Cryptography and Embedded System Security
CRAESS_I

Xiaolu Hou

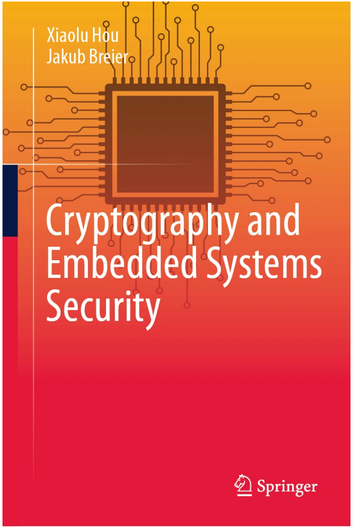
FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Abstract algebra and number theory
- Introduction to cryptography
- Symmetric block ciphers and their implementations
- RSA, RSA signatures, and their implementations
- Probability theory and introduction to SCA
- SPA and non-profiled DPA
- Profiled DPA
- SCA countermeasures
- FA on RSA and countermeasures
- FA on symmetric block ciphers
- FA countermeasures for symmetric block cipher
- Practical aspects of physical attacks
  - Invited speaker: Dr. Jakub Breier, Senior security manager, TTControl GmbH

Recommended reading

- Textbook
    - Sections
        - 5.3.1, 5.3.4
        - 5.4.1, 5.4.4



Lecture Outline

- Introduction to Fault Attacks
- Recall – RSA Signatures
- Bellcore Attack and Countermeasures
- Safe-error Attack and Countermeasure

FA on RSA and countermeasures

- Introduction to Fault Attacks
- Recall – RSA Signatures
- Bellcore Attack and Countermeasures
- Safe-error Attack and Countermeasure

Why are we interested in physical attacks?

- Cryptography provides algorithms that enable secure communication in theory
- In the real world, these algorithms have to be implemented on real devices:
  - Software implementations: general-purpose devices
  - Hardware implementations: dedicated secure hardware devices
- To evaluate the security level of cryptographic implementations, it is necessary to include a physical security assessment

Targets and Attack Goals

Targets

- Credit cards
- Passports
- Key Fob
- ...

Goals:

- Recovery of the secret key
- Privilege escalation
- IP theft
- ...

picture source: https://goodtimes.ca/how-many-credit-cards-is-too-many/

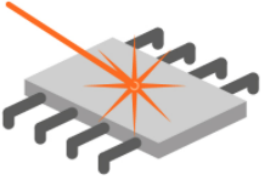
Different Physical Attack Methods

- Side-channel attacks
  - EM/Power analysis
  - Timing analysis
  - Cache attacks
- Fault attacks
  - Optical fault injection
  - Electromagnetic fault injection
  - Clock/voltage glitch
- Hardware Trojans
- ...

picture source: https://nl.dreamstime.com/stock-foto-s-hamer-die-computer-boos-tonen-raken-image34210923

High Level Description of Fault Attacks

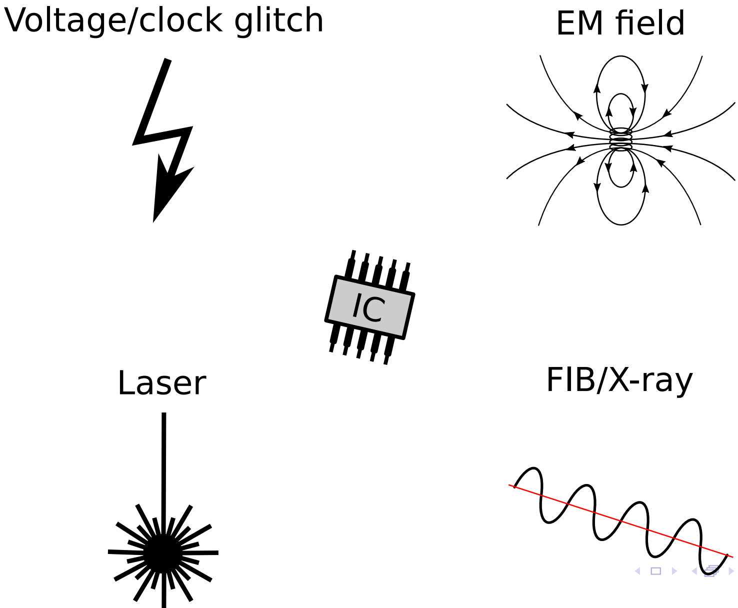
- Active attacks, the attacker tries to perturb the internal computations by external means
- Exploit a scenario where the attacker has access to the device and can tamper with it
- There exist also techniques that can achieve fault attacks remotely, such as Rowhammer[1]



[1] Kim, Yoongu, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors." ACM SIGARCH Computer Architecture News 42.3 (2014): 361-372.

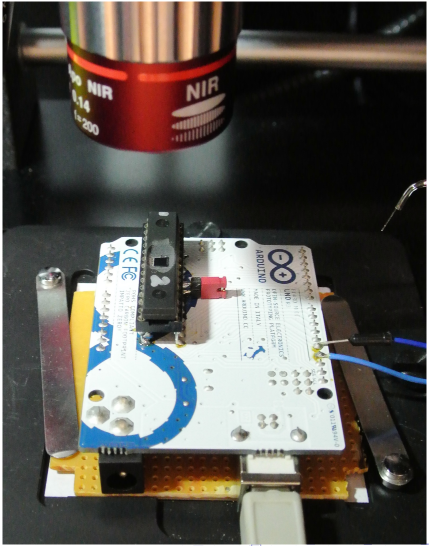
picture source: https://blog.applus.com/

Fault Injection Techniques



Laser Fault Injection Setup

By carefully tuning the beam's energy level below a destructive threshold, it is possible to inject faults into a device and it will not suffer any permanent damage



Fault Effects

- Instruction skip/change
  - Perturbs the instruction being executed by modifying the opcode for the instruction
- Bit flip
  - Flips the bits in the data.
  - The number of bits affected is normally limited by the size of the registers.
  - For example, for an AVR device, we can have $m-$bit flips for $m = 1, 2, \ldots, 8$
- Bit set/reset
  - Fixes the bit value to be 1 (set) or 0 (reset)
- Random byte fault
  - Changes the byte value to a random number
- Stuck-at faults
  - Permanently changes the value of one bit to 0 (stuck-at-0) or 1 (stuck-at-1)
- ...

Fault Types

- *Permanent fault*
  - Destructive fault that changes the value of a memory cell permanently and hence affects data during the computations
- *Transient fault*
  - The circuit recovers its original behavior after the fault stimulus ceases (usually just one instruction) or after the device reset
  - Can perturb both data and instruction
- In this course, we only consider transient faults

Fault Attack

- First introduced by Boneh et al. to attack implementation of RSA with CRT[1]
- After the fault injection, there are two possible scenarios
  - The output (ciphertext) is faulty
  - Fault is ineffective and the ciphertext is not changed
  - Both scenarios can be exploited
- Attacker goal: recover secret key
- Developed on the algorithmic level
- There are also implementation-specific vulnerabilities

[1] Boneh, D., DeMillo, R. A., & Lipton, R. J. (1997, May). On the importance of checking cryptographic protocols for faults. In International conference on the theory and applications of cryptographic techniques (pp. 37-51). Springer, Berlin, Heidelberg.

Remarks

- Fault attacks on public key ciphers depend on the underlying intractable problem and we do not have a systematic methodology.
- However, the general attack concept can be applied to ciphers based on similar intractable problems.
- We will discuss a few attacks on RSA signatures and the corresponding countermeasures
- The attacks can also be applied to RSA decryption process

FA on RSA and countermeasures

- Introduction to Fault Attacks
- Recall – RSA Signatures
- Bellcore Attack and Countermeasures
- Safe-error Attack and Countermeasure

RSA

Definition (RSA)

Let $n = pq$, where p, q are distinct prime numbers. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, $\mathcal{K} = \mathbb{Z}^*_{\varphi(n)} - \{ 1 \}$. For any $e \in \mathcal{K}$, define encryption

$$E_e : \mathbb{Z}_n \to \mathbb{Z}_n, \quad m \mapsto m^e \bmod n,$$

and the corresponding decryption

$$D_d : \mathbb{Z}_n \to \mathbb{Z}_n, \quad c \mapsto c^d \bmod n,$$

where $d = e^{-1} \bmod \varphi(n)$. The cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{E} = \{ E_e : e \in \mathcal{K} \}$, $\mathcal{D} = \{ D_d : d \in \mathcal{K} \}$, is called *RSA*.

- $\varphi(n) = (p - 1)(q - 1)$
- Public key: n, e, RSA modulus, encryption exponent
- Private key: d, decryption exponent

RSA signatures

- To use RSA for digital signature, let p and q be two distinct primes.
- $n = pq$, choose $e \in \mathbb{Z}^*_{\varphi(n)} - \{1\}$ and compute $d = e^{-1} \mod \varphi(n)$.
- Same as for RSA, the public key consists of e and n.
- d is the private key.
- p, q and $\varphi(n)$ should also be kept secret.

RSA signatures

To sign a message m, Alice computes the signature

$$s = m^d \bmod n.$$

Then Alice sends both m and s to Bob. To verify the signature, Bob computes

$$s^e \bmod n.$$

If $s \equiv m \bmod n$, then the verification algorithm outputs true, and false otherwise.

- Up to now, the only method known to compute s from $m \bmod n$ is using d, so if the verification algorithm outputs true, Bob can conclude that Alice is the owner of d.
- RSA signatures are commonly used together with a fast public hash function h – m will be the hashed value of the message

RSA signatures – Example

Example

- Alice chooses $p = 5$ and $q = 7$.
- Then
$$n = 35, \quad \varphi(n) = 24$$
- Suppose Alice chooses $e = 5$, which is coprime to 24.
- By the extended Euclidean algorithm
$$d = e^{-1} \bmod \varphi(n) = ?$$

RSA signatures – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5.$$

- By the extended Euclidean algorithm

$$24 = 5 \times 4 + 4,\ 5 = 4 + 1 \Longrightarrow 1 = 5 - (24 - 5 \times 4) = 24 \times (-4) + 5 \times 5,$$

  and $d = e^{-1} \bmod 24 = 5$.
- To sign message (hashed value) $m = 10$, Alice computes

$$s = m^d \bmod n = ?$$

- Alice sends both the message and signature to Bob.
- Bob verifies the signature

$$s^e \bmod n = ?$$

RSA signatures – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

- To sign message (hashed value) $m = 10$, Alice computes

$$s = m^d \bmod n = 10^5 \bmod 35 = 5.$$

- Alice sends both the message (hashed value) $m = 10$ and signature $s = 5$ to Bob.
- Bob verifies the signature

$$s^e \bmod n = 5^5 \bmod 35 = 10 = m.$$

CRT-based RSA implementation

By the Chinese Remainder Theorem, finding the solution for $x \equiv a^d \bmod n$ is equivalent to solving

$$x \equiv a^d \bmod p, \quad x \equiv a^d \bmod q.$$

We can compute

$$x_p := a^{d \bmod (p-1)} \bmod p, \quad x_q := a^{d \bmod (q-1)} \bmod q,$$

and solve for

$$x \equiv x_p \bmod p, \quad x \equiv x_q \bmod q.$$

An implementation that computes $a^d \bmod n$ by solving the above equation is called *CRT-based RSA*.

CRT-based RSA implementation

To compute $a^d \bmod n$. We calculate

$$x_p := a^{d \bmod (p-1)} \bmod p, \qquad x_q := a^{d \bmod (q-1)} \bmod q,$$

$$M_q = q, \quad M_p = p,$$

$$y_q = M_q^{-1} \bmod p = q^{-1} \bmod p, \quad y_p = M_p^{-1} \bmod q = p^{-1} \bmod q,$$

Gauss's algorithm

$$a^d \bmod n = x_p y_q q + x_q y_p p \bmod n$$

Garner's algorithm

$$a^d \bmod n = x_p + ((x_q - x_p) y_p \bmod q) p.$$

CRT-based RSA implementation – Example

CRT-based RSA implementation

$$x_p := a^{d \bmod (p-1)} \bmod p, \qquad x_q := a^{d \bmod (q-1)} \bmod q,$$

$$M_q = q, \quad M_p = p,$$

$$y_q = M_q^{-1} \bmod p = q^{-1} \bmod p, \quad y_p = M_p^{-1} \bmod q = p^{-1} \bmod q,$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

To sign message (hashed value) $m = 10$, with CRT-based RSA implementation

$$s_p = ? \quad s_q = ? \quad y_p = ? \quad y_q = ?$$

CRT-based RSA implementation – Example

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

To sign message (hashed value) $m = 10$, with CRT-based RSA implementation, Alice computes

$$\begin{aligned} s_p &= m^{d \bmod (p-1)} \bmod p = 10^{5 \text{ mod } 4} \bmod 5 = 0, \\ s_q &= m^{d \bmod (q-1)} \bmod q = 10^{5 \text{ mod } 6} \bmod 7 = 5. \end{aligned}$$

By the extended Euclidean algorithm

$$7 = 5 + 2,\ 5 = 2 \times 2 + 1 \Longrightarrow 1 = 5 - 2 \times (7 - 5) = 5 \times 3 - 2 \times 7$$

$$\begin{aligned} y_p &= p^{-1} \bmod q = 3 \bmod 7 = 3, \\ y_q &= q^{-1} \bmod p = -2 \bmod 5 = 3. \end{aligned}$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$a^d \bmod n = x_p y_q q + x_q y_p p \bmod n$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Gauss's algorithm

$$s = ?$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$a^d \bmod n = x_p y_q q + x_q y_p p \bmod n$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Gauss's algorithm

$$s = s_p y_q q + s_q y_p p \bmod n = 5 \times 3 \times 5 \bmod 35 = 5.$$

CRT-based RSA implementation – Example

Garner's algorithm

$$a^d \bmod n = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Garner's algorithm

$$s = ?$$

CRT-based RSA implementation – Example

Garner's algorithm

$$a^d \bmod n = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Garner's algorithm

$$s = s_p + ((s_q - s_p)y_p \bmod q)p = 0 + (5 \times 3 \bmod 7) \times 5 = 1 \times 5 = 5.$$

FA on RSA and countermeasures

- Introduction to Fault Attacks
- Recall – RSA Signatures
- Bellcore Attack and Countermeasures
- Safe-error Attack and Countermeasure

Background

- Boneh, D., DeMillo, R. A., & Lipton, R. J. (1997, May). On the importance of checking cryptographic protocols for faults. In International conference on the theory and applications of cryptographic techniques (pp. 37-51). Springer, Berlin, Heidelberg.
- *Transient fault*
  - The circuit recovers its original behavior after the fault stimulus ceases (usually just one instruction) or after the device reset
  - Can perturb both data and instruction
- Implementation dependent – CRT-based
- Given one faulty signature, with knowlege of correct signature, attacker can factor the RSA modulus
- Bellcore – name of the company
- The first paper that introduced fault attacks to cryptographic implementations

Bellcore attack

- y_p and y_q can be precomputed – assume no faults
- By the design of s_p and s_q, we have

$$s \equiv s_q \bmod q, \quad s \equiv s_p \bmod p,$$

- Suppose a malicious fault was induced during the signing of the signature and the computation of s_p or s_q, but not both, is corrupted.

Bellcore attack

$$s \equiv s_q \bmod q, \quad s \equiv s_p \bmod p$$

- Assume that s_p is faulty and s_q is computed correctly.
- A similar attack applies if s_q is faulty and s_p is correct.
- Let s' denote the faulty signature, then

$$s' \equiv s \equiv s_q \bmod q, \quad s' \not\equiv s \bmod p.$$

- In other words,

$$q | (s' - s), \quad p \nmid (s' - s).$$

- n and e are public.
- If the attacker further has the knowledge of s and s', then they can compute

$$q = \gcd(s' - s, n), \quad p = \frac{n}{q}.$$

- How does the attacker compute the private key?

Bellcore attack

$$s \equiv s_q \bmod q, \quad s \equiv s_p \bmod p,$$

- Assume that s_p is faulty and s_q is computed correctly.
- Let s' denote the faulty signature, then

$$s' \equiv s \equiv s_q \bmod q, \quad s' \not\equiv s \bmod p \Longrightarrow q|(s'-s), \quad p \nmid (s'-s).$$

- If the attacker further has the knowledge of s and s', then they can compute

$$q = \gcd(s'-s, n), \quad p = \frac{n}{q}.$$

- After factorizing n, the attacker can compute

$$\varphi(n) = (p-1)(q-1)$$

- And eventually, recover the private key

$$d = e^{-1} \bmod \varphi(n)$$

by the extended Euclidean algorithm

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

We have computed that $y_q = 3$ and $y_p = 3$. Suppose $m = 6$. With CRT-based RSA, to calculate the signature, Alice computes

$$s_p = m^{d \bmod (p-1)} \bmod p = ?$$
$$s_q = m^{d \bmod (q-1)} \bmod q = ?$$

And the signature

$$s = s_p + ((s_q - s_p)y_p \bmod q)p = ?$$

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

With CRT-based RSA, to calculate the signature, Alice computes

$$\begin{aligned} s_p &= m^{d \bmod (p-1)} \bmod p = 6^{5 \bmod 4} \bmod 5 = 1, \\ s_q &= m^{d \bmod (q-1)} \bmod q = 6^{5 \bmod 6} \bmod 7 = 6. \end{aligned}$$

$$s = s_p + ((s_q - s_p)y_p \bmod q)p = 1 + ((6-1) \times 3 \bmod 7) \times 5 = 6.$$

We can verify that

$$s^e \bmod n = 6^5 \bmod 35 = 6 = m.$$

Now suppose the computation of s_p is faulty and $s'_p = 3$. Then the faulty signature $s' = ?$

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

$$s_p = 1, \quad s_q = 6, \quad s = 6$$

Now suppose the computation of s_p is faulty and $s'_p = 3$. Then we have

$$s' = s'_p + ((s_q - s'_p)y_p \bmod q)p = 3 + ((6-3) \times 3 \bmod 7) \times 5 = 3 + 2 \times 5 = 13.$$

If the attacker has the knowledge of $s = 6$ and $s' = 13$, they can compute

$$q = \gcd(s' - s, n) = ?$$

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

$$s_p = 1, \quad s_q = 6, \quad s = 6$$

Now suppose the computation of s_p is faulty and $s'_p = 3$. Then we have

$$s' = s'_p + ((s_q - s'_p)y_p \bmod q)p = 3 + ((6-3) \times 3 \bmod 7) \times 5 = 3 + 2 \times 5 = 13.$$

If the attacker has the knowledge of $s = 6$ and $s' = 13$, they can compute

$$q = \gcd(s' - s, n) = \gcd(13 - 6, 35) = \gcd(7, 35) = 7.$$

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

$$s_p = 1, \quad s_q = 6, \quad s = 6$$

Similarly, suppose the computation of s_q is faulty and $s'_q = 2$. Then

$$s' = s_p + ((s'_q - s_p)y_p \bmod q)p = ?$$

If the attacker has the knowledge of s and s', they can compute

$$p = \gcd(s' - s, n) = ?$$

Bellcore attack – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

$$s_p = 1, \quad s_q = 6, \quad s = 6$$

Similarly, suppose the computation of s_q is faulty and $s'_q = 2$. Then

$$s' = s_p + ((s'_q - s_p)y_p \bmod q)p = 1 + ((2-1) \times 3 \bmod 7) \times 5 = 16.$$

If the attacker has the knowledge of $s = 6$ and $s' = 16$, they can compute

$$p = \gcd(s' - s, n) = \gcd(16 - 6, 35) = \gcd(10, 35).$$

By the Euclidean algorithm

$$35 = 10 \times 3 + 5, \quad \gcd(10, 35) = \gcd(10, 5) = 5.$$

Bellcore attack – a different attack

- Lenstra, A. K. (1996). Memo on RSA signature generation in the presence of faults.
- Assume the attacker does not have the correct signature s
- But has the knowledge of the faulty signature s' as well as the original message hash value m.
    - For example, the attacker can request Alice for the signature of a chosen message.

Bellcore attack – a different attack

- y_p and y_q can be precomputed – assume no faults
- By the design of s_p and s_q, we have

$$s \equiv s_q \bmod q, \quad s \equiv s_p \bmod p,$$

- which gives

$$m \equiv s^e \equiv s_q^e \bmod q, \quad m \equiv s^e \equiv s_p^e \bmod p.$$

- A malicious fault was induced during the signing of the signature and the computation of s_p or s_q, but not both, is corrupted.
- Suppose s_p is faulty
- Then

$$s'^e \equiv m \bmod q, \quad s'^e \not\equiv m \bmod p,$$

i.e.

$$q|(s'^e - m), \quad p \nmid (s'^e - m).$$

- How can the attacker find q?

Bellcore attack – a different attack

- y_p and y_q can be precomputed – assume no faults
- By the design of s_p, s_q, y_p and y_q, we have

$$s \equiv s_q \bmod q, \quad s \equiv s_p \bmod p,$$

- which gives

$$m \equiv s^e \equiv s_q^e \bmod q, \quad m \equiv s^e \equiv s_p^e \bmod p.$$

- Suppose a malicious fault was induced during the signing of the signature and the computation of s_p or s_q, but not both, is corrupted.
- Then

$$s'^e \equiv m \bmod q, \quad s'^e \not\equiv m \bmod p,$$

i.e.

$$q | (s'^e - m), \quad p \nmid (s'^e - m).$$

- The attacker can compute

$$q = \gcd(s'^e - m, n), \quad p = \frac{n}{q}.$$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad \varphi(n) = 10 \times 12 = 120.$$

- Choose $e = 11$, which is coprime with $\varphi(n)$.
- By the extended Euclidean algorithm

$$d = ?$$

$$y_q = q^{-1} \bmod p = 13^{-1} \bmod 11 = ? \quad y_p = p^{-1} \bmod q = ?$$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad e = 11$$

$$120 = 11\times 10+10,\ 11 = 10\times 1+1 \Longrightarrow 1 = 11-(120-11\times 10) = 11\times 11-120 \Longrightarrow d = 11$$

$$13 = 11\times 1+2,\ 11 = 2\times 5+1 \Longrightarrow 1 = 11-2\times 5 = 11-5\times(13-11) = 11\times 6-13\times 5,$$

$$y_q = q^{-1} \bmod p = 13^{-1} \bmod 11 = -5 \bmod 11 = 6,$$

$$y_p = p^{-1} \bmod q = 11^{-1} \bmod 13 = 6.$$

- Suppose $m = 2$. To calculate the signature, Alice computes

$$s_p = m^{d \bmod (p-1)} \bmod p = ?$$

$$s_q = m^{d \bmod (q-1)} \bmod q = ?$$

By Garner's algorithm, $s = ?$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad e = 11, \quad d = 11, \quad y_q = 6, \quad y_p = 6$$

- Suppose $m = 2$.
- To calculate the signature, Alice computes

$$s_p = m^{d \bmod (p-1)} \bmod p = 2^{11 \bmod 10} \bmod 11 = 2 \bmod 11 = 2,$$
$$s_q = m^{d \bmod (q-1)} \bmod q = 2^{11 \bmod 12} \bmod 13 = 7.$$

- By Garner's algorithm,

$$s = s_p + ((s_q - s_p)y_p \bmod q)p = 2 + ((7-2) \times 6 \bmod 13) \times 11 = 2 + 4 \times 11 = 46.$$

- Suppose the computation of s_p is faulty and $s'_p = 7$.
- Then $s' = ?$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad y_q = 6, \quad y_p = 6, \quad s_p = 2, \quad s_q = 7, \quad s = 46, \quad s'_p = 7$$

- We have

$$s' = s'_p + ((s_q - s'_p)y_p \bmod q)p = 7 + ((7-7) \times 6 \bmod 13) \times 11 = 7.$$

- If the attacker has the knowledge of $s = 46$ and $s' = 7$, they can compute

$$q = \gcd(s' - s, n) = ?$$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad e = 11, \quad d = 11, \quad s = 46, \quad s' = 7$$

- If the attacker has the knowledge of $s = 46$ and $s' = 7$, they can compute

$$q = \gcd(s' - s, n) = \gcd(7 - 46, 143) = \gcd(-39, 143) = \gcd(39, 143).$$

- By the Euclidean algorithm,

$$\begin{aligned} &143 = 39 \times 3 + 26, && \gcd(39, 143) = \gcd(39, 26), \\ &39 = 26 + 13, && \gcd(39, 26) = \gcd(26, 13), \\ &26 = 13 \times 2, && q = \gcd(26, 13) = 13. \end{aligned}$$

- If the attacker has the knowledge of $s' = 7$ and $m = 2$, they can compute $q = \gcd(s'^e - m, n) = ?$

Bellcore attack – Example

Example

$$p = 11, \quad q = 13, \quad n = 143, \quad s_p = 2, \quad s_q = 7, \quad s = 46, \quad s'_p = 7, \quad s' = 7$$

If the attacker has the knowledge of $s' = 7$ and $m = 2$, they can compute

$$q = \gcd(s'^e - m, n) = \gcd(7^{11} - 2, 143) = \gcd(1977326741, 143).$$

By the Euclidean algorithm,

$$\begin{aligned}
&1977326741 = 143 \times 13827459 + 104, && \gcd(1977326741, 143) = \gcd(143, 104),\\
&143 = 104 + 39, && \gcd(143, 104) = \gcd(104, 39),\\
&104 = 39 \times 2 + 26, && \gcd(104, 39) = \gcd(39, 26),\\
&39 = 26 + 13, && \gcd(39, 26) = \gcd(26, 13)\\
&26 = 13 \times 2, && q = \gcd(26, 13) = 13.
\end{aligned}$$

Shamir's countermeasure

- A. Shamir, Method and apparatus for protecting public key schemes from timing and fault attacks, United States Patent No. 5,991,415, November 23, 1999. Also presented at the rump session of EUROCRYPT'97.
- Using an *extended modulus*

Shamir's countermeasure

- Let r be a random ℓ_r-bit prime number.
- Typically $\ell_r = 32$
- Instead of computing s_p and s_q as

$$s_p := m^{d \bmod (p-1)} \bmod p, \quad s_q := m^{d \bmod (q-1)} \bmod q,$$

- We compute

$$s_p^* = m^{d \bmod (p-1)(r-1)} \bmod pr, \quad s_q^* = m^{d \bmod (q-1)(r-1)} \bmod qr.$$

- Then we check if

$$s_p^* \equiv s_q^* \bmod r.$$

- If yes, the signature s is given by

$$s = s_p^* y_q q + s_q^* y_p p \bmod n.$$

Shamir's countermeasure

- Suppose the Bellcore attack is to be carried out and a malicious fault is injected during the computation of s_p^* or s_q^*, but not both.
- Without loss of generality, let us assume s_p^* is faulty and s_q^* is computed correctly.
- Let $s_p^{*'}$ denote the faulty s_p^*.
- The fault will be detected if

  $$s_p^{*'} \not\equiv s_q^* \bmod r,$$

  which means the probability of injecting an undetectable fault is the probability of producing $s_p^{*'}$ such that

  $$s_p^{*'} \equiv s_q^* \bmod r.$$

- The probability is $1/r$.
- Thus, with Shamir's countermeasure, the Bellcore attack will be successful with probability $1/r$.
- When the bit length of r is around 32 bits, this probability is about 2^{-32}.

Shamir's countermeasure – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3$$

Suppose $r = 3$.

$$s_p^* = m^{d \bmod (p-1)(r-1)} \bmod pr = ?$$
$$s_q^* = m^{d \bmod (q-1)(r-1)} \bmod qr = ?$$

Shamir's countermeasure – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3, \quad r = 3$$

$$s_p^* = m^{d \bmod (p-1)(r-1)} \bmod pr = 6^{5 \bmod (4\times 2)} = 6^5 \bmod 15 = 6,$$
$$s_q^* = m^{d \bmod (q-1)(r-1)} \bmod qr = 6^{5 \bmod (6\times 2)} = 6^5 \bmod 21 = 6.$$

We can check that

$$s_p^* \equiv s_q^* \equiv 0 \bmod 3.$$

The signature is given by

$$s = s_p^* y_q q + s_q^* y_p p \bmod n = 6 \times 3 \times 7 + 6 \times 3 \times 5 \bmod 35 = 6.$$

Suppose an error occurred and the faulty value $s_p^{*'} = 4$. Then $s_p^{*'} \bmod r = ?$

Shamir's countermeasure – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3, \quad r = 3$$

$$s_p^* = 6, \quad s_q^* = 6, \quad s_p^* \equiv s_q^* \equiv 0 \bmod 3, \quad s = 6$$

Suppose an error occurred during the computation of s_p^*, and the faulty value $s_p^{*'} = 4$. Then we would have

$$s_p^{*'} \not\equiv s_q^* \bmod r.$$

The fault will be detected. What if $s_p^{*'} = 9$?

Shamir's countermeasure – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3, \quad r = 3$$

$$s_p^* = 6, \quad s_q^* = 6, \quad s_p^* \equiv s_q^* \equiv 0 \bmod 3, \quad s = 6$$

Suppose an error occurred during the computation of s_p^*, and the faulty value $s_p^{*'} = 9$, we have

$$s_p^{*'} \equiv s_q^* \equiv 0 \bmod 3,$$

and the faulty signature will be

$$s' = s_p^{*'} y_q q + s_q^* y_p p \bmod n = ?$$

In this case, the attacker can repeat the Bellcore attack by computing

$$q = \gcd(s' - s, n) = ?$$

Shamir's countermeasure – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad e = 5, \quad d = 5, \quad m = 6, \quad y_q = 3, \quad y_p = 3, \quad r = 3$$

$$s_p^* = 6, \quad s_q^* = 6, \quad s_p^* \equiv s_q^* \equiv 0 \bmod 3, \quad s = 6$$

Suppose an error occurred during the computation of s_p^*, and the faulty value $s_p^{*'} = 9$, we have

$$s_p^{*'} \equiv s_q^* \equiv 0 \bmod 3,$$

and the faulty signature will be

$$s' = s_p^{*'} y_q q + s_q^* y_p p \bmod n = 9 \times 3 \times 7 + 6 \times 3 \times 5 \bmod 35 = 34.$$

In this case, the attacker can repeat the Bellcore attack by computing

$$q = \gcd(s' - s, n) = \gcd(34 - 6, 35) = 7.$$

Infective countermeasure

- Although Shamir's countermeasure can effectively protect RSA signature computations against the Bellcore attack, a simple improved attack is to bypass the check
$$s_p^* \equiv s_q^* \bmod r.$$
using an instruction skip.
- We will discuss a more sophisticated countermeasure against the Bellcore attack, the *infective countermeasure*
- The main goal of the countermeasure is to make s_p faulty if s_q is faulty, hence the name "infective".
- Sung-Ming, Y., Kim, S., Lim, S., & Moon, S. (2002). RSA speedup with residue number system immune against hardware fault cryptanalysis. In Information Security and Cryptology—ICISC 2001: 4th International Conference Seoul, Korea, December 6–7, 2001 Proceedings 4 (pp. 397-413). Springer Berlin Heidelberg.

Infective countermeasure

- Same as before, let p and q be distinct odd primes.
- $n = pq$.
- d is the private key for RSA signatures.
- $e = d^{-1} \mod \varphi(n)$.
- m is the hash value for the message.

$$y_q = q^{-1} \mod p, \quad y_p = p^{-1} \mod q.$$

- We select a random integer r such that $\gcd(d_r, \varphi(n)) = 1$ and e_r is a small integer, where
$$d_r = d - r, \quad e_r = d_r^{-1} \mod \varphi(n).$$
- Let
$$k_p = \left\lfloor \frac{m}{p} \right\rfloor, \quad k_q = \left\lfloor \frac{m}{q} \right\rfloor.$$

Infective countermeasure

We select a random integer r such that $\gcd(d_r, \varphi(n)) = 1$ and e_r is a small integer, where

$$d_r = d - r, \quad e_r = d_r^{-1} \bmod \varphi(n).$$

Let

$$k_p = \left\lfloor \frac{m}{p} \right\rfloor, \quad k_q = \left\lfloor \frac{m}{q} \right\rfloor.$$

The signature s is then computed as follows:

$$\begin{aligned}
s_p &= m^{d_r} \bmod p, \\
\hat{m} &= ((s_p^{e_r} \bmod p) + k_p p) \bmod q, \\
s_q &= \hat{m}^{d_r} \bmod q, \\
s_{dr} &= s_p y_q q + s_q y_p p \bmod n, \\
\widetilde{m} &= (s_q^{e_r} \bmod q) + k_q q, \\
s &= s_{dr} \widetilde{m}^r \bmod n.
\end{aligned}$$

Infective countermeasure

- Bellcore attack assumes one of s_p and s_q is faulty, but not both.
- For the infective countermeasure, it can be shown
  - When $p < q$, if s_p is faulty, s_q will also be faulty.
  - When $p > q$, if s_p is faulty, then s_q has a high probability to be faulty.
  - If s_q is faulty and s_p is not faulty, the attacker cannot repeat the attack without brute force.

Infective countermeasure – Example

Example

$$p = 11,\ q = 13,\ n = 143,\ m = 2,\ \varphi(n) = 120,\ d = 11,\ y_p = 6,\ y_q = 6.$$

Choose $r = 4$, then

$$d_r = d - r = 11 - 4 = 7.$$

By the extended Euclidean algorithm,

$$e_r = d_r^{-1} \bmod \varphi(n) =?$$

We also have

$$k_p = \left\lfloor \frac{m}{p} \right\rfloor =? \quad k_q = \left\lfloor \frac{m}{q} \right\rfloor =?$$

Infective countermeasure – Example

Example

$$p = 11,\ q = 13,\ n = 143,\ m = 2,\ \varphi(n) = 120,\ d = 11,\ y_p = 6,\ y_q = 6.$$

$$r = 4, \quad d_r = d - r = 11 - 4 = 7.$$

By the extended Euclidean algorithm,

$$120 = 7 \times 17 + 1 \Longrightarrow 1 = 120 - 7 \times 17,$$

hence

$$e_r = d_r^{-1} \bmod \varphi(n) = -17 \bmod 120 = 103.$$

We also have

$$k_p = \left\lfloor \frac{m}{p} \right\rfloor = \left\lfloor \frac{2}{11} \right\rfloor = 0, \quad k_q = \left\lfloor \frac{m}{q} \right\rfloor = \left\lfloor \frac{2}{13} \right\rfloor = 0.$$

Infective countermeasure – Example

Example

$$p = 11,\ q = 13,\ n = 143,\ m = 2,\ \varphi(n) = 120,\ d = 11,\ y_p = 6,\ y_q = 6.$$

$$r = 4, \quad d_r = 7, \quad e_r = 103, \quad k_p = 0, \quad k_q = 0$$

$$\begin{aligned}
s_p &= m^{d_r} \bmod p = ? \\
\hat{m} &= ((s_p^{e_r} \bmod p) + k_p p) \bmod q = ? \\
s_q &= \hat{m}^{d_r} \bmod q = ? \\
s_{dr} &= s_p y_q q + s_q y_p p \bmod n = ? \\
\widetilde{m} &= (s_q^{e_r} \bmod q) + k_q q = ? \\
s &= s_{dr} \widetilde{m}^r \bmod n = ?
\end{aligned}$$

Infective countermeasure – Example

Example

$$p = 11,\ q = 13,\ n = 143,\ m = 2,\ \varphi(n) = 120,\ d = 11,\ y_p = 6,\ y_q = 6.$$

$$r = 4, \quad d_r = 7, \quad e_r = 103, \quad k_p = 0, \quad k_q = 0$$

$$\begin{aligned}
s_p &= m^{d_r} \bmod p = 2^7 \bmod 11 = 128 \bmod 11 = 7, \\
\hat{m} &= ((s_p^{e_r} \bmod p) + k_p p) \bmod q = (7^{103} \bmod 11 + 0) \bmod 13 \\
&= (7^{103 \bmod 10}) \bmod 13 = (7^3 \bmod 11) \bmod 13 = 2, \\
s_q &= \hat{m}^{d_r} \bmod q = 2^7 \bmod 13 = 128 \bmod 13 = 11, \\
s_{dr} &= s_p y_q q + s_q y_p p \bmod n = 7 \times 6 \times 13 + 11 \times 6 \times 11 \bmod 143 = 128, \\
\widetilde{m} &= (s_q^{e_r} \bmod q) + k_q q = 11^{103} \bmod 13 + 0 = 11^7 \bmod 13 = 2, \\
s &= s_{dr} \widetilde{m}^r \bmod n = 128 \times 2^4 \bmod 143 = 2048 \bmod 143 = 46.
\end{aligned}$$

Suppose s_p is faulty and $s_p' = 2$. Then $\hat{m}' =?$ $s_q' =?$

Infective countermeasure – Example

Example

$$p = 11,\ q = 13,\ m = 2,\ y_p = 6,\ y_q = 6,\ r = 4,\ d_r = 7,\ e_r = 103,\ k_p = 0,\ k_q = 0$$

$$\begin{aligned}
s_p &= m^{d_r} \bmod p = 2^7 \bmod 11 = 128 \bmod 11 = 7,\\
\hat{m} &= ((s_p^{e_r} \bmod p) + k_p p) \bmod q = 2,\\
s_q &= \hat{m}^{d_r} \bmod q = 2^7 \bmod 13 = 128 \bmod 13 = 11.
\end{aligned}$$

Suppose s_p is faulty and $s'_p = 2$. Then

$$\begin{aligned}
\hat{m}' &= ((s_p'^{e_r} \bmod p) + k_p p) \bmod q = (2^{103} \bmod 11 + 0) \bmod 13 = 2^3 \bmod 11 = 8,\\
s'_q &= \hat{m}'^{d_r} \bmod q = 8^7 \bmod 13 = 5.
\end{aligned}$$

Thus s'_q is also faulty.

FA on RSA and countermeasures

- Introduction to Fault Attacks
- Recall – RSA Signatures
- Bellcore Attack and Countermeasures
- Safe-error Attack and Countermeasure

Introduction

- Right-to-left square and multiply algorithm
- Modular multiplication: Blakely's method
- The attack exploits the knowledge of whether an intermediate faulty value is used or not by observing whether the final output is changed, thus the name *safe error attack* [1].
- Since only knowing whether the output is changed or not is enough, if a countermeasure that repeats the computation, compares the final results, and outputs an error when a fault is detected is implemented, the safe error attack still applies.

[1] Yen, S. M., & Joye, M. (2000). Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on computers

Notations

- n has bit length ℓ_n,
$$2^{\ell_n - 1} \leq n < 2^{\ell_n}.$$
- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- ω: the computer's word size
  - for a 64-bit processor, the *word size* is 64
- Let $\kappa = \lceil \ell_n / \omega \rceil$, i.e. $(\kappa - 1)\omega < \ell_n \leq \kappa\omega$.
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)
$$a = a_{\kappa-1} || a_{\kappa-2} || \ldots || a_0,$$
- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have
$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Blakley's method

- We would like to compute
$$R = ab \bmod n.$$
- Since
$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i,$$
where $0 \le a_i < 2^\omega$.
- The product ab can be computed as follows
$$t = ab = \left(\sum_{i=0}^{\kappa-1} a_i (2^\omega)^i \right) b = \sum_{i=0}^{\kappa-1} (2^\omega)^i a_i b,$$

Algorithm 1: Blakely's method for computing modular multiplication.

Input: n, a, b// $n \in \mathbb{Z}, n \geq 2$ has bit length ℓ_n; $a, b \in \mathbb{Z}_n$

Output: $ab \bmod n$

$R = 0$

// $\kappa = \lceil \ell_n / \omega \rceil$, where ω is the word size of the computer

for $i = \kappa - 1$, $i \geq 0$, $i--$ **do**

  $R = 2^{\omega} R + a_i b$

  $R = R \bmod n$

return R

Blakely's method – Example

Input: $n,\ a,\ b$
Output: $ab \bmod n$

$R = 0$
for $i = \kappa - 1,\ i \geq 0,\ i - -$ **do**
  $R = 2^{\omega} R + a_i b$
  $R = R \bmod n$
return R

Example

$\omega = 2$, $a = 13 = 1101_2$, $b = 5$, $n = 15$ ($\ell_n = 4$), $\kappa = 2$.

$$a_0 = 01_2 = 1, \quad a_1 = 11_2 = 3.$$

For $i = 1$,

$$R = 0 + 3 \times 5 \bmod 15 = 0 \bmod 15.$$

For $i = 0$,

$$R = 0 + 1 \times 5 \bmod 15 = 5 \bmod 15$$

We have the final result $13 \times 5 \bmod 15 = 5$.

Attack on a simple algorithm

Algorithm 2: An algorithm involving computing modular multiplication with Blakely's method.

Input: n, a, b, c// $a, b \in \mathbb{Z}_n$; $c = 0, 1$

Output: $ab \bmod n$ if $c = 1$ and a otherwise

if $c = 1$ **then**

  $R = 0$

  // $\kappa = \lceil \ell_n / \omega \rceil$, where ω is the computer's word size

  for $i = \kappa - 1$, $i >= 0$, $i - -$ **do**

    $R = 2^\omega R + a_i b$

    $R = R \bmod n$

  $a = R$

return a

Attack on a simple algorithm

Input: $n,\ a,\ b,\ c$
Output: $ab \bmod n$ if $c = 1$
and a otherwise

```
1 if c = 1 then
2     R = 0
3     for i = κ − 1, i >= 0, i −− do
4         R = 2^ω R + a_i b
5         R = R mod n
6     a = R
7 return a
```

- Attacker has the knowledge of the correct output for a pair of a and b.
- Can rerun the algorithm with the same input, inject fault, and observe the output.
- Suppose $c = 1$ and a fault is injected during the loop starting from line 3 in the register containing a_{i_0} when $i < i_0$. Will the output be faulty?
- What if $c = 0$?

Attack on a simple algorithm

Input: $n, \ a, \ b, \ c$
Output: $ab \bmod n$ if $c = 1$
and a otherwise

1 **if** $c = 1$ **then**
2 $\quad R = 0$
3 $\quad$ **for** $i = \kappa - 1, \ i >= 0, \ i - -$ **do**
4 $\quad\quad R = 2^{\omega} R + a_i b$
5 $\quad\quad R = R \bmod n$
6 $\quad a = R$
7 **return** a

- Attacker has the knowledge of the correct output for a pair of a and b.
- Can rerun the algorithm with the same input, inject fault, and observe the output.
- Suppose $c = 1$ and a fault is injected during the loop starting from line 3 in the register containing a_{i_0} when $i < i_0$ – the fault in a_{i_0} will not affect the output since a_{i_0} is used when i is equal to i_0.
- If $c = 0$ and a fault is injected in the register containing a_{i_0} during the computation, then the final result will be faulty since the faulty value in a will be returned.

Attack on a simple algorithm

Input: $n, \; a, \; b, \; c$

Output: $ab \bmod n$ if $c = 1$ and a otherwise

1 **if** $c = 1$ **then**
2 $\quad R = 0$
3 $\quad$ **for** $i = \kappa - 1, \; i >= 0, \; i - -$ **do**
4 $\quad\quad R = 2^{\omega} R + a_i b$
5 $\quad\quad R = R \bmod n$
6 $\quad a = R$
7 **return** a

- Attacker knows: correct output, a, b.
- If $c = 1$ and a fault is injected during the loop starting from line 3 in the register containing a_{i_0} when $i < i_0$ – output will not be affected
- If $c = 0$, output will be faulty
- Now, if the attacker does not know the value of c and would like to recover it by fault injection attacks.
- Attacker assumes $c = 1$ and the loop in line 3 is executed.
- Injects fault in a_{i_0} at the time when i is less than i_0.
- By comparing the output with the correct one, how does the attacker recover c?

Attack on a simple algorithm

Input: $n, \ a, \ b, \ c$
Output: $ab \bmod n$ if $c = 1$ and a otherwise

```
1 if c = 1 then
2     R = 0
3     for i = κ − 1, i >= 0, i − − do
4         R = 2^ω R + a_i b
5         R = R mod n
6     a = R
7 return a
```

- Attacker knows: correct output, a, b.
- If $c = 1$ and a fault is injected during the loop starting from line 3 in the register containing a_{i_0} when $i < i_0$ – output will not be affected
- If $c = 0$, output will be faulty
- Now, if the attacker does not know the value of c and would like to recover it by fault injection attacks.
- Assume that $c = 1$ and the loop in line 3 is executed.
- Injects fault in a_{i_0} at the time when i is less than i_0.
- Compares the output with the correct one and recovers the value of c – if the output is correct, $c = 1$; otherwise $c = 0$.

Square and multiply algorithm

- $m \in \mathbb{Z}_n$
- Binary representation of $d = d_{\ell_d - 1} \dots d_2 d_1 d_0$, where $d_i = 0, 1$ and

$$d = \sum_{i=0}^{\ell_d - 1} d_i 2^i,$$

- Then we have

$$m^d = m^{\sum_{i=0}^{\ell_d - 1} d_i 2^i} = \prod_{i=0}^{\ell_d - 1} (m^{2^i})^{d_i} = \prod_{0 \le i < \ell_d, d_i = 1} m^{2^i}.$$

- To compute $m^d \bmod n$, we can
  - First compute m^{2^i} for $0 \le i < \ell_d$
  - Then m^d is a product of m^{2^i} for which $d_i = 1$

Right-to-left Square and Mulitply Algorithm

Algorithm 3: Computing RSA signature with the right-to-left square and multiply algorithm.

Input: n, m, d

Output: $s = m^d \bmod n$

$s = 1$, $t = m$

for $i = 0$, $i < \ell_d$, $i++$ **do**

    // ith bit of d is 1

    if $d_i = 1$ **then**

        // multiply by m^{2^i}

        $s = s * t \bmod n$

    // $t = m^{2^{i+1}}$

    $t = t * t \bmod n$

return s

Right-to-left square and multiply algorithm with Blakely's method

- Since ℓ_n is the bit length of n, the bit lengths of the variables s and t are at most ℓ_n.
- ω is the computer's word size

$$\kappa = \lceil \ell_n / \omega \rceil$$

- We can write

$$s = \sum_{j=0}^{\kappa-1} s_j (2^\omega)^j, \quad t = \sum_{j=0}^{\kappa-1} t_j (2^\omega)^j.$$

Right-to-left square and multiply algorithm with Blakely's method

Input: n, m, d. Output: $m^d \bmod n$

1 $s = 1, \quad t = m$
2 **for** $i = 0,\ i < \ell_d,\ i{+}{+}$ **do**
3   **if** $d_i = 1$ **then**
        `// lines 4 -- 8 implement` $s = s * t \bmod n$
4         $R = 0$
5   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
6             $R = 2^\omega R + s_j t$
7             $R = R \bmod n$
8         $s = R$
9     $R = 0$`// lines 9 -- 13 implement` $t = t * t \bmod n$
10   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
11         $R = 2^\omega R + t_j t$
12         $R = R \bmod n$
13     $t = R$
14 **return** s

Right-to-left square and multiply algorithm with Blakely's method

1 $s = 1, \quad t = m$
2 **for** $i = 0,\ i < \ell_d,\ i{+}{+}$ **do**
3   **if** $d_i = 1$ **then**
4         $R = 0$
5   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
6             $R = 2^{\omega} R + s_j t$
7             $R = R \bmod n$
8         $s = R$
9     $R = 0$
10   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
11         $R = 2^{\omega} R + t_j t$
12         $R = R \bmod n$
13     $t = R$
14 **return** s

Example

$$n = 15, \quad d = 3 = 11_2, \quad m = 2, \quad \omega = 2$$

$$\ell_n = 4 \quad \ell_d = 2, \quad \kappa = 2.$$

Line 1 gives:

$$s = 1,\ s_0 = 01,\ s_1 = 00. \quad t = 2,\ t_0 = 10,\ t_1 = 00.$$

For $i = 0$, $d_0 = 1$, loop from line 5 computes

$$j = 1 \quad R = 2^{\omega} R + s_1 t \bmod n = ?$$
$$j = 0 \quad R = 2^{\omega} R + s_0 t \bmod n = ?$$

Line 8: $s = ?, \quad s_0 = ?, \quad s_1 = ?$

Right-to-left square and multiply algorithm with Blakely's method

1 $s = 1, \quad t = m$
2 **for** $i = 0,\ i < \ell_d,\ i{+}{+}$ **do**
3   **if** $d_i = 1$ **then**
4         $R = 0$
5   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
6             $R = 2^\omega R + s_j t$
7             $R = R \bmod n$
8         $s = R$
9     $R = 0$
10   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
11         $R = 2^\omega R + t_j t$
12         $R = R \bmod n$
13     $t = R$
14 **return** s

Example

$$n = 15, \quad d = 3 = 11_2, \quad m = 2, \quad \ell_d = 2, \quad \kappa = 2.$$

$i = 0, d_0 = 1$	line 5	$j = 1$	$R = s_1 t \bmod n = 0$
		$j = 0$	$R = s_0 t \bmod n = 2$
	line 8	$s = 2$	$s_0 = 10, s_1 = 00$
	line 10	$j = 1$	$R = 0$
		$j = 0$	$R = 4$
	line 13	$t = 4$	$t_0 = 00, t_1 = 01$
$i = 1, d_1 = 1$	line 5	$j = 1$	$R = 0$
		$j = 0$	$R = 8$
	line 9	$s = 8$	

$$m^d \bmod n = 2^3 \bmod 15 = 8$$

Safe error attack on RSA Signatures

1 $s = 1, \quad t = m$
2 **for** $i = 0,\ i < \ell_d,\ i{+}{+}$ **do**
3   **if** $d_i = 1$ **then**
4         $R = 0$
5   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
6             $R = 2^{\omega} R + s_j t$
7             $R = R \bmod n$
8         $s = R$
9     $R = 0$
10   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
11         $R = 2^{\omega} R + t_j t$
12         $R = R \bmod n$
13     $t = R$
14 **return** s

- Suppose $d_i = 1$ and a fault is injected during the ith iteration of the outer loop and at the time when $j < j_0$ during the loop starting from line 5, in the register containing s_{j_0}.
  - The fault in s_{j_0} will not affect the output since s_{j_0} is used when j is equal to j_0 and the value in s is replaced by R in line 8.
- Suppose $d_i = 0$ and a fault is injected during the ith iteration of the outer loop in the register containing s_{j_0}, then the value in s will be changed and the final result will be different.
- How does the attacker recover the value of d_i?

Safe error attack on RSA Signatures

```
1  s = 1,    t = m
2  for i = 0, i < ℓ_d, i++ do
3      if d_i = 1 then
4          R = 0
5          for j = κ − 1, j ≥ 0, j−− do
6              R = 2^ω R + s_j t
7              R = R mod n
8          s = R
9      R = 0
10     for j = κ − 1, j ≥ 0, j−− do
11         R = 2^ω R + t_j t
12         R = R mod n
13     t = R
14 return s
```

- Suppose $d_i = 1$ and a fault is injected during the ith iteration of the outer loop and at the time when $j < j_0$ during the loop starting from line 5, in the register containing s_{j_0} – correct output
- If $d_i = 0$ and a fault is injected during the ith iteration of the outer loop in the register containing s_{j_0} – faulty output
- Similarly to the attack on the simple algorithm, the attacker first assumes $d_i = 1$, and injects fault in s_{j_0} at the time corresponding to $j < j_0$.
- If the final result is not changed, the attacker can conclude that $d_i = 1$, otherwise, $d_i = 0$.

Safe error attack on RSA signatures – Example

1 $s = 1, \quad t = m$
2 **for** $i = 0,\ i < \ell_d,\ i{+}{+}$ **do**
3   **if** $d_i = 1$ **then**
4         $R = 0$
5   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
6             $R = 2^{\omega} R + s_j t$
7             $R = R \bmod n$
8         $s = R$
9     $R = 0$
10   **for** $j = \kappa - 1,\ j \geq 0,\ j{-}{-}$ **do**
11         $R = 2^{\omega} R + t_j t$
12         $R = R \bmod n$
13     $t = R$
14 **return** s

Example

$$n = 15, \quad d = 3 = 11_2, \quad m = 2, \quad \ell_d = 2, \quad \kappa = 2.$$

$i = 0, d_0 = 1$	line 5	$j = 1$	$R = s_1 t \bmod n = 0$
		$j = 0$	$R = s_0 t \bmod n = 2$
	line 8	$s = 2$	$s_0 = 10, s_1 = 00$
	line 10	$j = 1$	$R = 0$
		$j = 0$	$R = 4$
	line 13	$t = 4$	$t_0 = 00, t_1 = 01$
$i = 1, d_1 = 1$	line 5	$j = 1$	$R = 0$
		$j = 0$	$R = 8$
	line 9	$s = 8$	

Makes guess $d_0 = 1$, injects fault into s_1 when $i = 0,\ j = 0$ (line 5)

Safe error attack on RSA signatures – Example

Example

$$n = 15, \quad d = 3 = 11_2, \quad m = 2, \quad \ell_d = 2, \quad \kappa = 2.$$

$i = 0, d_0 = 1$	line 5	$j = 1$	$R = s_1 t \bmod n = 0$
		$j = 0$	$R = s_0 t \bmod n = 2$
	line 8	$s = 2$	$s_0 = 10, s_1 = 00$
	line 10	$j = 1$	$R = 0$
		$j = 0$	$R = 4$
	line 13	$t = 4$	$t_0 = 00, t_1 = 01$
$i = 1, d_1 = 1$	line 5	$j = 1$	$R = 0$
		$j = 0$	$R = 8$
	line 9	$s = 8$	

- Makes guess $d_0 = 1$, injects fault into s_1 when $i = 0$, $j = 0$ (line 5)
- s_1 is used (blue s_1) before $j = 0$ and reassigned value in line 8 (orange s_1).
- Thus the computations are not affected and the final result is unchanged.

Safe error attack on RSA signatures

- The attacker can repeat the attack for different values of i to recover the entire private key.
- Similar techniques can also be applied to attack the left-to-right square and multiply algorithm with Blakely's method[1].

[1] Yen, S. M., & Joye, M. (2000). Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers

Countermeasure for the simple algorithm

Input: $n,\ a,\ b,\ c$

Output: $ab \bmod n$ if $c = 1$ and a otherwise

$R = 0$
if $c = 1$ **then**
  for $i = \kappa - 1,\ i >= 0,\ i - -$ **do**
    $R = 2^{\omega} R + a_i b$
    $R = R \bmod n$
  $a = R$
return a

Algorithm 4: Modified algorithm.

Input: $n,\ a,\ b,\ c$// $a, b \in \mathbb{Z}_n\,;\ \ c = 0, 1$

Output: $ab \bmod n$ if $c = 1$ and a otherwise

$R = 0$
if $c = 1$ **then**
  for $i = \kappa - 1,\ i >= 0,\ i - -$ **do**
    $R = 2^{\omega} R + b_i a$
    $R = R \bmod n$
  $a = R$
return a

Will the output be faulty if the fault is injected

- in b_{i_0} when $i < i_0$ and $c = 1$
- in b and $c = 0$
- in a

Countermeasure for the simple algorithm

Input: $n,\ a,\ b,\ c$ // $a, b \in \mathbb{Z}_n$; $c = 0, 1$
Output: $ab \bmod n$ if $c = 1$ and a otherwise

```
R = 0
if c = 1 then
    for i = κ − 1, i >= 0, i − − do
        R = 2^ω R + b_i a
        R = R mod n
    a = R
return a
```

- $c = 1$ and the fault is in b_{i_0} when $i < i_0$ – since b_{i_0} is used before the fault happens, the final result will not be affected.
- $c = 0$, a fault in b_{i_0} at any time will not change the final output.
- If a fault is injected in a, the output will be faulty no matter what value c takes.

Countermeasure for RSA Signatures

Input: n, m, d

$s = 1, \quad t = m$
for $i = 0,\ i < \ell_d,\ i++$ **do**
  if $d_i = 1$ **then**
    $R = 0$
    for $j = \kappa - 1,\ j \geq 0,\ j--$ **do**
      $R = 2^\omega R + s_j t$
      $R = R \bmod n$
    $s = R$
  $R = 0$
  for $j = \kappa - 1,\ j \geq 0,\ j--$ **do**
    $R = 2^\omega R + t_j t$
    $R = R \bmod n$
  $t = R$
return s

Output: $m^d \bmod n$.

$s = 1, \quad t = m$
for $i = 0,\ i < \ell_d,\ i++$ **do**
  if $d_i = 1$ **then**
    $R = 0$
    for $j = \kappa - 1,\ j \geq 0,\ j--$ **do**
      $R = 2^\omega R + t_j s$
      $R = R \bmod n$
    $s = R$
  $R = 0$
  for $j = \kappa - 1,\ j \geq 0,\ j--$ **do**
    $R = 2^\omega R + t_j t$
    $R = R \bmod n$
  $t = R$
return s