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

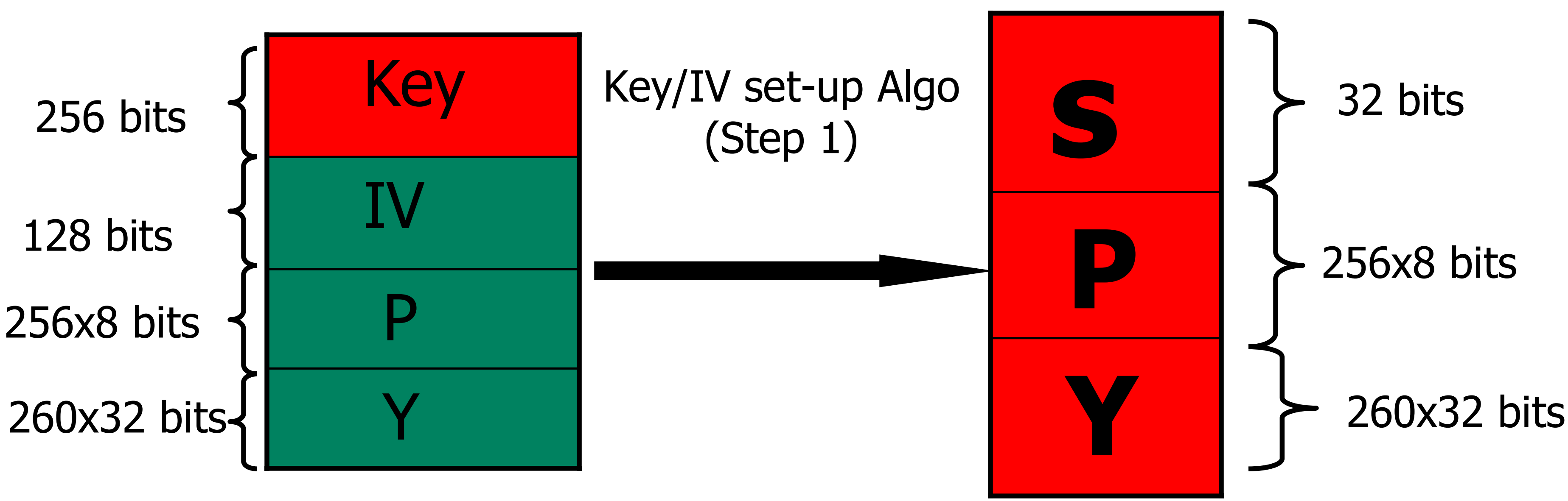
Outline

- 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

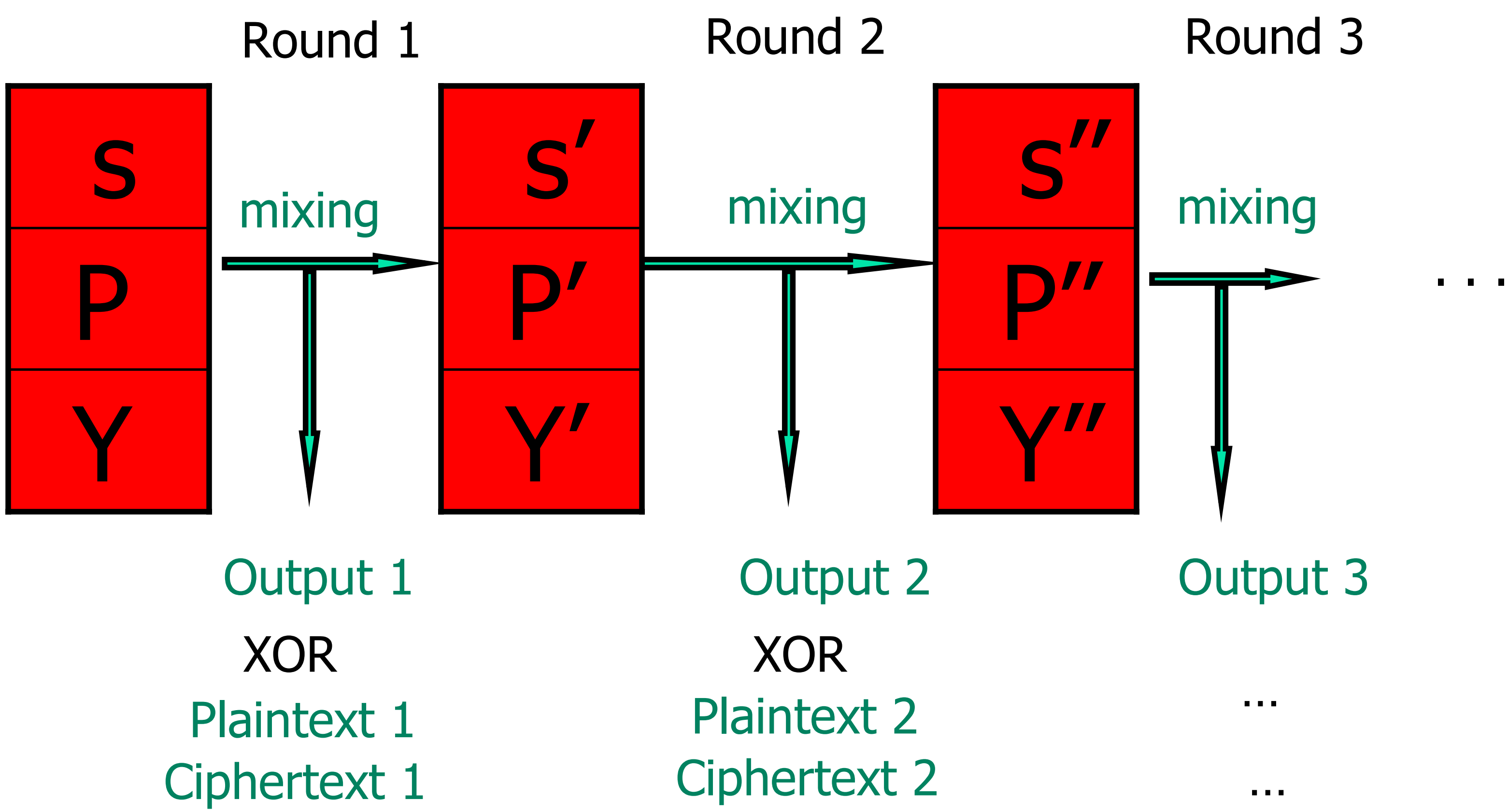
- RC4 (1987) by Rivest
- IA, IB, ISAAC (1996) by Jenkins Jr.
- RC4A (2004) by Paul and Preneel
- VMPC (2004) by Zoltak
- HC-256 (2004) by Wu
- GGHN (2005) by Gong *et al*.
- Py, Py6 (2005) by Biham and Seberry
- PyPy (2006) by Biham and Seberry

Stage I : Key/IV set-up of Py

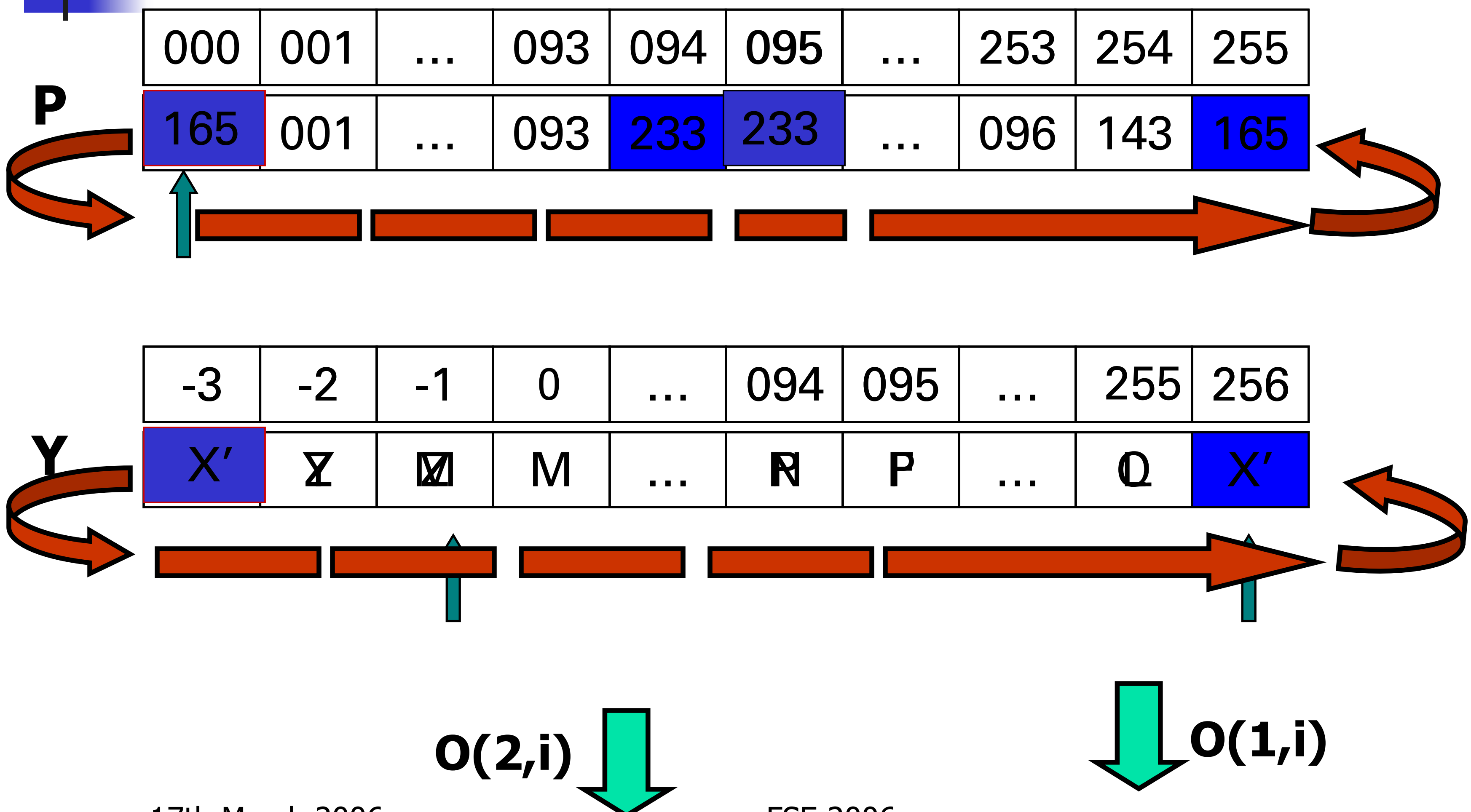


Initialization

Stage II : Keystream bytes generation of Py



Single round of Py: ith round



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

P

…	26	…	72	…	116	…	208	…	239	…
	1									

Round 1

P

…	26	…	72	…	116	…	208	…	239	…
			X		-18 mod32				Y	

Round 2

P

…	26	…	72	…	116	…	208	…	239	…
			Y+1		7 mod32		254		X+1	

Round 3

…rd rounds

	-…	…	254	255	256
Y				G	G

Round 3

Bias in the lsb's.

z=O(1,1)[0] XOR O(2,3)[0]

P(z=0)=1

O(1,1) = (S XOR G) + H

O(2,3) = (S XOR H) + G

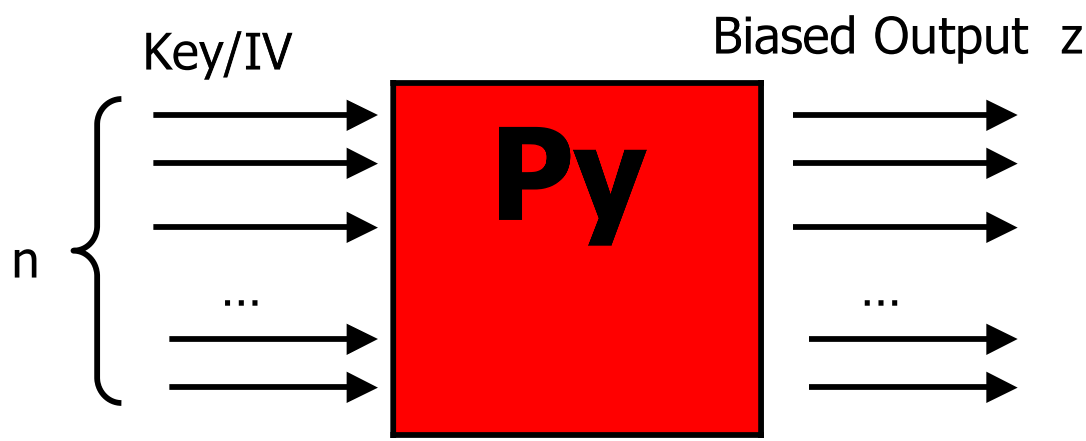
Quantifying the bias

- 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 XOR G) + H

O(2,3) = (S XOR H) +

Bias in the ith bits.

z=O(1,1)[i] XOR O(2,3)[i]

P(z=0)=1/2+μ

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.