Blinded Fault Resistant Exponentiation

Guillaume Fumaroli[1] David Vigilant[2]

[1] Thales Communications
guillaume.fumaroli@fr.thalesgroup.com

[2] Gemalto
david.vigilant@gemalto.com

FDTC ’06

Outline

1. Previous Work
   - Exponentiation in Cryptosystems
   - Algorithms and Attacks

2. Our Algorithm
   - Dialectic / Toward a secure algorithm
   - Algorithm
   - Analysis

Outline

1. Previous Work
   - Exponentiation in Cryptosystems
   - Algorithms and Attacks

2. Our Algorithm
   - Dialectic / Toward a secure algorithm
   - Algorithm
   - Analysis

Definition :

Definition (Group Exponentiation)

- Let $(\mathbb{G}, \times)$ be a group,
- x be an element of $\mathbb{G}$, and k be an integer :

$$x^k = \underbrace{x \times x \times \ldots \times x}_{k \ \ times}$$

Implemented in various cryptosystems – Example 1

RSA Signature (Straightforward Mode)

- *Group :* $(\mathbb{Z}_N^*, \times)$
- *Initialization :* $N = p \cdot q$ *with* p, q *prime*
- *Public Key :* $\{N,e\}$
- *Private Key :* $\{p,q,d\}$ *where* $d \equiv e^{-1} \bmod \varphi(N)$,
- *Let* M *be the message then :*

$$S \equiv \dot{M}^d \bmod N$$

Implemented in various cryptosystems – Example 2

ECDH over $\mathbb{F}_p$ (Static Mode)

- *Group : (E* $(\mathbb{F}_p)$, *Point Addition)*
- *Initialization :* $Q_A = d_A \cdot P$, $Q_B = d_B \cdot P$

$$Q_A = \underbrace{P + P + \ldots + P}_{d_A \ \text{times}}$$

- *Public Keys :* Q_A, Q_B
- *Private Keys :* d_A, d_B

$$A \xrightarrow{Q_A} B$$

$$A \xleftarrow{Q_B} B$$

$$K = d_A \cdot Q_B = d_B \cdot Q_A$$

gemalto

Exponentiation – goals and constraints :

Critical Operation

- *Plays a central role in PKC,*
- *Manipulates sensitive data.*

Constraints in Embedded Devices

- *Costly operation*
- *Variables reach critical sizes*
- *Some parameters not always available to the device*
- *Targeted by side-channel attacks*

$\Rightarrow$ **Build secure standalone exponentiation requiring neither extra parameters nor precomputations**

Exponentiation – goals and constraints :

Critical Operation

- *Plays a central role in PKC,*
- *Manipulates sensitive data.*

Constraints in Embedded Devices

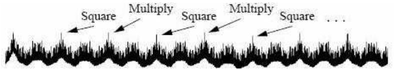
- *Costly operation*
- *Variables reach critical sizes*
- *Some parameters not always available to the device*
- *Targeted by side-channel attacks*

$\Rightarrow$ **Build secure standalone exponentiation requiring neither extra parameters nor precomputations**

Naive Implementation : Square-and-multiply

Input: $x \in \mathbb{G}$, $k = \sum_{i=0}^{t-1} k_i 2^i \in \mathbb{N}$
Output: $x^k \in \mathbb{G}$

$R_0 \leftarrow 1$; $R_1 \leftarrow x$
for $j = t-1$ **down to** 0 **do**
  $R_0 \leftarrow {R_0}^2$
  if $k_j = 1$ **then** $R_0 \leftarrow R_0 R_1$
end for
return R_0



Remark

Square-and-multiply broken by simple power analysis.

Square-and-multiply-always (CHES '99 Coron):

Input: $x \in \mathbb{G}$, $k = \sum_{i=0}^{t-1} k_i 2^i \in \mathbb{N}$
Output: $x^k \in \mathbb{G}$

$R_0 \leftarrow 1$; $R_2 \leftarrow x$
for $j = t-1$ **down to** 0 **do**
  $R_0 \leftarrow {R_0}^2$
  $R_{\bar{k_j}} \leftarrow R_{\bar{k_j}} R_2$
end for
return R_0

Remark

Square-and-multiply-always broken by safe-error attacks (CHES '02 Joye and others).

Montgomery Ladder (CHES '02 Joye and others):

Input: $x \in \mathbb{G}$, $k = \sum_{i=0}^{t-1} k_i 2^i \in \mathbb{N}$
Output: $x^k \in \mathbb{G}$

$R_0 \leftarrow 1$; $R_1 \leftarrow x$
for $j = t-1$ **down to** 0 **do**
  $R_{\bar{k}_j} \leftarrow R_{\bar{k}_j} R_{k_j}$
  $R_{k_j} \leftarrow R_{k_j}^2$
end for
return R_0

Properties

- *Atomic algorithm*
- *No dummy operation*

Montgomery-Ladder is practical and withstands aforementioned attacks

Montgomery Ladder:

Properties

$$R_1 = R_0 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1) = (1, x)$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = x$
- $R_1 \leftarrow R_1^2 = x^2$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3$
- $R_0 \leftarrow R_0^2 = x^2$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5$
- $R_1 \leftarrow R_1^2 = x^6$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11}$
- $R_1 \leftarrow R_1^2 = x^{12}$

Montgomery Ladder:

Properties

$$R_1 = R_0 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1) = (1, x)$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = x$
- $R_1 \leftarrow R_1^2 = x^2$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3$
- $R_0 \leftarrow R_0^2 = x^2$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5$
- $R_1 \leftarrow R_1^2 = x^6$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11}$
- $R_1 \leftarrow R_1^2 = x^{12}$

Montgomery Ladder:

Properties

$$R_1 = R_0 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1) = (1, x)$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = x$
- $R_1 \leftarrow R_1^2 = x^2$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3$
- $R_0 \leftarrow R_0^2 = x^2$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5$
- $R_1 \leftarrow R_1^2 = x^6$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11}$
- $R_1 \leftarrow R_1^2 = x^{12}$

Montgomery Ladder:

Properties

$$R_1 = R_0 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1) = (1, x)$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = x$
- $R_1 \leftarrow R_1^2 = x^2$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3$
- $R_0 \leftarrow R_0^2 = x^2$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5$
- $R_1 \leftarrow R_1^2 = x^6$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11}$
- $R_1 \leftarrow R_1^2 = x^{12}$

Montgomery Ladder:

Properties

$$R_1 = R_0 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1) = (1, x)$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = x$
- $R_1 \leftarrow R_1^2 = x^2$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3$
- $R_0 \leftarrow R_0^2 = x^2$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5$
- $R_1 \leftarrow R_1^2 = x^6$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11}$
- $R_1 \leftarrow R_1^2 = x^{12}$

Outline

Dialectic

Strong constraints example met on smart cards

Implement a blinded 2048-bit RSA signature in straightforward mode:

- *Maximal size of co-processor registers = 2048 bits*
- *p, q and e not available*

How to build a generic blinding of the exponentiation?

Remark

- *Additive mask on base element difficult*
- *Additive mask on private exponent difficult*
- *Precomputation of a mask / refresh (Coron CHES99) difficult*
- *Problem in randomizing projective coordinates (Goubin, Kunz-Jacques CHES05)*

$\Rightarrow$ **Need a generic DPA-immune algorithm**
Only multiplicative mask on the base element well-suited

Dialectic

Strong constraints example met on smart cards

Implement a blinded 2048-bit RSA signature in straightforward mode:

- *Maximal size of co-processor registers = 2048 bits*
- *p, q and e not available*

How to build a generic blinding of the exponentiation?

Remark

- *Additive mask on base element difficult*
- *Additive mask on private exponent difficult*
- *Precomputation of a mask / refresh (Coron CHES99) difficult*
- *Problem in randomizing projective coordinates (Goubin, Kunz-Jacques CHES05)*

$\Rightarrow$ **Need a generic DPA-immune algorithm**
Only multiplicative mask on the base element well-suited

Dialectic

Strong constraints example met on smart cards

Implement a blinded 2048-bit RSA signature in straightforward mode:

- *Maximal size of co-processor registers = 2048 bits*
- *p, q and e not available*

How to build a generic blinding of the exponentiation?

Remark

- *Additive mask on base element difficult*
- *Additive mask on private exponent difficult*
- *Precomputation of a mask / refresh (Coron CHES99) difficult*
- *Problem in randomizing projective coordinates (Goubin, Kunz-Jacques CHES05)*

$\Rightarrow$ Need a generic DPA-immune algorithm
Only multiplicative mask on the base element well-suited

Dialectic / Toward a secure algorithm

In our context, the only practical situation would be the multiplicative mask balanced exponentiation (MMBE):
Let r be a random element in $\mathbb{G}$

1. $S_1 = (\dot{M}r)^d$
2. $S_2 = (r^{-1})^d$
3. $S = S_1 \times S_2 = \dot{M}^d$

Remark

Very costly solution (2 atomic exponentiations required)

Our Algorithm:

Input: $x \in \mathbb{G}$, $k = \sum_{i=0}^{t-1} k_i 2^i \in \mathbb{N}$
Output: $x^k \in \mathbb{G}$

Pick a random $r \in \mathbb{G}$
$R_0 \leftarrow r$; $R_1 \leftarrow rx$; $R_2 \leftarrow r^{-1}$
for $j = t - 1$ **down to** 0 **do**
  $R_{\bar{k}_j} \leftarrow R_{\bar{k}_j} R_{k_j}$
  $R_{k_j} \leftarrow R_{k_j}^2$
  $R_2 \leftarrow R_2^2$
end for
return $R_2 R_0$

Our Algorithm:

Properties

$$R_1 \cdot R_2 = R_0 \cdot R_2 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1, R_2) = (r, xr, r^{-1})$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = xr^2$
- $R_1 \leftarrow R_1^2 = x^2 r^2$
- $R_2 \leftarrow R_2^2 = r^{-2}$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3 r^4$
- $R_0 \leftarrow R_0^2 = x^2 r^4$
- $R_2 \leftarrow R_2^2 = r^{-4}$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5 r^8$
- $R_1 \leftarrow R_1^2 = x^6 r^8$
- $R_2 \leftarrow R_2^2 = r^{-8}$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11} r^{16}$
- $R_1 \leftarrow R_1^2 = x^{12} r^{16}$
- $R_2 \leftarrow R_2^2 = r^{-16}$

gemalto

Our Algorithm:

Properties

$$R_1 \cdot R_2 = R_0 \cdot R_2 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1, R_2) = (r, xr, r^{-1})$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = xr^2$
- $R_1 \leftarrow R_1^2 = x^2 r^2$
- $R_2 \leftarrow R_2^2 = r^{-2}$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3 r^4$
- $R_0 \leftarrow R_0^2 = x^2 r^4$
- $R_2 \leftarrow R_2^2 = r^{-4}$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5 r^8$
- $R_1 \leftarrow R_1^2 = x^6 r^8$
- $R_2 \leftarrow R_2^2 = r^{-8}$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11} r^{16}$
- $R_1 \leftarrow R_1^2 = x^{12} r^{16}$
- $R_2 \leftarrow R_2^2 = r^{-16}$

Our Algorithm:

Properties

$$R_1 \cdot R_2 = R_0 \cdot R_2 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1, R_2) = (r, xr, r^{-1})$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = xr^2$
- $R_1 \leftarrow R_1^2 = x^2 r^2$
- $R_2 \leftarrow R_2^2 = r^{-2}$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3 r^4$
- $R_0 \leftarrow R_0^2 = x^2 r^4$
- $R_2 \leftarrow R_2^2 = r^{-4}$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5 r^8$
- $R_1 \leftarrow R_1^2 = x^6 r^8$
- $R_2 \leftarrow R_2^2 = r^{-8}$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11} r^{16}$
- $R_1 \leftarrow R_1^2 = x^{12} r^{16}$
- $R_2 \leftarrow R_2^2 = r^{-16}$

gemalto

Our Algorithm:

Properties

$$R_1 \cdot R_2 = R_0 \cdot R_2 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1, R_2) = (r, xr, r^{-1})$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = xr^2$
- $R_1 \leftarrow R_1^2 = x^2 r^2$
- $R_2 \leftarrow R_2^2 = r^{-2}$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3 r^4$
- $R_0 \leftarrow R_0^2 = x^2 r^4$
- $R_2 \leftarrow R_2^2 = r^{-4}$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5 r^8$
- $R_1 \leftarrow R_1^2 = x^6 r^8$
- $R_2 \leftarrow R_2^2 = r^{-8}$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11} r^{16}$
- $R_1 \leftarrow R_1^2 = x^{12} r^{16}$
- $R_2 \leftarrow R_2^2 = r^{-16}$

Our Algorithm:

Properties

$$R_1 \cdot R_2 = R_0 \cdot R_2 \times x$$

Example ($k = (1011)_2$)

Initialization: $(R_0, R_1, R_2) = (r, xr, r^{-1})$

Step 1: bit = 1

- $R_0 \leftarrow R_0 R_1 = xr^2$
- $R_1 \leftarrow R_1^2 = x^2 r^2$
- $R_2 \leftarrow R_2^2 = r^{-2}$

Step 2: bit = 0

- $R_1 \leftarrow R_1 R_0 = x^3 r^4$
- $R_0 \leftarrow R_0^2 = x^2 r^4$
- $R_2 \leftarrow R_2^2 = r^{-4}$

Step 3: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^5 r^8$
- $R_1 \leftarrow R_1^2 = x^6 r^8$
- $R_2 \leftarrow R_2^2 = r^{-8}$

Step 4: bit = 1

- $R_0 \leftarrow R_0 R_1 = x^{11} r^{16}$
- $R_1 \leftarrow R_1^2 = x^{12} r^{16}$
- $R_2 \leftarrow R_2^2 = r^{-16}$

Security Analysis:

Properties

1. *Simple Side-Channel Attacks*
    - *Keep Montgomery-Ladder atomicity*
    - *No conditional branching*
2. *Differential Side-Channel Attacks*
    - *Manipulated variables are randomized / decorrelated from inputs-outputs*
    - *Multiplicative mask changes at each loop*
3. *Fault Attacks*
    - *No Dummy Operation*
    - *Whenever a fault is injected:*

Security Analysis:

Properties

1. *Simple Side-Channel Attacks*
   - *Keep Montgomery-Ladder atomicity*
   - *No conditional branching*
2. *Differential Side-Channel Attacks*
   - *Manipulated variables are randomized / decorrelated from inputs-outputs*
   - *Multiplicative mask changes at each loop*
3. *Fault Attacks*
   - *No Dummy Operation*
   - *Whenever a fault is injected:*

Security Analysis:

Properties

1. *Simple Side-Channel Attacks*
   - *Keep Montgomery-Ladder atomicity*
   - *No conditional branching*
2. *Differential Side-Channel Attacks*
   - *Manipulated variables are randomized / decorrelated from inputs-outputs*
   - *Multiplicative mask changes at each loop*
3. *Fault Attacks*
   - *No Dummy Operation*
   - *Whenever a fault is injected:*
     - *Output modified*
     - *Consistency lost between* R_0, R_1 *and* $R_2 \Rightarrow$ *Random output not exploitable*

Security Analysis:

Avoiding exponent / Loop manipulation
(Boneh, DeMillo, Lipton JoC '01)

Input: $x \in \mathbb{G}$, $k = \sum_{i=0}^{t-1} k_i 2^i \in \mathbb{N}$,
$\mathrm{CKS}_{\mathrm{ref}}$ the checksum of k.
Output: $x^k \in \mathbb{G}$

Pick a random $r \in \mathbb{G}$
$R_0 \leftarrow r$; $R_1 \leftarrow rx$; $R_2 \leftarrow r^{-1}$
init(CKS)
for $j = t-1$ **down to** 0 **do**
  $R_{\bar{k_j}} \leftarrow R_{\bar{k_j}} R_{k_j}$
  $R_{k_j} \leftarrow R_{k_j}^2$
  $R_2 \leftarrow R_2^2$
  update(CKS, k_j)
end for
$R_2 \leftarrow R_2 \oplus \mathrm{CKS} \oplus \mathrm{CKS}_{\mathrm{ref}}$
return $R_2 R_0$

Security and Efficiency Analysis:

Exponentiation in Strong Constraints : Comparison				
Security Analysis				
Attacks	Naive	ML	MMBE	Our Algo
SPA, Timing	Not immune	Immune	Immune	Immune
Fault	Not immune	Immune	Immune	Immune
DPA	Not immune	Not immune	Immune	Immune
Complexity and Storage				
Complexity	tS,(t/2)M	tS,tM	2tS,(2t+1)M,1I	2tS,(t+1)M,1I
Buffers	2(or 3)	2(or 3)	3(or 4)	3(or 4)

($t = \lceil \log_2(k) \rceil$)

S: Square / M: Multiplication / I: inversion

Conclusion:

Summary

Blinded Fault Resistant Exponentiation Algorithm:

- *Inherently thwarts all known-attacks*
- *No extra parameters required*
- *At least 25 % decreases complexity compared to MMBE*

Suitable to strong embedded device constraints