Differential Fault Analysis of HC-128

Aleksandar Kircanski and Amr M. Youssef

AFRICACRYPT 2010
May 03-06, 2010, Stellenbosch, South Africa

Outline

- Fault analysis attacks
- DFA of array-based stream ciphers
- Specification of HC-128
- Attacking HC-128
- Conclusion

Main idea of fault analysis

- Induce an error in the device that performs encryption
  - Laser beam, voltage manipulation, overclocking
- Inspect the faulty output and deduce secret information

Some important works

- 1996: DFA of public-key crypto-systems (Boneh & DeMillo)
- 1998: DFA of block ciphers (Biham & Shamir)
- 2002: Fault induction made cheap (Skorobogatov & Anderson)
- 2004: DFA of stream ciphers (Hoch & Shamir)

DFA models

- Memory
  - Hamming weight
  - The ability to choose the memory location
- Durability
  - Transient
  - Permanent
- DFA of HC-128: faults occur in random inner state words

Natural approach for DFA of array-based ciphers

- Large state, slow update (RC4, HC-128, MV3,..)
- Let P be the inner state array
- $s_i = g(P[i_0], P[i_1], \dots P[i_k])$ the keystream output function

Then:

- Fault random $P[f]$
- Recover f
- Iterate until a faulty keystream word is encountered
- One of $\{i_1, \dots i_n\}$ indices had to be equal to f
- If the index depends on the inner state, information leaks

Problem

- Sometimes the approach above can not yield sufficient information
- Reason: untractable dependence between indices and the inner state content
- Example: HC-128: strategy does not lead to complete inner state recovery

Our approach: utilize the *reuse* of words

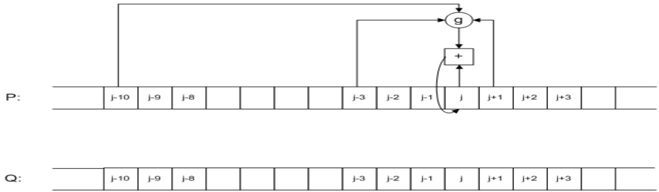
- Insert a random fault, corrupting $P[f]$ to $P'[f]$, recover f
- Clock the cipher until $P'[f]$ is *used* in the output [step i]: Non-faulty: $s_i(P[f], ..)$, faulty: $s'_i(P'[f], ..)$
- From $s_i(P[f], ..) \oplus s'_i(P'[f], ..)$ recover something about $P[f] \oplus P'[f]$
- Clock more, until $P'[f]$ is *reused* in the output [step j]: Non-faulty: $s_j(P[f], ..)$, faulty : $s'_j(P'[f], ..)$
- Consider $s_j(P[f], ..) \oplus s'_j(P'[f], ..)$: since $P[f] \oplus P'[f]$ is (partially) known, perform diff. cryptanlaysis on *other* values participating in $s_j()$

Why DFA via inner state *reuse* works for HC-128?

- HC-128: two tables P and Q, each 512 32-bit words
- Update function:
  $P[j] += (P[j \boxminus 10] \ggg 8) + (P[j \boxminus 3] \ggg 10) \oplus (P[j \boxminus 511] \ggg 23)$
- Output function:
  $s_i = (Q[A_i] + Q[B_i]) \oplus P[j]$, A_i, B_i pseudo random
- j public: ability to tell at which step is $P[f]$ is used
- Guarantee no update of $P[f]$ between *use* and *reuse*

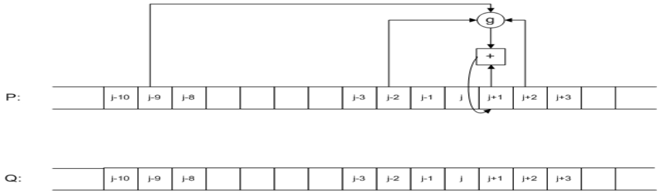
HC-128

- Member of eStream Software Portfolio
- 3.05 cycles/byte on Pentium M processor
- 128-bit key, 128-bit IV
- Inner state: $P[0], \ldots P[511]$, $Q[0], \ldots Q[511]$
- Update: 1 element per step, non-linear function ($\oplus, +,$ *rot*)
- Alternation of *runs* of length 512 of P-steps, Q-steps
- HC-128: likely to be widely implemented
- None of the security conjectures disproved



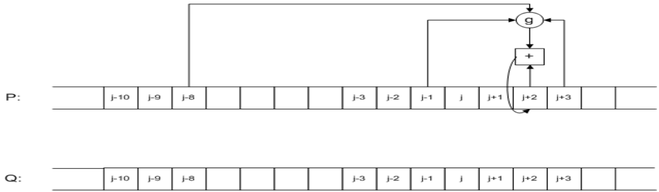
Update during “P-steps”

- 512 steps updating P table
- $P[j] += (P[j \boxminus 10] \ggg 8) + (P[j \boxminus 3] \ggg 10) \oplus (P[j \boxminus 511] \ggg 23)$
- Publicly known j increments



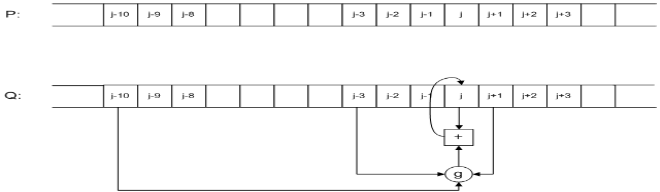
Update during "P-steps"

- 512 steps updating P table
- $P[j] += (P[j \boxminus 10] \ggg 8) + (P[j \boxminus 3] \ggg 10) \oplus (P[j \boxminus 511] \ggg 23)$
- Publicly known j increments



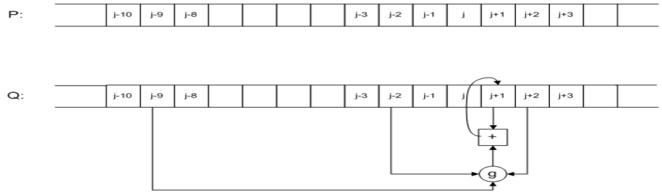
Update during “P-steps”

- 512 steps updating P table
- $P[j] += (P[j \boxminus 10] \ggg 8) + (P[j \boxminus 3] \ggg 10) \oplus (P[j \boxminus 511] \ggg 23)$
- Publicly known j increments



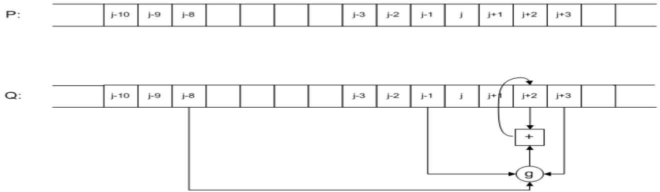
Update during "Q-steps"

- 512 steps updating Q table
- $Q[j] += (Q[j \boxminus 10] \lll 8) + (Q[j \boxminus 3] \lll 10) \oplus (Q[j \boxminus 511] \lll 23)$
- Publicly known j increments



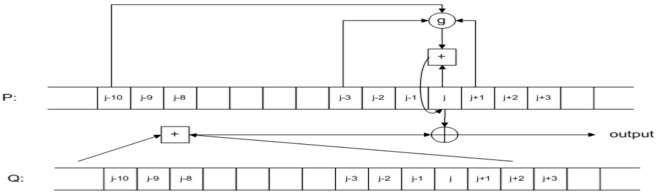
Update during "Q-steps"

- 512 steps updating Q table
- $Q[j] += (Q[j \boxminus 10] \lll 8) + (Q[j \boxminus 3] \lll 10) \oplus (Q[j \boxminus 511] \lll 23)$
- Publicly known j increments



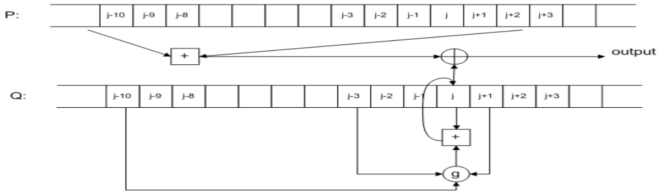
Update during "Q-steps"

- 512 steps updating Q table
- $Q[j] += (Q[j \boxminus 10] \lll 8) + (Q[j \boxminus 3] \lll 10) \oplus (Q[j \boxminus 511] \lll 23)$
- Publicly known j increments



Output during "P-steps"

- $s_i = h_1(P[j \boxminus 12]) \oplus P[j] =$
  $= (Q[A_i] + Q[B_i]) \oplus P[j]$
  where: $0 \leq A_i \leq 255$, $256 \leq B_i \leq 511$



Output during "Q-steps"

- $s_i = h_1(Q[j \boxminus 12]) \oplus Q[j] =$
  $= (P[A_i] + P[B_i]) \oplus Q[j]$
  where: $0 \leq A_i \leq 255$, $256 \leq B_i \leq 511$

Two auxiliary algorithms

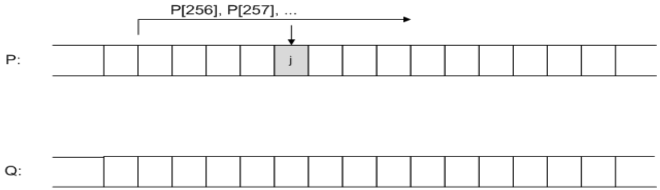
- Fault position recovery ($P[f]$ faulted: recover f)
- Difference between the original and the faulty value (recover $P[f] \oplus P'[f]$)

Collecting faulty information

- Until every P, Q word faulted at least once, repeat
  - Reset the cipher, iterate for 268 steps
  - Induce a fault
  - Store the resulting faulty keystream words

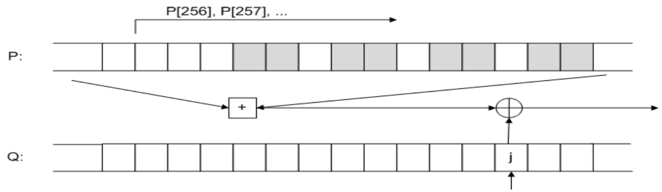
32 phases

- Inner state recovered
- Phase i: linear equations in i-th bit of $P[0], \ldots P[512]$, $Q[0], \ldots Q[512]$
- To ensure full rank: several different ways to generate equations



- Fault: second half of the P table
- Propagation only to $P[j]$ $j > f$, and not to Q table
- In Q-steps, the output depends on exactly one faulty value
- $s_i = (P[A_i] + P'[B_i]) \oplus Q[j]$: only $P'[B_i]$ faulty
- $P[B_i] \oplus P'[B_i]$ known, diff. analysis to recover $P[A_i]$ bits

- Fault: second half of the P table
- Propagation only to $P[j]$ $j > f$, and not to Q table
- In Q-steps, the output depends on exactly one faulty value
- $s_i = (P[A_i] + P'[B_i]) \oplus Q[j]$: only $P'[B_i]$ faulty
- $P[B_i] \oplus P'[B_i]$ known, diff. analysis to recover $P[A_i]$ bits



- Fault: second half of the P table
- Propagation only to $P[j]$ $j > f$, and not to Q table
- In Q-steps, the output depends on exactly one faulty value
- $s_i = (P[A_i] + P'[B_i]) \oplus Q[j]$: only $P'[B_i]$ faulty
- $P[B_i] \oplus P'[B_i]$ known, diff. analysis to recover $P[A_i]$ bits

Complexity of the attack

- 32 systems of linear bit equations in 1024 variables
- Sparse systems, each around 18000 equations
- The total expected number of faults: 7192

Future work

- Extend the attack to HC-256
- Reduce the number of faults

THANK YOU!