A realtime key recovery attack on the authenticated cipher FASER128[⋆]

Xiutao FENG and Fan ZHANG

Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, CAS, China (e-mail: fengxt@amss.ac.cn)

Abstract. FASER is a family of authenticated ciphers submitted to the CAESAR competition, which contains two parent ciphers: FASER128 and FASER256. In this work we only focus on FASER128 and present a key recovery attack to FASER128, which needs at most 64 key words and is realtime in a PC. The result shows that FASER128 is very insecure. What's more, our attack can be easily applied to FASER256 and break it entirely.

Keywords: CAESAR, stream ciphers, FASER128, key recovery attack.

1 Introduction

Authenticated cipher is a cipher combining encryption with authentication, which can provides confidentiality, integrity and authenticity assurances on the data simultaneously and has been widely used in session communications such as SSL/TLS [1, 2], etc. Since the security of authenticated ciphers depends on both encryption and authentication, and an attacker possesses extra information such as authentication tag except for plaintext/ciphertext and has more choices in the execution of attacks (he may attack any one of the encryption and the authentication, or attack both of them simultaneously), thus it is more difficult to design a good authenticated cipher than a usual encryption cipher from the viewpoint of designers. CAESAR is a new competition calling for submissions of authenticated ciphers [3]. This competition follows a long tradition of focused competitions in secret-key cryptography. It is expected to have a tremendous increase in confidence in the security of authentication ciphers.

FASER is a family of authenticated ciphers designed by F. Chaza et al and has been submitted to the CAESAR competition [4]. FASER is composed of two parent ciphers: FASER128 and FASER256. In this work we only focus on FASER128 and present a key recovery attack to FASER128, which needs at most 64 key words and is realtime in a PC. The result shows that FASER128 is very insecure. What's more, our attack can be easily applied to FASER256 and break it entirely.

[⋆] This work was supported by the Natural Science Foundation of China (Grant No. 61121062, 11071285), the 973 Program (Grant No. 2011CB302401)

The rest of this paper is organized as follows: in section 2 we recall FASER128 briefly, and in section 3 we present a key recovery attack on FASER128. Finally section 4 concludes the paper.

2 Description of FASER128

In this section we will recall FASER128 briefly, and more details on FASER128 can be found in [4].

FASER128 is composed of two state registers(E and A, one for encryption and one for authentication), three components (FSR, MIX, MAJ) and three procedures (*initialisation*, *update*, $finalisation$), shown in Fig. 1. Since our attack does not involve knowledge of authentication including the FSR A, the generation of authentication tag Tag and the procedure $finalisation$, etc., thus here we do not intend to introduce them, and if readers are interested in them, please refer to [4].

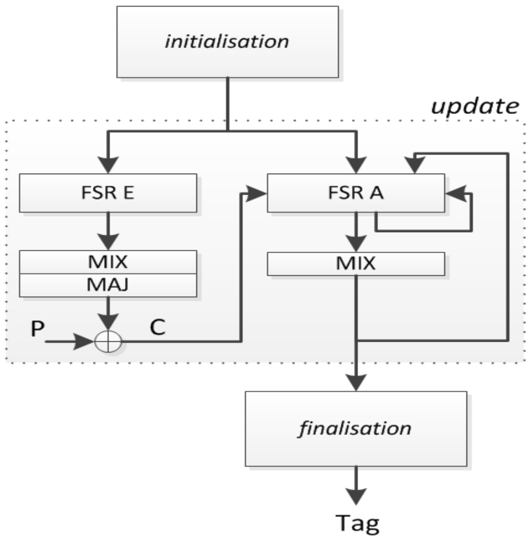

Fig. 1 The structure of FASER

2.1 The FSR E

The FSR E contains a 256-bit state register E and 8 sub-FSRs, where the register E is subdivided into 4 words, each word 64 bits, that is, $E = (E_3, E_2, E_1, E_0)$, and 8 sub-FSRs are defined as below:

$$\begin{aligned}
&FSR17(X): y \leftarrow x_{16} \oplus x_{15} \oplus x_{14} \cdot x_{13}, && (x_{16}, \cdots, x_1, x_0) \leftarrow (x_{15}, \cdots, x_0, y)\\
&FSR23(X): y \leftarrow x_{22} \oplus x_{21} \oplus x_{12} \cdot x_{11}, && (x_{22}, \cdots, x_1, x_0) \leftarrow (x_{21}, \cdots, x_0, y)\\
&FSR29(X): y \leftarrow x_{28} \oplus x_{27} \oplus x_{19} \cdot x_{12}, && (x_{28}, \cdots, x_1, x_0) \leftarrow (x_{27}, \cdots, x_0, y)\\
&FSR31(X): y \leftarrow x_{30} \oplus x_{11} \oplus x_{21} \cdot x_{13}, && (x_{30}, \cdots, x_1, x_0) \leftarrow (x_{29}, \cdots, x_0, y)\\
&FSR33(X): y \leftarrow x_{32} \oplus x_{19}, && (x_{32}, \cdots, x_1, x_0) \leftarrow (x_{31}, \cdots, x_0, y)\\
&FSR35(X): y \leftarrow x_{34} \oplus x_{32}, && (x_{34}, \cdots, x_1, x_0) \leftarrow (x_{33}, \cdots, x_0, y)\\
&FSR41(X): y \leftarrow x_{40} \oplus x_{37}, && (x_{40}, \cdots, x_1, x_0) \leftarrow (x_{39}, \cdots, x_0, y)\\
&FSR47(X): y \leftarrow x_{46} \oplus x_{41}, && (x_{46}, \cdots, x_1, x_0) \leftarrow (x_{45}, \cdots, x_0, y)
\end{aligned}$$

The feedback update of the FSR E can be described as

$$FeedFSR(X) = (FSR_3(X_3), FSR_2(X_2), FSR_1(X_1), FSR_0(X_0)), \qquad (1)$$

where

$$\begin{aligned}
FSR_0(X_0) &= FSR33(H_{33}(X_0)) \parallel FSR31(L_{31}(X_0)),\\
FSR_1(X_1) &= FSR35(H_{35}(X_1)) \parallel FSR29(L_{29}(X_1)),\\
FSR_2(X_2) &= FSR41(H_{41}(X_2)) \parallel FSR23(L_{23}(X_2)),\\
FSR_3(X_3) &= FSR47(H_{47}(X_3)) \parallel FSR17(L_{17}(X_3)),
\end{aligned}$$

$H_i(X)$ and $L_i(X)$ represent the i-th most significant (High) bits of 64-bit X and i-th least significant (Low) bits of 64-bit X respectively, and $\parallel$ denotes concatenation.

In one update of the FSR E, the FSR is clocked 8 times, that is,

$$FSR(X) = \underbrace{FeedFSR(FeedFSR(\cdots FeedFSR}_{8 \text{ times}}(X)\cdots)), \qquad (2)$$

where each clock of the FSR results in each sub-FSR being clocked. Consequently, there are 64-bits updated in each state register of the FSR E[1].

2.2 The functions MIX and MAJ

The MIX function combines information from the state register. The input is the entire state (X_3, X_2, X_1, X_0) and the output is three 64-bit words such that:

$$\begin{aligned}
Y_0 &= (X_0 \lll 3) \oplus (X_1 \lll 12) \oplus (X_2 \lll 43) \oplus (X_3 \lll 27),\\
Y_1 &= (X_0 \lll 22) \oplus (X_1 \lll 54) \oplus (X_2 \lll 5) \oplus (X_3 \lll 30),\\
Y_2 &= (X_0 \lll 50) \oplus (X_1 \lll 35) \oplus (X_2 \lll 14) \oplus (X_3 \lll 60),
\end{aligned}$$

where $\lll$ denotes bitwise rotation to the left.

[1] For accuracy, here we adjust the definition of the FSR instead of the definition of the designers in the original paper, which makes it easier to misunderstand that one update of the FSR clocks only once.

The MAJ function operates on 64-bit words as well. The Input is the output of MIX, i.e. Y_0, Y_1, Y_2, and the output of MAJ is Z:

$$Z = (Y_0 \cdot Y_1) \oplus (Y_0 \cdot Y_2) \oplus (Y_1 \cdot Y_2), \tag{3}$$

where "·" means the bitwise multiplication.

2.3 The procedure *initialisation*

The procedure *initialisation* of FASER128 is mainly used to update the state registers E and A using the key K and the public message number PMN. Here we are interested only in the update of the register E. The pseudo-code for the procedure of *initialisation* is

```
$INIT(E, K, PMN):$
   1. $E = 0x5A \parallel \cdots \parallel 0x5A \parallel PMN \parallel K$,
   2. for $i = 1$ to 8 do
   3.       $E = FSR(E)$,
   4.       $(Y_2, Y_1, Y_0) = MIX(E)$,
   5.       $E = (E_3, E_2 \oplus Y_2, E_1 \oplus Y_1, E_0 \oplus Y_0)$,
   6.       $E = (E_2, E_1, E_0, E_3)$,
   7. end for
   8. $TWEAK(E)$,
   9. for $i = 1$ to 8 do
  10.       $E = FSR(E)$,
  11. end for
```

where $TWEAK$ is defined as below:

$$TWEAK(X):\ (x_{63}, x_{62}, \cdots, x_2, x_1, x_0) \leftarrow (1, x_{62}, \cdots, x_2, 0, 1). \tag{4}$$

2.4 The procedure *update* (or *encryption*)

Here we only focus on the encryption of the procedure *update* and omit the authentication of *update*. The following describes one update of FASER128 to process one 64-bit plaintext word P_i. FASER128 continues to clock until all the input has been processed. The pseudo-code for the procedure *update* is

```
$UPDATE(E, P_i):$
                $E = FSR(E)$,
    $(Y_2, Y_1, Y_0) = MIX(E)$,
                $Z = MAJ(Y_2, Y_1, Y_0)$,
              $C_i = P_i \oplus Z$.
```

3 A realtime key recovery attack

In this section we will give a realtime key recovery attack on FASER128. Our attack can be divided into two phases: at the first phase we recover the state of the register E of the FSR E, and at the second phase we further recover the key K by the procedure *initialisation*.

3.1 Recovering the state of the register E

For simplicity we keep all notations in section 2 throughout the rest of the paper. For a 64-bit word variable X, we denote by $X^t[i]$ the i-th bit of the value of X at time $t \geq 0$, where $0 \leq i \leq 63$. Our main idea is to observe certain relationship between the key bits $Z^t[i]$ and $Z^{t+1}[i+8 \mod 64]$ for some i to get some linear equations among the state of the register E, and then recover the state of the FSR E by means of those linear equations.

For example, set $i = 54$. Then we have

$$\begin{aligned} Z^t[54] =& Y_0^t[54]Y_1^t[54] \oplus Y_1^t[54]Y_2^t[54] \oplus Y_2^t[54]Y_0^t[54], \\ Y_0^t[54] =& X_0^t[51] \oplus X_1^t[42] \oplus X_2^t[11] \oplus X_3^t[27], \\ Y_1^t[54] =& X_0^t[32] \oplus X_1^t[0] \oplus X_2^t[49] \oplus X_1^t[24], \\ Y_2^t[54] =& X_0^t[4] \oplus X_1^t[19] \oplus X_2^t[40] \oplus X_3^t[58] \end{aligned} \tag{5}$$

and

$$\begin{aligned} Z^{t+1}[62] =& Y_0^{t+1}[62]Y_1^{t+1}[62] \oplus Y_1^{t+1}[62]Y_2^{t+1}[62] \oplus Y_2^{t+1}[62]Y_0^{t+1}[62], \\ Y_0^{t+1}[62] =& X_0^{t+1}[59] \oplus X_1^{t+1}[50] \oplus X_2^{t+1}[19] \oplus X_3^{t+1}[35], \\ Y_1^{t+1}[62] =& X_0^{t+1}[40] \oplus X_1^{t+1}[8] \oplus X_2^{t+1}[57] \oplus X_3^{t+1}[32], \\ Y_2^{t+1}[62] =& X_0^{t+1}[12] \oplus X_1^{t+1}[27] \oplus X_2^{t+1}[48] \oplus X_3^{t+1}[2]. \end{aligned} \tag{6}$$

By the update of the FSR, it is easy to see that the following holds:

- $X_0^t[j] = X_0^{t+1}[j+8]$ for $j = 51, 32, 4$,
- $X_1^t[j] = X_1^{t+1}[j+8]$ for $j = 42, 0, 19$,
- $X_2^t[j] = X_2^{t+1}[j+8]$ for $j = 11, 49, 40$, and
- $X_3^t[j] = X_3^{t+2}[j+8]$ for $j = 27, 24$.

Thus we have

$$Y_0^{t+1}[62] = Y_0^t[54],\ Y_1^{t+1}[62] = Y_1^t[54],\ Y_2^{t+1}[62] = Y_2^t[54] \oplus X_3^t[58] \oplus X_3^{t+1}[2],$$

and

$$Z^t[54] \oplus Z^{t+1}[62] = (X_3^t[58] \oplus X_3^{t+1}[2])(Y_0^t[54] \oplus Y_1^{t+1}[54]). \tag{7}$$

In particular, when $Z^t[54] \oplus Z^{t+1}[62] = 1$, we can get two linear equations:

$$X_3^t[58] \oplus X_3^{t+1}[2] = 1, \tag{8}$$

$$Y_0^t[54] \oplus Y_1^{t+1}[62] = 1. \tag{9}$$

The above observation holds also for $i = 55$. Indeed we have

$$Z^t[55] \oplus Z^{t+1}[63] = (X_3^t[59] \oplus X_3^{t+1}[3])(Y_0^t[55] \oplus Y_1^{t+1}[55]). \tag{10}$$

Thus when $Z^t[55] \oplus Z^{t+1}[63] = 1$, we have as well

$$X_3^t[59] \oplus X_3^{t+1}[3] = 1, \tag{11}$$
$$Y_0^t[55] \oplus Y_1^{t+1}[63] = 1. \tag{12}$$

Recovering $\boldsymbol{X_3}$: It is noticed that equations (8) and (11) involve only the states of X_3 at different times, thus we can recover the state of X_3 after we get about 64 linear equations. More precisely, the process of recovering the state of X_3 is: first we collect about 64 linear equations by those key bits satisfying $Z^t[54] \oplus Z^{t+1}[62] = 1$ or $Z^t[55] \oplus Z^{t+1}[63] = 1$ for each possible time $t+j (j \geq 0)$. Note that those linear equations involve 47-bit state variables $H_{47}(X_3^t)$ of the linear sub-FSR and 17-bit state variables $L_3(X_3^t)$ of the nonlinear sub-FSR of X_3, if we guess the 17-bit $L_3(X_3^t)$ at time t, then $L_3(X_3^{t+j})$ are known, and 64 linear equations are simplified to linear equations only on 47 variables $H_{47}(X_3^t)$ of the linear sub-FSR of X_3. Thus we can recover $H_{47}(X_3^t)$ with 47 out of 64 equations. The rest of those equations is used to check the correctness of recovered X_3^t. Finally X_3^t can be determined uniquely.

In the above process of recovering X_3^t, we can collect one equation for each possible j on average (since either of equations (8) and (11) holds on average). Thus we need about 64 key words for collecting 64 equations. After collecting 64 linear equations, we need to further guess 2^{17} possible states $L_3(X_3^t)$ of the nonlinear sub-FSR of X_3, and solve a linear system on 47 bit variables $H_{47}(X_3^t)$ for each possible $L_3(X_3^t)$. Since the feedback of the linear sub-FSR of X_3 is very simple and is just the addition of two variables, solving this linear system is faster than that we expect in practice.

Recovering $\boldsymbol{X_2^t}$: Set $i = 3$. Similarly to equations (5), we have

$$\begin{aligned} Y_0^{t+1}[11] &= Y_0^t[3] \\ Y_1^{t+1}[11] &= Y_1^t[3] \oplus X_2^t[62] \oplus X_2^{t+1}[6] \\ Y_2^{t+1}[11] &= Y_2^t[3] \end{aligned}$$

With the expressions of $Z^t[3]$ and $Z^{t+1}[11]$, we get

$$Z^t[3] \oplus Z^{t+1}[11] = (X_2^t[62] \oplus X_2^{t+1}[6]) \cdot (Y_0^t[3] \oplus Y_2^t[3]).$$

If $Z^t[3] \oplus Z^{t+1}[11] = 1$, then we have

$$X_2^t[62] \oplus X_2^{t+1}[6] = 1 \tag{13}$$
$$Y_0^t[3] \oplus Y_2^t[3] = 1 \tag{14}$$

It is noticed that equation (13) involves only the state variables of X_2. Similarly to the recovery of X_3^t, we collect 64 linear equations by those key bits satisfying $Z^{t+j}[3] \oplus Z^{t+j+1}[11] = 1$. Then we guess 23-bit state $L_{23}(X_2^t)$ of the nonlinear sub-FSR of X_2 and solve 41-bit state variables $H_{41}(X_2^t)$ of the linear sub-FSR of X_2 with 41 out of 64 linear equations. The rest of those linear equations is used check the correctness of recovered X_2^t. Finally X_2^t can be determined uniquely.

Remark 1 *Since X_3^t has been recovered, we have another method to collect more linear equations on X_2^t in practice. Consider $i = 22$. Note that*

$$\begin{aligned} Y_0^{t+1}[30] =& Y_0^t[22] \oplus c_1, \\ Y_1^{t+1}[30] =& Y_1^t[22] \oplus X_2^t[17] \oplus X_2^{t+1}[25] \oplus c_2, \\ Y_2^{t+1}[30] =& Y_2^t[22], \end{aligned}$$

where $c_1 = X_3^t[59] \oplus X_3^{t+1}[3]$ and $c_2 = X_3^t[56] \oplus X_3^{t+1}[0]$ are known constants, if $c_1 = 0$ and $Z^t[22] \oplus Z^{t+1}[30] = 1$, then we have

$$X_2^t[17] \oplus X_2^{t+1}[25] \oplus c_2 = 1, \tag{15}$$

$$Y_0^t[22] \oplus Y_2^t[22] = 1. \tag{16}$$

The equation (15) involves only the state variables of X_2 as well. What is more, it is easy to check that the same conclusion holds for $i = 23, 24, 25$ or 41. Hence we can collect 64 equations with less key words, and the total data complexity is reduced.

Recovering X_1^t: When X_3^t and X_2^t have been recovered, we collect linear equations only on the state of X_1 similarly to (8) for $i \in \{37, 38, 56, 57\}$ and similarly to (15) for $i \in \{4, 5, 6, 7, 8, 35, 36, 39, 40, 53, 58\}$ respectively. Once enough linear equations are collected, we guess the state $L_{29}(X_1^t)$ of the nonlinear sub-FSR of X_1 directly and solve the state variables $H_{35}(X_1^t)$ of the linear sub-FSR of X_1 with 35 out of those linear equations, and the rest is used to check the correctness of X_1^t. Finally we get the unique X_1^t.

Recovering X_0^t: After X_3^t, X_2^t and X_1^t are recovered, it is easy to recover X_0^t. It is noticed that equations (9) and (12) got in the recovery of X_3^t are indeed some linear equations only on the state $H_{33}(X_0^t)$ of the linear sub-FSR of X_0, thus by them we can directly determine $H_{33}(X_0^t)$.

Finally for $i \in \{0, 1, 2, 17, 18, \cdots, 21, 34, \cdots, 49, 53, \cdots, 63\}$, we consider equations

$$\begin{aligned} Z^t[i] =& Y_0^t \cdot Y_1^t \oplus Y_1^t \cdot Y_2^t \oplus Y_2^t \cdot Y_0^t \\ =& a \cdot b \oplus (b \oplus c) \cdot (X_0^t[j] \oplus c), \end{aligned}$$

where a, b, c are known constants and $0 \le j < 31$. If $b \oplus c = 1$, then we get $X_0^t[j]$ directly. As for the rest unknown variables among $L_{31}(X_0^t)$, one method is to guess and check them directly, and another method is to get them from the linear equations on Y_0^t, Y_1^t and Y_2^t got during the recovery of X_2 and X_1. Up to now we have recovered the state of the register E of the FSR E entirely.

3.2 Recovering the key K

In this section we consider how to recover the key K from the state of the register E of the FSR E.

By the definition of the FSR E, one can easily check that the FSR E is invertible. Thus when the state E after initialisation are known, we can invert the FSR E (i.e., steps 9, 10 and 11 in the procedure *initialisation*) and get the intermediate state E after step 8. Since $TWEAK$ is not a permutation, we will get 8 possible values of E corresponding to the intermediate state E after step 8, denoted by $E_1, E_2, \cdots, E_8$. For each possible value E_i ($1 \le i \le 8$), we deal with the inversion of steps from 7 to 2 in turn. It is surprising that the linear operations determined by steps 4 and 5 is NOT invertible! Indeed one can check that the rank of the matrix of the linear transformation determined by steps 4 and 5 is 191 (note: the dimension is 192). Thus for an arbitrary output after step 5, there are 2 possible inputs before step 4 corresponding to it. Note that steps $2 \sim 7$ loop eight times, so we can get totally 2^8 possible values before step 2 for each E_i, denoted by $E_{i,j}$, where $1 \le j \le 2^8$. Finally we verify whether the prefix of each possible $E_{i,j}$ (totally 2^{11} possible values) is equal to 0x5a5a$\cdots$5a or not. If some $E_{i,j}$ passes the verification, then one candidate K is written down. Here it should be emphasized that all candidates K are valid and they are equivalent to each other.

4 Conclusion

In this paper we discuss the security of FASER128, and give a key recovery attack on FASER128, which need only a few of key words and can recover all possible K realtime in a PC. Our results show that FASER128 is very insecure.

References

1. RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0.
2. RFC 5246: The Transport Layer Security (TLS) Protocol, Version 1.2.
3. CAESER: http://competitions.cr.yp.to/index.html.
4. FASER: F. Chaza, C. McDonald and R. Avanzi, submited to CAESAR, available from: http://competitions.cr.yp.to/round1/faser.pdf.