Impact of Extending Side Channel Attack on Cipher Variants: A Case Study with the HC Series of Stream Ciphers

Goutam Paul and Shashwat Raizada
Jadavpur University, Kolkata and Indian Statistical Institute, Kolkata

Outline

Motivation

Impact of Extending Side Channel Attack on Cipher Variants: A Case Study with the HC Series of Stream Ciphers.

- Cipher Variants executes with identical ease on similar hardware .
- The best possible replacement for a compromised cipher is its variant.
- Example : When WEP running on all WiFi devices had a flaw immediate solution was RC4 variant in WPA prior a cipher change in WPA2.
- Hence the all important question attempted by trying successful side channel attacks of HC-128 on HC-256 and vice versa

Basics of Stream Cipher

- Symmetric Key Cryptosystem, both sender and the receiver has the same key.
- The best possible scenario that the sender and receiver have a common stream of bits that they have generated sitting in the same table and tossing an unbiased coin. One Time Pad (never used repeatedly). Practically not possible.
- Use a Pseudorandom generator based on a seed (secret key).
- The pseudorandom bit stream will be used to generate the ciphertext by exoring with the plaintext bits.
- Decryption is by exoring the ciphertext with the keystream.

The eSTREAM Project

- An effort to get some secure stream ciphers satisfying the current requirements.
- http://www.ecrypt.eu.org/stream/
- This multi-year effort running from 2004 to 2008 has identified a portfolio of promising new stream ciphers.
- "For the future, we expect that research on the eSTREAM submissions in general, and the portfolio ciphers in particular, will continue. We therefore welcome any ongoing contributions to any of the eSTREAM submissions. It is also possible that changes to the eSTREAM portfolio might be needed in the future. If so, any future revisions will be made available via these pages. "

The eSTREAM Portfolio

The eSTREAM Portfolio (revision 1) (September 2008): The eSTREAM portfolio has been revised and the portfolio now contains the following ciphers:

Profile 1 (SW)	Profile 2 (HW)
HC-128	Grain v1
Rabbit	MICKEY v2
Salsa20/12	Trivium
SOSEMANUK	

HC-128 and HC-256

- Designed by Hongjun Wu
- HC-128 is a scaled down version of HC-256 that has been presented in FSE 2004
- A synchronous stream cipher with 32-bit word output in each step
- A software stream cipher with 128 and 256 bit keys for HC-128 and HC-256 respectively.
- Intellectual Property: Free for any use.

Notations

$+$: $x + y$ means $x + y \bmod 2^{32}$, where $0 \leq x < 2^{32}$ and $0 \leq y < 2^{32}$.
$\boxminus$: $x \boxminus y$ means $x - y \bmod 512$ in HC-128.
$\boxminus$: $x \boxminus y$ means $x - y \bmod 1024$ in HC-256.
$\oplus$: bit-wise exclusive OR.
$\|$: concatenation.
$\gg$: right shift operator. $x \gg n$ means x being right shifted n bits.
$\ll$: left shift operator. $x \ll n$ means x being left shifted n bits.
$\ggg$: right rotation operator. $x \ggg n$ means $((x \gg n) \oplus (x \ll (32 - n))$, where $0 \leq n < 32$, $0 \leq x < 2^{32}$.
$\lll$: left rotation operator. $x \lll n$ means $((x \ll n) \oplus (x \gg (32 - n))$, where $0 \leq n < 32$, $0 \leq x < 2^{32}$.

Data Structures

- Two tables P and Q, each with 512 many 32-bit elements are used as internal states of HC-128.
- A 128-bit key array $K[0, \ldots, 3]$ and a 128-bit initialization vector $IV[0, \ldots, 3]$ are used, where each entry of the array is a 32-bit element.
- Two tables P and Q, each with 1024 many 32-bit elements are used as internal states of HC-256.
- A 256-bit key array $K[0, \ldots, 7]$ and a 256-bit initialization vector $IV[0, \ldots, 7]$ are used, where each entry of the array is a 32-bit element.
- s_t denotes the keystream word generated at the t-th step, $t = 0, 1, 2, \ldots$.

Functions in HC-128

$$f_1(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3),$$

$$f_2(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10),$$

$$g_1(x, y, z) = ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8),$$

$$g_2(x, y, z) = ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8),$$

$$h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}],$$

$$h_2(x) = P[x^{(0)}] + P[256 + x^{(2)}],$$

where $x = x^{(3)} \| x^{(2)} \| x^{(1)} \| x^{(0)}$, x is a 32-bit word and $x^{(0)}$ (least significant byte) , $x^{(1)}, x^{(2)}$ and $x^{(3)}$ (most significant byte) are four bytes.

Functions in HC-256

$$f_1(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3),$$

$$f_2(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10),$$

$$g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024],$$

$$g_2(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + P[(x \oplus y) \bmod 1024],$$

$$h_1(x) = Q[x^{(0)}] + Q[256 + x^{(1)}] + Q[512 + x^{(2)}] + Q[768 + x^{(3)}],$$

$$h_2(x) = P[x^{(0)}] + P[256 + x^{(1)}] + P[512 + x^{(2)}] + P[768 + x^{(3)}],$$

where $x = x^{(3)} \| x^{(2)} \| x^{(1)} \| x^{(0)}$, x is a 32-bit word and $x^{(0)}$ (least significant byte) , $x^{(1)}, x^{(2)}$ and $x^{(3)}$ (most significant byte) are four bytes.

Key and IV setup of HC-128

Let $K[0,\ldots,3]$ be the secret key and $IV[0,\ldots,3]$ be the initialization vector. Let $K[i+4] = K[i]$ and $IV[i+4] = IV[i]$ for $0 \leq i \leq 3$.
The key and IV are expanded into an array $W[0,\ldots,1279]$ as follows.

$$W[i] = \begin{cases} K[i] & 0 \leq i \leq 7; \\ IV[i-8] & 8 \leq i \leq 15; \\ f_2(W[i-2]) + W[i-7] + \\ f_1(W[i-15]) + W[i-16] + i \\ & 16 \leq i \leq 1279. \end{cases}$$

Update the tables P and Q with the array W as follows.

$$P[i] = W[i+256], \text{ for } 0 \leq i \leq 511$$
$$Q[i] = W[i+768], \text{ for } 0 \leq i \leq 511$$

Key and IV setup of HC-128 (Contd.)

Run the cipher 1024 steps and use the outputs to replace the table elements as follows.
for $i = 0$ to 511, do $P[i] =$
$(P[i] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])) \oplus h_1(P[i \boxminus 12]);$
for $i = 0$ to 511, do $Q[i] =$
$(Q[i] + g_2(Q[i \boxminus 3], Q[i \boxminus 10], Q[i \boxminus 511])) \oplus h_2(Q[i \boxminus 12]);$

Key and IV setup of HC-256

- Similar to HC-128.
- Key and IV is expanded using SHA2 like message expansion.
- Two tables *P* and *Q*, each with 1024 many 32-bit elements are used as internal states of HC-256.
- Cipher is run 2048 times without generating any output.

The Keystream Generation Algorithm of HC-128

```
$i = 0$;
repeat until enough keystream bits are generated
{
    $j = i \bmod 512$;
    if $(i \bmod 1024) < 512$
    {
        $P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$;
        $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
    }
    else
    {
        $Q[j] = Q[j] + g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$;
        $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
    }
    end-if
    $i = i + 1$;
} end-repeat
```

The Keystream Generation Algorithm of HC-256

```
$i = 0$;
repeat until enough keystream bits are generated
{
    $j = i \bmod 1024$;
    if $(i \bmod 2048) < 1024$
    {
        $P[j] = P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$;
        $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
    }
    else
    {
        $P[j] = P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$;
        $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
    }
    end-if
    $i = i + 1$;
} end-repeat
```

Existing Side Channel Results on HC series of Ciphers

- *Differential Fault Analysis of HC-128.* A. Kircanski and A. M. Youssef.In Africacrypt 2010, pages 360-377, vol. 6055, Lecture Notes in Computer Science, Springer.
- *A Cache Timing Analysis of HC-256.* E. Zenner . In SAC 2008, pages 199-213, vol. 5381, Lecture Notes in Computer Science, Springer.

Fault Analysis

- Fault attacks are an invasive side channel cryptanalytic technique in which faults are inserted into the cryptographic device.
- The goal maybe to corrupt the value of an internal state register or memory location or to make a change in the execution flow, such as skipping an instruction or changing a memory address etc.
- The corresponding change in the cipher output obtained are used to extrapolate the internal state. A differential fault attack against a stream cipher resets the cipher with the same key, but injecting different faults. The resulting keystreams have small differences and the attack exploits these differentials.
- In this paper, we extend the differential fault attack on HC-128 to HC-256.

Keystream Generation of HC-128 and existing Fault Attack

i	*Parray*	*Qarray*	s_i
0	$P[0]$ *Updated*	*NoChange*	$s_0 = h_1(P[0 \boxminus 12]) \oplus P[0]$
1	$P[1]$ *Updated*	*NoChange*	$s_1 = h_1(P[1 \boxminus 12]) \oplus P[1]$
2	$P[2]$ *Updated*	*NoChange*	$s_2 = h_1(P[2 \boxminus 12]) \oplus P[2]$
...	...	...	...
512	*NoChange*	$Q[0]$ *updated*	$s_{512} = h_2(Q[0 \boxminus 12]) \oplus Q[0]$
513	*NoChange*	$Q[1]$ *updated*	$s_{513} = h_2(Q[1 \boxminus 12]) \oplus Q[1]$
...	...	...	
1023			

In P-block, the keystream is generated as

$$\begin{aligned} s_{P,j} &= h_1(P[j \boxminus 12]) \oplus P[j] \\ &= \Big(Q\big[(P[j \boxminus 12])^{(0)}\big] + Q\big[256 + (P[j \boxminus 12])^{(3)}\big]\Big) \oplus P[j]. \end{aligned}$$

Fault Attack in HC-256

In P-block, the keystream is generated as

$$\begin{aligned} s_{P,j} &= h_1(P[j \boxminus 12]) \oplus P[j] \\ &= \Big(Q\big[(P[j \boxminus 12])^{(0)}\big] + Q\big[256 + (P[j \boxminus 12])^{(1)}\big] + Q\big[512 + \\ &\quad (P[j \boxminus 12])^{(2)}\big] + Q\big[768 + (P[j \boxminus 12])^{(3)}\big]\Big) \oplus P[j]. \end{aligned}$$

The P term is updated as
$P[j] = P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$
where
$g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024]$

Two distinct types of fault in HC-256

- Suppose we inject a fault at $Q[f]$ before $P[0]$ is updated in P-block.
- We rerun the key generation algorithm 1024 times to generate 1024 *faulty* keystream words corresponding to the current P-block.
- We compare $s_{P,j}$ with $s'_{P,j}$ for $j = 0, \ldots, 511$.
- Whenever $s_{P,j} \neq s'_{P,j}$, we know that the faulty keystream has been accessed.
- The noticeable faults observed in the keystream are either due to a faulty value of Q entering the h_1 function or a faulty value of Q entering the update function of P.

Two distinct types of fault in HC-256 contd.

Definition

When a faulty Q (or P) array element enters in h_1 (or h_2), we call this an ***opportune*** event.

Definition

When a faulty Q (or P) array element is referred inside g_1 (or g_2), we all this a ***traverse*** event.

Opportune Event : Faulty Q entering h_1

Suppose an *opportune* event occurs and f is the location of the fault, i.e., $Q[f]$ is the faulty value accessed inside h_1. Here the location of a faulty Q element gives information of a byte of P. The keystream index j where $s_{P,j}$ and $s'_{P,j}$ differs,refers to:-

- byte 0 of $P[j \boxminus 12]$ if $0 \leq f \leq 255$,
- byte 1 of $P[j \boxminus 12]$ if $256 \leq f \leq 511$,
- byte 2 of $P[j \boxminus 12]$ if $512 \leq f \leq 767$ or
- byte 3 of $P[j \boxminus 12]$ if $768 \leq f \leq 1023$.

Complete P array would be revealed but for occurrence of *traverse* event blocking subsequent leaks.

Theoretical Estimate of bytes Obtainable

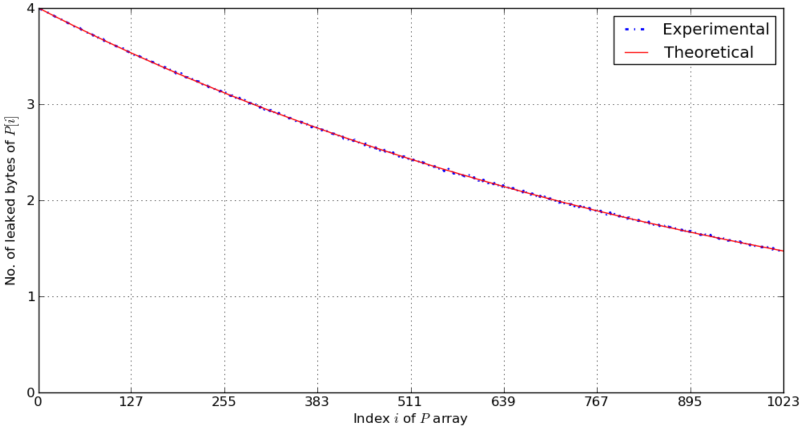
Theorem

The expected number of bytes of $P[i]$ leaked through h_1 function is given by $4(\frac{1023}{1024})^{i+1}$, $0 \leq i \leq 1023$.

Corollary

The expected number of words of the array P leaked through h_1 function is 647.

Bytes Leaked in Opportune Event



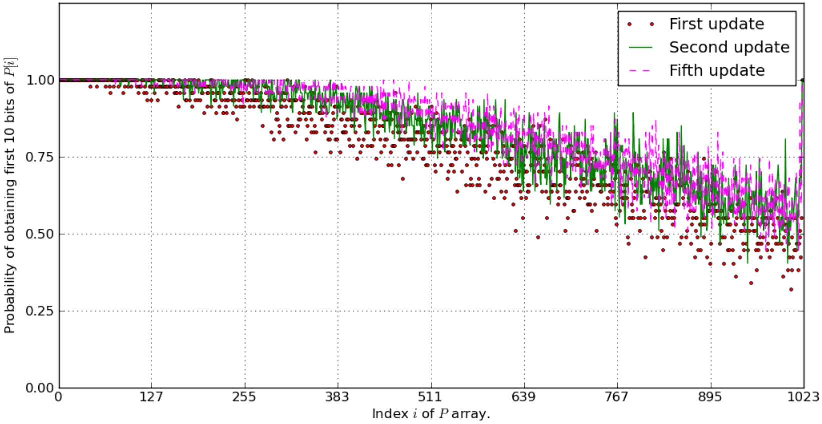
Traverse Event

The update of P involves Q (unlike HC-128), faulty $Q[f]$ is eventually referred inside g_1.
Such a *traverse* event does not yield any more information within the particular update.
However, this case assists in finding elements of the previous updates as follows.
Recall the update of P.

$$P[j] \quad = \quad P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023]),$$

where
$g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024]$.

Number of first ten bits obtained Combining Both Event



Number of first ten bits obtained using Both Events

Type	Update No.	Terms Obtained
only *h* fault	1	646.8
both *h* and *g* fault	1	805.1
both *h* and *g* fault	2	860.0
both *h* and *g* fault	3	871.0
both *h* and *g* fault	4	875.0
both *h* and *g* fault	5	877.6

Table: Number of words with first ten bits leaked vs. number of subsequent updates incorporated.

Total words obtained approx 719

Cache Analysis Attacks

- Cache analysis is a side channel cryptanalysis technique that has been introduced independently by Bernstein and Osvik et al primarily for the AES block cipher.
- *Cache* is a temporary storage area that is closer to the CPU compared to the RAM for enabling faster access
- *Cache hit* if data is in cache it avoids re-fetching from the RAM.Otherwise a *cache miss* occurs and the cache must be immediately loaded with the requisite data from the RAM.
- Cache management in modern processors divides the available cache memory into blocks of *b* bytes. For a given block, all the *b* bytes must be loaded together.

Cache Analysis Attacks (contd.)

- To ensure consistency between the data in the RAM and the cache, a record of cache blocks being loaded into the cache is maintained.
- This record serves as the initial raw-data for commencing the cache analysis. The adversary first fills the entire cache with his own data.
- Next, during normal computation, user's data replaces some data of the adversary.
- This pattern of Cache loads gives details of the cipher internals

Cache Access Points in HC-256

In *P*-block, the keystream is generated as

$$\begin{aligned} s_{P,j} &= h_1(P[j \boxminus 12]) \oplus P[j] \\ &= \Big(Q\big[(P[j \boxminus 12])^{(0)}\big] + Q\big[256 + (P[j \boxminus 12])^{(1)}\big] + Q\big[512 + \\ &\quad (P[j \boxminus 12])^{(2)}\big] + Q\big[768 + (P[j \boxminus 12])^{(3)}\big]\Big) \oplus P[j]. \end{aligned}$$

The *P* term is updated as
$P[j] = P[j] + P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$
where
$g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 1024]$

Applicability on HC-128

HC-128 ... has a slightly smaller inner state ... and surprisingly big changes of the internal workings. Most state update equations are modified, and this has a profound impact on the above cache timing attack. It turns out that the attack can not be transferred to HC-128 in a straightforward way. Thus, further analysis of HC-128 is necessary to determine its resistance against cache timing attacks.

Bits Obtainable in HC-128

Key-stream is generation uses two similiar functions h_1, h_2 :-

$$\begin{aligned} s_j &= h_1(P[j \boxminus 12]) \oplus P[j] \quad \text{(keystream when } P \text{ is updated)}, \\ s_j &= h_2(Q[j \boxminus 12]) \oplus Q[j] \quad \text{(keystream when } Q \text{ is updated)}. \end{aligned}$$

Every h_1 (or h_2) function call, it uses bytes 0 and 2 of $P[j \boxminus 12]$ (or $Q[j \boxminus 12]$) to access the elements of Q (or P) array.
Cache with block size b bytes holds $b/4$ array words, the cache blocks can be numbered using the first $8 - \log_2(b/4)$ bits.
For each call, two elements of the array are accessed.
This gives $16 - 2\log_2(b/4)$ bits of each element of the arrays P and Q for each cache fill sought (h_1 or h_2 call).

Bits Obtainable in HC-128

The Bits obtained depends on *b* and is given in the table.

Cache Block Size *b*:	16	32	64	128
No. of bits obtained:	12	10	8	6

Table: Cache block size vs. number of bits learnt.

Constructing Byte 0 and 2 of each element

For $0 \leq j \leq 500$, the keystream generation equation $s_j = h_1(P[j \boxminus 12]) \oplus P[j]$ is rewritten.

$$Q[u] + Q[v] = s(j+12) \oplus P_1[j+12]), \qquad (1)$$

where $u = P[j]^{(0)}$ and $v = P[j]^{(2)}$. The keystream s is known.

- Example for a cache block b of 32 bytes, the first 5 most significant bits of each of u and v are known.
- Thus, $u \gg 3$ and $32 + (v \gg 3)$ denote the cache blocks loaded for computing h_1.
- Within these blocks (in Q_0), we exhaustively search the elements that would give a sum identical to the RHS These additions are across two chunks of 5 bits and so additional carry bits need to be accounted.
- For each element $P[j]$, there are $(b/4).(b/4) = b^2/16$ calculations.

Finding remaining Sixteen bits of each element

- For the remaining 16 bits for each element in case of HC-128, we propose to use the techniques of differential fault analysis in combination with the cache analysis suggested here.
- Present attack needs to solve a set of 32 systems of linear equations over $\mathbb{Z}_2$ in 1024 variables.
- If one performs the cache analysis proposed first, immediately the problem reduces to 16 systems of linear equations over $\mathbb{Z}_2$ in 1024 variables.

Case Study with HC series of Stream Ciphers

Thus our contributions constitute of

- extending the differential fault attack on HC-128 cipher onto HC-256 with a slightly weaker assumption giving 719 words of each internal array;
- extending the cache attack on HC-256 onto HC-128 yielding half state exposure.

Concluding Remarks

....the side channel vulnerability for a particular implementation of a cipher may percolate to its variants also, albeit in a different degree. This vulnerability is still exploitable through refinement of the attack vectors. So, while selecting a cipher variant to thwart side channel vulnerabilities, extra caution must be exercised.

Thank You