Cryptography and Embedded System Security CRAESS_I

Xiaolu Hou

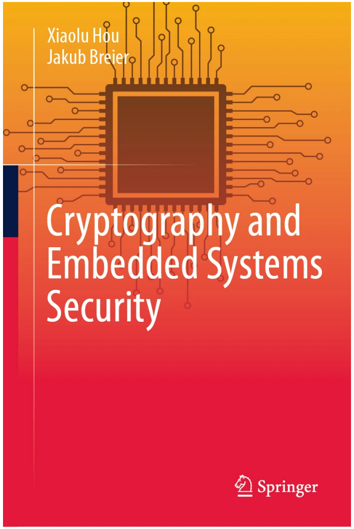
FIIT, STU
xiaolu.hou @ stuba.sk

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 4.6



Lecture Outline

- Introduction
- Square and multiply-always
- Blinding for RSA
- Masking for PRESENT

SCA countermeasures

- Introduction
- Square and multiply-always
- Blinding for RSA
- Masking for PRESENT

Countermeasures

- Protocol level: design cryptographic protocols to survive leakage analysis
    - Limiting the number of communications that can be performed with any given key, fewer measurements can be done by the attacker for the same key
    - Re-keying[1]
- Cryptographic primitive level
    - Proposal of new cipher design

[1]Medwed, M., Standaert, F. X., Großschädl, J., & Regazzoni, F. (2010, May). Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In International Conference on Cryptology in Africa (pp. 279-296). Springer, Berlin, Heidelberg.

Countermeasures

- Implementation level
  - Time randomization[1]
  - Encryption of the buses[2]
  - Hiding
  - Masking and blinding

[1] May, D., Muller, H. L., & Smart, N. P. (2001, May). Random register renaming to foil DPA. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 28-38). Springer, Berlin, Heidelberg.

[2] Brier, E., Handschuh, H., & Tymen, C. (2001, May). Fast primitives for internal data scrambling in tamper resistant hardware. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 16-27). Springer, Berlin, Heidelberg.

Countermeasures

- Architecture level
  - Use non-deterministic processor to randomly change the sequence of the executed program during each execution[1]
  - Integrate secure instructions into a non-secure processor[2]

[1] May, D., Muller, H. L., & Smart, N. P. (2001, July). Non-deterministic processors. In Australasian Conference on Information Security and Privacy (pp. 115-129). Springer, Berlin, Heidelberg.

[2] Saputra, H., Vijaykrishnan, N., Kandemir, M., Irwin, M. J., & Brooks, R. (2003). Masking the energy behaviour of encryption algorithms. IEE Proceedings-Computers and Digital Techniques, 150(5), 274-284.

Countermeasures

- Hardware level
  - Conforming glues[1]
  - Protective coating[2]
  - Detachable power supplies[3]

[1]Anderson, R., & Kuhn, M. (1996, November). Tamper resistance – a cautionary note. In Proceedings of the second Usenix workshop on electronic commerce (Vol. 2, pp. 1-11).

[2]Tuyls, P., Schrijen, G. J., Škorić, B., Geloven, J. V., Verhaegh, N., & Wolters, R. (2006, October). Read-proof hardware from protective coatings. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 369-383). Springer, Berlin, Heidelberg.

[3]Shamir, A. (2000, August). Protecting smart cards from passive power analysis with detached power supplies. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 71-77). Springer, Berlin, Heidelberg.

Hiding and masking/blinding

- We have seen how the dependency of a device's leakages (power consumption) on data and operations can be exploited to recover the secret keys of a cryptographic implementation.
- Goal: make the leakage of the DUT independent of the operations or the intermediate values of the executed cryptographic implementation.
- Hiding – remove the operation/data dependency of leakage
    - Change the leakage of the DUT in a way that every operation requires a similar (balance leakages) or a random (randomize leakages) amount of energy.
- Masking/blinding – remove the data dependency of leakage by randomizing the intermediate values that the DUT is processing
    - The rationale is that since the value being processed in the DUT is randomized and independent of the intermediate value of the cryptographic computation, we cannot capture information on the actual intermediate value from the leakages.
    - Symmetric block cipher: masking
    - Asymmetric cipher: blinding

Hiding – randomizing power consumption

- Insert random delay (jitter)[1]
- Shuffle the execution order of independent operations
  - Sboxes in AES[2]
  - Randomize the sequence of square and multiply[3]
- Using residue number systems allow randomizing the representation of finite field elements for computing exponentiation[4]

[1]Coron, J. S., & Kizhvatov, I. (2009, September). An efficient method for random delay generation in embedded software. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 156-170). Springer, Berlin, Heidelberg.

[2]Herbst, C., Oswald, E., & Mangard, S. (2006, June). An AES smart card implementation resistant to power analysis attacks. In International conference on applied cryptography and network security. Springer.

[3]Walter, C. D. (2002, February). MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In Cryptographers' Track at the RSA Conference. Springer.

[4]Bajard, J. C., Imbert, L., Liardet, P. Y., & Teglia, Y. (2004, August). Leak resistant arithmetic. In International Workshop on Cryptographic Hardware and Embedded Systems. Springer.

Hiding – balancing power consumption

- Cell level, logic designs
  - Dual-rail precharge logic (DPL)[1], in pre-charge phase, values in the weries are set to a precharge value (either 0 or 1), during evaluation phase, one wire carries the signal 0 and the other wire carries the signal 1
    - Using code $\{01, 10\}$ for encoding $0, 1$
  - Dynamic and differential logic styles [2]

[1] Tiri, K., & Verbauwhede, I. (2006). A digital design flow for secure integrated circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(7), 1197-1208.

[2] Tiri, K., Akmal, M., & Verbauwhede, I. (2002, September). A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In Proceedings of the 28th European solid-state circuits conference (pp. 403-406). IEEE.

Hiding – balancing power consumption

- Software level
  - DPL in software for symmetric block ciphers[1]
  - DPL in software with proved security for bitsliced implementation of PRESENT[2]
  - Linear complementary dual code[3]
  - Error detecting and correcting code in software[4]
  - Square and multiply-always for computing exponentiation in RSA[5]

[1] Hoogvorst, P., Duc, G., & Danger, J. L. (2011). Software implementation of dual-rail representation. COSADE, 51, 24-25.

[2] Rauzy, P., Guilley, S., & Najm, Z. (2013). Formally Proved Security of Assembly Code Against Leakage. IACR Cryptol. ePrint Arch., 2013, 554.

[3] Carlet, C., & Guilley, S. (2015). Complementary dual codes for counter-measures to side-channel attacks. In Coding Theory and Applications (pp. 97-105). Springer, Cham.

[4] Breier, J., & Hou, X. (2017, February). Feeding two cats with one bowl: On designing a fault and side-channel resistant software encoding scheme. In Cryptographers' Track at the RSA Conference (pp. 77-94). Springer, Cham.

[5] Coron, J. S. (1999, August). Resistance against differential power analysis for elliptic curve cryptosystems. In International workshop on cryptographic hardware and embedded systems (pp. 292-302). Springer, Berlin, Heidelberg.

Masking and blinding

- Let $\boldsymbol{v}$ be the secret intermediate value that we would like to mask.
- The masked value, denoted $\boldsymbol{v}_{\mathfrak{m}}$, concealed by a random value $\mathfrak{m}$, called a *mask*, with a binary operation $\cdot$ such that

$$\boldsymbol{v}_{\mathfrak{m}} = \boldsymbol{v} \cdot \mathfrak{m}.$$

Masking

- Boolean masking, the binary operation is bitwise `XOR`
- Arithmetic masking, the binary operation is modular addition or modular multiplication
- Affine masking[1]
- Polynomial masking[2]
- Inner product masking[3]

[1] Willich, M. V. (2001, December). A technique with an information-theoretic basis for protecting secret data from differential power attacks. In IMA International Conference on Cryptography and Coding (pp. 44-62). Springer, Berlin, Heidelberg.

[2] Goubin, L., & Martinelli, A. (2011, September). Protecting AES with Shamir's secret sharing scheme. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 79-94). Springer, Berlin, Heidelberg.

[3] Balasch, J., Faust, S., Gierlichs, B., & Verbauwhede, I. (2012, December). Theory and practice of a leakage resilient masking scheme. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 758-775). Springer, Berlin, Heidelberg.

Masking – more methods in hardware implementations

- Masking buses[1]
- Boolean masking with DLP[2]
- Random precharging[3]

[1]Benini, L., Galati, A., Macii, A., Macii, E., & Poncino, M. (2003, August). Energy-efficient data scrambling on memory-processor interfaces. In Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED'03. (pp. 26-29). IEEE.

[2]Popp, T., & Mangard, S. (2005, August). Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 172-186). Springer, Berlin, Heidelberg.

[3]Bucci, M., Guglielmo, M., Luzzi, R., & Trifiletti, A. (2004, September). A power consumption randomization countermeasure for DPA-resistant cryptographic processors. In International Workshop on Power and Timing Modeling, Optimization and Simulation (pp. 481-490). Springer, Berlin, Heidelberg.

SCA countermeasures

- Introduction
- Square and multiply-always
- Blinding for RSA
- Masking for PRESENT

Motivation

- We have seen an SPA attack on RSA implementations that exploit the part of the square and multiply algorithm that multiplication is carried out only when the secret key bit is 1.
- A natural countermeasure is that we always compute multiplication no matter what the value of the secret key bit.
- Such an algorithm is called *square and multiply-always algorithm*

Recall – 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 – Example

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 2 \times 4 = 8$$

- From $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$, choose public key $e = 3$.
- By the extended Euclidean algorithm

  $$8 = 3 \times 2 + 2,\ 3 = 2 \times 1 + 1 \Longrightarrow 1 = 3 - 3 \times 1 = 3 - (8 - 3 \times 2) = -8 + 3 \times 3.$$

  The private key $d = 3^{-1} \bmod 8 = 3$.
- Suppose Alice would like to send plaintext $m = 2$ to Bob. Alice computes

  $$c = ?$$

- After receiving the ciphertext c from Alice, Bob computes:

  $$m = ?$$

RSA – Example

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 2 \times 4 = 8, e = 3, d = 3$$

- Suppose Alice would like to send plaintext $m = 2$ to Bob, using Bob's public key $n = 15, e = 3$. Alice computes

$$c = m^e \bmod n = 2^3 \bmod 15 = 8 \bmod 15.$$

- After receiving the ciphertext c from Alice, Bob computes the plaintext using his private key

$$m = c^d \bmod n = 8^3 \bmod 15 = 512 \bmod 15 = 2 \bmod 15.$$

Square and multiply algorithm

- Let $n \geq 2$ be an integer, $d \in \mathbb{Z}_{\varphi(n)}$, $a \in \mathbb{Z}_n$
- Binary representation of $d = d_{\ell_d - 1} \ldots d_2 d_1 d_0$, where $d_i = 0, 1$ and

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

- Then we have

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

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

Square and multiply algorithms

```
Algorithm 1: Right-to-left
Input: $n$, $a$, $d$// $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$;
    $d \in \mathbb{Z}_{\varphi(n)}$ has bit length $\ell_d$
Output: $a^d \bmod n$
1 result $= 1$, $t = a$
2 for $i = 0$, $i < \ell_d$, $i++$ do
      // $i$th bit of $d$ is 1
3     if $d_i = 1$ then
          // mutiply by $a^{2^i}$
4         result $=$ result $* t \bmod n$// $a^d = \prod_{0 \leq i < \ell_d, d_i = 1} a^{2^i}$
      // $t = a^{2^{i+1}}$
5     $t = t * t \bmod n$
6 return result
```

```
Algorithm 2: Left-to-right
Input: $n$, $a$, $d$// $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$;
    $d \in \mathbb{Z}_{\varphi(n)}$
Output: $a^d \bmod n$
1 $t = 1$
2 for $i = \ell_d - 1$, $i \geq 0$, $i--$ do
3     $t = t * t \bmod n$
      // $i$th bit of $d$ is 1
4     if $d_i = 1$ then
5         $t = a * t \bmod n$
6 return t
```

Right-to-left square and multiply-always

```
Input: $n,\ a,\ d$ // $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$;
    $d \in \mathbb{Z}_{\varphi(n)}$ has bit length $\ell_d$
Output: $a^d \bmod n$
1 result $= 1$, $t = a$
2 for $i = 0,\ i < \ell_d,\ i++$ do
      // $i$th bit of $d$ is 1
3     if $d_i = 1$ then
          // mutiply by $a^{2^i}$
4         result = result $* t \bmod n$ // $a^d = \prod_{0 \leq i < \ell_d, d_i = 1} a^{2^i}$
      // $t = a^{2^{i+1}}$
5     $t = t * t \bmod n$
6 return result
```

```
Algorithm 3: Square and multiply-always
Input: $n,\ a,\ d$
Output: $a^d \bmod n$
1 result $= 1$, $t = a$
2 for $i = 0,\ i < \ell_d,\ i++$ do
      // $i$th bit of $d$ is 1
3     if $d_i = 1$ then
          // mutiply by $a^{2^i}$
4         result = result $* t \bmod n$
5     else
          // compute multiplication and
             discard the result
6         tmp = result $* t \bmod n$
      // $t = a^{2^{i+1}}$
7     $t = t * t \bmod n$
8 return result
```

Left-to-right square and multiply-always

```
Input: $n$, $a$, $d$ // $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$;
    $d \in \mathbb{Z}_{\varphi(n)}$ has bit length $\ell_d$
Output: $a^d \bmod n$
1 $t = 1$
2 for $i = \ell_d - 1$, $i \geq 0$, $i - -$ do
3     $t = t * t \bmod n$
      // $i$th bit of $d$ is 1
4     if $d_i = 1$ then
5         $t = a * t \bmod n$
6 return t
```

Algorithm 4: Square and multiply-always

```
Input: $n$, $a$, $d$
Output: $a^d \bmod n$
1 $t = 1$
2 for $i = \ell_d - 1$, $i \geq 0$, $i - -$ do
3     $t = t * t \bmod n$
      // $i$th bit of $d$ is 1
4     if $d_i = 1$ then
5         $t = a * t \bmod n$
6     else
          // $i$th bit of $d$ is $0$, compute
             multiplication and discard the
             result
7         tmp $= a * t \bmod n$
8 return t
```

SPA on left-to-right square and multiply algorithm

For our experiment, we have set

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747$$

Algorithm 5: Left-to-right square and multiply algorithm for computing modular exponentiation with parameters from above.

```
Input: $a$// $a \in \mathbb{Z}_{1189}$
Output: $a^{747} \bmod 1189$
1 $n = 1189$
2 $dbin = [1,1,0,1,0,1,1,1,0,1]$// binary representation of $d = 747$, $d_0 = 1$, $d_1 = 1$
3 $\ell_d$ = length of $dbin$// bit length of $d$
4 $t = 1$
5 for $i = \ell_d - 1$, $i \geq 0$, $i --$ do
6     $t = t * t \bmod n$
      // $i$th bit of $d$ is 1
7     if $d_i = 1$ then
8         $t = a * t \bmod n$
9 return t
```

SPA on left-to-right square and multiply algorithm

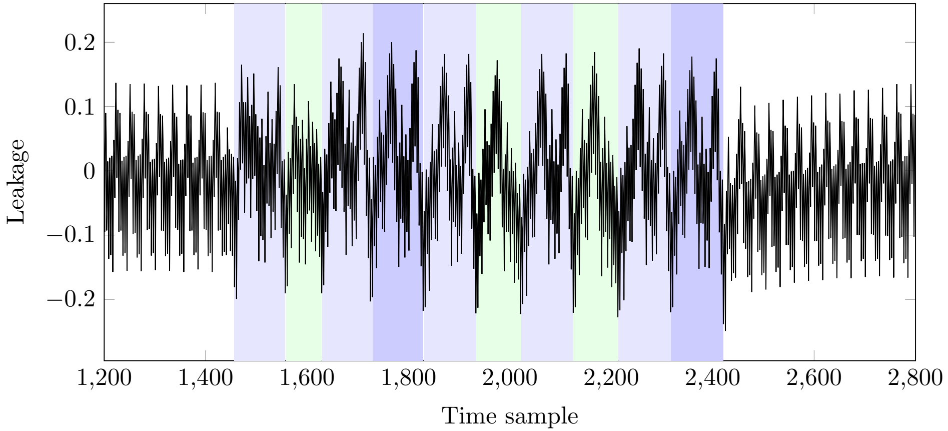
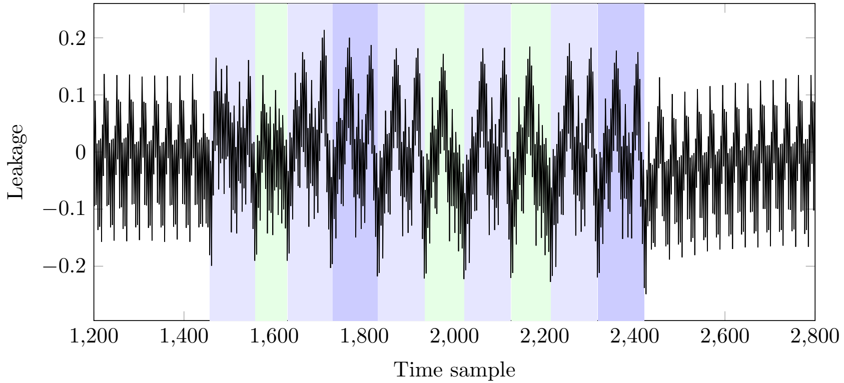


Figure: Green patterns (single peak cluster) $\rightarrow d_i = 0$; blue patterns (multiple peak clusters) $\rightarrow d_i = 1$

We can then read out the value of bits d_i $(i = \ell_d - 1, \ldots, 0, 1)$

SPA on left-to-right square and multiply algorithm



We can then read out the value of bits d_i $(i = \ell_d - 1, \ldots, 0, 1)$:

$$1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1.$$

Finally, we recover the secret key

$$d = 1011101011_2 = 747.$$

Implementation with countermeasure

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747$$

Algorithm 6: Left-to-right square and multiply-always algorithm.

```
Input: $a$// $a \in \mathbb{Z}_{1189}$
Output: $a^{747} \bmod 1189$
$n = 1189$
$dbin = [1,1,0,1,0,1,1,1,0,1]$// binary representation of $d = 747$, $d_0 = 1$, $d_1 = 1$
$\ell_d =$ length of $dbin$// bit length of $d$
$t = 1$
for $i = \ell_d - 1$, $i \geq 0$, $i - -$ do
    $t = t * t \bmod n$
    if $d_i = 1$ then
        $t = a * t \bmod n$
    else
        // $i$th bit of $d$ is $0$, compute multiplication and discard the result
        tmp $= a * t \bmod n$
return t
```

The power trace

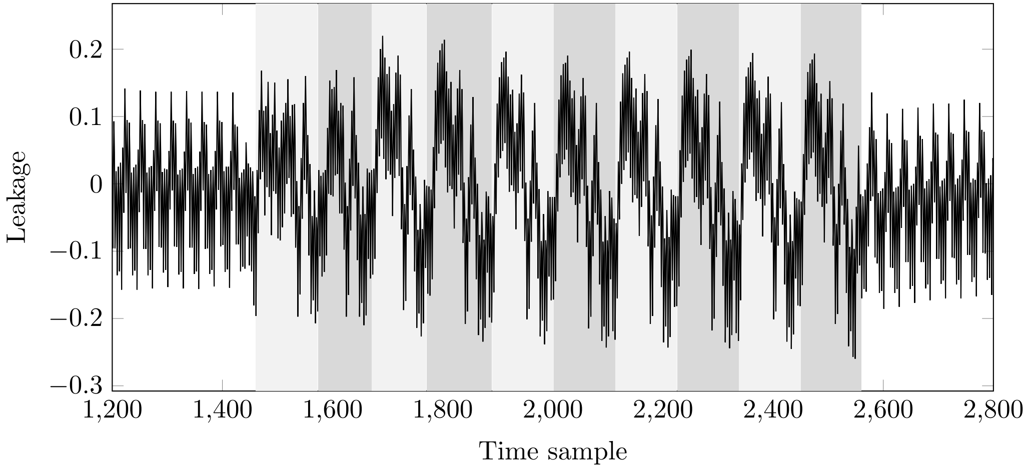
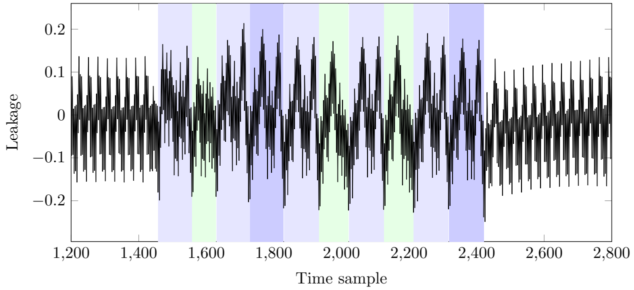
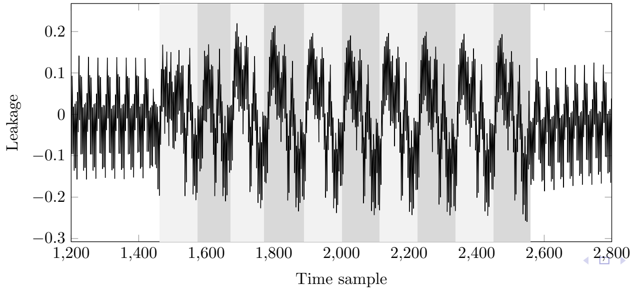


Figure: One trace corresponding to the computation of left-to-right square and multiply-always algorithm. We can see ten similar patterns. But in this case, all of them have more than one peak cluster.

Comparison



Leakage
0.2
0.1
0
−0.1
−0.2
1,200
1,400
1,600
1,800
2,000
2,200
2,400
2,600
2,800
Time sample



Leakage
0.2
0.1
0
−0.1
−0.2
−0.3
1,200
1,400
1,600
1,800
2,000
2,200
2,400
2,600
2,800
Time sample

SCA countermeasures

- Introduction
- Square and multiply-always
- Blinding for RSA
- Masking for PRESENT

Some history

- The application of arithmetic masks in the context of public cryptosystems is called blinding
- First suggested by Kocher[1]
- Formalized by J.-S. Coron[2]
- There is also a patent related to blinding methods[3]

[1]Kocher, P. C. (1996, August). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Annual International Cryptology Conference (pp. 104-113). Springer, Berlin, Heidelberg.

[2]Coron, J. S. (1999, August). Resistance against differential power analysis for elliptic curve cryptosystems. In International workshop on cryptographic hardware and embedded systems (pp. 292-302). Springer, Berlin, Heidelberg.

[3]Kocher, P. C., & Jaffe, J. M. (2001). U.S. Patent No. 6,298,442. Washington, DC: U.S. Patent and Trademark Office.

Notations

- Let p, q be two distinct odd primes, $n = pq$ be the RSA modulus
- $d \in \mathbb{Z}^*_{\varphi(n)}$: the private key for RSA
- $e = d^{-1} \mod \varphi(n)$: public key for RSA
- The SPA attack we have discussed exploits leakages during the computation of

  $$a^d \mod n$$

  for some $a \in \mathbb{Z}_n$.
- The attack can be during the RSA signature signing process or RSA decryption.
- DPA attacks can also be applied to RSA implementations, e.g. DPA on left-to-right square and multiply algorithm with Montgomery's method for modular multiplication[1]

[1] Amiel, F., Feix, B., & Villegas, K. (2007). Power analysis for secret recovering and reverse engineering of public key algorithms. In Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers 14 (pp. 110-125). Springer Berlin Heidelberg.

Blinding

- Exponent blinding, message blinding, and modulus blinding
- Mainly against DPA attacks
- It is recommended to blind the secret values during the computation.
- It is also required that the masks and blinded values should be updated frequently, or even during the computations.
- In this case, it will be difficult for the attacker to combine whatever partial information obtained from the leakages of the previously blinded value and the newly leaked information.
- Berzati, A., Canovas-Dumas, C., & Goubin, L. (2010, August). Public key perturbation of randomized RSA implementations. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 306-319). Springer, Berlin, Heidelberg.
- Kocher, P., Jaffe, J., Jun, B., & Rohatgi, P. (2011). Introduction to differential power analysis. Journal of Cryptographic Engineering, 1(1), 5-27.

Exponent blinding

- Generate a random number $\lambda \in [0, 2^\ell - 1]$
- Typically, for RSA modulus of bit length 1024, we take $\ell = 20$ or 30 to guarantee a reasonable overhead
- Instead of computing
$$a^d \bmod n,$$
- We compute
$$a^{d+\lambda\varphi(n)} \bmod n.$$

Exponent blinding

Theorem (Fermat's Little Theorem)

Let p be a prime. For any $a \in \mathbb{Z}$, if $p \nmid a$, then $a^{p-1} \equiv 1 \mod p$.

Example

- Let $p = 3$. $2^2 = 4 \equiv 1 \bmod 3$.
- Let $p = 5$. $2^4 = 16 \equiv 1 \bmod 5$.

Exponent blinding

Corollary

Let p be a prime. Then for any $a, b, c \in \mathbb{Z}$ such that $b \equiv c \; mod \; (p-1)$, we have

$$a^b \equiv a^c \; mod \; p, \quad \textit{in particular, } a^b \equiv a^{b \; mod \; (p-1)} \; mod \; p.$$

Proof.

By Fermat's Little Theorem,

$$a^{p-1} \equiv \begin{cases} 1 \bmod p & \text{if } p \nmid a \\ 0 \bmod p & \text{otherwise} \end{cases}.$$

Since $b \equiv c \bmod (p-1)$, $b - c = (p-1)k$ for some $k \in \mathbb{Z}$. And

$$a^b \equiv a^{c+(p-1)k} \equiv a^c a^{(p-1)k} \equiv \begin{cases} a^c \bmod p & \text{if } p \nmid a \\ 0 \bmod p & \text{otherwise} \end{cases} \equiv a^c \bmod p.$$

Exponent blinding

Corollary

Let p and q be two distinct primes and $n = pq$. For any $a, b \in \mathbb{Z}$, we have

$$a^b \equiv a^{b \ mod \ \varphi(n)} \ mod \ n.$$

Proof.

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

$$b \bmod \varphi(n) \equiv b \bmod (p-1), \quad b \bmod \varphi(n) \equiv b \bmod (q-1).$$

By the previous corollary,

$$a^b \equiv a^{b \bmod \varphi(n)} \bmod p, \quad a^b \equiv a^{b \bmod \varphi(n)} \bmod q.$$

By Chinese Remainder Theorem,

$$a^b \equiv a^{b \bmod \varphi(n)} \bmod n.$$

Exponent blinding

- Instead of computing
$$a^d \bmod n,$$
- We compute
$$a^{d+\lambda\varphi(n)} \bmod n.$$

Corollary

Let p and q be two distinct primes and $n = pq$. For any $a, b \in \mathbb{Z}$, we have
$$a^b \equiv a^{b \; mod \; \varphi(n)} \; mod \; n.$$

$$d + \lambda\varphi(n) \bmod \; \equiv d \bmod \varphi(n)$$

Example for exponent blinding

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3$$

Take $a = 8$ and $\lambda = 2$.

$$a^d \bmod n = 8^3 \bmod 15 = 2.$$

With the countermeasure, we have

$$a^{d+\lambda\varphi(n)} \bmod n = ?$$

Example for exponent blinding

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3$$

Take $a = 8$ and $\lambda = 2$.

$$a^d \bmod n = 8^3 \bmod 15 = 2.$$

With the countermeasure, we have

$$\begin{aligned} a^{d+\lambda\varphi(n)} \bmod n &= 8^{3+2\times 8} \bmod 15 = 8^{19} \bmod 15 = 8 \times (8^2)^9 \bmod 15 \\ &= 8 \times 4^9 \bmod 15 = 8 \times 4 \times (4^2)^4 \bmod 15 = 32 \bmod 15 = 2. \end{aligned}$$

Effectiveness against our SPA attack

- With one decryption computation, even if we recover the bits of the exponent, it will be $d+$ a random number
- But if we attack two decryption computations, we get

$$d + \lambda_1 \varphi(n), \quad d + \lambda_2 \varphi(n)$$

  then we will have the value of $\varphi(n)$
- Effective against certain DPA attacks

Message blinding

Take a random number λ such that $\gcd(\lambda, n) = 1$, compute

$$a_1 = \lambda^e \bmod n, \quad a_2 = \lambda^{-1} \bmod n.$$

And to get

$$a^d \bmod n,$$

we calculate

$$(((aa_1)^d \bmod n)a_2) \bmod n.$$

Recall corollary

We just proved the following corollary

Corollary

Let p and q be two distinct primes and $n = pq$. For any $a, b \in \mathbb{Z}$, we have

$$a^b \equiv a^{b \textit{ mod } \varphi(n)} \textit{ mod } n.$$

Example

$$p = 5, \quad q = 3, \quad n = 15, \quad a = 2, \quad b = 6, \varphi(n) = 8$$

Then

$$2^{10} \equiv 2^{10 \bmod 8} \equiv 2^2 \equiv 4 \bmod 15.$$

We can verify that indeed

$$2^{10} \equiv 1024 \equiv 4 \bmod 5.$$

Message blinding

- Since
$$ed \equiv 1 \bmod \varphi(n),$$
by the corollary,
$$\lambda^{ed} \equiv \lambda \bmod n \Longrightarrow \lambda^{ed-1} \bmod n = 1.$$
- Then
$$\begin{aligned}(((aa_1)^d \bmod n)a_2) \bmod n &= (((a\lambda^e \bmod n)^d \bmod n)(\lambda^{-1} \bmod n)) \bmod n \\ &= ((a^d \bmod n)(\lambda^{ed-1} \bmod n)) \bmod n = a^d \bmod n.\end{aligned}$$
- The first mask a_1 randomizes the input of the computation
- The second mask a_2 corrects the output to the expected result.

Example for message blinding

$$a_1 = \lambda^e \bmod n, \quad a_2 = \lambda^{-1} \bmod n, \quad a^d \bmod n = (((aa_1)^d \bmod n)a_2) \bmod n.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, which is coprime with n. Then

$$a_1 = ?,$$

Example for message blinding

$$a_1 = \lambda^e \bmod n, \quad a_2 = \lambda^{-1} \bmod n, \quad a^d \bmod n = (((aa_1)^d \bmod n)a_2) \bmod n.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, which is coprime with n. Then

$$a_1 = \lambda^e \bmod n = 4^3 \bmod 15 = 64 \bmod 15 = 4.$$

By the extended Euclidean algorithm

$$a_2 =?$$

Example for message blinding

$$a_1 = \lambda^e \bmod n, \quad a_2 = \lambda^{-1} \bmod n, \quad a^d \bmod n = (((aa_1)^d \bmod n)a_2) \bmod n.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, which is coprime with n. Then $a_1 = 4$. By the extended Euclidean algorithm

$$15 = 4 \times 3 + 3, \quad 4 = 3 + 1 \Longrightarrow 1 = 4 - 3 = 4 - (15 - 4 \times 3) = 4 \times 4 - 15$$

and

$$a_2 = \lambda^{-1} \bmod n = 4.$$

Finally,

$$(((aa_1)^d \bmod n)a_2) \bmod n =?$$

Example for message blinding

$$a_1 = \lambda^e \bmod n, \quad a_2 = \lambda^{-1} \bmod n, \quad a^d \bmod n = (((aa_1)^d \bmod n)a_2) \bmod n.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, which is coprime with n. Then

$$a_1 = 4, \quad a_2 = 4.$$

$$\begin{aligned} (((aa_1)^d \bmod n)a_2) \bmod n &= (((8 \times 4)^3 \bmod 15) \times 4) \bmod 15 \\ &= ((2^3 \bmod 15) \times 4) \bmod 15 = 32 \bmod 15 = 2. \end{aligned}$$

We can compute

$$a^d \bmod n = 8^3 \bmod 15 = 512 \bmod 15 = 2.$$

Modulus blinding

Generate a random number λ and compute

$$(a^d \bmod (\lambda n)) \bmod n.$$

It is easy to see that

$$(a^d \bmod (\lambda n)) \bmod n = a^d \bmod n.$$

Example for modulus blinding

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, then

$$(a^d \bmod (\lambda n)) \bmod n = ?$$

Example for modulus blinding

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad e = 3, \quad d = 3, \quad a = 8, \quad \varphi(n) = 8$$

Take $\lambda = 4$, then

$$\begin{aligned} (a^d \bmod (\lambda n)) \bmod n &= (8^3 \bmod (4 \times 15)) \bmod 15 \\ &= (512 \bmod 60) \bmod 15 = 32 \bmod 15 = 2. \end{aligned}$$

We can check that

$$a^d \bmod n = 2.$$

Effectiveness on DPA attacks

- DPA attacks that rely on knowing certain intermediate values related to a cannot be carried out when a or n is randomized

Attacks on Countermeasures

- Fouque, P. A., & Valette, F. (2003, September). The doubling attack–why upwards is better than downwards. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 269-280). Springer, Berlin, Heidelberg.
  - Recover a blinded secret exponent by SPA
  - Only works for left to right implementation for square and multiply algorithm
- Witteman, M. F., van Woudenberg, J. G., & Menarini, F. (2011, February). Defeating RSA multiply-always and message blinding countermeasures. In Cryptographers' Track at the RSA Conference (pp. 77-88). Springer, Berlin, Heidelberg.
  - DPA attack
- Fouque, P. A., Réal, D., Valette, F., & Drissi, M. (2008, August). The carry leakage on the randomized exponent countermeasure. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 198-213). Springer, Berlin, Heidelberg.
  - Exploit leakage during the computation of the random exponent

SCA countermeasures

- Introduction
- Square and multiply-always
- Blinding for RSA
- Masking for PRESENT

Boolean masking

- Proven to be secure given that the source of randomness is truly random[1]
- The cryptographic algorithm needs to be changed a bit for us to carry out computations with the masked intermediate values and keep track of all the masks
- At the end of the encryption, we can remove the masks to output the original ciphertext

[1] Prouff, E., & Rivain, M. (2013, May). Masking against side-channel attacks: A formal security proof. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 142-159). Springer, Berlin, Heidelberg.

Masking scheme

- A *masking scheme* specifies how masks are applied to the plaintext and intermediate values, as well as how they are removed from the ciphertext.
- There are a few principles we follow for a masking scheme design
  - All intermediate values should be masked during the computation. In particular, we would apply masks to the plaintext (and the key).
  - We assume the attacker does not have knowledge of the masks – otherwise, the attacker can carry out a DPA attack by making hypotheses about the key values.
  - When some intermediate values are to be `XOR`-ed with each other (e.g. in AES MixClomns operation), different masks should be applied to each of them.
  - Each encryption has a different set of randomly generated masks.
- For any function f, the mask that is applied to an input of f is called the *input mask* of f.
- The corresponding mask for the output is called the *output mask* for f.

Linear function

Definition

Let $f : \mathbb{F}_2^{m_1} \to \mathbb{F}_2^{m_2}$ be a function, where m_1 and m_2 are positive integers. f is said to be *linear* (w.r.t. $\oplus$) if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^{m_1}$, we have

$$f(\boldsymbol{x} \oplus \boldsymbol{y}) = f(\boldsymbol{x}) \oplus f(\boldsymbol{y}).$$

f is *non-linear* if it is not linear.

Example

- AddRoundKey operation in AES round function is ?
- DES Sboxes ?
- pLayer in PRESENT round function is ?

Linear function

Definition

Let $f : \mathbb{F}_2^{m_1} \to \mathbb{F}_2^{m_2}$ be a function, where m_1 and m_2 are positive integers. f is said to be *linear* (w.r.t. $\oplus$) if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^{m_1}$, we have

$$f(\boldsymbol{x} \oplus \boldsymbol{y}) = f(\boldsymbol{x}) \oplus f(\boldsymbol{y}).$$

f is *non-linear* if it is not linear.

Example

- AddRoundKey operation in AES round function is a linear function. In fact, bitwise `XOR` with a round key is a linear function in general.
- DES Sboxes are non-linear functions. Any Sbox proposed so far for symmetric block ciphers is non-linear.
- pLayer in PRESENT round function is linear.

Boolean masking for linear functions

- With Boolean masking, it is easy to keep track of the masks with linear operations.
- Let f be a linear operation and take any input of f, $\boldsymbol{v}$, with a corresponding mask $\mathfrak{m}$, we have
$$f(\boldsymbol{v} \oplus \mathfrak{m}) = f(\boldsymbol{v}) \oplus f(\mathfrak{m}).$$
- When the input mask is $\mathfrak{m}$, the output mask is given by $f(\mathfrak{m})$.
- One of the main challenges in designing a masking scheme is to find ways to keep track of masks for non-linear operations.

PRESENT

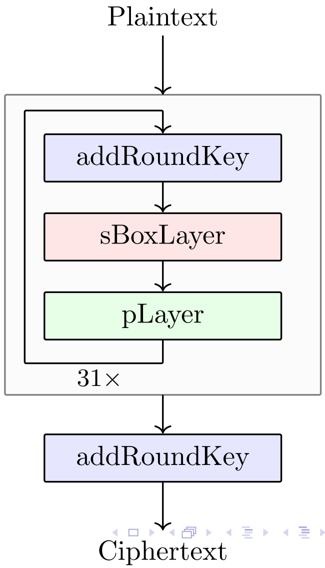
- Proposed in 2007 as a symmetric block cipher optimized for hardware implementation.
- Block length: 64
- Number of rounds: 31
- Key length: 80 or 128.

PRESENT – encryption

- Round function: addRoundKey, sBoxLayer, and pLayer.
- After 31 rounds, addRoundKey is applied again before the ciphertext output

Remark

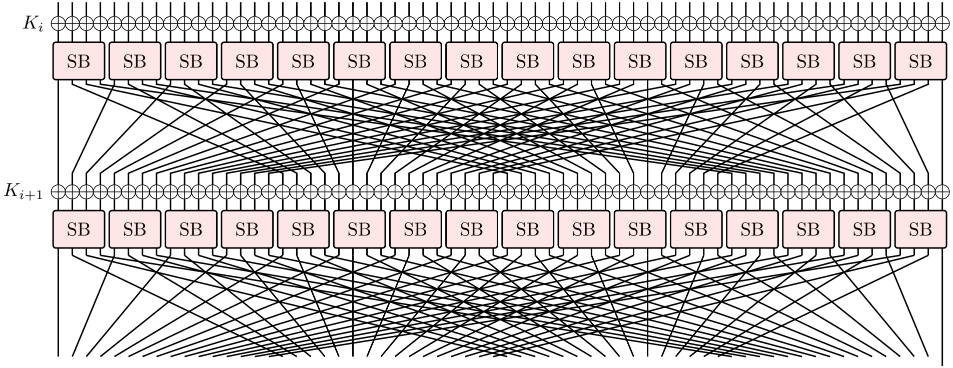
For PRESENT specification, we consider the 0th bit of a value as the rightmost bit in its binary representation. For example, the 0th bit of $3 = 011_2$ is 1, the 1st bit is 1 and the 2nd bit is 0.



PRESENT – sBoxLayer

- sBoxLayer applies sixteen $4-$bit Sboxes to each nibble of the current cipher state.
- For example, if the input is `0`, the output is `C`.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2



PRESENT – pLayer

pLayer permutes the 64 bits using the following formula:

$$\text{pLayer}(j) = \left\lfloor \frac{j}{4} \right\rfloor + (j \bmod 4) \times 16,$$

where j denotes the bit position.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Masking PRESENT Sbox

- Compute a lookup table T1 such that for any $v \in \mathbb{F}_2^4$, any input mask $\mathfrak{m}_{\mathsf{in}}$ and its corresponding output mask $\mathfrak{m}_{\mathsf{out}}$ for PRESENT Sbox,

$$\mathsf{T1}[v \oplus \mathfrak{m}_{\mathsf{in}}, \mathfrak{m}_{\mathsf{in}}] = \mathsf{SB}(v) \oplus \mathfrak{m}_{\mathsf{out}}.$$

- Table T2 helps us keep track of the masks

$$\mathsf{T2}[\mathfrak{m}_{\mathsf{in}}] = \mathfrak{m}_{\mathsf{out}}, \quad \mathfrak{m}_{\mathsf{in}} = 0, 1, \dots, \mathtt{F}.$$

- Do not need to generate a masked Sbox lookup table whenever the input mask for the Sbox changes.
- The size of T1 is 8×4, and the storage required is $2^8 \times 4 = 2^{10}$ bits, or 2^7 bytes.
- The table T2 requires $2^4 \times 4 = 64$ bits of memory.

Masked cipher state

- Since the pLayer operation is linear, we can simply apply pLayer to the masks to keep track of their changes.
- We represent the intermediate values of PRESENT encryption as

$$\boldsymbol{b}_{15}, \boldsymbol{b}_{14}, \ldots, \boldsymbol{b}_1, \boldsymbol{b}_0,$$

  where each $\boldsymbol{b}_j$ denotes a nibble of the cipher state.
- At the beginning of one encryption, we randomly generate 16 masks, and each is applied to one nibble of the plaintext.
- Suppose the cipher state at the input of round i is of the following format:

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}^{i-1}_{15,\text{in}}, \boldsymbol{b}_{14} \oplus \mathfrak{m}^{i-1}_{14,\text{in}}, \ldots, \boldsymbol{b}_1 \oplus \mathfrak{m}^{i-1}_{1,\text{in}}, \boldsymbol{b}_0 \oplus \mathfrak{m}^{i-1}_{0,\text{in}}.$$

Masked addRoundKey

Suppose the cipher state at the input of round i is of the following format:

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}_{15,\text{in}}^{i-1}, \boldsymbol{b}_{14} \oplus \mathfrak{m}_{14,\text{in}}^{i-1}, \ldots, \boldsymbol{b}_{1} \oplus \mathfrak{m}_{1,\text{in}}^{i-1}, \boldsymbol{b}_{0} \oplus \mathfrak{m}_{0,\text{in}}^{i-1}.$$

- We do not apply masks to the round keys. Consequently, after the addRoundKey operation, the cipher state will be?

Masked addRoundKey

Suppose the cipher state at the input of round i is of the following format:

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}^{i-1}_{15,\text{in}}, \boldsymbol{b}_{14} \oplus \mathfrak{m}^{i-1}_{14,\text{in}}, \ldots, \boldsymbol{b}_{1} \oplus \mathfrak{m}^{i-1}_{1,\text{in}}, \boldsymbol{b}_{0} \oplus \mathfrak{m}^{i-1}_{0,\text{in}}.$$

- We do not apply masks to the round keys. Consequently, after the addRoundKey operation, each nibble of the cipher state still has the same mask

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}^{i-1}_{15,\text{in}}, \boldsymbol{b}_{14} \oplus \mathfrak{m}^{i-1}_{14,\text{in}}, \ldots, \boldsymbol{b}_{1} \oplus \mathfrak{m}^{i-1}_{1,\text{in}}, \boldsymbol{b}_{0} \oplus \mathfrak{m}^{i-1}_{0,\text{in}},$$

Masked sBoxLayer

Let

$$\mathfrak{m}_{j,\text{out}}^{i-1} = \text{T2}\left[\mathfrak{m}_{j,\text{in}}^{i-1}\right], \quad j = 0, 1, \ldots, 15,$$

denote the output mask for PRESENT Sbox corresponding to the input mask $\mathfrak{m}_{j,\text{in}}^{i-1}$.
Then after sBoxLayer, the cipher state is ?

Masked sBoxLayer

Let

$$\mathfrak{m}_{j,\mathsf{out}}^{i-1} = \mathsf{T2}\left[\mathfrak{m}_{j,\mathsf{in}}^{i-1}\right], \quad j = 0, 1, \ldots, 15,$$

denote the output mask for PRESENT Sbox corresponding to the input mask $\mathfrak{m}_{j,\mathsf{in}}^{i-1}$.
Then after sBoxLayer, the cipher state is as follows:

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}_{15,\mathsf{out}}^{i-1}, \boldsymbol{b}_{14} \oplus \mathfrak{m}_{14,\mathsf{out}}^{i-1}, \ldots, \boldsymbol{b}_{1} \oplus \mathfrak{m}_{1,\mathsf{out}}^{i-1}, \boldsymbol{b}_{0} \oplus \mathfrak{m}_{0,\mathsf{out}}^{i-1},$$

Masked pLayer

- We apply the pLayer operation to both the cipher state and the mask for the whole cipher state, i.e. the string obtained by concatenating all 16 masks $\mathfrak{m}^{i-1}_{j,\mathsf{out}}$:

$$\mathfrak{m}^{i-1}_{15,\mathsf{out}}, \mathfrak{m}^{i-1}_{14,\mathsf{out}}, \dots, \mathfrak{m}^{i-1}_{1,\mathsf{out}}, \mathfrak{m}^{i-1}_{0,\mathsf{out}}.$$

- After pLayer, masks for each nibble of the cipher state will be changed and the cipher state will become:

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}^{i}_{15,\mathsf{in}}, \boldsymbol{b}_{14} \oplus \mathfrak{m}^{i}_{14,\mathsf{in}}, \dots, \boldsymbol{b}_{1} \oplus \mathfrak{m}^{i}_{1,\mathsf{in}}, \boldsymbol{b}_{0} \oplus \mathfrak{m}^{i}_{0,\mathsf{in}}.$$

- Where

$$\mathfrak{m}^{i}_{15,\mathsf{in}}, \mathfrak{m}^{i}_{14,\mathsf{in}}, \dots, \mathfrak{m}^{i}_{1,\mathsf{in}}, \mathfrak{m}^{i}_{0,\mathsf{in}} = \mathsf{pLayer}(\mathfrak{m}^{i-1}_{15,\mathsf{out}}, \mathfrak{m}^{i-1}_{14,\mathsf{out}}, \dots, \mathfrak{m}^{i-1}_{1,\mathsf{out}}, \mathfrak{m}^{i-1}_{0,\mathsf{out}}).$$

- Consequently, $\mathfrak{m}^{i}_{j,\mathsf{in}}$ will be the input mask for the jth Sbox in round $i+1$.

Final addRoundKey

- Finally, after 31 rounds, we have another addRoundKey operation, which does not change the masks of the cipher state.
- The cipher state will be

$$\boldsymbol{b}_{15} \oplus \mathfrak{m}^{31}_{15,\mathrm{in}}, \boldsymbol{b}_{14} \oplus \mathfrak{m}^{31}_{14,\mathrm{in}}, \ldots, \boldsymbol{b}_{1} \oplus \mathfrak{m}^{31}_{1,\mathrm{in}}, \boldsymbol{b}_{0} \oplus \mathfrak{m}^{31}_{0,\mathrm{in}}.$$

- To get the unmasked ciphertext, we remove the masks by `XOR`ing the cipher state with

$$\mathfrak{m}^{31}_{15,\mathrm{in}}, \mathfrak{m}^{31}_{14,\mathrm{in}}, \ldots, \mathfrak{m}^{31}_{1,\mathrm{in}}, \mathfrak{m}^{31}_{0,\mathrm{in}}.$$

Algorithmic description

- Input: p, T1, T2, K_i $(i = 1, 2, \ldots, 32)$
- Output: ciphertext

```
randomly generate 16 masks   $\mathfrak{m}_0, \mathfrak{m}_1, \ldots, \mathfrak{m}_{15}$
array of size 16   state $= p \oplus \mathfrak{m}_{15}, \mathfrak{m}_{14}, \ldots, \mathfrak{m}_1, \mathfrak{m}_0$// mask the $j$th nibble of the
    plaintext with $\mathfrak{m}_j$, each entry of the array is one masked nibble
array of size 16   masks $= \mathfrak{m}_{15}, \mathfrak{m}_{14}, \ldots, \mathfrak{m}_1, \mathfrak{m}_0$
for $i = 0$, $i < 31$, $i{+}{+}$ do
    state = addRoundKey(state, $K_i$)
    for $j = 0$, $j < 16$, $j{+}{+}$ do
        // for each nibble
        state[$j$] = T1[state[$j$], masks[$j$]]// masked Sbox computation
        masks[$j$] = T2[masks[$j$]]// record the output masks of Sbox computation
    state = pLayer(state)// apply pLayer to the cipher state
    masks = pLayer(masks)// apply pLayer to the masks
state = addRoundKey(state, $K_i$)
state = state $\oplus$ masks// remove mask from the ciphertext
return state
```

Design of T2

- It is suggested that T2 be designed such that all possible values of $\mathfrak{m}_{\text{in}} \oplus \mathfrak{m}_{\text{out}}$ appear.
- For example, one possible choice of T2 is as follows

$\mathfrak{m}_{\text{in,SB}}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\mathfrak{m}_{\text{out,SB}} = \text{T2}[\mathfrak{m}_{\text{in,SB}}]$	E	4	F	9	0	3	D	5	7	8	A	2	B	1	6	C
$\mathfrak{m}_{\text{in,SB}} \oplus \mathfrak{m}_{\text{out,SB}}$	E	5	D	A	4	6	B	2	F	1	0	9	7	C	8	3

Sasdrich, P., Bock, R., & Moradi, A. (2018). Threshold Implementation in Software: Case Study of PRESENT. In Constructive Side-Channel Analysis and Secure Design: 9th International Workshop, COSADE 2018, Singapore, April 23–24, 2018, Proceedings 9 (pp. 227-244). Springer International Publishing.

Higher order DPA

- In fact, in general, we have the following observations:
- Let f be any function and let $\mathfrak{m}_{\text{in},f}$ (resp. $\mathfrak{m}_{\text{out},f}$) denote its input mask (resp. output mask).
- For any input $\boldsymbol{x}$ of f, we have

$$(\boldsymbol{x} \oplus f(\boldsymbol{x})) \oplus (\mathfrak{m}_{\text{in},f} \oplus \mathfrak{m}_{\text{out},f}) = (\boldsymbol{x} \oplus \mathfrak{m}_{\text{in},f}) \oplus (f(\boldsymbol{x}) \oplus \mathfrak{m}_{\text{out},f}).$$

- Thus, when choosing the input mask $\mathfrak{m}_{\text{in},f}$ and output mask $\mathfrak{m}_{\text{out},f}$ of f, we need to ensure that all possible values of $\mathfrak{m}_{\text{in},f} \oplus \mathfrak{m}_{\text{out},f}$ appear.
- Otherwise, the distribution induced by $(\boldsymbol{x} \oplus f(\boldsymbol{x})) \oplus (\mathfrak{m}_{\text{in},f} \oplus \mathfrak{m}_{\text{out},f})$ will not be uniform, and the signal corresponding to the value of $\boldsymbol{x} \oplus f(\boldsymbol{x})$ cannot be properly concealed, making it vulnerable to DPA attacks.
- Second-order DPA

Higer-order masking

- The masked value $\boldsymbol{v}_m$ is related to the original value $\boldsymbol{v}$ and the mask $\mathfrak{m}$ through a binary operation $\boldsymbol{v}_{\mathfrak{m}} = \boldsymbol{v} \cdot \mathfrak{m}$.
- In the language of *secret sharing*[1], we can say that the secret value $\boldsymbol{v}$ is represented by two *shares*, $\boldsymbol{v}_{\mathfrak{m}}$ and $\mathfrak{m}$.
- Given only one of the two shares, no information about $\boldsymbol{v}$ can be revealed.
- Instead of two shares (or one mask), we can also use several shares, resulting in a higher-order masking.
- In particular, a d*th order masking* apply $d-1$ masks to the secret value $\boldsymbol{v}$.

[1] Beimel, A. (2011, May). Secret-sharing schemes: A survey. In International conference on coding and cryptology (pp. 11-46). Berlin, Heidelberg: Springer Berlin Heidelberg.

Higher order DPA

- The DPA attack we have discussed uses one intermediate value – such a DPA attack is also called *first-order DPA*.
- When information leakage from several intermediate values (e.g. by combining leakages from a few time samples) is analyzed, we have a *higher-order DPA*[1]
- The number of traces needed for a higher-order DPA to succeed is exponential in the standard deviation of the noise. The exponent is given by $d + 1$, where $d + 1$ is the order of the masking (i.e. d masks are applied)[2]

[1] Chari, S., Jutla, C. S., Rao, J. R., & Rohatgi, P. (1999, August). Towards sound approaches to counteract power-analysis attacks. In Annual International Cryptology Conference (pp. 398-412). Springer, Berlin, Heidelberg.

[2] Prouff, E., & Rivain, M. (2013, May). Masking against side-channel attacks: A formal security proof. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 142-159). Springer, Berlin, Heidelberg.

Security of Boolean masking

- Let us take a secret value $\boldsymbol{v}$ of bit length at most m_v.
- The masked value is given by $\boldsymbol{v} \oplus \mathfrak{m}$, where $\mathfrak{m} \in \mathbb{F}_2^{m_v}$.
- We can consider the value of $\mathfrak{m}$ as a discrete random variable.
- In case the distribution induced by this random variable is uniform on $\mathbb{F}_2^{m_v}$, the distribution induced by the value of $\boldsymbol{v} \oplus \mathfrak{m}$ is also uniform on $\mathbb{F}_2^{m_v}$ regardless of the value of $\boldsymbol{v}$.
- Thus, we expect the leakage to be independent of $\boldsymbol{v}$ when only first-order DPA is carried out.
- For Boolean masking schemes with one mask, security proofs have been given[1]
- Security proofs for higher order masking is also given[2]

[1] Blömer, J., Guajardo, J., & Krummel, V. (2004, August). Provably secure masking of AES. In International workshop on selected areas in cryptography (pp. 69-83). Springer, Berlin, Heidelberg.

[2] Rivain, M., & Prouff, E. (2010, August). Provably secure higher-order masking of AES. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 413-427). Springer, Berlin, Heidelberg.

Further Reading

- Masking was first proposed by Goubin and Patarin[1] and Chari et al.[2] independently
- Hardware implementations of masking are vulnerable to DPA attacks due to glitches in CMOS circuits[3]
- Threshold Implementation, which is based on multiparty computation[4], was introduced as a proper way to realize Boolean masking in hardware platforms[5]

[1] Goubin, L., & Patarin, J. (1999, August). DES and differential power analysis the "Duplication" method. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 158-172). Springer, Berlin, Heidelberg.

[2] Chari, S., Jutla, C. S., Rao, J. R., & Rohatgi, P. (1999, August). Towards sound approaches to counteract power-analysis attacks. In Annual International Cryptology Conference (pp. 398-412). Springer, Berlin, Heidelberg.

[3] Mangard, S., Popp, T., & Gammel, B. M. (2005, February). Side-channel leakage of masked CMOS gates. In Cryptographers' Track at the RSA Conference. Springer, Berlin, Heidelberg.

[4] Cramer, R., & Damgård, I. (2005). Multiparty computation, an introduction. In Contemporary cryptology (pp. 41-87). Birkhäuser Basel.

[5] Nikova, S., Rijmen, V., & Schläffer, M. (2011). Secure hardware implementation of nonlinear functions in the presence of glitches. Journal of Cryptology, 24(2), 292-321.