A New Algorithm for Switching from Arithmetic to Boolean Masking

Jean-Sébastien Coron and Alexei Tchulkine

Gemplus Card International

34 rue Guynemer, 92447 Issy-les-Moulineaux, France

GEMPLUS

Differential Power Analysis

- Differential Power Analysis
  - Introduced by Paul Kocher and al. in 1998
  - Consists in extracting information about the secret key of a cryptographic algorithm, by studying the power consumption during the execution of the algorithm
  - All algorithms are vulnerable (DES, AES, RSA, HMAC...)
- Countermeasures
  - Hardware countermeasures: add noise, random delay...
  - Software countermeasures: random masking.

Random masking

- Random masking
  - Proposed by Chari et al. at Crypto 99.
  - Consists in masking all intermediary data with a random.
  - The masked data and the random are processed separately.
- Boolean masking:
  - A variable x is written as:

$$x = x' \oplus r$$

    where x' is the masked variable and r a random.
  - x' and r are manipulated separately (instead of x).

Random masking

- Advantage: increased security.
  - The data is shared in two (or more) variables.
  - The power leakage of an individual share does not reveal any information to the attacker
  - The attacker must correlate the shares to get useful information
    - ✓ Exponentially more curves are needed.
- Drawback: decreased efficiency.
  - Two shares are processed instead of one.
  - More RAM needed for non-linear functions, such as SBOXes.
  - Issue for smart-cards.

Boolean/arithmetic masking

- Boolean masking: $x = x' \oplus r$
  - is applicable when $\oplus$ are used, *e.g.* DES.
  - Let $x_1 = (x'_1, r_1) = x'_1 \oplus r_1$ and $x_2 = (x'_2, r_2)$.
  - To compute $x_3 = x_1 \oplus x_2 = (x'_3, r_3)$
    - ✓ Compute $x'_3 = x'_1 \oplus x'_2$.
    - ✓ Compute $r_3 = r_1 \oplus r_2$.
- Arithmetic masking:
  - A variable x is written as:

$$x = A + r \mod 2^k$$

  - Applicable when arithmetic operations are used
  - IDEA, RC6, SHA.

Conversion

- For algorithms combining boolean and arithmetic operations:
  - IDEA, RC6, SHA.
  - Conversion required between boolean and arithmetic masking.
- The conversion must be secure:
  - Let x', r such that $x = x' \oplus r$. We want to compute A such that $x = A + r \mod 2^k$.
  - We can not compute $A = (x' \oplus r) - r \mod 2^k$ directly,
  - since otherwise $x = x' \oplus r$ is leaked.

From boolean to arithmetic masking

- Very efficient and elegant technique invented by Louis Goubin.
  - Provably secure and constant number of operations (CHES 2001).
  - Based in the fact that for all x', the function $f_{x'}(r) = (x' \oplus r) - r$ is affine in r
- Let x', r such that $x = x' \oplus r$.
  - We want to compute $A = (x' \oplus r) - r \mod 2^k$.
  - Generate a random k-bit integer r_1. Then:

$$\begin{aligned} A &= f_{x'}(r) = f_{x'}((r_1 \oplus r) \oplus r_1) \\ &= f_{x'}(r_1 \oplus r) \oplus (f_{x'}(r_1) \oplus x') \end{aligned}$$

From arithmetic to boolean

- Method proposed by Goubin:
  - Also provably secure.
  - Less efficient than boolean to arithmetic.
  - Number of operations: $5k + 5$ for k-bit variables.
  - Bottleneck in some implementations, for example SHA.
- We propose a more efficient algorithm
  - Provably secure.
  - Based on pre-computed tables.

Conversion for small size

- Arithmetic to boolean conversion.
  - Given A, r, we must compute $x' = (A + r) \oplus r$.
- Precomputed table G of 2^k values of k-bits.
  - Generate a random k-bit r.
  - For $A = 0$ to $2^k - 1$ do $G[A] \leftarrow (A + r) \oplus r$
  - Output G and r.
- Conversion from arithmetic to boolean:

$$x = x' \oplus r = A + r \mod 2^k$$

  - Given A, return $x' = G[A]$.
  - Provably resistant to DPA (like classical SBOX randomization).

Performances

- Comparison between our method and Goubin.

	Our method	Goubin's method
Pre-computation time	2^{k+1}	0
Conversion time	1	$5k+5$
Table size	2^k	0

- Main limitation of our method:
  - Pre-computation time and memory required.
  - But pre-computation is done once and every subsequent conversion requires only one step.
  - Only feasible for conversion with small sizes ($k=4$ or $k=8$ bits).

Extension for larger sizes

- Conversion for $\ell \cdot k$-bit variables.
  - We use two k-bit tables G and C.
  - Example: $k = 4$ and $\ell = 8$ for 32-bit variables: two 4-bit tables require 16 bytes of RAM.
- Otherview of the algorithm
  - We separate the 32-bit variable into 8 nibbles of 4 bits.
  - We apply the previous conversion method to each nibble using table G.
  - We propagate the carry among the nibbles, using a randomized carry table C.

The algorithm for large size

- Let A, R such that $x = A + R \mod 2^{\ell \cdot k}$.
  - A and R are $\ell \cdot k$ bit variables.
  - Let $A = A_1 \| A_2$, $R = R_1 \| R_2$ where A_2, R_2 are k-bit.

$$x = (A_1 \| A_2) + (R_1 \| R_2) \mod 2^{\ell k}$$

- Splitting via carry computation.
  - If $A_2 + R_2 \geq 2^k$, let $A_1 \leftarrow A_1 + 1 \mod 2^{(\ell-1)k}$.
  - Then if $x = x_1 \| x_2$, we have:

$$\begin{aligned} x_1 &= A_1 + R_1 \mod 2^{(\ell-1)k} \\ x_2 &= A_2 + R_2 \mod 2^k \end{aligned}$$

  - We can apply the conversion recursively to (A_1, R_1) and (A_2, R_2).

The algorithm (2)

- Conversion of $x_2 = A_2 + R_2 \mod 2^k$
  - We use the previous table G with $r = R_2$

$$x'_2 \leftarrow G[A_2]$$

  - We obtain $x_2 = x'_2 \oplus R_2$.
- We apply the same method recursively to $x_1 = A_1 + R_2 \mod 2^{(k-1)\cdot\ell}$.
  - We obtain x'_1 such that $x_1 = x'_1 \oplus R_1$.
  - Letting $x' = x'_1 \| x'_2$, we obtain as required:

$$x = x' \oplus R$$

Carry computation

- Problem with carry computation:
  - We cannot compute $A_2 + R_2$ directly, since this would leak information about x.
- Instead, we use a carry table C:
  - Randomized carry table generation:
    1. Generate a random k-bit γ.
    2. For $A = 0$ to $2^k - 1$ do

$$C[A] \leftarrow \begin{cases} 0 + \gamma, \text{if } A + R_2 < 2^k \\ 1 + \gamma \mod 2^k, \text{if } A + R_2 \geq 2^k \end{cases}$$

  - Instead of testing if $A_2 + R_2 \geq 2^k$, we let:

$$A_1 \leftarrow (A_1 + C[A_2]) - \gamma \mod 2^{(\ell-1)k}$$

Security of the new method

- The new algorithm is secure against first order DPA.
  - All intermediate data have the uniform distribution
  - The attacker learns nothing by observing an individual step.
- The attacker must correlate the power consumption of at least two steps (High-Order DPA).
  - This requires more curves.
  - This might be infeasible if there is a counter that limits the number of executions with the same key.

Performances

- Number of elementary operations for i-bit variables with a j-bit microprocessor with $k = 4$.
  - Our new method: $T_{i,j}$.
  - Goubin's method: $G_{i,j}$

	$T_{8,8}$	$T_{8,32}$	$T_{32,8}$	$T_{32,32}$	$G_{8,8}$	$G_{8,32}$	$G_{32,8}$	$G_{32,32}$
Pre-computation time	64	64	64	64	0	0	0	0
Conversion time	10	10	76	40	45	45	660	165
Table size	32	32	32	32	0	0	0	0

- Our method is more advantageous for 32-bit variables on 8-bit microprocessor.
  - Our method works with intermediate 4 bits variable, whereas Goubin's method always works with full 32-bit variables.

Application to SHA-1

- Motivation:
  - MAC algorithms:

$$\text{MAC}_K(x) = \text{SHA-1}(K_1 \| x \| K_2)$$

$$\text{HMAC}_K(x) = \text{SHA-1}(K_2 \| \text{SHA-1}(x \| K_1))$$

- Without appropriate countermeasure:
  - A straightforward DPA recovers the secret-key K.
- Masking Countermeasure:
  - SHA-1 combines 32-bit boolean operations with 32-bit arithmetic operations
  - Conversion is required.

Performances for SHA-1

- Number of elementary operations for each of the 80 iterations step.

	8-bit micro	32-bit micro
Our method	344	155
Goubin's method	864	216

- Conclusion:
  - An implementation of SHA-1 secure against DPA will be roughly 2.7 times faster using our method than using Goubin's method on a 8-bit microprocessor.