# RC4 : Rivest Cipher 4

## Generalities

| System | Type | Year |
|---|---|---|
| RC2 | Block | 1987 |
| **RC4** | **Stream** | **1987** |
| RC5 | Block | 1994 |
| RC6 | Block | 1997 |

- RC4 was certainly the most used cipher : WEP, WPA, XBOX, Skype. . .
- . . . despite many people promising to stop using it : `https://www.rc4nomore.com`
- **However, RC4 is weak !**

# RC4 : characteristics

- **Key :** length $40 \leq \ell \leq 128$ bits.
- **Internal state :**
  — RC4 work on an array $S$ of 256 bytes.
  — 2 counters.
- **Initialization function** depends on the key $K$.
- **Key stream generation :**
  — Update function $f$.
  — Filtering function $\phi$.

- *To simplify the description, let reduce the array size of RC4 to 8 bytes (and let call it TinyRC4).*

# TinyRC4

## Initialization function

- The goal of the initialization function is to set the internal state with a **pseudo-random permutation** which depends on the key and on the positive integers smaller than 8 .

- After initialization (example) : $\{0, 1, 3, 7, 4, 6, 5, 2\}$

# TinyRC4

## Initialization function

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | $i$ | $j$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | $i$ | $j$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 0 | 0 |

# TinyRC4

## Initialization function

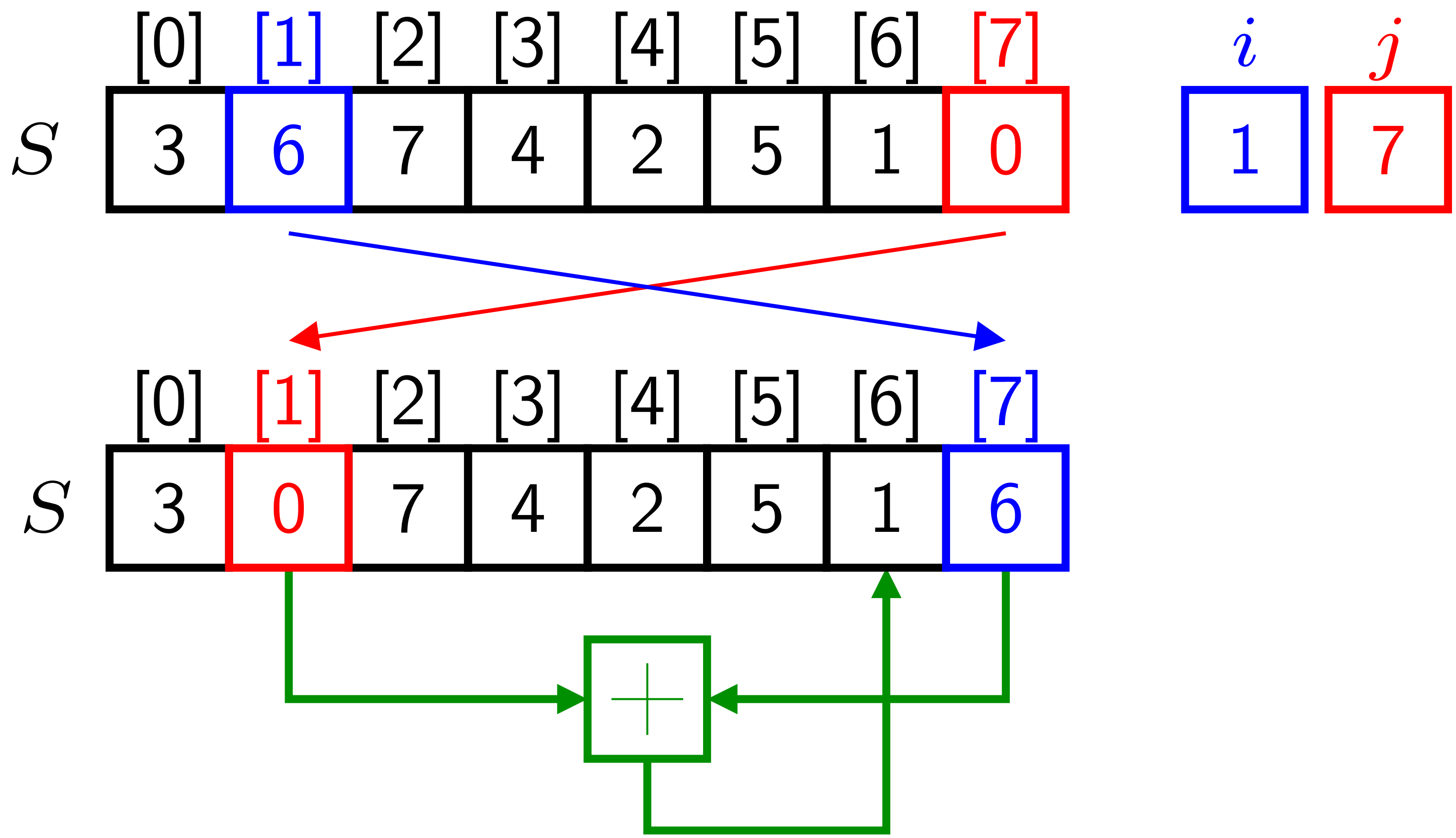
| | [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|---|
| $K$ | 2 | 114 | 84 | 0 | 201 | 48 |

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|---|---|---|---|---|---|---|---|---|
| $S$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| $i$ | $j$ |
|---|---|
| 0 | 0 |

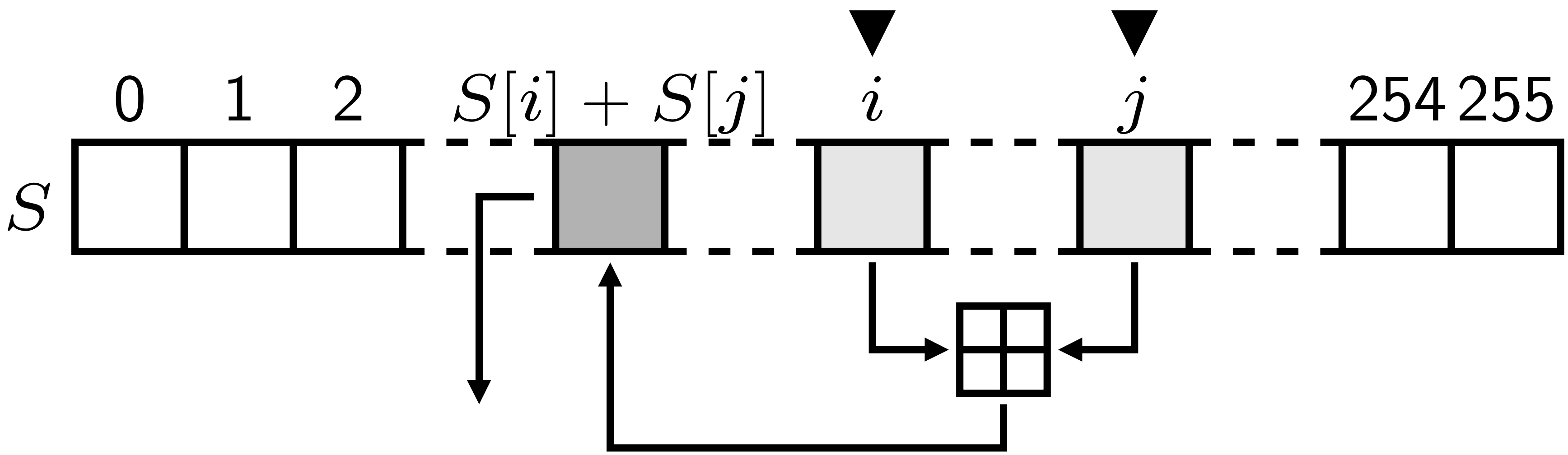
- Update :
  — $i_{t+1} = i_t + 1$
  — $j_{t+1} = j_t + S[i_t] + K[i_t] \bmod 8$
  — $\texttt{swap}(S[i_t], S[j_{t+1}])$

# Keystream generation




- If it works for TinyRC4, it will work for RC4.

# Keystream generation



# Code C of RC4

- A few line!

```
int i=0,j=0,x,t;
for (x=0; x < len; ++x)
{
  i = (i + 1) % 256;
  j = (j + state[i]) % 256;
  t = state[i];
  state[i] = state[j];
  state[j] = t;
  out[x] = state[(state[i] + state[j]) % 256];
}
```

# Code C de RC4

## Initialisation

```
int i,j=0,t;
for (i=0; i < 256; ++i)
  state[i] = i;
for (i=0; i < 256; ++i)
{
  j = (j + state[i] + key[i % len]) % 256;
  t = state[i];
  state[i] = state[j];
  state[j] = t;
}
```

# Cryptanalysis

- **Key recovery attacks** attempt **to recover the secret key** of the stream cipher from the keystream.
- The most powerful attack !

# Cryptanalysis

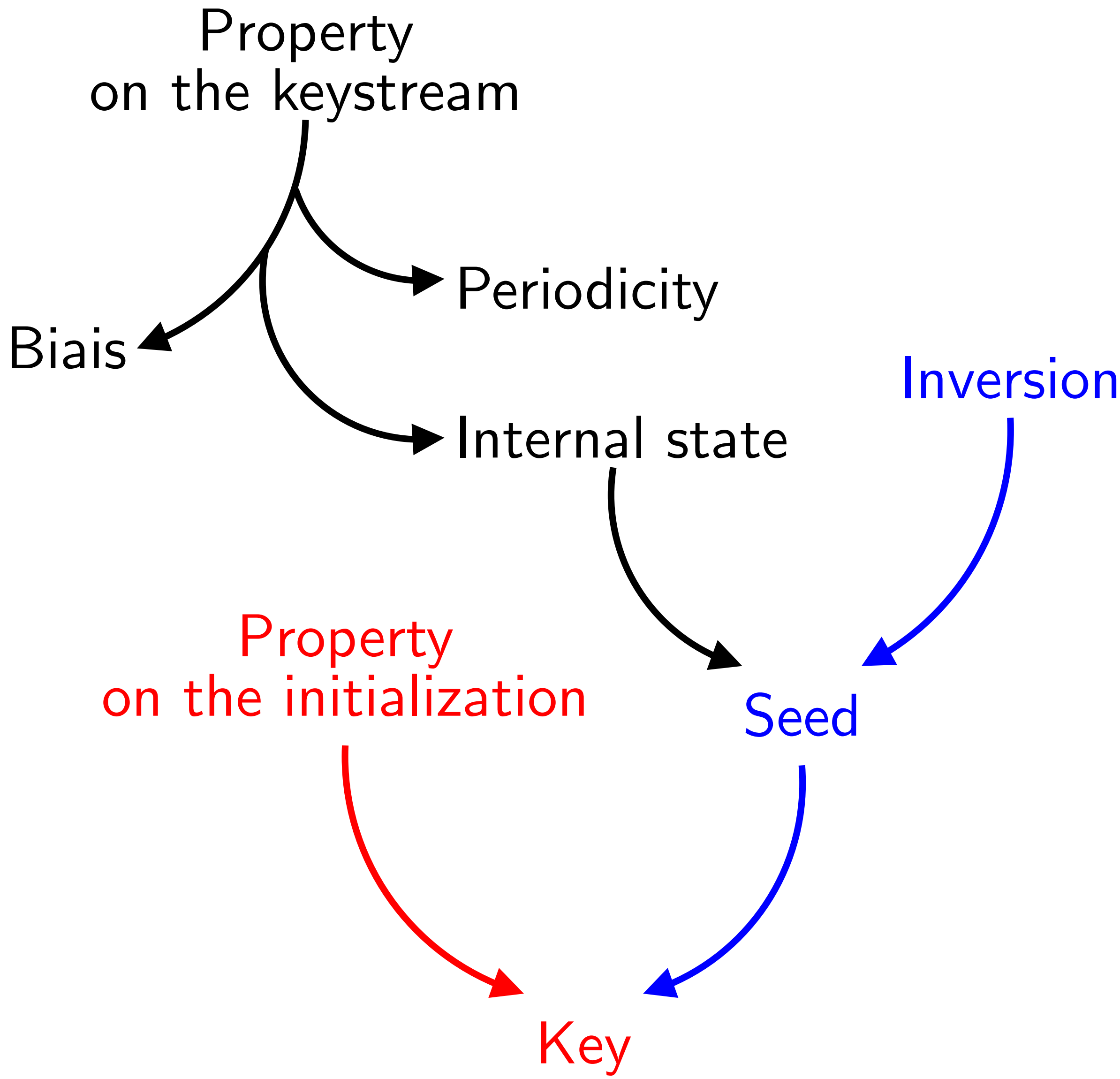
- **Initialization recovery attacks** attempt **to recover the initial state** of the stream cipher from the keystream.
- Knowing the key is enough to recover the initial state but it is not necessary reciprocity.

# Cryptanalysis

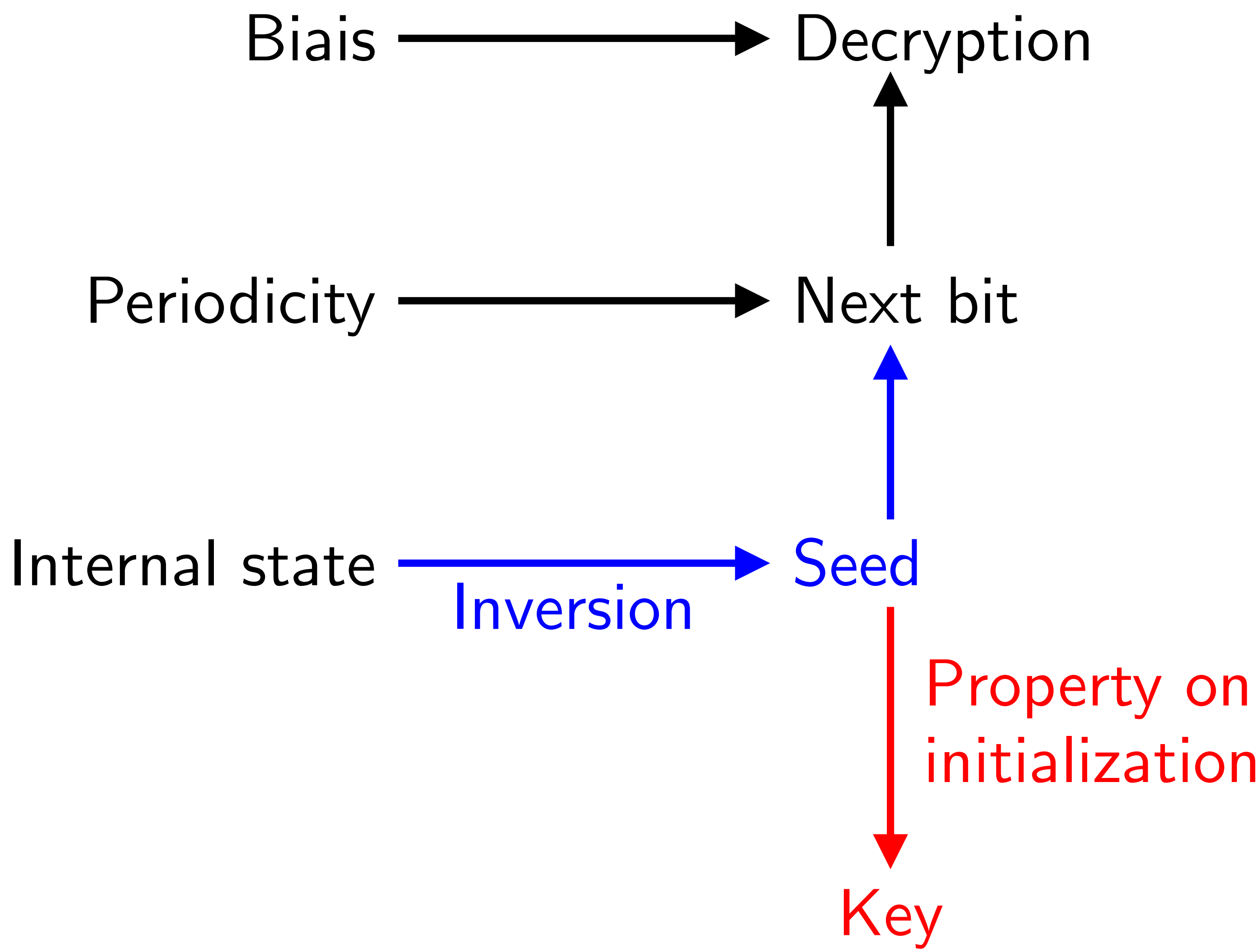
- **Next bit prediction attacks** consist from a keystream of $n$ to predict the next bit of the keystream

- **Distinguishing attacks** allow to determine if a keystream of $n$ bits can be distinguish from the output of an $ideal\ random\ number\ generator$.

# Attacks against stream ciphers



# Impact of the attacks



# Attacks against RC4

- To break a stream ciphers, we focus on 3 kinds of weakness :

  ▷ **Properties on the keystream**

  ▷ **Properties of inversion**

  ▷ **Properties on initialization**

# Inversion of TinyRC4

## and so of RC4

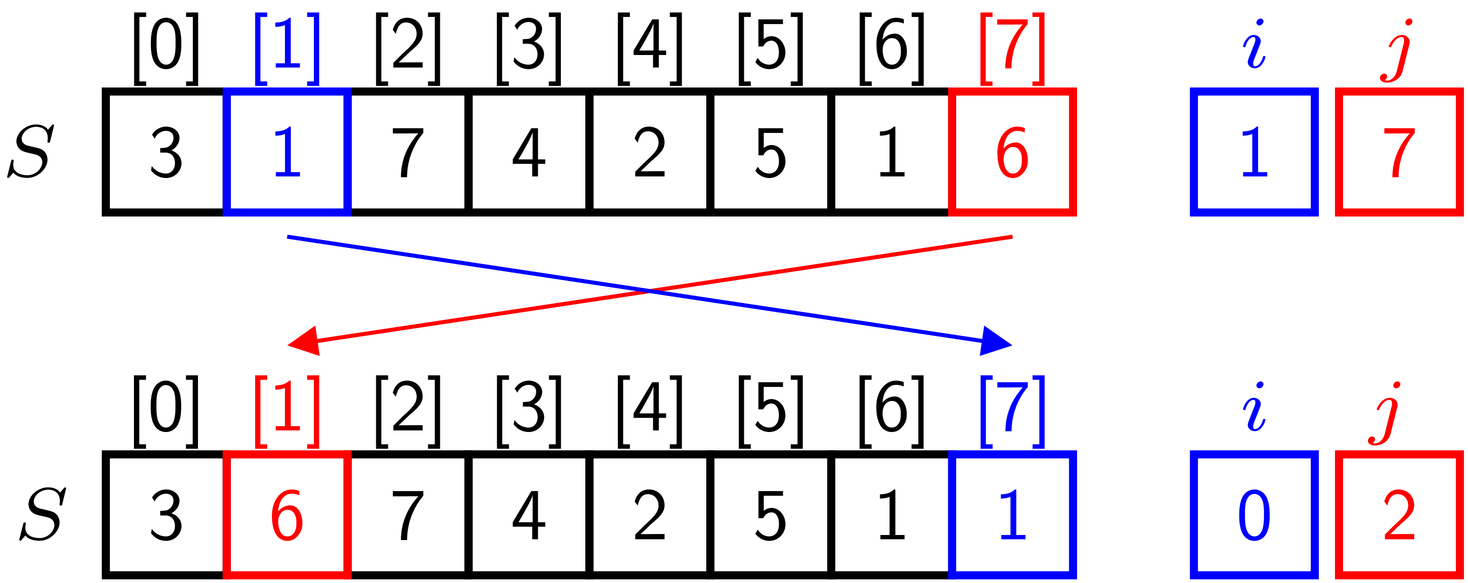
- Let assume that we know the internal state after $t$ rounds and the last byte of the keystream.
- We know $S$ et $i_t = t$.
- It is easy to recover $j_t$ because :

$$k_t = S[S[i_t] + S[j_t]].$$

We know $k_t$, $i_t$ et $S$, we can recover $j$.

# Inversion of TinyRC4

## Example




- Inversion Function :
  - — $\texttt{SWAP}(S[i],S[j])$
  - — $i_{t-1} = i_t - 1 \bmod 8$
  - — $j_{t-1} = j_t - S[i_{t-1}] \bmod 8$

# Initialization Property

Roos Biais

- If the initial state of $S$ was computed by a random permutation then we would have :

$$\forall y \in \mathbb{Z}/256\mathbb{Z}, \mathbf{Pr}(S[y] = c) = 1/256$$

with $\forall c \in \mathbb{Z}/256\mathbb{Z}$.

- Is the initialization of RC4 behaving like a random permutation ?

# Initialization property

## Roos biais

- We define $f_y$ by :

$$\begin{aligned} f_y &= \sum_{x=0}^{y} K[x] + x \\ &= \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]. \end{aligned}$$

- The most likely valuefor $S[y]$ at the end of the initialization is $S[y] = f_y$. **[Roos 1995]**

# Initialization property

## Roos biais

| $y$ | $\mathbf{Pr}(S[y] = f_y)$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0-7 | 0.370 | 0.368 | 0.362 | 0.358 | 0.349 | 0.340 | 0.330 | 0.322 |
| 8-15 | 0.309 | 0.298 | 0.285 | 0.275 | 0.260 | 0.245 | 0.229 | 0.216 |
| 16-23 | 0.203 | 0.189 | 0.173 | 0.161 | 0.147 | 0.135 | 0.124 | 0.112 |
| 24-31 | 0.101 | 0.090 | 0.082 | 0.074 | 0.064 | 0.057 | 0.051 | 0.044 |
| 32-39 | 0.039 | 0.035 | 0.030 | 0.026 | 0.023 | 0.020 | 0.017 | 0.014 |
| 40-47 | 0.013 | 0.012 | 0.010 | 0.009 | 0.008 | 0.007 | 0.006 | 0.006 |

Having noticed that : $\frac{1}{256} = 0.00390625$

# Roos biais

- Let assume a key of 40 bits :

$$K = \{106, 59, 220, 65, 34\}$$

| $y$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $f_y$ | 106 | 166 | 132 | 200 | 238 | 93 | 158 | 129 |
| $S_{256}[y]$ | 230 | 166 | 87 | 48 | 238 | 93 | 68 | 239 |
| $y$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $f_y$ | 202 | 245 | 105 | 175 | 151 | 229 | 21 | 142 |
| $S_{256}[y]$ | 202 | 83 | 105 | 147 | 151 | 229 | 35 | 142 |

# Roos biais

- For $S[1]$ and $S[2]$ we have :

$$\begin{aligned}
\mathbf{Pr}(S[1] = f_1) &\approx \left(\frac{256-1}{256}\right)^{256} \\
\mathbf{Pr}(S[2] = f_2) &\approx \left(\frac{256-1}{256}\right)^{256}
\end{aligned}$$

$$\begin{aligned}
f_1 &= K[0] + K[1] + 1 \\
f_2 &= K[0] + K[1] + K[2] + 3
\end{aligned}$$

- We obtain equations on the key from the seed.

# Generalization

- We have :

$$\mathbf{Pr}(S[y] = f_y) = \left(\frac{256 - y}{256}\right) \cdot \left(\frac{256 - 1}{256}\right)^{256 + \frac{(y+1)y}{2}} + \frac{1}{256}$$

# Key recovery

- We know $S$ et $K = \{106, 59, 220, 65, 34\}$

| $y$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $f_y$ | 106 | 166 | 132 | 200 | 238 | 93 | 158 | 129 |
| $S[y]$ | 230 | 166 | 87 | 48 | 238 | 93 | 68 | 239 |
| $y$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $f_y$ | 202 | 245 | 105 | 175 | 151 | 229 | 21 | 142 |
| $S[y]$ | 202 | 83 | 105 | 147 | 151 | 229 | 35 | 142 |

- Each time we have $S[y] = f_y$, we obtain an equation on the key and then we can build a system of equations :

# Key recovery

$$1 + \sum_{x=0}^{1} K[x] = 166 \qquad (1)$$

$$10 + \sum_{x=0}^{4} K[x] = 238 \qquad (2)$$

$$15 + \sum_{x=0}^{5} K[x] = 93 \qquad (3)$$

$$36 + \sum_{x=0}^{8} K[x] = 202 \qquad (4)$$

# Sources


- **RC4 Stream Cipher and Its Variants.** *P. Goutam et S. Maitra, 2011 CRC Press.*
- **Weaknesses in the Key Scheduling Algorithm of RC4.** *S. R. Fluhrer, I. Mantin et A. Shamir, SAC 2001, Springer-Verlag.*
- **A Class of Weak Keys in the RC4 Stream Cipher.** *Andrew Roos, posté sur sci.crypt, 1995.*