^{t}, L_{5,2}^{t}, L_{4,2}^{t}, L_{5,4}^{t}, L_{4,1}^{t}$	2^{160}

provided by some nonlinear equations in the attack, e.g., if $a = bc$, then the complexity of guessing a conditioned on b or c is 2^8 instead of 2^{16} on average.

Another approach to launch a guess-and-determine attack on Sablier is to collect sufficiently many algebraic equations which associate the internal state words with the keystream words and try to solve these equations by some algebraic methods. Here is just one try. To recover 13 internal state words at time i, we first express the keystream words z_j's as polynomials of those state words, the degrees of which are listed in Table 3.3. For convenience, we ignore the impact of the rotational shift in step 2 and 7 in the keystream generation, which will introduce some new variables otherwise. Then we test and find out that whatever we choose at most 5 words to be guessed, we can only get an algebraic equation system with at least 8 unknown variables and degree of at least 8. Meanwhile, these equations systems are not sparse, and it seems hard to solve the system out in a reasonable time. If the impact of the rotational shift in step 2 and 7 is considered, or we express Sablier in a bitwise way, the equation systems will be more complex and be more difficult to solve.

Maybe what we found is not the best guess and determine attack, we feel

Table 3.3: Degree of z_j's

j	$\leq i-1$	$i-1$	i	$i+1$	$i+2$	$i+3$	$i+4$	$\geq i+5$
Degree of z_j	≥ 12	7	1	2	4	7	10	≥ 12

that it is infeasible to launch a guess and determine attack on Sablier with a complexity less than 2^{80}.

3.6 Rotational Cryptanalysis

In the rotational cryptanalysis, the adversary tries to investigate the propagation of the rotational relations through the cryptographic transformations. The first step of such an attack is to define a rotational pair of the input state. For example, in [12], the state of the block cipher Threefish is defined as a word I, and a rotational counterpart of I is created by bitwise rotation operation of itself; in [13], a rotational pair of the hash function Keccak is created by bitwise rotation operation of the corresponding lane in the internal state.

Note that Sablier is a stream cipher which have the output function to generate the keystream and due to the impact of the bitwise rotation of the concatenations in step 2 and 7, it seems hard to define a rotational pair of the internal state and less possible to observe some kind of rotational difference in the keystream word. Further, there are constants both in the initialization and in the keystream generation phase, which will make the compensated vectors used to facilitate the rotational attack more difficult to find than the case in a block cipher.

Now let us consider a modified version of Sablier, where the bitwise rotations in step 2 and 7 are applied to the two internal state words separately. Thus we can create a rotational pair by the bitwise rotation operation of each state word. Here we use some techniques in [13] to analyze Sablier.

We first introduce three types of rotational relations between the bits. For a bit x and its corresponding rotational bit y, if $x = y$ with probability 1, we call them having an equal relation(E); if $x = y$ with probability 0, we call them having an opposite relation(O); if $x = y$ with probability $\frac{1}{2}$, we call them having an unknown relation(U). In Sablier, there are three basic operations: bitwise xor($\oplus$), bitwise and($\cdot$) and bitwise rotation($\ggg$). The bitwise rotation operation preserves the rotational relations and the other two basic bitwise operations propagate the rotational relations as follows. Another impact of

$\oplus$	E	O	U
E	E	O	U
O	O	E	U
U	U	U	U

$\cdot$	E	O	U
E	E	U	U
O	U	U	U
U	U	U	U

rotational relation is xoring with a constant. For a bit x and its corresponding rotational bit y, if x and y is xored with the same constant, their rotational relation is preserved; if only x (or y) is xored with an '1', their rotational relation is changed as $E \to O$, $O \to E$, and $U \to U$.

For the cryptanalysis point of view, the more rotational pairs with the rotational relations of E and O exist in the internal state, the more advantages he can gain to mount an attack. We have tested for the modified version of Sablier and found that after 24 steps of initialization steps, far from the 128 steps, all the rotational pairs have an unknown relation. Thus, we feel that the original full-round version of Sablier will be even more strong against the rotational cryptanalysis.

3.7 Security of the Authenticated Mechanism

Now we consider the security of the authenticated mechanism of Sablier.

3.7.1 Toeplitz Matrices

Toeplitz matrices are characterized by having fixed diagonals. More precisely, each left-to-right diagonal is fixed, i.e., if $k-i{=}l-j$ for any indices $1 \leq i, k \leq n$, $1 \leq j, l \leq m$, then $A_{i,j} = A_{k,l}$. Note that an $n \times m$ Toeplitz matrix is fully described by its first column and first row, i.e., by $n+m-1$ elements.

Any given sequence s of $n+m-1$ bits is associated with an $n \times m$ Toeplitz matrix T_s. We map the first n elements of s into the first column of T_s starting from the bottom, i.e., $T_s(n,1) = s_1, ..., T_s(1,1) = s_n$ and then the last m bits of s into the first row of T_s, i.e., $T_s(1,1) = s_n, T_s(1,2) = s_{n+1}, ..., T_s(1,m) = s_{n+m-1}$. We say that s generates T_s.

Toeplitz matrices of dimension $n \times m$ can be used to hash message of length m by multiplying the message, seen as a column vector, by the matrix. The resultant hash value has length n. See the following equation.

$$\begin{bmatrix} S_{1,1} & S_{1,2} & \dots & S_{1,m} \\ S_{2,1} & S_{2,2} & \dots & S_{2,m} \\ \vdots & \vdots & & \vdots \\ S_{n,1} & S_{n,2} & \dots & S_{n,m} \end{bmatrix} \cdot \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_m \end{bmatrix} = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{bmatrix} \tag{3.1}$$

Where $S_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ is the bit in Toeplitz matrix, M_j is the bit of a message and H_i is the bit of the resultant hash value. For each M_j, if it is zero we do nothing and if it is one we update $H \leftarrow H \oplus S_j$, where S_j is the jth column of the matrix.

It is well-known that the family of Toeplitz matrices T_s with s chosen at random constitutes a strongly universal family of hash functions [20]. This scheme has been improved by [17] by using LFSR to construct the Toeplitz matrix, which only requires n-bit random bits instead of $n+m-1$ bits. The resultant family is still almost universal, or ϵ-balanced.

3.7.2 ϵ-biased Distribution Sequences

ϵ-biased distribution were introduced by Naor et.al. [21] as a tool for constructing small sample spaces, or more generally as a tool for replacement of truly random sequences which is more compact and easier to generate sequences.

Definition 1 *Let S be a distribution on sequences of length l. Let $\langle \alpha, s \rangle$ denote the inner product modulo 2 between $\alpha \in \{0,1\}^l$ and $s \in \{0,1\}^l$. Then,*

1. *S is said to pass the linear test α with bias ϵ if $|Prob(\langle \alpha, s \rangle = 1) - 1/2| \leq \epsilon$ (the probability taken over the choice of s from the distribution S).*
2. *S is said to be an ϵ-biased distribution if it passes all linear tests $\alpha \neq 0$ with bias ϵ.*

The case $\epsilon = 0$ corresponds to the uniform distribution. Therefore, ϵ-biased distribution can be viewed as approximations to the uniform distribution. However, even for very small ϵ, there may be significant distinctions between ϵ-biased and uniform distributions.

3.7.3 Substitution Attacks

The substitution attack is a powerful attack [22], in which the attacker tries to replace a legitimate message-tag pair (m, t) with another pair (m', t') of his choice and the receiver cannot notice this forged pair. Here we will consider this kind of attack as the main threat to the authentication scheme and analyze the success probability of this attack.

The authentication scheme used in Sablier can be regarded as a $L \times t$ Toeplitz matrix based hashing scheme [17, 18], where L is the message length and t is the tag length. It has been proved in [18] that if the Toeplitz matrix is generated by a sequence selected from an ϵ-biased distribution S, then the hashing scheme that uses multiplication of the message by the Toeplitz matrix as the message hash is $(\frac{1}{2^t} + \epsilon)$-opt-secure, where n is the length of the tag size. This means that the probability of a successful substitution attack is lower than or equal to $\frac{1}{2^t} + \epsilon$. This bound was corrected and extended to $\frac{1}{2^t} + 2 \cdot \epsilon$ in [5]. Here ϵ can be regarded as a measurement of the randomness in the keystream sequence used for authentication, from the previous linear cryptanalysis in Section 3.2, we know that $\epsilon \ll 2^{-32}$ for Sablier. We can thus claim that the success probability of the substitution attack is bounded by approximately 2^{-32} and the best attack is to exhaustively guess the tag for each message.

Since the sequences used to construct the Toeplitz matrix are extracted from the state $L_{4,1}$ during the keystream generation phase and it is hard to evaluate the ϵ theoretically, we have conducted a random experiment to evaluate ϵ according to the definition of ϵ-biased distribution sequences.

From Fig 3.1, the bias ϵ decreases with the growth of the sequence length. Note that these results are better than that in the Figure 3 of [5], e.g., give sequence length $L = 20$, the bias in Fig 3.1 is lower than 2^{-9} while the biases

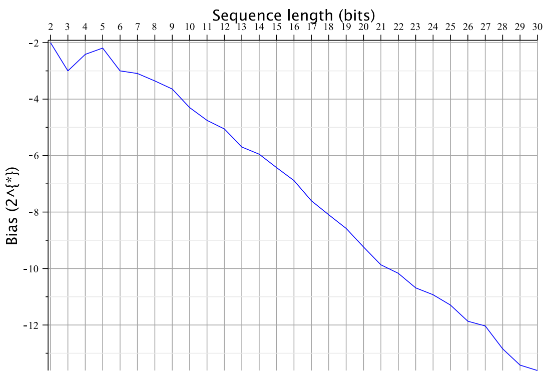

Figure 3.1: Evaluation of ϵ

of the construction in [5] and [17] are approximately 2^{-4} and 2^{-5} respectively, which demonstrates that the masking sequences used for authentication in Sablier behave more randomness.

From the study in [5, 15], avoiding reuse of the key-IV pair is crucial to the security of the authentication. An attacker who is able to tweak a message-tag pair and have it accepted, this happens with probability 2^{-w}, will be able to perform subsequent forgeries with probability 1 if the key-IV pair is reused.

3.7.4 Comparisons

The 3GPP algorithm 128-EIA3 [10], which uses stream cipher ZUC [11], utilizes two different keys with similar IVs or the equal ones to generate two keystream sequences: one for encryption and one for authentication. Hence, one needs to utilize two implementations of ZUC or an expansive buffering. While the authentication mechanism used in Sablier is more compact from a hardware point of view as both the authentication and encryption share a single instance of the cipher.

The stream cipher Helix [16] and a tweaked cipher Phelix (a submission to eSTREAM), which use the message as part of the input to the keystream generator, allow message authentication for free. After encrypting the plaintext, one can generate a constant number of additional stream blocks and output those blocks as an authenticator of the plaintext. However, incorporating the plaintext into the generator takes time for each block and the attack can have some limited control to the keystream generator, which is a security threat. Thus, we do not follow the design criteria of Helix to design our authentication mechanism.

Chapter 4

Features

Sablier has the following useful features.

4.1 A New Stream Cipher Structure

Sablier is designed according to a new internal state structure in stream cipher design, called the Sablier structure. In general, this structure can be imaged as a sandglass or a cocktail shaker. The 3-dimensional sandglass has two connected vertical glass bulbs allowing a regulated trickling of the sand/material form top to the bottom. The registers L_1, L_2 are the higher glass bulb and L_4, L_5 are the lower glass bulb. L_3 is the connected passage of the sandglass. The first half round of our algorithm is the process of sand trickling from the top to the bottom. For the sandglass, once the top bulb is empty, it can be turned upside down to begin the sand trickling process again. Thus the second half round

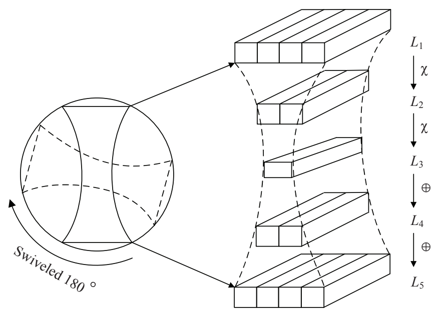

Figure 4.1: The Sablier structure

of Sablier just exchange the registers L_1, L_2 and L_4, L_5, then execute the same process as in the first half round. As we can see, one round of our algorithm is just like the process of the sandglass regulated trickling from top to the bottom twice.

Table 4.1: The gate count used for different functions.

Function	Gate Count
D flip flop	6
NOT	0.5
AND	1.33
XOR2	2.63

This is a new internal structure in stream ciphers. Unlike the previous LFSR-based structure and the usual linear/nonlinear combined structure in Grain and Trivium, this new structure natively have some immunity against the previously developed cryptanalysis methods such as the fast correlation attacks [19]. In the linear cryptanalysis of Sablier, we find that this new structure has good resistance against the linear cryptanalysis. Besides, a new structure also seems good for bio-diversity in the design of stream cipher.

The structure of Sablier can be transform to a special generalized Feistel structure, which is shown in Section 5.1. Further, from L_1 to L_2 and L_2 to L_3 we use a nonlinear function F, which consists of 16 the Keccak χ functions in parallel. This nonlinear function χ is translation-invariant in all directions and has algebraic degree two. It has good differential propagation and correlation properties. We have chosen it for its simple nonlinear propagation properties, its simple algebraic expression and its low gate count: one *xor*, one *and* and one *not* operation per state bit. Moreover, we added a linear transform layer in L_5 and cyclic shift in L_3 to provide the linear diffusion.

The authentication mechanism of Sablier is similar to the one used in Grain-128a. But here we made some modifications to make it more efficient without a penalty in security. In Grain-128a, the keystream is used to mask the message bits which will reduce the speed of the whole algorithm when authentication is activated. Instead, here we adopt the leak extraction strategy in [3] to avoid this problem.

4.2 Hardware Performance

The hardware implementation of Sablier is simple and efficient due to the simple nonlinear function and linear transformation in the design. Precisely, Sablier uses 208-bit memory for storing the state registers L_1, L_2, L_3, L_4, L_5. In one round of the algorithm, we use three nonlinear functions, which means that we need 48 χ functions. From the keystream generation of Sablier in Chapter 1, there are 6 16-bit *xor* in L_5 and 2 16-bit *xor* in L_4 and 1 16-bit *xor* in L_3. Thus in total, for one round of Sablier, we need 9 16-bit *xor*. In the output function we need 2 16-bit *xor*. We choose a gate cont of 6 for a flip flop. The following table lists the factors chosen in our implementation. Thus in total in one round we need 176 *xor*2 and 48 χ functions. The Sablier may require 224 *xor*2s, 48 *and*s, 48 *not*s and 208 flip flops, i.e., about 1925GE. We feel that Sablier is definitely more efficient than AES-GCM in constrained hardware environments. Besides, since Sablier will output 16-bit per clock cycle, it is expected to be 16 times faster than Trivium in hardware.

4.3 Justification for the Recommended Parameter Sets

From the security analysis in Chapter 3, we feel that Sablier is secure with respect to 80-bit key length and 32-bit tag size.

Chapter 5

Design Rationale

In this section, we motivate the choices we have made in the processing of designing Sablier. The generic criterion we follow in the design is a relatively good tradeoff between the security, efficiency and agility in constrained hardware environments. Of course, we put more weight on the security aspect, as we consider the security as the most important criterion. We expect to provide a comfortable security margin against both the known and unknown attacks. Here we claim that the designers have not hidden any weaknesses in this cipher.

5.1 Design Strategy and Algorithm Structure

In general, we follow the leak extraction strategy in [3] in the design of Sablier, but with some small differences. Precisely, the original leak extraction is applied to a block cipher, while in Sablier, we expect the updating function of the internal state to be a random permutation after sufficiently many initialization rounds and the leak extraction is applied to the underlying permutation at a low ratio, i.e., the ratio between the leaked state and the whole state is as small as possible to thwart the guess-and-determine attacks. Besides, the direct leak extraction is only applied to the authentication process, not in the encryption. We adopt a simple linear output function in the keystream generation phase to make the leak extraction more difficult to exploit in various attacks.

Unlike the traditional LFSR-based stream ciphers and the usual nonlinear/linear shift registers combined structure in Grain and Trivium, Sablier adopts a new internal structure to generate the keystream, only using bitwise xor, bitwise logical and and bitwise intra-word rotation. The new internal structure is motivated by the mixing process of the sand in a sandglass, or can be seen as the shaking of the cocktail in a container in a bar. We first mix or shake the container for a large number of times, which is just the initialization phase of Sablier. After that, something will be emitted after each mixing or shaking round as keystream.

Fig.5.1 gives an equivalent high-level structure of one-round Sablier. From

Fig.5.1, it is easy to prove that the state updating function of Sablier is a permutation of the internal state.

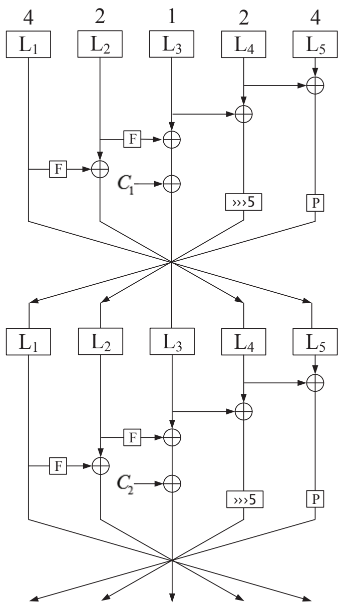

Figure 5.1: The equivalent high-level structure of one-round Sablier

5.2 Choice of the Nonlinear Function χ

Since we want to only use the bitwise logical and in our new design, we adopt the nonlinear function χ in Keccak to be the only nonlinear function in Sablier. During the cryptanalysis of Keccak, the χ function is known to have good cryptographic properties, i.e., its simple nonlinear propagation properties, simple algebraic expression and its low gate count. For more details, please refer to [2] for a full description.

5.3 Choice of Linear Transformation Matrix

The updating of the L_5 layer can be expressed as:

$$\begin{pmatrix} L_{5,1} \\ L_{5,2} \\ L_{5,3} \\ L_{5,4} \end{pmatrix} = M \begin{pmatrix} L_{5,1} \\ L_{5,2} \\ L_{5,3} \\ L_{5,4} \end{pmatrix} + \begin{pmatrix} L_{4,2} \\ 0 \\ 0 \\ L_{4,1} \end{pmatrix}, M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

Obviously, the matrix M has a functionality of linear diffusion in L_5. The branch number of M does not reach the maximum, however, M has an advantage in hardware implementing that the updating of states in L_5 layer will not slow down the updating of the other states. Besides, the concatenation and 32-bit right rotation are used together to destroy the data unit boundary and provide some linear diffusion functionality as well.

5.4 Choice of Authentication Mechanism

Since Sablier aims to be efficient in hardware implementation, we refer to the authentication mechanism of Grain-128a which is cost-efficient in hardware. However, compared to Grain-128a, in which the first 64-bit pre-output keystream bits and half of the keystream bits generated are used for authentication, we extract a different position in the internal state of Sablier as the masking sequence for the authentication. In this way, the authentication mechanism will not slow down the encryption process. Both the cipher with authentication and the one without authentication have the same throughput rate, which can ensure the encryption (decryption) speed.

Note that we could have created two keystreams: one for encryption and one for authentication. This would in a sense allow us to double the throughput rate, but could have disastrous drawbacks if we are not careful about the details, and we have decided to stick with the much safer approach now adopted in the design.

5.5 Choice of Support for Variable Tag Lengths

Since Sablier is a lightweight cipher and its potential applications are for some resource constrained environments, we choose 32 as an upper bound of the tag size. This bound can easily be upgrade to 64 bits if we double the size of both the accumulator and the shift register, requiring more hardware resources. Besides, the size of the pre-output keystream blocks for the initialization of the authentication is raised to 8, which means a prolonged initialization rounds of the cipher. Sablier without authentication will not output the keystream used for the authentication. This will avoid an attack scenario mentioned in section 5.7 of [4].

5.6 The Unknown Weaknesses

In the linear cryptanalysis of Sablier, we have considered the LSCA method. In the LSCA method, we need continuous 209 keystream words to find a bias. We find that our algorithm is extremely strong against this type of linear cryptanalysis. Maybe there exist other linear cryptanalysis methods which could find a better correlation coefficient, we strongly feel that since the security margin in the LSCA method is quite large, it is infeasible to find a linear distinguisher with the correlation coefficient greater than 2^{-40}. Further, we expect the extremely large security margin in Sablier with respect to the linear distinguishing attacks will provide some resistance against the unknown attacks.

Chapter 6

Intellectual Property

Sablier is not covered by any patent and is free for any application in practice.

If any of this information changes, the submitter will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

Chapter 7

Consent

The submitter hereby consents to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter understands that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter understands that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter acknowledges that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter understands that if he disagrees with published analyses then he is expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter understands that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Chapter 8

Test Vectors

This section provides the test vectors consisting of a 80-bit key, a 80-bit initialization vector, and the first 256 bytes of message.

The message is as follow.

```
0001 0203 0405 0607 0809 0a0b 0c0d 0e0f 1011 1213 1415 1617 1819
1a1b 1c1d 1e1f 2021 2223 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435 3637 3839 3a3b 3c3d 3e3f 4041 4243 4445 4647 4849 4a4b 4c4d
4e4f 5051 5253 5455 5657 5859 5a5b 5c5d 5e5f 6061 6263 6465 6667
6869 6a6b 6c6d 6e6f 7071 7273 7475 7677 7879 7a7b 7c7d 7e7f 8081
8283 8485 8687 8889 8a8b 8c8d 8e8f 9091 9293 9495 9697 9899 9a9b
9c9d 9e9f a0a1 a2a3 a4a5 a6a7 a8a9 aaab acad aeaf b0b1 b2b3 b4b5
b6b7 b8b9 babb bcbd bebf c0c1 c2c3 c4c5 c6c7 c8c9 cacb cccd cecf
d0d1 d2d3 d4d5 d6d7 d8d9 dadb dcdd dedf e0e1 e2e3 e4e5 e6e7 e8e9
eaeb eced eeef f0f1 f2f3 f4f5 f6f7 f8f9 fafb fcfd feff
```

```
IV = 0000 0000 0000 0000 0000
key = 0000 0000 0000 0000 0000
ciphertext =
b05c d25c b607 004b b324 1c01 2d2b 3298 66eb 2f86 b74f 7810 a24a
5da4 268e ebe2 289e c311 a2a0 6da4 dde2 b70c 5ff4 5a22 178e 351f
5a2e d16c 31bf 15c8 d890 5a6d 72ab 2292 3dac ccbe 0023 6260 8dc3
7a62 78cb 0af1 c24c b74b 634e aaa1 9aad dcae fc13 45de 5c86 3f4c
99b4 10bc 995c 9fd5 f4a9 e45b e035 374c be89 9653 3eab 1070 4440
1821 1d85 feea 3796 ee38 9ed7 4423 ca03 2ddf e1ae ceb7 ecd2 9a50
fe4d ac85 b304 80af 1cf3 1974 8624 f102 6184 5dd4 2a13 804c 71b4
24d4 b4a1 db10 0fb3 2321 568d 0dad dd47 7ad1 702b df72 b4c4 5ae6
283d 0ebb 8fe3 7dbc f7f0 da49 897a 26a1 1c6a 59e2 6329 43cd 7492
7f5e 14d5 9f88 d734 1b4d 66ea 6570 77da 1693 e1b 15e8f
tag = 7287 db93
```

```
IV = 0001 0203 0405 0607 0809
```

```
key = 0001 0203 0405 0607 0809
ciphertext =
a97f 21f9 e4ad 323d 5d00 18f8 2aed 79fd 1a2e 822e 424b 8187 e0f9
6507 8534 2f8a 8f41 cc23 a7e5 a979 dfd9 cd9a 4cb3 1c25 4c0c ee6c
dbbb befe ccc0 f49d 399a 303c 1ae1 6a6c 9ee6 dbe6 28f9 2dd8 0ee9
2783 eed4 14b0 9b51 36c4 bca5 3309 3c28 b73e 64ac c124 aa76 7192
9f16 f5f5 5665 1d2b d3f7 4ee9 e0c5 fd1a 0a6f b9f1 73ae 0104 c617
7406 9dbc 7b00 681c e931 5d85 739d 4925 7e06 6798 fd90 7668 8aa2
d409 29ff 5e7c c50e db75 8e27 2a5d e657 4d47 49dc fe3e 0212 f4ce
a26c 9b12 8830 9025 24f1 21db 75e4 7a36 ef44 4173 0592 e600 2c36
e0dd 9749 873e 35fa f360 28fb 330a 5e5f df5c a799 b92d d8ba f1d5
688a 16de e070 94ef 48e8 1af1 e6f9 7cef a49c 3fd3 4fee
tag = 5706 fbdd
```

Bibliography

[1] Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium, *Fast Software Encryption-FSE'2009*, LNCS vol. 5665, Springer, Heidelberg,(2009), pp. 1-22.

[2] Bertoni, G., Daemen, J., Peeters, M. and Van Assche G.: The Keccak reference, available at `http://keccak.noekeon.org/Keccak-reference-3.0.pdf.`

[3] Biryukov, A.: The design of a stream cipher Lex, Selected Areas in Cryptography-SAC 2006, Lecture Notes in Computer Science vol. 4356, Springer-Verlag. pp. 67-75, (2007).

[4] Agren M., Hell, M., Johansson T., and Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing (IJWMC)* vol. 5, No. 1, pp. 48-59. (2011)

[5] Agren, M., Hell, M., Johansson, T.: On hardware-oriented message authentication with applications towards RFID, in Proceedings of the 2011 Workshop on Lightweight Security and Privacy: Devices, Protocols, and Applications, E. Savas, A. A. Selcuk, and U. Uludag, Eds., pp. 26-33. (2011).

[6] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials, *Advances in Cryptology-EUROCRYPT'2009*, LNCS vol. 5479, Springer, Heidelberg,(2009), pp. 278-299.

[7] Dinur, I., Shamir, A.: Breaking Grain-128 with dynamic cube attacks, *Fast Software Encryption-FSE'2011*, LNCS vol. 6733, Springer, Heidelberg,(2011), pp. 167-187.

[8] Handschuh, H., Preneel, B.: Key-recovery attacks on universal hash function based MAC algorithms, in Advances in Cryptology—CRYPTO 2008, ser. Lecture Notes in Computer Science, D. Wagner, Ed., vol. 5157. Springer-Verlag. pp. 144-161. (2008).

[9] Golić Jovan Dj.: Correlation properties of a general binary combiner with memory, *Journal of Cryptology*, vol.9, Springer-Verlag. pp. 111-126, (1996).

[10] 3GPP, Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 specification, 3rd Generation Partnership Project (3GPP), TS, Jul. 2010. [Online]. Available at: `http://gsmworld.com/our-work/programmesand-initiatives /fraud-and-security/gsm security algorithms.htm`

[11] Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC specification, 3rd Generation Partnership Project (3GPP), TS, Jul. 2010. [Online]. Available at: `http://gsmworld.com/our-work/programmes-andinitiatives /fraud-and-security/gsm security algorithms.htm`

[12] Khovratovich D., Nikolić I.: Rotational cryptanalysis of ARX. The 17th international conference on Fast Software Encryption-FSE'2010. pp. 333-346. LNCS vol.6147, Springer-Verlag (2010)

[13] Pawe M., Josef P., and Marian S.: Rotational cryptanalysis of round-reduced Keccak. Accepted to FSE 2013, March 11-13, 2013, Singapore. available at https://eprint.iacr.org/2012/546.pdf

[14] Ferguson, N., Whiting, D., Schneier, B., Kelsey, J., Lucks, S., Kohno, T.: Helix: fast encryption and authentication in a single cryptographic primitive, in [16], pp. 330C346 (2003), `http://www.macfergus.com/helix/`

[15] Handschuh, H., Preneel, B.: Key-recovery attacks on universal hash function based MAC algorithms, Advances in Cryptology–CRYPTO'2008, D. Wagner, Ed. Lecture Notes in Computer Science, vol. 5157. Springer-Verlag. pp. 144-161. (2008).

[16] Ferguson, N., Whiting, D., Schneier, B., Kelsey, J., Lucks, S., Kohno, T.: Helix: fast encryption and authentication in a single cryptographic primitive, in [16], pp. 330-346 (2003), `http://www.macfergus.com/helix/`

[17] H. Krawczyk. LFSR-based hashing and authentication. In *Adavnces in Cryptology–CRYPTO'94*, pages 129-139. Springer-Verlag, 1994.

[18] H. Krawczyk. New hash functions for message authentication. In *Adavnces in Cryptology–EUROCRYPTO'95*, pages 301-310. Springer-Verlag, 1995.

[19] Willi, M., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3): 159-176, 1989.

[20] Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hash functions. *Theoretical Computer Science*, 107(1):121-133, 1993.

[21] Naor, J., Naor, M.: Small bias probability spaces: efficient construction an applications. SIAM *Jour. on Computing*, vol. 22, No. 4, 1993, pp. 838-856.

[22] T. Johansson, G. Kabatianskii, and B. Smeets. On the relation between A-codes and codes correcting independent errors. In T. Helleseth, editor, *Advances in Cryptology–EUROCRYPT'93*, volume 765 of Lecture Notes in Computer Science, pp. 1-11. Springer-Verlag, 1994