Key Recovery Attack on Py and Pypy with Chosen IVs

Hongjun Wu and Bart Preneel

Katholieke Universiteit Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{wu.hongjun,bart.preneel}@esat.kuleuven.be

Abstract. In this paper we extend our previous attack to recover the key of Py and Pypy. If the IV size is at least ten bytes, the chosen IV attack can be applied to recover the key information of Py and Pypy. In general, $ivsiveb - 9$ bytes of the key can be recovered, where *ivsizeb* indicates the size of the IV in bytes. For example, for 256-bit key and 256-bit IV, the key is reduced to 72 bits with about 2^{24} chosen IVs. For 128-bit key and 256-bit IV, the key can be recovered easily with about 2^{24} chosen IVs.

1 Introduction

Py [1] and Pypy [2] are stream ciphers submitted to the ECRYPT eSTREAM project. A distinguishing attack on Py was found by Paul, Preneel and Sekar [4]. In that attack, the keystream can be distinguished from random with $2^{89.2}$ outputs. Later, the attack was improved by Crowley [3], and the data required in the attack is reduced to 2^{72}. In order to resist the distinguishing attack on Py, the designers decided to discard half of the outputs, i.e., the first output of the two outputs at each step is discarded. The new version is called Pypy.

The IV setup of Py and Pypy are identical. In [5], we showed that there is serious flaw in the IV setup of Py and Pypy. For IVs with special difference, two keystreams can be identical for every 2^{16} IVs.

In this paper, we extend our attack to recover the key of Py and Pypy. We show that Py and Pypy are vulnerable to the chosen IV attack.

This paper is organized as follows. In Sect. 2, we illustrate the IV setup of Py and Pypy. Section 3 describes the attack of generating identical keystreams. The key recovery attack is given in Sect. 4. Section 5 concludes this paper.

2 The Initialization of Py and Pypy [1]

The initializations of Py and Pypy are identical. The initialization consists of two stages: key setup and IV setup. In key setup, the key is expanded into the internal state of the cipher. In the IV setup, the IV is introduced into the state.

2.1 The key setup

In the following, we give the description of the key setup. P is an array with 256 8-bit elements. Y is an array with 260 32-bit elements, s is 32-bit. $YMININD = -3$, $YMAXIND = 256$. The table 'internal_permutation' is a constant permutation table with 256 elements.

```
keysizeb=size of key in bytes;
ivsizeb=size of IV in bytes;
YMININD = -3; YMAXIND = 256;
s = internal_permutation[keysizeb1];
s = (s<<8) | internal_permutation[(s ^ (ivsizeb1))&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[0])&0xFF];
s = (s<<8) | internal_permutation[(s ^ key[keysizeb1])&0xFF];
for(j=0; j<keysizeb; j++)
{
     s = s + key[j];
     s0 = internal_permutation[s&0xFF];
     s = ROTL32(s, 8) ^ (u32)s0;
}
/* Again */
for(j=0; j<keysizeb; j++)
{
     s = s + key[j];
     s0 = internal_permutation[s&0xFF];
     s ^= ROTL32(s, 8) + (u32)s0;
}
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;
}
```

2.2 The IV setup

The IV setup is given below. EIV is a byte array with the same size as the IV.

```
/* Create an initial permutation */
u8 v= iv[0] ^ ((Y(0)>>16)&0xFF);
u8 d=(iv[1 mod ivsizeb] ^ ((Y(1)>>16)&0xFF))|1;
for(i=0; i<256; i++)
{
     P(i)=internal_permutation[v];
     v+=d;
```

```
}
/* Now P is a permutation */
/* Initial s */
s = ((u32)v<<24)^((u32)d<<16)^((u32)P(254)<<8)^((u32)P(255));
s ^= Y(YMININD)+Y(YMAXIND);
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;
}
/* Again, but with the last words of Y, and update EIV */
for(i=0; i<ivsizeb; i++)
{
     s = s + iv[i] + Y(YMAXINDi);
     u8 s0 = P(s&0xFF);
     EIV(i) += s0;
     s = ROTL32(s, 8) ^ (u32)s0;
}

/*updating the rolling array and s*/
for(i=0; i<260; i++)
{
     u32 x0 = EIV(0) = EIV(0)^(s&0xFF);
     rotate(EIV);
     swap(P(0),P(x0));
     rotate(P);
     Y(YMININD)=s=(s^Y(YMININD))+Y(x0);
     rotate(Y);
}
s=s+Y(26)+Y(153)+Y(208);
if(s==0)
     s=(keysizeb*8)+((ivsizeb*8)<<16)+0x87654321;
```

3 Identical Keystreams

At the beginning of the IV setup, only 15 bits of the IV (IV[0] and IV[1]) are applied to initialize the array P and s (the least significant bit of IV[1] is not used). For an IV pair, if those 15 bits are identical, then the resulting P are the same. Then we notice that the IV is applied to update the values of s and EIV as follows.

```
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;
}
```

From [5], we know that for an IV pair with special difference, identical keystreams appear with probability as high as $2^{-22.9}$. The identical keystreams also leaks the information of the secret key, as illustrated in the rest of the paper.

4 Key Recovery Attack on Py and Pypy

In this section, we use the following IVs differences to illustrate the key recovery attack on Py and Pypy (the other differences can be used in a similar way to recover the key). Let two IVs iv_1 and iv_2 are different at only two bytes, $iv_1[i] \oplus iv_2[i] = 1$ and $iv_1[i+1] \neq iv_2[i+1]$ $(i \geq 1)$. Let the least significant bit of $iv_1[i]$ be 1. This type of IV pair results in identical keystreams with probability $2^{-23.2}$.

4.1 Recover part of Y from the identical IV pairs

We are interested in the following algorithm in 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;
}
```

Denote the s at the end of the ith step of this algorithm as s_i, and denote the least and most significant bytes of s_i as $s_{i,0}$ and $s_{i,3}$, respectively. Denote the least and most significant bytes of $Y(i)$ as $Y_{i,0}$ and $Y_{i,3}$, respectively. Denote ξ as a binary noise with value 0 with probability 0.5. Denote $B(x)$ as a function that gives the least significant byte of x. For the above IV pair, if the two keystreams are identical, then we know that at the end of the $(i+1)$th step, the two s are the same. It means that

$$\begin{aligned}&(P(B(s_{i-1,0} + iv_1[i] + Y_{-3+i,0})) \oplus B(s_{i-1,3} + Y_{-3+i,3} + \xi_1)) + 256 + iv_1[i+1] \\ &= (P(B(s_{i-1,0} + iv_2[i] + Y_{-3+i,0})) \oplus B(s_{i-1,3} + Y_{-3+i,3} + \xi_2)) + iv_2[i+1] \quad (1)\end{aligned}$$

where ξ_1 and ξ_2 are introduced by the carry bits when $IV[i]$ and $Y(-3+i)$ are introduced, and $\xi_1 = \xi_2$ with probability very close to 1 since the $IV[i]$ has negligible effect on the value of ξ_1 and ξ_2.

Suppose that there are several equations (1) with the same $s_{i-1,0}$ and $s_{i-1,3}$ (it can be achieved if the first i bytes of all the IVs are the same), then we can recover the values of $B(s_{i-1,0} + Y_{-3+i,0})$ and $B(s_{i-1,3} + Y_{-3+i,3} + \xi)$. From the experiment, we find that if there are two equations (1), in average the values can be recovered together with 5.22 wrong values. If there are three equations (1), in average the values can be recovered together with 1.29 wrong values. If there are four equations (1), in average the values can be recovered together with 0.54 wrong values. If there are five equations (1), in average the values can be recovered together with 0.25 wrong values. If there are six equations (1), in average the values can be recovered together with 0.12 wrong values. If there are seven equations (1), in average the values can be recovered together with 0.06 wrong values. It shows that the values of $B(s_{i-1,0} + Y_{-3+i,0})$ and $B(s_{i-1,3} + Y_{-3+i,3} + \xi)$ can be determined with only a few equations (1).

After recovering a number of $B(s_{i-1,0} + Y_{-3+i,0})$ and $B(s_{i-1,3} + Y_{-3+i,3} + \xi)$ for $i \geq 1$, we proceed to recover part of the information of the array Y. Note that

$$s_{i,0} = P(B(s_{i-1,0} + iv[i] + Y_{-3+i,0})) \oplus B(s_{i-1,3} + Y_{-3+i,3} + \xi) \qquad (2)$$

Denote iv^θ as a fixed IV with the first i bytes being identical to all the IVs with differences at $iv[i]$ and $iv[i+1]$. From $iv^\theta[i]$, and the values of $B(s^\theta_{i-1,0} + Y_{-3+i,0})$, $B(s^\theta_{i-1,3} + Y_{-3+i,3} + \xi)$ and (2), we find the values of $s^\theta_{i,0}$. Suppose that when we introduce the IV difference at the $(i+1)$th and $(i+2)$th bytes, the first $i+1$ bytes of each IV are identical to that of iv^θ. Then we recover the value of $B(s^\theta_{i,0} + Y_{-3+i+1,0})$. From the values of $B(s^\theta_{i,0} + Y_{-3+i+1,0})$ and $s^\theta_{i,0}$, we determine the value of $Y_{-3+i+1,0}$.

Generating the equations (1). The above attack can be successful if we can find several equations (1) with the same $s_{i-1,0}$ and $s_{i-1,3}$. In the following, we illustrate how to obtain these equations for $2 \leq i \leq ivsizeb - 3$. To ensure that the same $s_{i-1,0}$ and $s_{i-1,3}$ appear in these equations, we require that the values of $iv_1[j]$ and $iv_2[j]$ $(0 \leq j < i)$ are fixed. Let the least significant bit of $iv[i]$ and the 8 bits of $iv[i+1]$ choose all the 512 values, and the other 119 bits remain unchanged, then we obtain a $255 \times 255 \approx 2^{16}$ desired IV paris. We call these 512 IVs as a desired IV group. From the analysis given in [5], this type of IV pair results in identical keystreams with probability $2^{-23.2}$, we thus obtain $\frac{2^{-23.2}}{2^{16}} = 2^{-7.2}$ identical keystream pairs from one desired IV group. It means that we can obtain $2^{-7.2}$ equations (1) from one desired IV group. We modify the values of the 7 most significant bits of $iv_1[i]$ and $iv_2[i]$, and 3 bits of $iv_1[i+2]$ and $iv_2[i+2]$, then we obtain $2^7 \times 2^3 = 2^{10}$ desired IV groups. From these desired IV groups, we obtain $2^{10} \times 2^{-7.2} = 7$ equations (1). There are $2^7 \times 2^3 \times 2^9 = 2^{19}$ IVs being used in the attack. To find all the $s_{i,0}$ for $2 \leq i \leq ivsizeb - 3$, we need $(ivsizeb - 4) \times 2^{19}$ IVs in the attack.

To ease the generation of the equations, we can set one fixed iv^{θ}. When the differences are introduced at $iv[i]$ and $iv[i+1]$, then all the first i bytes of each IV are chosen to be identical to that of iv^{θ}.

We are able to recover $s_{i,0}$ for $2 \leq i \leq ivsizeb - 3$. It indicates that we can recover the values of $Y_{-3+i,0}$ for $3 \leq i \leq ivsizeb - 3$.

4.2 Recover the key

In the above analysis, we recovered the values of $Y_{-3+i,0}$ for $3 \leq i \leq ivsizeb - 3$. Then we look at the last part of the key schedule.

```
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 above algorithm, we obtain the following relation:

$$B(Y_{-3+i,0} + key[i+1] + \xi_i') \oplus P(B(Y_{-3+i+3,0} + key[i+4])) = Y_{-3+i+4}\,, \quad (3)$$

where ξ_i' indicates the carry bit noise introduced by $key[i+2]$ and $key[i+3]$, it is computed as $\xi_i' \approx (key[i+2] + Y_{-3+i+1,0}) \gg 8$. The value of the binary ξ_i' is 0 with probability about 0.5.

Once the values of $Y_{-3+i,0}$ $(3 \leq i \leq ivsizeb - 3)$ are known, we find an equation (3) linking $key[i+1]$ and $key[i+4]$ for $(3 \leq i \leq ivsizeb - 7)$. Each relation leaks at least 7 bits of $key[i+1]$ and $key[i+4]$. There are $(ivsizeb - 9)$ relations, so at least $7 \times (keysizeb - 9)$ bits information of the key is leaked. We note that the randomness of ξ_i' does not affect the overall attack (once we guess the values of $key[4]$, $key[5]$ and $key[6]$, then we obtain the other key bytes $key[j]$ $(6 < j \leq ivsizeb - 3)$ and all the ξ_j' $(3 \leq j \leq ivsizeb - 7)$). Thus the leaked information is $(ivsizeb - 9)$ bytes.

If the IV size is 9 more bytes larger than the key size, then all information of the key can be recovered.

4.3 Other attacks

In [5], the IV differences at three bytes are introduced. This type of IV differences can also be applied to recover the key in a similar approach. We ignore here the details of the attack exploiting the three-byte IV differences.

5 Conclusion

Py and Pypy are vulnerable to the chosen IV attack.

References

1. E. Biham, J. Seberry, "Py: A Fast and Secure Stream Cipher Using Rolling Arrays." Available at http://www.ecrypt.eu.org/stream/ciphers/py/py.ps .
2. E. Biham, J. Seberry, "Pypy: Another Version of Py." Available at http://www.ecrypt.eu.org/stream/papersdir/2006/038.pdf
3. P. Crowley, "Improved Cryptanalysis of Py." Available at http://www.ecrypt.eu.org/stream/papersdir/2006/010.pdf .
4. S. Paul, B. Preneel, S. Sekar, "Distinguishing Attack on the Stream Cipher Py." *Fast Software Encryption – FSE 2006*, to appear.
5. H. Wu, B. Preneel, "Attacking the IV Setup of Py and Pypy." Available at http://www. ecrypt.eu.org/stream/papersdir/2006/050.pdf