Differential Cryptanalysis of the Stream Ciphers Py, Py6 and Pypy

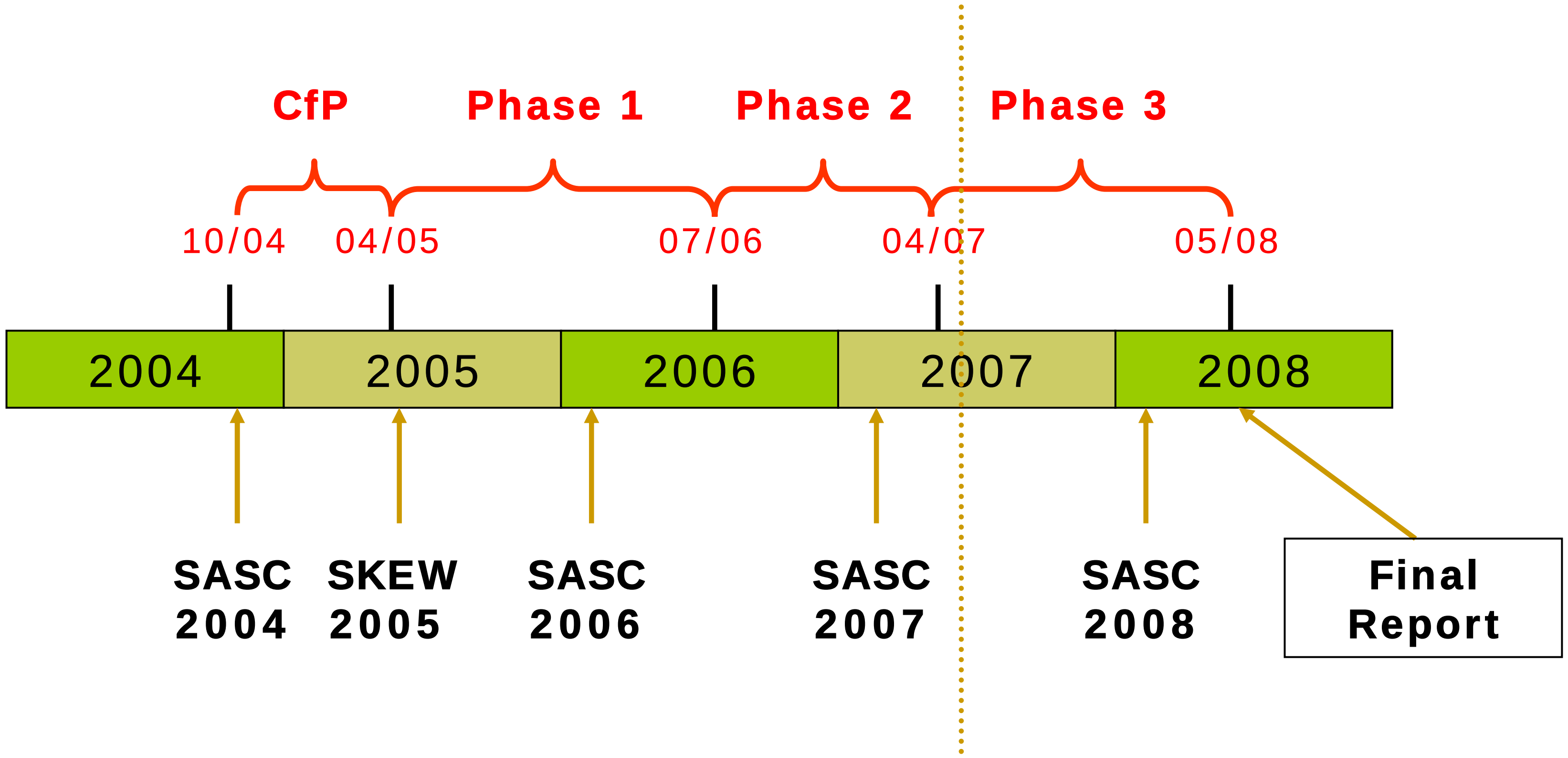
Hongjun Wu and Bart Preneel

Katholieke Universiteit Leuven
ESAT/COSIC

eSTREAM – ECRYPT Stream Cipher Project

- A multi-year project to identify new stream ciphers that might become suitable for widespread adoption
- eSTREAM considers stream ciphers for two types of environments
  - Profile 1
    - Stream ciphers for high throughput SW applications
    - 128-bit key, 64 and 128-bit IVs
  - Profile 2
    - Stream ciphers for HW with restricted resources
    - 80-bit key, 32-bit and 64-bit IVs
- Associated authentication mechanisms are also studied

eSTREAM Timeline http://www.ecrypt.eu.org/stream



eSTREAM Submission status

- 34 submissions
  - 23 for software profile
  - 25 for hardware profile

- Phase 3 candidates
  - Eight software profile ciphers:

    CryptMT Dragon HC LEX NLS Rabbit Salsa20 SOSEMANUK
  - Eight hardware profile ciphers:

    DECIM Edon80 F-FCSR Grain MICKEY Moustique Pomaranch Trivium

Overview

1. Py, Pypy and Py6 (1)

Py, Pypy

designed by Biham and Seberry

submissions to eSTREAM

eSTREAM phase 2 software focus ciphers

Main Features of Py and Pypy:

RC4 type stream ciphers

Rolling array: using extra memory space to achieve fast array rotation !!

Fast: Py – 2.85 cycles/byte

Pypy – 4.88 cycles/byte (Pentium M)

1. Py, Py6 and Pypy (2)

Py6 is the reduced version of Py (smaller internal state)

Internal state size: Py (10,400 bits)

Py6 (2,592 bits)

=> the key/IV setup of Py6 is about 3 times faster than that of Py

1. Py, Py6 and Pypy (3)

Previous Distinguishing Attacks against Py and Py6

Py – 2^{89} outputs (FSE'06, Paul, Preneel and Sekar)
improved to 2^{72} outputs (Crowley, 2006)

Py6 – 2^{69} outputs (Asiacrypt'06, Paul and Preneel)

Pypy – proposed to resist the distinguishing attacks by
discarding half of the outputs of Py

1. Py, Py6 and Pypy (4)

Our attacks against Py, Pypy and Py6

1) Identical keystreams appear with high probability

Py and Pypy – every 2^{16} IVs with special differences

Py6 – every 2^{7} IVs with special differences

=> insecure

1. Py, Py6 and Pypy (5)

2) Key recovery attack against Py and Pypy

for 16-byte key and 16-byte IV, with 2^{23} chosen IVs, effective key size reduced to 3 bytes

for 32-byte key and 32-byte IV, with 2^{24} chosen IVs, effective key size reduced to 3 bytes

=> insecure

2. Identical Keystreams of Py, Pypy (1)

The Key/IV setup of Py and Pypy are identical

=> The attacks against Py and Pypy are the same

IV setup is non-invertible

=> Resulting in collision of the internal state

=> Identical keystreams for different IVs

2. Identical Keystreams of Py, Pypy (2)

We examine part of the IV setup of Py and Pypy

IV is applied to update *s* and *EIV* twice!

Y: secret array with 260 32-bit elements

P: 8-bit permutation table (with 15-bit secret info)

s: 32-bit internal state

EIV: temporary byte array

```
for(i=0; i<ivsizeb; i++)
{
      s = s + iv[i] + Y(YMININD+i);
      u8 s0 = P(s&0xFF);
      EIV(i) = s0;
      s = ROTL32(s, 8) ^ (u32)s0;
}
for(i=0; i<ivsizeb; i++)
{
      s = s + iv[i] + Y(YMAXIND-i);
      u8 s0 = P(s&0xFF);
      EIV(i) += s0;
      s = ROTL32(s, 8) ^ (u32)s0;
}
```

2. Identical Keystreams of Py, Pypy (3)

In the above algorithm, IV is applied to update *s* and the array *EIV* twice. After that *s* and *EIV* are applied to update the array *Y* and permutation table *P*

If different IVs result in the same *s* and *EIV*, then the keystreams will be identical

The attack is to find the special IV differences that lead to the same *s* and *EIV*

2. Identical Keystreams of Py, Pypy (4)

Attack 1. Two adjacent IV bytes are different:

$$iv_1[i] \oplus iv_2[i] = 1\ , \quad iv_1[i+1] \neq iv_2[i+1]$$

At the *i*-th step in the first 'for' loop,

```
s = s + iv[i] + Y(YMININD+i);
u8 s0 = P(s&0xFF);
EIV(i) = s0;
s = ROTL32(s, 8) ^ (u32)s0;
```

At the end of the *i*th step, $EIV_1[i] \neq EIV_2[i]$, $s_1 - s_2 = 0x100 + \delta_1$

2. Identical Keystreams of Py, Pypy (5)

$$iv_1[i] \oplus iv_2[i] = 1, \quad iv_1[i+1] \neq iv_2[i+1]$$

At the (i+1)-th step in the first 'for' loop,

```
s = s + iv[i+1] + Y(YMININD+i+1);
u8 s0 = P(s&0xFF);
EIV(i+1) = s0;
s = ROTL32(s, 8) ^ (u32)s0;
```

At the end of the (i+1)-th step, if $iv_2[i+1] - iv_1[i+1] = \delta_1$, then the two *s* will be identical, and $EIV_1[i+1] = EIV_2[i+1]$ with high probability => for the others steps in the first 'for' loop, the two *s* remain identical

2. Identical Keystreams of Py, Pypy (6)

$$iv_1[i] \oplus iv_2[i] = 1, \quad iv_1[i+1] \neq iv_2[i+1]$$

At the *i*-th step in the second ‘for’ loop,

```
s = s + iv[i] + Y(YMAXIND-i);
u8 s0 = P(s&0xFF);
EIV(i) += s0;
s = ROTL32(s, 8) ^ (u32)s0;
```

At the end of the *i*-th step, $EIV_1[i] = EIV_2[i]$ with probability $\frac{1}{255}$ and $s_1 - s_2 = 0x100 + \delta_2$

2. Identical Keystreams of Py, Pypy (7)

$$iv_1[i] \oplus iv_2[i] = 1, \quad iv_1[i+1] \neq iv_2[i+1]$$

At the (i+1)-th step in the second 'for' loop,

```
s = s + iv[i+1] + Y(YMAXIND-i-1);
u8 s0 = P(s&0xFF);
EIV(i+1) += s0;
s = ROTL32(s, 8) ^ (u32)s0;
```

At the end of the (i+1)-th step, if $iv_2[i+1] - iv_1[i+1] = \delta_2$, then the two *s* will be identical, and $EIV_1[i+1] = EIV_2[i+1]$ with high probability

2. Identical Keystreams of Py, Pypy (8)

$$iv_1[i] \oplus iv_2[i] = 1, \quad iv_1[i+1] \neq iv_2[i+1]$$

From the simulation, the above difference results in identical keystream with probability about $2^{-23.2}$.

With the lsb of IV[i] and all the 8 bits of IV[i+1] choose all the possible values, 2^{16} IV pairs can be obtained from these 512 IVs.

=> one identical keystream pair appears for about 2^{16} IVs

2. Identical Keystreams of Py, Pypy (9)

Attack 2. Three-byte IV differences

$$iv_1[i] - iv_2[i] = iv_2[i+4] - iv_1[i+4]$$
$$iv_1[i+1] \neq iv_2[i+1]$$

(details omitted)

From the simulation, the above difference results in identical keystream with probability about 2^{-23}.

=> one identical keystream pair appears for about 2^{16} IVs

3. Key Recovery Attacks (1)

From the collision of the internal state, a number of equations can be generated to recover the key of Py and Pypy

Three steps:

1) Recovering part of the array *Y* from the IV setup

2) Recovering the 15-bit secret information in *P* from IV setup

3) Recovering the key from the key setup

3. Key Recovery Attacks (2)

Recovering part of the array *Y* from the IV setup

```
for(i=0; i<ivsizeb; i++)   {
             s = s + iv[i] + Y(YMININD+i);
            u8 s0 = P(s&0xFF);
            EIV(i) = s0;
            s = ROTL32(s, 8) ^ (u32)s0;
}
```

In the above algorithm (the first ‘for’ loop), if two keystreams are identical, the difference in *s* will be eliminated by the difference in IVs

3. Key Recovery Attacks (3)

Recovering part of the array *Y* from the IV setup

For the two-byte IV differences, the following equation can be obtained:

$$(P(B(s^{\theta}_{i-1,0} + iv_1[i] + Y_{-3+i,0})) \oplus B(s^{\theta}_{i-1,3} + Y_{-3+i,3} + \xi_i)) + 256 + iv_1[i+1]$$
$$= (P(B(s^{\theta}_{i-1.0} + iv_2[i] + Y_{-3+i,0})) \oplus B(s^{\theta}_{i-1.3} + Y_{-3+i,3} + \xi_i)) + iv_2[i+1]$$

where iv_{θ} is a fixed IV, the first *i* bytes of iv_1 and iv_2 are identical to that of iv_{θ}, $iv_1[i] \oplus iv_2[i] = 1$ and $iv_1[i+1] \neq iv_2[i+1]$

with 7 equations above, the values of $B(s^{\theta}_{i-1,0} + Y_{-3+i,0})$ and $B(s^{\theta}_{i-1,3} + Y_{-3+i,3} + \xi_i)$ can be determined almost uniquely.

3. Key Recovery Attacks (4)

Recovering part of the array *Y* from the IV setup

From the values of $B(s^{\theta}_{i-1,0} + Y_{-3+i,0})$ and $B(s^{\theta}_{i-1,3} + Y_{-3+i,3} + \xi_i)$, the value of $s^{\theta}_{i,0}$ can be computed since

$$s_{i,0} = P(B(s_{i-1,0} + iv[i] + Y(-3+i))) \oplus B(s_{i-1,3} + Y(-3+i) + \xi_i)$$

From the values of $B(s^{\theta}_{i,0} + Y_{-3+i+1,0})$ and $s^{\theta}_{i,0}$, the value of $Y_{-3+i+1,0}$ is known

=> The values of $Y_{-3+i,0}$ for $3 \le i \le ivsizeb - 3$ can be determined with about $(ivsizeb - 4) \times 2^{19}$ IVs.

3. Key Recovery Attacks (5)

Recovering part of the array *Y* from the IV setup

Exploiting to the second ‘for’ loop

```
for   (i=0; i<ivsizeb; i++)   {
       s = s + iv[i] + Y(YMAXIND-i);
      u8 s0 = P(s&0xFF);
      EIV(i) += s0;
      s = ROTL32(s, 8) ^ (u32)s0;
}
```

=> The values of $Y_{256-i,0}$ for $3 \le i \le ivsizeb - 3$ can be determined with about $(ivsizeb - 4) \times 2^{19}$ IVs.

3. Key Recovery Attacks (6)

Recovering the 15-bit secret information in Permutation P

When recovering part of *Y*, only the difference elimination in *s* is considered.

Now consider that the difference in *EIV* is also eliminated in the second ‘for’ loop for identical keystream pair, the 15-bit secret information in P can be easily determined (details omitted here)

3. Key Recovery Attacks (7)

Recovering the key

```
for(i=YMININD, j=0; i<=YMAXIND; i++)  {
        s = s + key[j];
        s0 = internal_permutation[s&0xFF];
        Y(i) = s = ROTL32(s, 8) ^ (u32)s0;
        j = (j+1) mod keysizeb;
}
```

From the last part of the key setup given above, the following equation is obtained to link the key bytes:

$$B(Y_{-3+i,0} + key[i+1 \bmod keysizeb] + \xi'_i)$$
$$\oplus P'(B(Y_{-3+i+3,0} + key[i+4 \bmod keysizeb])) = Y_{-3+i+4,0}$$

3. Key Recovery Attacks (8)

Recovering the key

1) If three key bytes key[4], key[5], key[6] are guessed, then the one-bit error in the above equation for other *i's* can be computed recursively.

   => each equation leaks one-byte key information

2) From the values of $Y_{-3+i,0}$ and $Y_{256-i,0}$ for $3 \leq i \leq ivsizeb - 3$ we obtain $2 \times (ivsizeb - 9)$ equations

So the maximum leaked key information is $2 \times (ivsizeb - 9)$ bytes (the number of equations may be more than the number of key bytes being involved, thus the actual leaked key information may be less)

3. Key Recovery Attacks (9)

Recovering the key

1) For 16-byte key and 16-byte IV, 13 key bytes involved in 14 equations

   => recovering 13 key bytes

2) For 32-byte key and 32-byte IV, 29 key bytes involved in 46 equations

   => recovering 29 key bytes

4. Security of Py6

Running the C code of Py6 with the 3-byte IV differences:

$iv_1[i] - iv_2[i] = 32$, $iv_1[i+1] \neq iv_2[i+1]$, $iv_1[i+1] \gg 6 = iv_2[i+1] \gg 6$,
and $iv_2[i+5] - iv_1[i+5] = 8$ $(i \geq 2)$

identical keystream pair appears with probability $2^{-11.45}$. (extremely weak)

with 2^7 chosen IVs

(set the 5th lsb of *iv*[i] and 6 least significant bits of *iv*[i+1] to all the values)

=> about 2^{12} IV pairs

=> about one identical keystream pair

5 TPy, TPy6 and TPypy

TPy, TPy6 and TPypy:

tweaks proposed by Biham and Seberry in 2007

main change: using the IV only once in the IV setup

secure against the attacks given in this presentation

more security analysis needed for the tweaked versions

6 Conclusions

1. The initializations of Py, Pypy and Py6 are insecure

2. IV setup of stream cipher

   To ensure that the IV setup is invertible to preclude attacks based on internal collisions.

3. Similar problems:
   ANSI Retail MAC, COMP-128, Hasty Pudding,
   ABCvx (x<4),... and many more

Thank you!

Q & A