A practical forgery and state recovery attack on the authenticated cipher PANDA-s*

Xiutao FENG, Fan ZHANG and Hui WANG

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

Abstract. PANDA is a family of authenticated ciphers submitted to CARSAR, which consists of two ciphers: PANDA-s and PANDA-b. In this work we present a state recovery attack against PANDA-s with time complexity about 2^{41} under the known-plaintext-attack model, which needs 137 pairs of known plaintext/ciphertext and about 2GB memories. Our attack is practical in a small workstation. Based on the above attack, we further deduce a forgery attack against PANDA-s, which can forge a legal ciphertext (C, T) of an arbitrary plaintext P. The results show that PANDA-s is insecure.

Keywords: CAESAR, PANDA, state recovery attack, forgery attack.

1 Introduction

Authenticated cipher is a cipher combining encryption with authentication, which can provide confidentiality, integrity and authenticity assurances on the data simultaneously and has been widely used in many network session protocols such as SSL/TLS [1, 2], IPSec [3], etc. Currently a new competition, namely CAESAR, is calling for submissions of authenticated ciphers [4]. This competition follows a long tradition of focused competitions in secret-key cryptography, and is expected to have a tremendous increase in confidence in the security of authenticated ciphers.

PANDA is a family of authenticated ciphers designed by D. Ye et al and has been submitted to the CAESAR competition [5]. PANDA consists of two ciphers: PANDA-s and PANDA-b, and both are based on a simple round function. PANDA-s is similar to authenticated encryption (in short AE) with sponge structures [6] and is a mixture of a stream cipher and a MAC. PANDA-b is an online cipher like APE [7] with a permeation. In [8] Y. Sasaki et al present a forgery attack against PANDA-s under the condition of nonce reuse. It should be pointed that the nonce is usually a counter and is used once, thus it is easy to avoid launching Y. Sasaki et al' attack in practice. As for PANDA-s, in this work we present a practical state recovery attack with time complexity about

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

2^{41} under the known-plaintext-attack model, which needs 137 pairs of known plaintext/ciphertext and about 2GB memories. What is more, based on the above attack, we further deduce a forgery attack against PANDA-s which can forge a legal ciphertext (C, T) of an arbitrary plaintext P. The results show that PANDA-s is insecure.

The rest of this paper is organized as follows: in section 2 we recall PANDA-s briefly, and in section 3 we provide a state recovery attack and an evaluation of the time, data and memory complexity of our attack. Finally we further deduce a forgery attack against PANDA-s in section 4.

2 Description of PANDA-s

In this section we recall PANDA-s briefly. Since our attack does not involve in the initialization and the process of associated data of PANDA-s, thus here we omit them, and more details of PANDA-s can be found in [5].

PANDA-s takes in a 128-bit key K, a 128-bit nonce N, a variable-length associated data A and a variable-length plaintext P and outputs a variable-length ciphertext (C, T), where T is a 128-bit authentication tag. The main part of PANDA-s is a round function RoundFunc, which is a bijection from an eight 64-bit-block input to an eight 64-bit-block output. The state of PANDA-s is seven 64-bit blocks, which is a part of the input and output of RoundFunc. RoundFunc consists of four non-linear transformations SubNibbles and a linear transformation LinearTrans, as shown in Fig. 1.

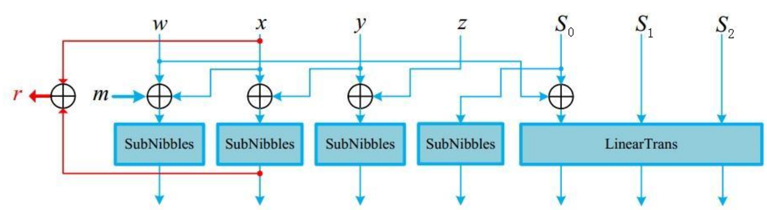

Fig. 1 The round function RoundFunc in PANDA-s

Let $(w, x, y, z, S_0, S_1, S_2, m)$ and $(w', x', y', z', S'_0, S'_1, S'_2, r)$ be the input and the output of RoundFunc respectively. Then the specific process of RoundFunc is defined as follows:

RoundFunc$(w, x, y, z, S_0, S_1, S_2, m)$

$w' \leftarrow \text{SubNibbles}(w \oplus x \oplus m)$

$x' \leftarrow \text{SubNibbles}(x \oplus y)$

$y' \leftarrow \text{SubNibbles}(y \oplus z)$

$z' \leftarrow \text{SubNibbles}(S_0)$

$(S'_0, S'_1, S'_2) \leftarrow \text{LinearTrans}(S_0 \oplus w, S_1, S_2)$

$r \leftarrow x \oplus x'$

return $(w', x', y', z', S'_0, S'_1, S'_2, r)$

2.1 SubNibbles

SubNibbles is a nonlinear transformation from a 64-bit input to a 64-bit output, and is shown in Fig. 2. Let $a_0 a_1 \cdots a_{63}$ and $b_0 b_1 \cdots b_{63}$ be the input and the output of SubNibbles respectively. Then $b_i b_{i+16} b_{i+32} b_{i+48} = S(a_i a_{i+16} a_{i+32} a_{i+48})$, where $S(\cdot)$ represents a 4×4 S-box and is defined as in [5], $i = 0, 1, \cdots, 15$.

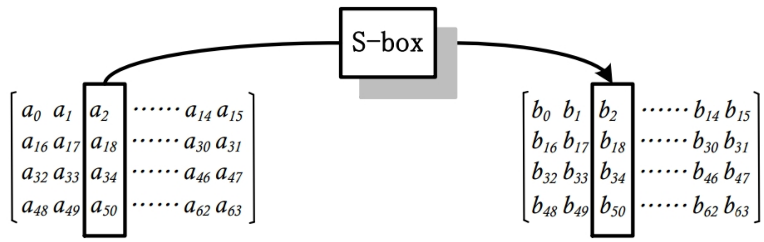

Fig. 2 SubNibbles acts on the individual columns of its input block

2.2 LinearTrans

The linear transformation uses the operations of a finite field. The finite field $\mathbb{F}_{2^{64}}$ is defined by an irreducible polynomial $p(x) = x^{64} + x^{30} + x^{19} + x + 1$, i.e., $\mathbb{F}_{2^{64}} = \mathbb{F}_2(\theta)$, where θ is a root of $p(x)$. The block $a_0 a_1 \cdots a_{63}$ corresponds to $a_0 + a_1\theta + \cdots + a_{62}\theta^{62} + a_{63}\theta^{63} \in \mathbb{F}_{2^{64}}$. The linear transformation LinearTrans is defined as LinearTrans(S_0, S_1, S_2)=$(S_0, S_1, S_2)\mathcal{A}$, where the matrix

$$\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & \alpha & \alpha + 1 \end{pmatrix}^7$$

and $\alpha = \theta^{32} \in \mathbb{F}_{2^{64}}$.

2.3 Encryption

Let $p_0 p_1 \cdots p_{m-1}$ be the plaintext and *state* be the internal state of PANDA-s after initialization. Then the encryption is described as below:

$(state, r) \leftarrow$ RoundFunc$(state, 0)$
for $t = 0$ to $m - 1$
  $c_t \leftarrow p_t \oplus r$
  $(state, r) \leftarrow$ RoundFunc$(state, p_t)$

2.4 The tag T

Use $tempt_i$ to update *state* with RoundFunc 14 times, and then output the XOR of some of state bits as the authentication tag T, where $tempt_i = adlen$ when i is even, $tempt_i = mslen$ when i is odd, *adlen* and *mslen* are the bit-length of the associated data and the plaintext repectively. More specifically,

for $i = 0$ to 13

$state \leftarrow \text{RoundFunc}(state, tempt_i)$

$T \leftarrow (w \oplus y, x \oplus z)$

3 A state recovery attack on PANDA-s

In this section we assume that an attacker has known a phase of the plaintext p_{t+i} corresponding to the ciphertext c_{t+i} after time $t \geq 0$, where $i = 0, 1, \cdots, m-1$, and m is large enough for the attacker to launch his attack. Since $r_{t+i} = p_{t+i} \oplus c_{t+i}$ for $i \geq 0$, thus the attacker knows the key words $\{r_{t+i}\}_{0 \leq i \leq m-1}$ as well. Below we first introduce some notations.

Let $(w, x, y, z, S_0, S_1, S_2)$ be the registers of PANDA-s and $(w_t, x_t, y_t, z_t, S_{0,t}, S_{1,t}, S_{2,t})$ be the state of these registers at time $t \geq 0$. For an arbitrary 64-bit word $x = x_0 x_1 \cdots x_{63}$, we denote

$$x[j] = x_j x_{j+16} x_{j+32} x_{j+48},$$

where $0 \leq j \leq 15$. Observe the update of the state of PANDA-s, and we have the following conclusion:

Lemma 1 *1. If $x_t[j]$ is known for some $0 \leq j \leq 15$, then all the sequences $\{ x_{t+i}[j] \}_{i \geq 0}$, $\{ y_{t+i}[j] \}_{i \geq 0}$, $\{ z_{t+i}[j] \}_{i \geq 0}$ and $\{ S_{0,t+i}[j] \}_{i \geq 0}$ are known;*

2. If both $x_t[j]$ and $w_t[j]$ are known for some $0 \leq j \leq 15$, then the sequence $\{ w_{t+i}[j] \}_{i \geq 0}$ is known.

Proof. It is noticed that $x_{t+i+1}[j] = x_{t+i}[j] \oplus r_{t+i}[j]$ for any $i \geq 0$, thus we have

$$x_{t+i+1}[j] = x_t[j] \oplus \bigoplus_{k=0}^{i} r_{t+k}[j].$$

If $x_t[j]$ is known, then the whole sequence $\{ x_{t+i}[j] \}_{i \geq 0}$ is known.

By the definition of the SubNibbles, we have

$$y_{t+i}[j] = S^{-1}(x_{t+i+1}[j]) \oplus x_{t+1}[j], \quad (1)$$

$$z_{t+i}[j] = S^{-1}(y_{t+i+1}[j]) \oplus y_{t+1}[j], \quad (2)$$

$$S_{0,t+i}[j] = S^{-1}(z_{t+i+1}[j]), \quad (3)$$

thus the sequences $\{y_{t+i}[j]\}_{i \geq 0}$, $\{z_{t+i}[j]\}_{i \geq 0}$ and $\{S_{0,t+i}[j]\}_{i \geq 0}$ are known.

Item 2 follows directly from $w_{t+i+1}[j] = S(w_{t+i}[j] \oplus p_{t+i}[j] \oplus x_{t+i}[j])$ for any $i \geq 0$. ■

3.1 A state recovery attack

In this section we will provide a state recovery attack against PANDA-s. The details are described as below:

1. **Get equations on $\{w_{t+i}\}_{i\geq 0}$ and $\{S_{0,t+i}\}_{i\geq 0}$.**

   By the definition of the LinearTrans, we need only three equations got at three distinct times to eliminate the variables $S_{1,t}$ and $S_{2,t}$. More precisely, the process is shown below:
   First we get three equations at time $t+1$, $t+2$ and $t+2$:

$$S_{0,t+1} = (S_{0,t} \oplus w_t, S_{1,t}, S_{2,t})\mathcal{A}\ \mathbf{e_1}, \tag{4}$$

$$S_{0,t+2} = ((S_{0,t} \oplus w_t, S_{1,t}, S_{2,t})\mathcal{A}^2 + (w_{t+1}, 0, 0)\mathcal{A})\ \mathbf{e_1}, \tag{5}$$

$$S_{0,t+3} = ((S_{0,t} \oplus w_t, S_{1,t}, S_{2,t})\mathcal{A}^3 + (w_{t+2}, 0, 0)\mathcal{A} + (w_{t+1}, 0, 0)\mathcal{A}^2)\ \mathbf{e_1}, \tag{6}$$

   where $\mathbf{e_1} = (1, 0, 0)'$ is a basic column vector.
   Second, we eliminate the variables $S_{1,t}$ and $S_{2,t}$ from the above equations and get

$$w_{t+2} \oplus C_5 w_{t+1} \oplus C_6 w_t = C_0, \tag{7}$$

   where $C_0 = C_1 S_{0,t+3} \oplus C_2 S_{0,t+2} \oplus C_3 S_{0,t+1} \oplus C_4 S_{0,t}$, and $C_1, C_2, \cdots, C_6$ are constants as defined in Appendix A.

2. **Find a multiple of $x^2 \oplus C_5 x \oplus C_6$ with coefficients 0 or 1.**

   It is noticed that the computation of the S-boxes in the SubNibbles can be done in parallel, we need to find a nonzero multiple of $x^2 \oplus C_5 x \oplus C_6$ with coefficients 0 or 1 in $F_{2^{64}}$ in order to solve equation (7) faster. Indeed we do it easily. One can check the following polynomial $f(x)$

$$f(x) = \bigoplus_{i \in I} x^i$$

   such that $x^2 \oplus C_5 x \oplus C_6 | f(x)$, where

$$\begin{aligned} I = \{ \ & 0,\ 4,\ 6,\ 7,\ 8,\ 10,\ 11,\ 14,\ 15,\ 17,\ 18,\\ & 19,\ 21,\ 23,\ 26,\ 30,\ 31,\ 32,\ 33,\ 34,\ 35,\ 37,\\ & 39,\ 43,\ 45,\ 46,\ 47,\ 49,\ 50,\ 51,\ 52,\ 55,\ 59,\\ & 61,\ 63,\ 64,\ 67,\ 68,\ 70,\ 72,\ 73,\ 74,\ 77,\ 78,\\ & 79,\ 83,\ 85,\ 89,\ 91,\ 94,\ 96,\ 97,\ 99,\ 100,\ 101,\\ & 103,\ 105,\ 106,\ 107,\ 108,\ 109,\ 110,\ 112,\ 113,\ 115,\ 117,\\ & 118,\ 119,\ 122,\ 124,\ 125,\ 127\}. \end{aligned}$$

   So we have

$$\bigoplus_{i \in I} w_{t+i} = C_t, \tag{8}$$

   where C_t is a linear relation of $S_{0,t+i}$ $(i = 0, 1, \cdots 127)$, or is viewed as an expression only on x_t.

3. **Set up the tables T_j in order to solve w_t and x_t faster.**

   Set $W_t = \bigoplus_{i \in I} w_{t+i}$. First we subdivide equation (8) into 16 equations:

   $$W_t[j] = C_t[j], \quad 0 \leq j \leq 15. \tag{9}$$

   For each equation, for example j, by Lemma 1, the left $W_t[j]$ depends on $w_t[j]$ and $x_t[j]$, and the right $C_t[j]$ depends on $x_t[j]$ ($j = 0, 1, \cdots, 15$). Let k be a positive integer such that $k \leq 15$. We consider the case $j = 0$ and further rewrite $C_t[0]$ as below:

   $$C_t[0] = F_t \oplus G_t,$$

   where F_t relies on $S_{0,t+i}[0], S_{0,t+i}[1], \cdots, S_{0,t+i}[k-1]$, that is, $x_t[0]$, $x_t[1]$, $\cdots$, $x_t[k-1]$, and G_t relies on $S_{0,t+i}[k], S_{0,t+i}[k+1], \cdots, S_{0,t+i}[15]$, that is, $x_t[k], x_t[k+1], \cdots, x_t[15]$, $0 \leq i \leq 15$. Hence we have

   $$W_t[0] = F_t \oplus G_t.$$

   Consider $k+1$ successive times t, $t+1$, $\cdots$, $t+k$, and we get an equation system

   $$\begin{cases} W_t[0] \oplus F_t = G_t \\ W_{t+1}[0] \oplus F_{t+1} = G_{t+1} \\ \quad \cdots \\ W_{t+k}[0] \oplus F_{t+k-1} = G_{t+k} \end{cases} \tag{10}$$

   and write it as $\mathcal{E}(w_t[0], x_t[0], \cdots, x_t[k-1]) = (G_t, G_{t+1}, \cdots G_{t+k})$ in short. For any $(k+1)$-tuple $(G_t, G_{t+1}, \cdots, G_{t+k})$, we set up a table T_0 to record $(w_t[0], x_t[0], \cdots, x_t[k-1])$, where

   $$\mathcal{E}(w_t[0], x_t[0], \cdots, x_t[k-1]) = (G_t, G_{t+1}, \cdots G_{t+k}).$$

   On the other hand, for any $1 \leq j \leq 15$, we set up a table T_j whose input is $(x_t[j], C_t[j])$ and output is $w_t[j]$, where $w_t[j], x_t[j], C_t[j]$ meet equation (9).

4. **Recover the state by looking up the tables T_j.**

   After the tables T_j are set up, we can recover the state $(w_t, x_t, y_t, z_t, S_{0,t}, S_{1,t}, S_{2,t})$ by looking up the tables T_j. More precisely, the process is shown below:
   (a) FOR each possible value of $(x_t[k], \cdots, x_t[15])$, DO:
   (b) Compute the $(k+1)$-tuple $(G_t, \cdots, G_{t+k})$; Look up the table T_0 to recover $w_t[0]$ and $x_t[0], \cdots, x_t[k-1]$;
   (c) Recover y_t, z_t, $S_{0,t}$ and compute C_t by x_t;
   (d) Look up the table T_j to recover $w_t[j]$ by $x_t[j]$ and $C_t[j]$ for $1 \leq j \leq 15$;
   (e) Recover $S_{1,t}$ and $S_{2,t}$ by the LinearTrans.
   (f) Check whether the recovered state $(w_t, x_t, y_t, z_t, S_{0,t}, S_{1,t}, S_{2,t})$ is correct or not. YES, output the current state and stop; NO, go to (a).

3.2 The time, data and memory complexity

In our attack we take $k = 6$. The most time-consuming operations in our attack mainly include the establishment of the table T_0 and the traversal of $(x_t[6], \cdots, x_t[15])$. As for the former, namely, establishing the table T_0, we first set up a temporary table *temp* which records $(w_t[0], x_t[0], x_t[1], x_t[2])$ for any $(G'_t, G'_{t+1}, G'_{t+2}, G'_{t+3})$, where $(w_t[0], x_t[0], x_t[1], x_t[2])$ meets the following equations:

$$\begin{cases} W_t[0] \oplus F'_t = G'_t \\ W_{t+1}[0] \oplus F'_{t+1} = G'_{t+1} \\ W_{t+2}[0] \oplus F'_{t+2} = G'_{t+2} \\ W_{t+3}[0] \oplus F'_{t+3} = G'_{t+3} \end{cases},$$

where F'_t means an expression only on $x_t[0], x_t[1], x_t[2]$ split from F_t. At the worst case, for any $(G'_t, G'_{t+1}, G'_{t+2}, G'_{t+3})$, we go through all possible values of $(w_t[0], x_t[0], x_t[1], x_t[2])$ and get the correct one, whose time complexity is at most $(2^{4\times4})^2 = 2^{32}$. Second, we set up the table T_0 by means of the temporary table *temp*. For any $(G_t, \cdots, G_{t+6})$, we guess the possible value of $(x_t[3], x_t[4], x_t[5])$ and look up the temporary table *temp* to recover $(w_t[0], x_t[0], x_t[1], x_t[2])$. Then we further check whether the recovered solution $(w_t[0], w_t[0], \cdots, w_t[5])$ meets the rest 3 equations in (10) or not, and record the correct one. The time complexity of the second step is about $2^{4\times(3+7)} = 2^{40}$. Finally we delete the temporary table *temp* as soon as the table T_0 is set up. Thus the total time complexity of setting up the table T_0 is about $2^{40} + 2^{32} \approx 2^{40}$. As for the latter, namely, the traversal of $(x_t[6], \cdots, x_t[15])$, since it has totally $2^{4\times10} = 2^{40}$ possible values, thus the time complexity of the traversal of $(x_t[6], \cdots, x_t[15])$ is about 2^{40}. So the total time complexity of our attack is about $2^{40} + 2^{40} = 2^{41}$.

As for the data complexity, in order to compute G_t, we need to compute $S_{0,t+i}[6], \cdots, S_{0,t+i}[15]$ $(i = 0, 1, 2, \cdots, 127)$. The latter needs about 131 pairs of known plaintext/ciphertext. Further we need more 6 pairs of known plaintext/ciphertext for computing $G_{t+1}, \cdots, G_{t+6}$. Thus we need totally 137 pairs of known plaintext/ciphertext, and it is very low.

As for the memory complexity, in order to store the table T_0, we need about $7 \times 2^{4\times7}\text{B} \approx 2^{31}\text{B} = 2\text{GB}$ memories, and store the tables T_j $(j = 1, 2, \cdots, 15)$, we need $15 \times 2^8\text{B} < 4\text{KB}$. Thus the memory complexity is about 2GB.

4 A forgery attack

Let (C, T) be the ciphertext and the authentication tag transported in some communication session. If an attacker has known a small phase of plaintext P which corresponds to some phase of the ciphertext C, then he can recover all corresponding plaintext of the ciphertext C and forge arbitrary legal ciphertext C' and the authentication tag T', where we assume that the plaintext P contains at least 137 of 64-bit blocks. The process is shown blow: based on the above attack, first the attacker recovers the state of PANDA-s at the beginning of processing the plaintext P with the plaintext/ciphertext pairs (P, C) ; second, since the update of the state of PANDA-s is invertible, he further recovers the initial state of PANDA-s in the process of encryption and decrypts the ciphertext C to get the whole plaintext P; finally, the attacker chooses an arbitrary plaintext P' and encrypts them with the recovered initial state to get C' and further generates the tag T'. The attacker sends the message (C', T') to a legal receiver (note: he has the legal secret key). The receiver decrypts C' and verifies T' to get P'.

References

1. A. Frier, P. Karlton, and P. Kocher, The SSL 3.0 Protocol, Netscape Communications Corp., 1996. http://home.netscape.com/eng/ssl3/ssl-toc.html.
2. T. Dierks and C. Allen, The TLS Protocol, RFC 2246, 1999.
3. S. Kent and R. Atkinson, Security Architecture for the Internet Protocol, RFC 2401, 1998.
4. CAESAR: http://competitions.cr.yp.to/index.html.
5. PANDA v1: D. Ye, P. Wang, L. Hu, L. Wang, Y. Xie, S. Sun, P. Wang, submission to CAESAR, available from: http://competitions.cr.yp.to/round1/pandav1.pdf.
6. G. Bertoni, J. Daemen, M. Peeters, G. Assche, Duplexing the sponge: Single-pass authenticated encryption and other applications, SAC 2011, LNCS 7118, pp. 320-337, 2011.
7. E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, K. Yasuda, APE: Authenticated permutation-based encryption for lightweight cryptography, http://eprint.iacr.org/2013/791.
8. Y. Sasaki and L. Wang, A Forgery Attack against PANDA-s, http://eprint.iacr.org/2014/217.

A The constants $C_1, C_2, \cdots, C_6$

The bit representation is with regard to the primitive element θ, and the most significant bit is at the left.

$$C_1 = 1000001101111000010001000110010000101100001101000000100110100100 1$$
$$C_2 = 11100101010001100111110010011011110111011111100111100110010110 00$$
$$C_3 = 0011100010111000000101010111111011100001111010001100110010101100 1$$
$$C_4 = 1000001101111000010001000110010000101100001101000000100110100100 1$$
$$C_5 = 1100110000010111011110011111000010001000110010110001110011110011$$
$$C_6 = 1000001101111000010001000110010000101100001101000000100110100100 1$$