A Method for Preventing “Skipping” Attacks

Marc Joye

Exponentiation-Based Cryptography

**Definition (Discrete logarithm problem)**
Let $\mathbb{G}$ be a group of (prime) order $q$
Given elements $x, y \in \mathbb{G}$, find $d$ such that $y = x^d$

- Diffie-Hellman key exchange, ElGamal encryption/signature, . . .
- e.g., $\mathbb{G} \subseteq \mathbb{F}_p^*$ or $\mathbb{G}$ subgroup of an elliptic curve over $\mathbb{F}_p$

**Definition ($e^{\text{th}}$ root problem)**
Let $\mathbb{G}$ be a group of [unknown] order $n$
Given element $x \in \mathbb{G}$ and integer $e > 1$ with $\gcd(e, n) = 1$, find $y$ such that $x = y^e$ (i.e., $y = x^d$ where $d = e^{-1} \mod n$)

- RSA encryption/signature, . . .
- e.g., $\mathbb{G} = (\mathbb{Z}/N\mathbb{Z})^*$ with $N = pq \implies n = \phi(N)$ where $\phi(N) = (p-1)(q-1)$
This Talk

- Efficient method for preventing fault attacks in practical settings for exponentiation-based cryptosystems
  - cryptographic primitives vs. cryptographic protocols
- Most known fault attacks are directed to cryptographic primitives
- Notable exception
  - skipping attacks [Schmidt and Herbst, 2008]
  - fault model experimentally validated

Notation

**Multiplicative notation**
- \((\mathbb{Z}/N\mathbb{Z})^*\)
- \(x \cdot y \pmod{N}\)
- \(y = x^d \pmod{N}\)  
  [exponentiation]

**Additive notation**
- \(E_{F_p} : y^2 = x^3 + ax + b\)
- \(P + Q\)
- \(Q = dP\)  
  [scalar multiplication]
**RSA Primitive**

- **Key generation**
  - **Input** keylength $k$ and $e$
  - **Output** $N = pq$ such that $|N|_2 = k$ and $\gcd(e, \phi(N)) = 1$
    
    $$d = e^{-1} \mod \phi(N)$$
  
  $pk = \{e, N\}$ and $sk = \{d\}$

- **[Plain] RSA encryption**
  - **Input** message $m$ and **public key** $pk$
  - **Output** ciphertext $c = m^e \mod N$

- **[Plain] RSA decryption**
  - **Input** ciphertext $c$ and **private key** $sk$
  - **Output** message $m = c^d \mod N$

(Can be used for signature by “exchanging” the roles of $pk$ and $sk$)

---

**Chinese Remaindering**

- **Computation of a signature $S = m^d \mod N$ using CRT**
  1. $s_p = m^{d_p} \mod p$
  2. $s_q = m^{d_q} \mod q$
  3. $S = \text{CRT}(s_p, s_q) = s_q + q[i_q(s_p - s_q) \mod p]$
Random Errors Against RSA Primitive

![Diagram of RSA errors]

\[ \gcd(\hat{S}^e - m \pmod{N}, N) = q \]

- Equally applies if
  - there is a fault on \( i_q \)
  - a [deterministic] padding is applied to \( m \) (e.g., FDH)

---

EC Primitive

- EC primitive = point multiplication (a.k.a. scalar multiplication)
  \[ E(K) \times \mathbb{Z} \rightarrow E(K), (P, d) \mapsto Q = [d]P \]
  - one-way function

- Cryptographic elliptic curves
  - \( K = \mathbb{F}_q \) with \( q = p \) (a prime) or \( q = 2^m \)
  - \( \#E(K) = hn \) with \( h \in \{1, 2, 3, 4\} \) and \( n \) prime
  - typical size: \( |n|_2 = 160 \) (\( \approx |K|_2 \))

**Definition (ECDL Problem)**

Let \( G = \langle P \rangle \subseteq E(K) \) a subgroup of prime order \( n \)

Given points \( P, Q \in G \), compute \( d \) such that \( Q = [d]P \)
Random Errors Against EC Primitive

Attack model

- EC parameters are in non-volatile memory
  - permanent faults in a unknown position, in any system parameter
  - transient fault during parameter transfer

Adversary’s goal

- Recover the value of \( d \) in the computation of \( Q = [d]P \)

Key Observation (1/2)

Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \)

\[ P + Q = (x_3, y_3) \] where

\[
x_3 = \lambda^2 + a_1 \lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3) \lambda - y_1 - a_1 x_3 - a_3
\]

with \( \lambda = \begin{cases} 
\frac{y_1 - y_2}{x_1 - x_2} & \text{[addition]} \\
\frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3} & \text{[doubling]}
\end{cases} \)

- Parameter \( a_6 \) is not involved in point addition (or point doubling)
Key Observation (2/2)

\[
E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6
\]

- If a ‘point’ \( \tilde{P} = (\tilde{x}, \tilde{y}) \in \mathbb{F}_q \times \mathbb{F}_q \) but \( \tilde{P} \notin E \) then the computation of \( \tilde{Q} = [d]\tilde{P} \) will take place on the curve

\[
\tilde{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + \tilde{a}_6
\]

where \( \tilde{a}_6 = \tilde{y}^2 + a_1\tilde{x}\tilde{y} + a_3\tilde{y} - \tilde{x}^3 - a_2\tilde{x}^2 - a_4\tilde{x} \)

- Now if
  1. \( \text{ord}_{\tilde{E}}(\tilde{P}) = t \) is small
  2. discrete logarithms are computable in \( \langle \tilde{P} \rangle \)

then

\[ d \pmod{t} \]

can be recovered from \( \tilde{Q} \)

---

Chosen Input Point Attack

- Construct a ‘point’ \( \tilde{P}_i = (\tilde{x}_i, \tilde{y}_i) \in \tilde{E}_i \) such that
  1. \( \text{ord}_{\tilde{E}_i}(\tilde{P}_i) = t_i \) is small
  2. discrete logarithms are computable in \( \langle \tilde{P}_i \rangle \)
- Query the device with \( \tilde{P}_i \) and receive \( \tilde{Q}_i = [d]\tilde{P}_i \)
- Solve the discrete logarithm and recover \( d \pmod{t_i} \)
- Iterating the process gives
  - \( d \pmod{t_i} \) for several \( t_i \)
  - \( d \) by Chinese remaindering

(This attack can easily be prevented using the curve equation)
Faults in the Curve Parameters

Recover $d$ in $Q = [d]P$ on $E_{\mathbb{F}_p}$: $y^2 = x^3 + a_4x + a_6$

- **Fault:** $a_4 \rightarrow \hat{a}_4$
- **Device outputs** $\hat{Q} = [d]P$ on $\hat{E}$: $y^2 = x^3 + \hat{a}_4x + \hat{a}_6$
- $\hat{Q} = [d](x_1, y_1) = (\hat{x}_d, \hat{y}_d) \in \hat{E}$
- **Two equations:**
  
  \[
  \begin{align*}
  y_1^2 &= x_1^3 + \hat{a}_4x_1 + \hat{a}_6 \\
  \hat{y}_d^2 &= \hat{x}_d^3 + \hat{a}_4\hat{x}_d + \hat{a}_6
  \end{align*}
  \]
  
  $\Rightarrow \hat{a}_4 = \ldots, \hat{a}_6 = \ldots$

- **Compute** $d \mod t$ from $\hat{Q} = [d]P$

---

Basic Countermeasures

- **Add CRC checks**
- **Randomize the computation**
  - e.g., $d \leftarrow d + r \phi(N)$
- **Compute the signatures twice**
  - doubles the **running time**
- **Verify the signatures**
  - efficient (as public verification exponent $e$ is small)
  - . . . but requires the **knowledge of $e$**
- (Also applies for [plain] decryption)
Signature Verification

- Most natural way to protect a signature scheme:

  **Check that a signature is correct before outputting it**

  - requires little overhead as public exponent $e$ is typically small in practice... for RSA
  - but assumes that the value of exponent $e$ is available!

---

Shamir’s Method and Variants

**Shamir’s countermeasure**

1. Choose a (small) random integer $r$
2. Compute $S^* = \hat{m}^d \mod rN$ and $Z = \hat{m}^d \mod r$
3. If $S^* \equiv Z \pmod{r}$ then output $S = S^* \mod N$, otherwise return error

- Variants
  - can be adapted to CRT mode
  - state-of-the-art (many variants were proposed)
    - Berzati, Canovas, and Goubin (FDTC 2008) / Vigilant (CHES 2008)

- Drawbacks
  - cannot guarantee to detect all faults with a 100% detection
  - impact the performance and usability
    - running time and memory requirements
    - [personalization process]
Giraud’s Method and Variants

Giraud’s countermeasure

1. Compute $\hat{m}^d \mod N$ using Montgomery ladder and obtain the pair $(Z, S) = (\hat{m}^{d-1} \mod N, \hat{m}^d \mod N)$

2. If $Z \hat{m} \equiv S \pmod{N}$ then output $S$, otherwise return error

- **Variants**
  - infective computation
  - can be adapted to CRT mode
  - Rivain (CT-RSA 2009): double ladder $(\hat{m}^a, \hat{m}^b)$
    - set $a = d$ and $b = \phi(N) - d$
    - check whether $\hat{m}^d \hat{m}^{\phi(N)-d} \equiv 1 \pmod{N}$

- **Drawbacks**
  - imposes the exponentiation algorithm
  - Rivain’s method more suited to CRT mode

Performance vs. Security

**Encryption schemes**
- Standard security notion: IND-CCA2
  - RSA-OAEP, ECIES, . . .
  - decryption process includes some built-in validity check
  - protection against errors (faults) appears unnecessary

**Signature schemes**
- No such built-in protection
- But signatures can be explicitly checked
  - not always possible or costly
- State-of-the-art
  - RSA-PSS (probabilistic padding):
    - no known [practical] attacks
    - (even proved secure against random errors)
  - RSA-FDH (deterministic padding):
    - GCD attack in CRT mode
    - skipping attack in standard or CRT modes
  - DSA and ECDSA:
    - skipping attack...
Skipping Attacks

Attack assumes that the attacker manages to skip a squaring operation
- can be seen as a random error at the bit level

Algorithm 1 Square-and-multiply

Input: $x \in \mathbb{G}$, $d = (d_{t-1}, \ldots, d_0)_2$
Output: $y = x^d$

1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: for $i = t - 1$ down to 0 do
    3: $R_0 \leftarrow R_0^2$
    4: if $d_i = 1$ then $R_0 \leftarrow R_0 \cdot R_1$
5: return $R_0$

Application to DSA-like Signature Schemes

- DSA-like signatures
  - Key generation: group $\mathbb{G} = \langle g \rangle$ of prime order $n$;
    conversion function $F : \mathbb{G} \rightarrow \mathbb{Z}$; hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}/n\mathbb{Z}$;
    $y = g^d$ for some secret exponent $d \in \mathbb{Z}/n\mathbb{Z}$
  - $pk = \{g, y\}$ and $sk = \{d\}$

- Signing
  - Input message $m$ and private key $sk$
  - Output signature $S = (r, s)$

1. pick a random $k \in \{1, \ldots, n - 1\}$
2. compute $z = g^k$ and set $r = F(z) \mod n$
3. if $r = 0$ then goto Step 1
4. compute $s = (H(m) + d \cdot r) / k \pmod{n}$
5. return $S = (r, s)$

- Verification

1. compute $u_1 = H(m) / s \mod n$ and $u_2 = r / s \mod n$
2. check whether $F(g^{du_1} y^{u_2}) \equiv r \pmod{n}$
 Skipping Attack Against DSA-like Signatures

- squaring skipped at iteration $j$
- $z \sim \hat{z}$ where
  $$\hat{z} = \prod_{i=j+1}^{t-1} g^{k_i 2^{i-1}} \cdot \prod_{i=0}^{j} g^{k_i 2^i} = (z \cdot g^\tilde{k})^{1/2}$$
- $(r, s) \sim (\hat{r}, \hat{s})$

Algorithm 2 Square-and-multiply

| Input: $g \in \mathbb{G}$, $k = (k_{t-1}, \ldots, k_0)_2$ |
| Output: $z = g^k$ |

1: $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow g$
2: for $i = t - 1$ down to 0 do
3: \hspace{1cm} $R_0 \leftarrow R_0^2$
4: \hspace{1cm} if $k_i = 1$ then $R_0 \leftarrow R_0 \cdot R_1$
5: \hspace{1cm} return $R_0$

Observation:

$$g^{\hat{u}_1} y^{\hat{u}_2} = g^{h(m)/s} y^{\hat{r}} = g^{h(m) + d\hat{r}} = g^k$$

$$\hat{r} \equiv F((z \cdot g^\tilde{k})^{1/2}) \pmod{n}$$

with $z = g^{\hat{u}_1} y^{\hat{u}_2} \implies \tilde{k} = \ldots$

---

General Description

Exponentiation-based cryptosystem

On input an element $x$ in a group $\mathbb{G}$ and an exponent $d$, one has to evaluate $y = x^d$

Idea

Compute \textbf{in parallel} with $y = x^d$ in $\mathbb{G}$ the value of $f = e^d$ in a group $\mathbb{G}'$ such that

1. computations in $\mathbb{G}'$ are fast
2. evaluating discrete logarithms in $\mathbb{G}'$ is easy

Examples

- $\mathbb{G}' = \mathbb{Z}^+$; i.e., the additive group of integers
- $\mathbb{G}' = (\mathbb{Z}/\Omega\mathbb{Z})^+$; i.e., the additive group of integers modulo $\Omega$
Preventing Skipping Attacks (I)

- $G' = \mathbb{Z}^+$; i.e., the additive group of integers
- group operation is addition over $\mathbb{Z}$ \iff $f = d \cdot e$
- choosing $e = 1$ yields $f = d$

Algorithm 3 Protected square-and-multiply (I)

Input: $x \in G$, $d = (d_{t-1}, \ldots, d_0)_2$
Output: $y = x^d$

1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: $T_0 \leftarrow 0$; $T_1 \leftarrow 1$
3: for $i = t - 1$ down to 0 do
4: \quad $(R_0, T_0) \leftarrow (R_0^2, 2 \cdot T_0)$
5: \quad if $(d_i = 1)$ then $(R_0, T_0) \leftarrow (R_0 \cdot R_1, T_0 + T_1)$
6: \quad if $(T_0 \not\equiv d \pmod{\Omega})$ then return error
7: return $R_0$

Preventing Skipping Attacks (II)

- $G' = (\mathbb{Z}/\Omega \mathbb{Z})^+$; i.e., the additive group of integers modulo $\Omega$
- group operation is addition modulo $\Omega$ \iff $f = d \cdot e \pmod{\Omega}$
- choosing $e = 1$ yields $f = d \mod{\Omega}$

Algorithm 4 Protected square-and-multiply (II)

Input: $x \in G$, $d = (d_{t-1}, \ldots, d_0)_2$
Output: $y = x^d$

1: $R_0 \leftarrow 1$; $R_1 \leftarrow x$
2: $T_0 \leftarrow 0$; $T_1 \leftarrow 1$
3: for $i = t - 1$ down to 0 do
4: \quad $(R_0, T_0) \leftarrow (R_0^2, 2 \cdot T_0 \pmod{\Omega})$
5: \quad if $(d_i = 1)$ then $(R_0, T_0) \leftarrow (R_0 \cdot R_1, T_0 + T_1 \pmod{\Omega})$
6: \quad if $(T_0 \not\equiv d \pmod{\Omega})$ then return error
7: return $R_0$
Glued Multiplication

Algorithm 5 Example of glued multiplication

Input: $R_0, R_1, T_0, T_1, r, r'$

Output: $(R_0 \cdot R_1, T_0 + T_1)$

1: $A \leftarrow r'$
2: $T_0 \leftarrow T_0$
3: $A \leftarrow R_0 \cdot R_1$
4: $A \leftarrow A \oplus r$
5: $T_0 \leftarrow T_0 + T_1$
6: $R_0 \leftarrow A \oplus r$
7: return $(R_0, T_0)$

- If one of the above instructions is skipped, this will result in
  - a random value for $R_0 \rightarrow$ will be of no use for the attacker; or
  - an incorrect value for $T_0 \rightarrow$ this will be detected

Summary

- Efficient method for preventing skipping attacks
  - induced overhead is minimal and does not impact the overall performance
- Generic method
  - can accommodate any exponentiation algorithm
  - can easily be combined with other counter-measures