Distinguishing Attacks on the Stream Cipher Py (Roo)

Speaker: Souradyuti Paul

(work jointly with B. Preneel and G. Sekar)

Computer Security and Industrial Cryptography (COSIC)
Department of Electrical Engineering-ESAT
Katholieke Universiteit Leuven, Belgium

Email: Souradyuti.Paul@esat.kuleuven.be
Outline

- Py and a Short History
- Description of Py
- Basic Idea of Attack and Assumptions
- Observation: Input-Output Correlation
- The Bias and the Distinguisher
- Complexities of the Attack
- Biases in other Pairs of Bits
- Conclusions and Remarks
Py and the evolution of RC4

- RC4 (1987) by Rivest
- VMPC (2004) by Zoltak
- HC-256 (2004) by Wu
- GGHN (2005) by Gong et al.
- Py, Py6 (2005) by Biham and Seberry
- PyPy (2006) by Biham and Seberry
Stage I: Key/IV set-up of Py

Key/IV set-up Algo (Step 1)

Key

IV

P

Y

Initialization

256 bits

128 bits

256x8 bits

260x32 bits

32 bits

256x8 bits

260x32 bits
Stage II: Keystream bytes generation of Py

Round 1

Round 2

Round 3

Output 1
XOR
Plaintext 1
Ciphertext 1

Output 2
XOR
Plaintext 2
Ciphertext 2

Output 3

...
Single round of Py: $i$th round

```
000  001  ...  093  094  095  ...  253  254  255
165  001  ...  093  233  233  ...  096  143  165
```

```
-3   -2   -1   0   ...   094  095  ...   255  256
X'   Y   Z  M  ...  N   P  ...  O  X'
```

$O(2,i)$  $O(1,i)$
The basic idea of our attacks and assumptions

- **Assumption**: Key/IV set-up is perfect
- **Focus**: mixing of bits in a round
- **Identify**: a class of internal states introducing bias in the outputs
- **Observe**: rest of the states do not cancel bias (reason: rigorous mixing)
- **Conclude**: output is biased on a randomly chosen internal state
Main observation: A lucky case in the array $P$

<table>
<thead>
<tr>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>116</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>208</td>
<td>208</td>
<td>208</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>239</td>
<td>239</td>
<td>239</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$Y \equiv -18 \mod 32$

$X \equiv 7 \mod 32$

$X+1 \equiv 254 \mod 32$
Outputs at 1st and 3rd rounds

Round 1

\[ O(1,1) = (S \text{ XOR } G) + H \]

Round 2

\[ O(2,3) = (S \text{ XOR } H) + G \]

Bias in the lsb’s.

\[ z = O(1,1)[0] \text{ XOR } O(2,3)[0] \]

\[ P(z=0) = 1 \]
Quantifying the bias

- The lucky case $L$ occurs with prob. $2^{-41.9}$

- For the lucky case the $P(z=0|L)=1$

- For the rest of the cases, we observe that $P(z=0|L') = 1/2$ (see the paper)

- The overall prob. $P(z=0) = \frac{1}{2} \cdot (1 + 2^{-41.9})$
The distinguisher (I)

- **Optimal Distinguisher**: If # of 0’s ≥ # of 1’s then Py else Random
- The advantage is close to 0% for n=1
- If n=2^{84.7} then advantage is more than 50%
The distinguisher (II)

Requirements:
- \# of Key/IV’s = 2^{84.7}
- key stream per Key/IV = 24 bytes
- time = 2^{84.7} \cdot T_{ini}

The distinguisher works
- within Py specifications
- with less than exhaustive search
The distinguisher (III)

- A variant of the distinguisher works in a single keystream but takes longer outputs than specified $2^{64}$.

- To reduce work load, a hybrid distinguisher with many key/IV’s and less than $2^{64}$ output bytes per Key/IV is also possible within the scope of the Py specification.
Bias in other pairs of bits

\[ O(1,1) = (S \text{ XOR } G) + H \]
\[ O(2,3) = (S \text{ XOR } H) + G \]

Bias in the \( i \)th bits.

\[ z = O(1,1)[i] \text{ XOR } O(2,3)[i] \]
\[ P(z=0) = 1/2 + \mu \]
Conclusion and remarks

- **Latest News:** Paul Crowley reduced the workload of the distinguisher to $2^{72}$ by combining all the individual biased bits.

- The modified version PyPy certainly does not contain this weakness.

- A completely unsubstantiated personal opinion: PyPy may come under distinguishing attack with workload less than exhaustive search.
Thanks.