Secure Implementation on Smart Cards

Emmanuel Prouff
emmanuel.prouff@ssi.gouv.fr

ANSSI

Course UVSQ – Jan. 2018

Introduction

Partie II. Algorithms for Embedded Systems

Partie III. Passive Attacks

Partie iV. Active Attacks

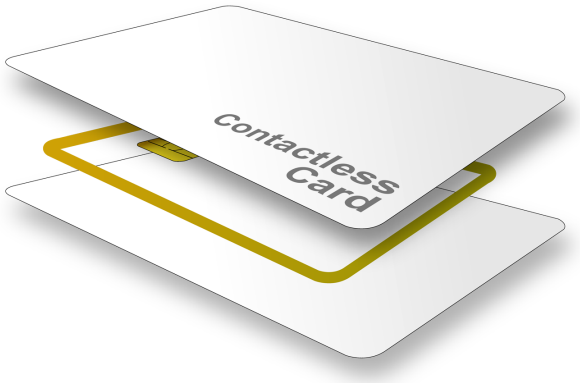
Première partie I

Introduction

Smart Card

- A Smart Card is a circuit embedded on a plastic support. It moreover has communication means, storage capacities and computation capacities.
- The physical characteristics of a smart card are standardized.
- The smart card enables the secure storage of sensitive data : a part of its memory is indeed protected in both writing and reading modes.

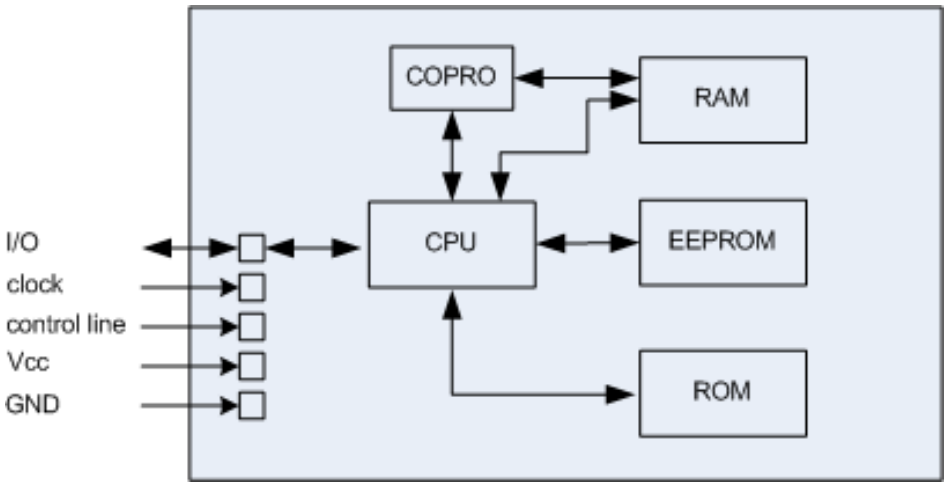
A Smart Card ?



- A plastic Support
- Storage and computation means
  - Micro-controller (ST, Atmel, NXP, Samsung, Infineon, etc.)
- Communication means
  - Connecters
  - Antenna

- Main Goal : embed private data and manipulate them in a secure way.

A Smart Card ?

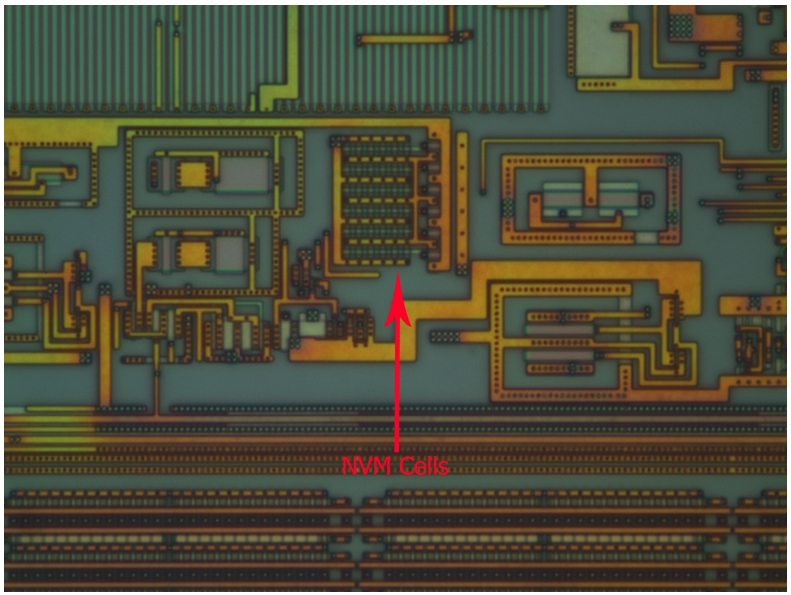


- ROM contains the smart card OS.
- RAM is dedicated to the storage of local and volatile variables during the processings.
- EEPROM contains code and some data.
- Co-processor is dedicated to particular cryptographic (*e.g. arithmetic) calculations.*

A Smart Card ?

- ROM contains the smart card OS.
- RAM is dedicated to the storage of local and volatile variables during the processings.
- EEPROM contains code and some data.
- Co-processor is dedicated to particular cryptographic (*e.g. arithmetic) calculations.*

A Smart Card ?

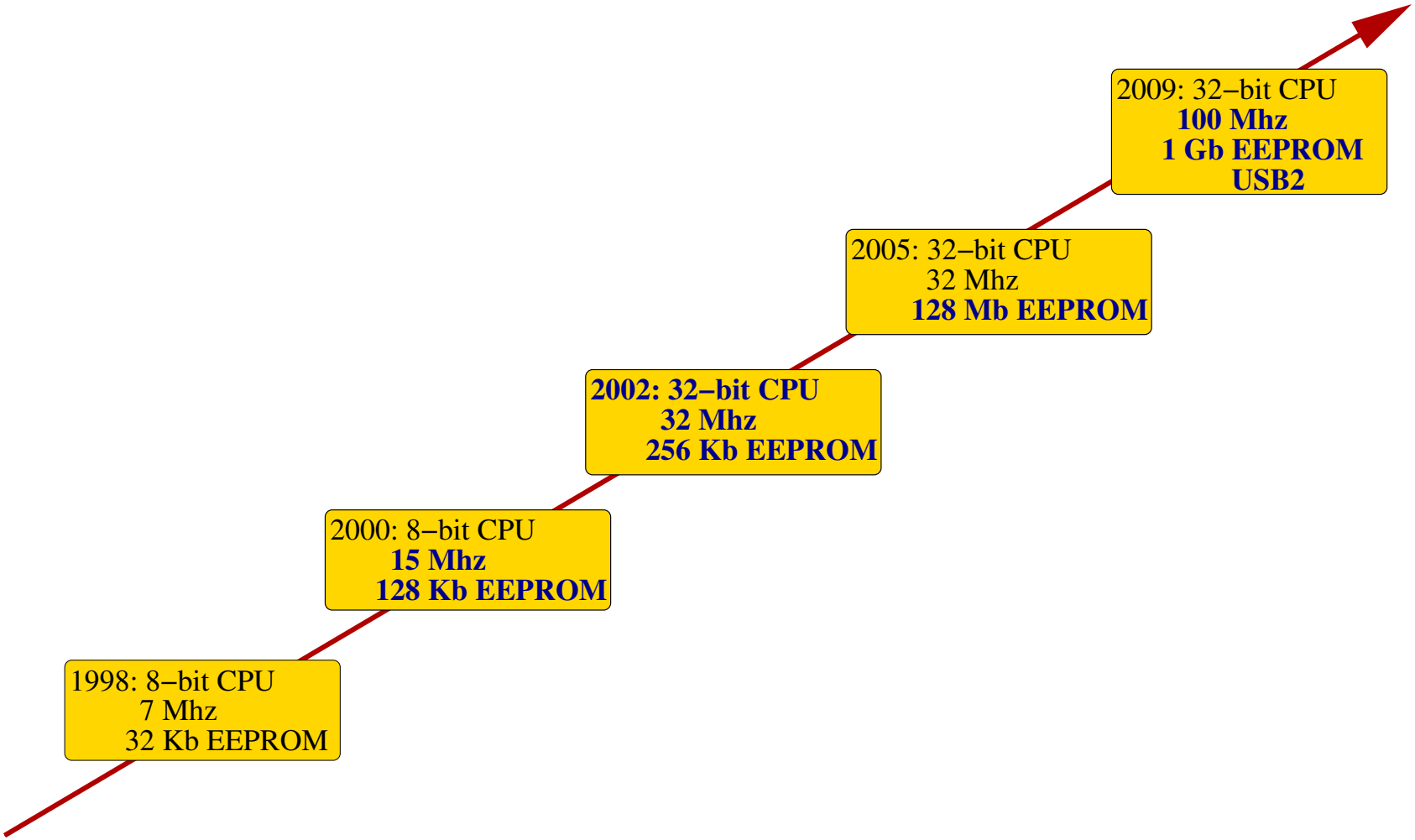


- ROM contains the smart card OS.
- RAM is dedicated to the storage of local and volatile variables during the processings.
- EEPROM contains code and some data.
- Co-processor is dedicated to particular cryptographic (*e.g. arithmetic) calculations.*

Smart Card Evolution

- The first smart cards (Bull and Motorola) only had 36 bytes of RAM and 1600 bytes of ROM.
- Today, a smart card has ;
  - between 16 and 512 Kbytes of ROM,
  - between 1 and 32 Kbytes of RAM,
  - a processor running at 100 MHz.
- ... it can ;
  - embed several mega-bytes of Flash memory,
  - communicate in USB2.0,
  - embed a web server.

Smart Card Evolution



Which Algorithm for which Security Issue ?

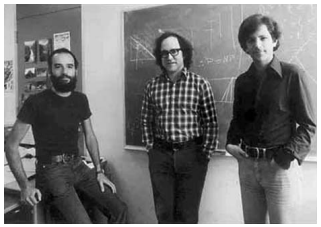
- Client (Bank, Operator, Government, etc.) has to deal with security issues : securing transactions, protecting citizen anonymity, limiting access to services, etc.
- In more than 95% of the cases it asks its internal security experts to find a solution.
  - A standard exists : it will certainly be chosen !
  - No satisfying standard exists : a new standardization process is initiated (*e.g. ETSI 3GPP, ISO, ICAO).*
  - *No satisfying standard exists : a solution is designed by the internal experts (proprietary algorithms).*
- Sometimes (in less than 5% of the cases) the Client asks Industrial experts to find a solution.
- As a consequence, the Smart Card Industry essentially implements standards (and even a very few of them !).

Cryptography in Smart Cards

Smart Cards implement a wide range of cryptographic algorithms :

- Block Ciphers : (Triple-)DES, AES, proprietary algorithms
- Hash functions : SHA family
- Data authentication : CBC-MAC, HMAC
- Symmetric key cryptography : RSA (OAEP, PKCS1-v1.5)
- Signature : RSA (PKCS1-v1.5, PSS), DSA, ECDSA
- Key exchange protocols : Diffie-Hellman, Diffie-Hellman on elliptic curves

RSA algorithm : introduction



RSA primitive

Let p, q be two prime numbers and let $N = pq$.
Let $e, d \in \mathbb{Z}_{\phi(N)}$ be such that $ed = 1 \mod \phi(N)$.

Public key $PK = (N, e)$	modulus N, *public exponent* e
Private key $SK = (N, d)$	private exponent d

Encryption	Decryption
$c = m^e \mod N$	$m = c^d \mod N$
Signature	**Verification**
$s = m^d \mod N$	$m \stackrel{?}{=} s^e \mod N$

Application : Banking cards 1/5

A banking smartcard contains the following (more or less sensitive) data :

- Public information :
  - the serial card number,
  - validity dates,
  - smart card holder identity,
  - an authentication value *VA* on 320 bits [before 1999].
- Sensitive information :
  - transactions history,
  - the PIN code.

Application : banking smart cards 2/5

The authentication value VA allows the terminal to check that the card has been issued by an allowed authority.

- The GIE Banking Cards Group generates $n = pq$ and spreads it in every banking terminal
- The value d is kept secret : $3d \equiv 1 \mod n$.

VA computation :

1. Construction of a binary chain I from the public information
2. Build $\tilde{I}$ such that $\tilde{I} = I \| I$
3. Compute the signature $VA = \tilde{I}^d \mod n$
4. Store VA in the card

Application : banking smart cards 3/5

Protocol

1. Terminal reads I and VA
2. Terminal checks the signature : $VA^3 \equiv I||I \mod n$
3. Terminal requests the smart-card holder authentication : PIN
4. Terminal asks the card to check the entered PIN
5. Terminal checks whether the transaction can continue
6. Carte writes the transaction in the history file
7. An invoice and transaction certificate are generated

Optional Step : Terminal calls the bank servers to involve it in the risk management process.

Application : banking smart cards 4/5

Humpich 1997

Since 1991, methods were known to factor 100-digits numbers in 1 week on a @450MHz PC (130-digits numbers in 1997).

- The cloning of a banking card is possible by copying all the data in its memory.
- Until 1999, n was a 96-digits integer (≈ 318 bits)...

Evolution :

- n is today 768-bits long.
- New international norm EMV in 1997
  - an authority generates a certificate for the public key of the smart card issuer. This certificate is stored in the card.
  - The smart card issuer stores on the card data signed with the corresponding private key.

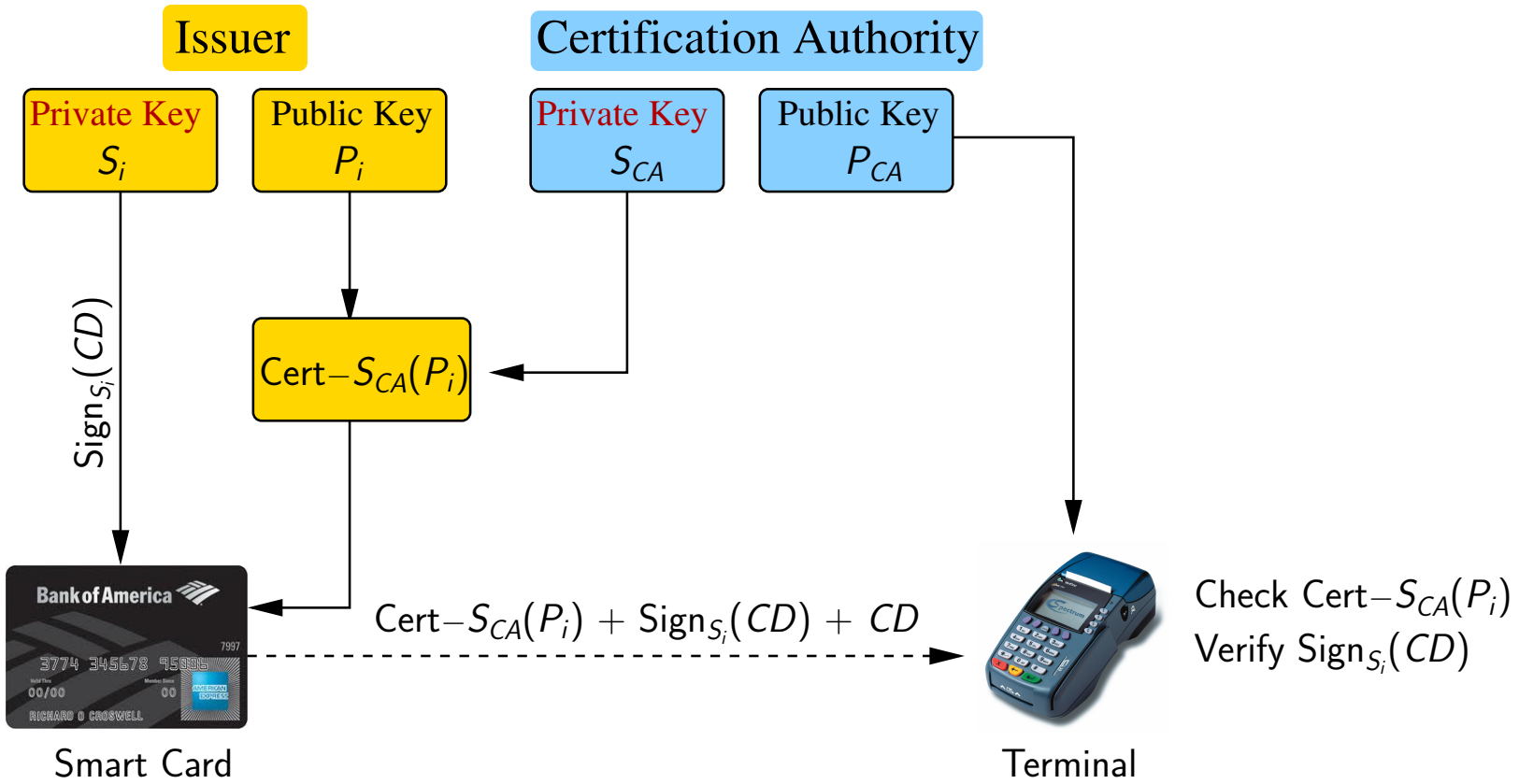
Application : banking smart cards 5/5

Norm EMV 4.1 : Integrated Circuit Card Specifications for Payment Systems (based on norm ISO/IEC 7816

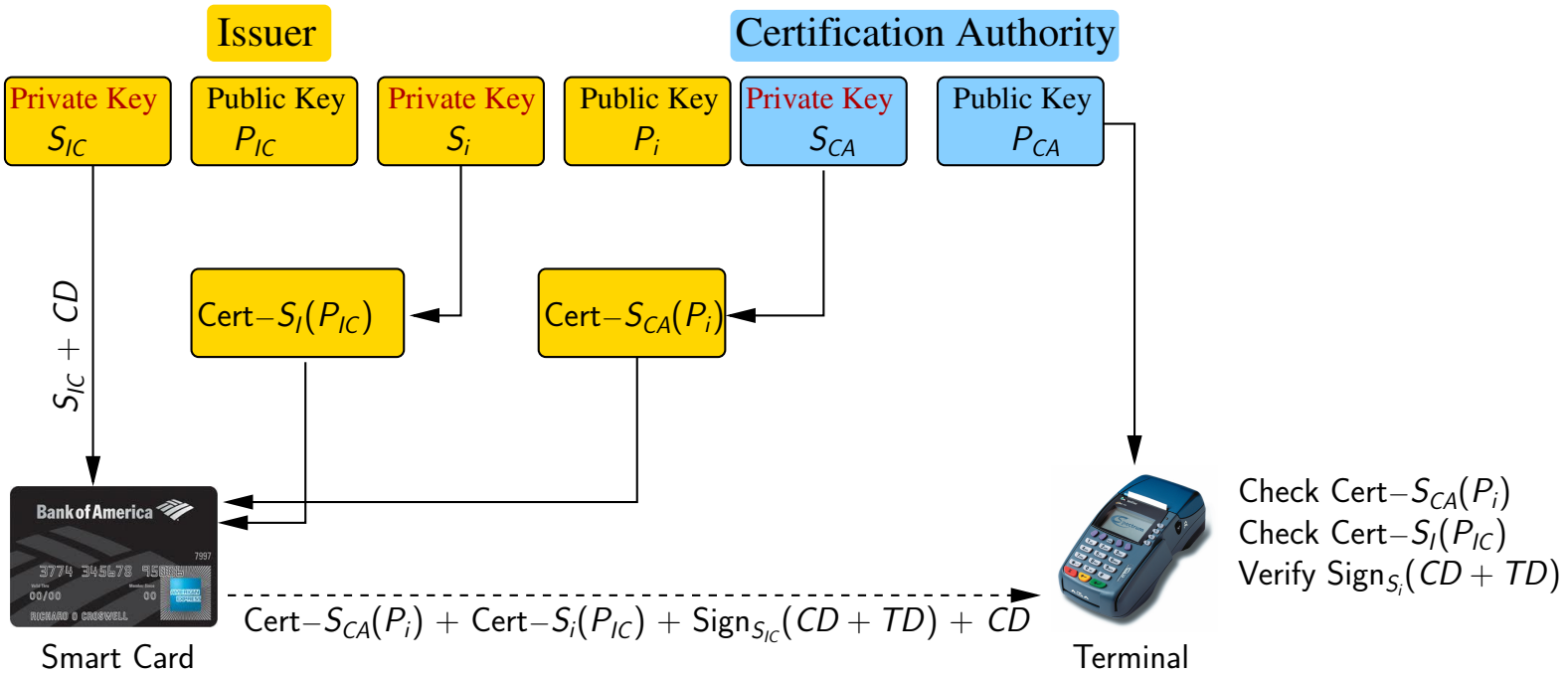
Static Data Authentication (SDA) : Terminal verifies the static signature provided by the card thanks to the Issuer public key.

Dynamic Data Authentication (DDA) : the smart card has its own signature key certified by the Issuer and the Terminal verifies a dynamic signature computed by the card .

Banking – SDA


Issuer
Certification Authority
Private Key S_i
Public Key P_i
Private Key S_{CA}
Public Key P_{CA}
$\text{Sign}_{S_i}(CD)$
Cert$-S_{CA}(P_i)$
Bank of America
3774 345678 95005
00/00
RICHARD O CROSWELL
Cert$-S_{CA}(P_i)$ + $\text{Sign}_{S_i}(CD)$ + CD
Check Cert$-S_{CA}(P_i)$
Verify $\text{Sign}_{S_i}(CD)$
Smart Card
Terminal

Banking – DDA


Issuer
Certification Authority
Private Key S_{IC}
Public Key P_{IC}
Private Key S_i
Public Key P_i
Private Key S_{CA}
Public Key P_{CA}
$S_{IC} + CD$
Cert$-S_I(P_{IC})$
Cert$-S_{CA}(P_i)$
Bank of America
Smart Card
Cert$-S_{CA}(P_i)$ + Cert$-S_i(P_{IC})$ + Sign$_{S_{IC}}(CD + TD)$ + CD
Terminal
Check Cert$-S_{CA}(P_i)$
Check Cert$-S_I(P_{IC})$
Verify Sign$_{S_i}(CD + TD)$

Attacks against Smart Card Implementations

- Classical Model : black-box model. The adversary observes only the input/output channel.
  - passively (non-adaptive attacks),
  - actively (adaptive attacks).
- Opposite Model : white-box model. The adversary has a total access to the implementation.
  - The secrets are hidden in the implementation
  - The adversary must not be able to extract them.
- In the middle : gray-box model. Embedded cryptography is vulnerable to other kinds of attacks through side channels.
  - the device behaviour indeed strongly depends on the value of the manipulated data.
  - active attacks : execution is disturbed during sensitive manipulations
  - passive attacks : execution is observed to retrieve information

First Example : the modular exponentiation

- The modular exponentiation is the main operation during the RSA processing and other algorithms :
  - Diffie-Helmann
  - DSA
  - primality tests
- This is an operation costly both in terms of memory and timing :
  $\Longrightarrow$ Need for optimisation
- This is an operation that manipulates secret data :
  $\Longrightarrow$ Need for security

Second Example : the AES encryption/decryption algorithm

- Involved in many protocols :
  - Authentication in cellular networks
  - Data encryption in E-Passport
  - Key derivation in Pay-TV
- This is an operation costly both in terms of memory and timing :
  $\Longrightarrow$ Need for optimisation
- This is an operation that manipulates secret data :
  $\Longrightarrow$ Need for security

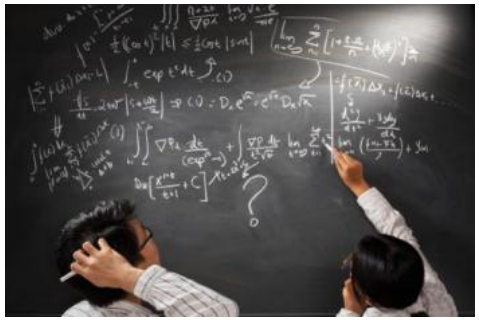
Deuxième partie II

Algorithms for Embedded Systems

Outline

Modular Exponentiation

Why is it an issue ?



Straightforward algorithm to process $m^d \mod m$: $d-1$ modular multiplications.

To improve it, we can :

- improve the modular exponentiation,
  - To be solved at hardware level.
- reduce the number of multiplications.
  - To be solved at software level.

In both cases, this implies to go into the details of the calculus.

Representation

- A n-bit processor can make operations on n-bits words :
  - arithmetic : addition, subtraction, multiplication, division
  - logic : or, and, bitwise complement, shifts
- Smart cards usually have 8, 16 or 32-bit processor.
- To process on larger numbers, we need to choose a representation and specify the operations for this representation.

Base-b representation

If $b \geq 2$ is an integer, then every positive integer x can be written in an unique way under the form : $x = x_n b^n + x_{n-1} b^{n-1} + \ldots + x_1 b + x_0$, with x_i integers such that $0 \leq x_i < b$ for $0 \leq i \leq n$ and $x_n \neq 0$.

Addition

Algo Multi-precision Addition

INPUT(S) :
$x = (x_n, x_{n-1}, \ldots, x_1, x_0)_b, y = (y_n, y_{n-1}, \ldots, y_1, y_0)_b$
OUTPUT(S) : $w = x + y = (w_{n+1}, w_n, \ldots, w_1, w_0)_b$

1: $c \leftarrow 0$ {Carry initialisation}
2: **for** $i = 0$ to n **do**
3: $w_i \leftarrow (x_i + y_i + c) \mod b$
4: **if** $(x_i + y_i + c) < b$ **then**
5: $c \leftarrow 0$ {no carry}
6: **else**
7: $c \leftarrow 1$ {carry}
8: $w_{n+1} \leftarrow c$
9: **return** $(w_{n+1}, w_n, \ldots, w_1, w_0)$

Multiplication

Algo Multi-precision Multiplication

INPUT(S) :
$x = (x_n, x_{n-1}, \ldots, x_1, x_0)_b, y = (y_t, y_{t-1}, \ldots, y_1, y_0)_b$
OUTPUT(S) : $w = x.y = (w_{n+t+1}, w_{n+t}, \ldots, w_1, w_0)_b$

1: **for** $i = 0$ Ã $n + t + 1$ **do**
2: $w_i \leftarrow 0$
3: **for** $i = 0$ to t **do**
4: $c \leftarrow 0$ {Carry initialisation}
5: **for** $j = 0$ to n **do**
6: Compute $(uv)_b = (w_{i+j} + x_j y_i + c)$
7: $w_{i+j} \leftarrow v$
8: $c \leftarrow u$ {carry}
9: $w_{i+n+1} \leftarrow u$
10: **return** $(w_{n+t+1}, w_{n+t}, \ldots, w_1, w_0)$

Co-processor

For operations on large numbers, some cards include a co-processor :

- it can manipulate numbers (much) larger than the processor can,
- it includes *super operations* that enable to directly perform operations on large numbers.

Modular Operations

Definition

Let x be an integer, then $x \mod m$ is called modular reduction of x modulo m. This is the remainder of the "classical" division of x by m. One denote by $\mathbb{Z}_m$ the set of all possible remainders modulo m.

Algo Modular Addition

INPUT(S) : $x, y \in \mathbb{Z}_m$
OUTPUT(S) : $w = x + y \mod m$

1: $w \leftarrow x + y$
2: **if** $w > m$ **then**
3: $\quad w \leftarrow w - m$
4: **return** w

Algo Modular Subtraction

INPUT(S) : $x, y \in \mathbb{Z}_m$ such that $x \geq y$
OUTPUT(S) : $w = x - y \mod m$

1: $w \leftarrow x - y$
2: **return** w

Modular Multiplication

Algo Modular Multiplication

INPUT(S) : $x, y \in \mathbb{Z}_m$
OUTPUT(S) : $w = xy \mod m$

1: $w \leftarrow xy$
2: Calculer $w \mod m$
3: **return** w

A straightforward method in order to perform a modular reduction is to first compute a classical division and then to take the remainder.

Algo Multi-precision Division

INPUT(S) : $x = (x_n, \ldots x_1 x_0)_b, y = (y_t, \ldots y_1 y_0)_b$ with $n \geq t \geq 1$, $y_t \neq 0$
OUTPUT(S) : $q = (q_{n-t} \ldots q_1 q_0)_b$ and $r = (r_t, \ldots r_1 r_0)_b$ such that $x = qy + r, 0 \leq r < y$

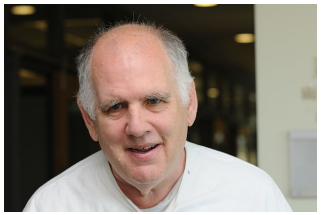
1: **for** $j = 0$ to $n - t$ **do**
2: $\quad q_j \leftarrow 0$
3: **while** $x \geq yb^{n-t}$ **do**
4: $\quad q_{n-t} \leftarrow q_{n-t} + 1$
5: $\quad x \leftarrow x - yb^{n-t}$
6: **for** $i = n$ to $t + 1$ **do**
7: $\quad$ **if** $x_i = y_t$ **then**
8: $\quad\quad q_{i-t-1} \leftarrow b - 1$
9: $\quad$ **else**
10: $\quad\quad q_{i-t-1} \leftarrow \lfloor (x_i b + x_{i-1})/y_t \rfloor$
11: $\quad$ **while** $q_{i-t-1}(y_t b + y_{t-1}) > x_i b^2 + x_{i-1} b + x_{i-2}$ **do**
12: $\quad\quad q_{i-t-1} \leftarrow q_{i-t-1} - 1$
13: $\quad x \leftarrow x - q_{i-t-1} y b^{i-t-1}$
14: $\quad$ **if** $x < 0$ **then**
15: $\quad\quad x \leftarrow x + yb^{i-t-1}$
16: $\quad\quad q_{i-t-1} \leftarrow q_{i-t-1} - 1$
17: $r \leftarrow x$
18: **return** (q, r)

$\approx (t + 4)(n - t)$ MULT
$n - t$ DIV

Cost

- This is complex and costly operation : it implies the processing of "true" divisions.
- More efficient methods have been proposed in the literature.

Montgomery Arithmetic



Definition

Let R be an integer an integer such that $R > m$ and $\gcd(m, R) = 1$. For y with $0 \leq y < mR$, $yR^{-1} \mod m$ is called Montgomery's reduction of y modulo m with respect to R.

- For R well chosen, this reduction can be processed efficiently.
- Montgomery's reduction enables to compute the modular multiplication without costly modular reductions.

Principle

- Principle : add a multiple of m in order to enable divisions by R.
- Let :
  - R be an integer such that $R > m$ and $\gcd(m, R) = 1$
  - $m' = -m^{-1} \mod R$

Algo MontRed

INPUT(S) : $0 \leq T < mR$
OUTPUT(S) : $TR^{-1} \mod m$

1: $U \leftarrow Tm' \mod R$
2: $t \leftarrow (T + Um)/R$
3: **if** $t > m$ **then**
4: $\quad u \leftarrow t - m$
5: **return** u

Second Step

t is indeed divisible by R :

$Um \equiv Tmm' \equiv -T \mod R$

Result

- $t \equiv TR^{-1} \mod m$
- $0 \leq t < 2m$

If the integers are represented in b-base, we can choose $R = b^n$:

$\mod R$: take the n last digits
$/R$: right shift of n digits

Montgomery's reduction

Soient :

- $m = (m_{n-1} \dots m_1 m_0)_b$
- $R = b^n$, $\gcd(m, b) = 1$
- $m' = -m^{-1} \mod b$

Step 3 : 1 MULT
Step 4 : n MULT
Total : $n(n+1)$ MULT and 0 DIV

Algo MontRed

INPUT(S) : $T = (t_{2n-1} \dots t_1 t_0)_b < mR$
OUTPUT(S) : $TR^{-1} \mod m$

1: $A \leftarrow T$
2: **for** $i = 0$ to $n-1$ **do**
3: $\quad u_i \leftarrow a_i m' \mod b$
4: $\quad A \leftarrow A + u_i m b^i$
5: $A \leftarrow A/b^n$
6: **if** $A \geq m$ **then**
7: $\quad A \leftarrow A - m$
8: **return** A

Illustration : $x^5 \mod m$

Let $\tilde{x} = xR \mod m$ and $\tilde{y} = yR \mod m$. Montgomery's reduction of $\tilde{x}\tilde{y}$ is $\tilde{x}\tilde{y}R^{-1} = xyR \mod m$.

Algo Exponentiation with MontRed

1: $\tilde{x} = xR \mod m$
2: $t = \text{MontRed}(\tilde{x}\tilde{x}) = \tilde{x}^2R^{-1} \mod m$
3: $u = \text{MontRed}(t^2) = \tilde{x}^4R^{-3} \mod m$
4: $v = \text{MontRed}(u\tilde{x}) = \tilde{x}^5R^{-4} \mod m = x^5R \mod m$
5: $w = vR^{-1} \mod m = x^5 \mod m$

Algo Classical Exponentiation

1: $t = x^2 \mod m$
2: $u = t^2 \mod m = x^4 \mod m$
3: $v = ux \mod m = x^5 \mod m$

If the processing of MontRed(xy) is faster than that of $xy \mod m$, this method is more efficient than the classical method. $\tilde{x} = xR \mod m$ is called Montgomery representation of x.

Montgomery Multiplication 1/2

Let $\tilde{x} = xR \mod m, \tilde{y} = yR \mod m$.
To process $\tilde{z} = xyR \mod m$, we can :

1. Compute the classical multiplication :

$$z' = \tilde{x}\tilde{y} = xyR^2 < mR$$

2. Apply Montgomery's reduction :

$$\text{MontRed}(z') = \tilde{z}$$

Better : the multiplication and reduction steps can be interleaved.

Montgomery's Multiplication 2/2

Let :

- $m = (m_{n-1} \dots m_1 m_0)_b$
- $R = b^n$, $\gcd(m, b) = 1$
- $m' = -m^{-1} \mod b$

Step 3 : 2 MULT
Step 4 : $2n$ MULT
Total : $2n(n+1)$ MULT et 0 DIV

Algo MontMul

INPUT(S) : $x = (x_{n-1} \dots x_1 x_0)_b$,
$y = (y_{n-1} \dots y_1 y_0)_b$
OUTPUT(S) : $xyR^{-1} \mod m$

1: $A \leftarrow 0$
2: **for** $i = 0$ to $n-1$ **do**
3: $\quad u_i \leftarrow (a_0 + x_i y_0) m' \mod b$
4: $\quad A \leftarrow (A + x_i y + u_i m)/b$
5: **if** $A \geq m$ **then**
6: $\quad A \leftarrow A - m$
7: **return** A

To get $xy \mod m$:

- we pre-compute $R^2 \mod m$ et we compute the Montgomery representation of the inputs
- we delete the final factor R by multiplying with 1

Exponentiation in Montgomery's representation

Algo Exponentiation with MontMul

1: Compute $C = R^2 \mod m$ {pre-processing}
2: $\tilde{x} = \text{MontMul}(x, C) = xR \mod m$
3: $t = \text{MontMul}(\tilde{x}, \tilde{x}) = \tilde{x}^2 R^{-1} \mod m$
4: $u = \text{MontMul}(u, u) = \tilde{x}^4 R^{-3} \mod m$
5: $v = \text{MontMul}(v, \tilde{x}) = \tilde{x}^5 R^{-4} \mod m = x^5 R \mod m$
6: $w = \text{MontMul}(v, 1) = x^5 \mod m$

Montgomery's arithmetic is efficient for exponentiations involving large numbers.

Usage

- Montgomery arithmetic enables to efficiently chain modular multiplications.
- Other methods exist (e.g. Barrett).
- Crypto-processors usually use those representations for the modular multiplications.
- The choice of the modular exponentiation algorithm is done by the developer himself.

Introduction

Goal : process $x \leftarrow x^d$ efficiently.

Note : exponentiation in a multiplicative group corresponds to a multiplication in an additive group ; the same algorithms are involved to process the scalar multiplication of a point in a group of points in an elliptic curve (ECDSA, ECDH).

Algo Trivial Exponentiation

INPUT(S) : m, d
OUTPUT(S) : m^d

1: $t \leftarrow m$
2: **for** $i = 1$ to $d - 1$ **do**
3: $\quad t \leftarrow t \times m$
4: **return** t

Algo Additive Version

INPUT(S) : G, d
OUTPUT(S) : dG

1: $T \leftarrow G$
2: **for** $i = 1$ to $d - 1$ **do**
3: $\quad T \leftarrow T + G$
4: **return** T

Operations : $d - 1$ MULT/ADD
Memory : t/T

Square-and-Multiply

Observation :

$$d = \sum_{i=0}^{i=n} d_i 2^i = d_0 + 2(d_1 + 2(d_2 + \ldots + 2(d_{n-1} + 2(d_n))) \ldots)$$

$$m^d = m^{d_0} \times (m^{d_1} \times (m^{d_2} \times \ldots (m^{d_{n-1}} \times (m^{d_n})^2)^2 \ldots)^2$$

Square-and-Multiply

Algo S&M binary from left to right

INPUT(S) : $m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t \leftarrow 1$
2: **for** $i = n$ to 0 **do**
3: $\quad t \leftarrow t \times t$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad t \leftarrow t \times m$
6: **return** t

Operations :

- n SQ
- $HW(d) - 1$ MULT.

Memory : m, t

Example : m^{283}

$$283 = (100011011)_2$$

d		1	0	0	0	1	1	0	1	1
t	1	m	m^2	m^4	m^8	m^{17}	m^{35}	m^{70}	m^{141}	m^{283}

AES : Brief History

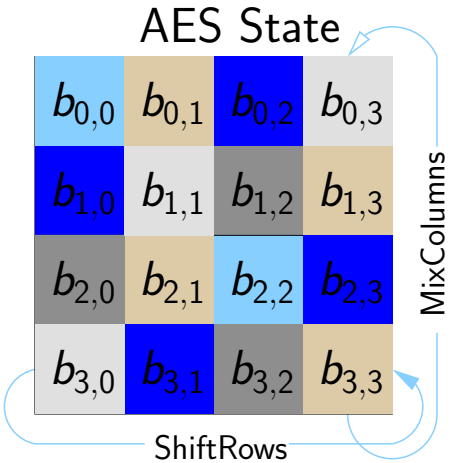
- Advanced Encryption Standard is the result of an international competition started in 1997.
- The purpose was to replace the DES standard since its key size was became too small.
- Some of the new requirements ware : a block size equal to 128 bits (or 256 bits) and a scalable key size with possible values 128, 192 or 256.
- Among the 15 last candidates, the proposal of the Belgium researchers Daemen and Rijmen has been selected. It was called RIJNDAEL.

AES : Short Description

- Iterative block cipher (but not based on a Feistel structure).
- Substitution and Permutation Network (SPN).
  1. the ciphertext $Y(i)$ produced by the previous round is modified thanks to a non-linear substitution operation (that ensures data confusion).
  2. Then a linear permutation is applied (that ensures data diffusion).
  3. Eventually, the round key is bit-wisely added to produce $Y(i+1)$.
- The number of rounds is 10 for a 128-bits key and 14 for a 256-bits key.

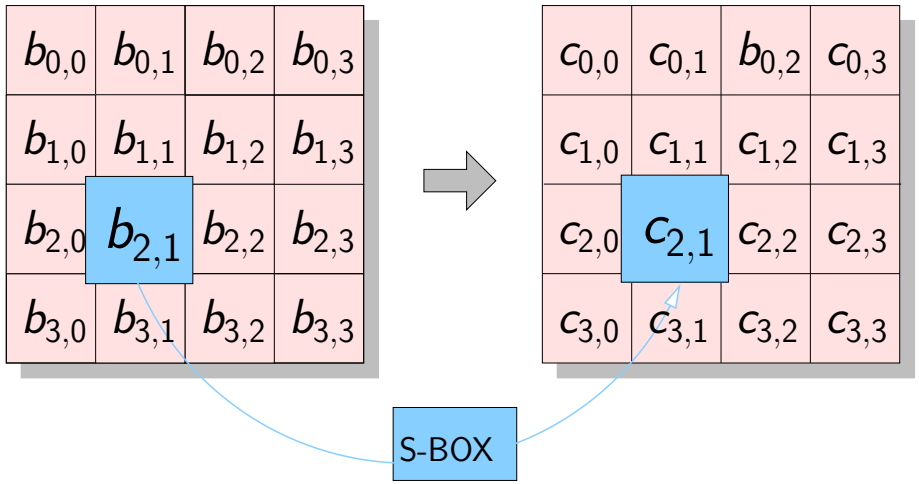
AES : details

- AES is an iterative block cipher with key sizes 128, 192 or 256 bits.
- Based on a Permutation/Substitution network. Each round is composed of :
  - A byte substitution (SubBytes).
  - A shifting of the bytes (ShiftRows)
  - A matrix product (MixColumns).
  - A bitwise addition with the round key (AddRoundKey).



AES

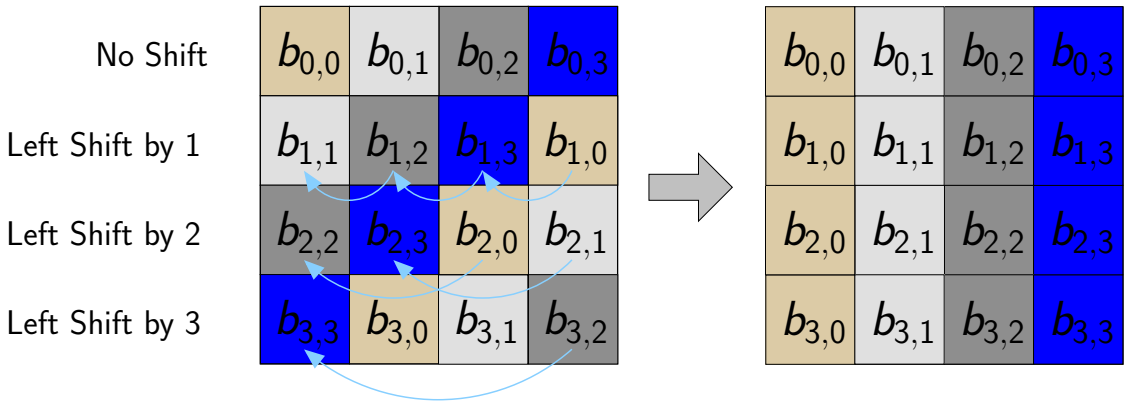
SubBytes



- Each byte is replaced by a value stored in a table of 256 bytes. This table is called s-box.

AES

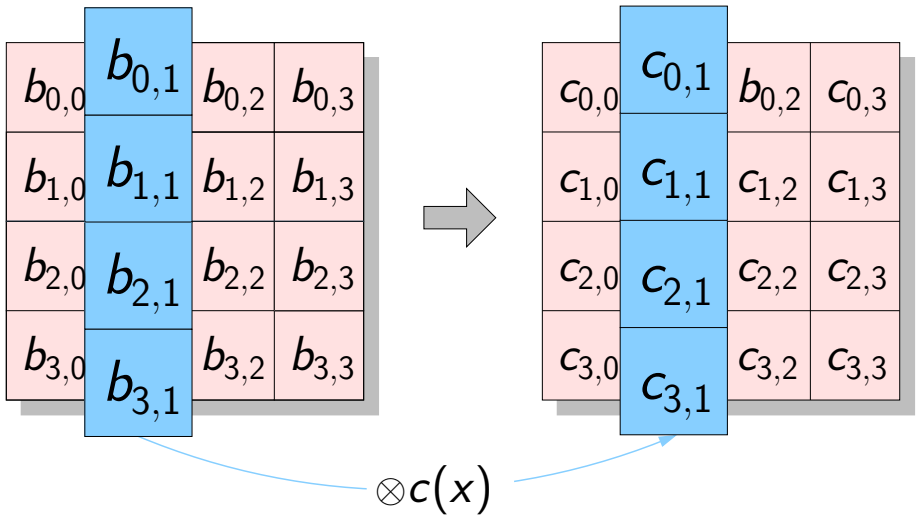
ShiftRows



- The bytes of each row are permuted. The permutation is simply left cycling.

AES

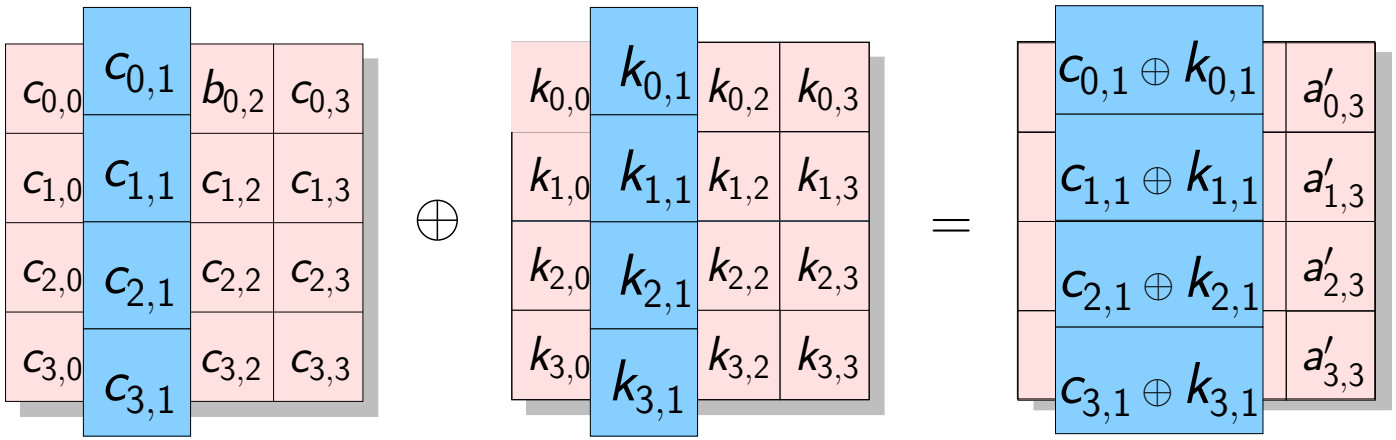
MixColumns



- The columns of the state are considered as polynomials over GF(256)
- Each column is multiplied by the polynomial defined on GF(256) by $c(X) = 3X^3 + X^2 + X \mod X^4 + 1$.
- For the operations on $GF(256)$, the representation $GF(2)[X]/(X^8 + X^4 + X^3 + X + 1)$ is used.

AES

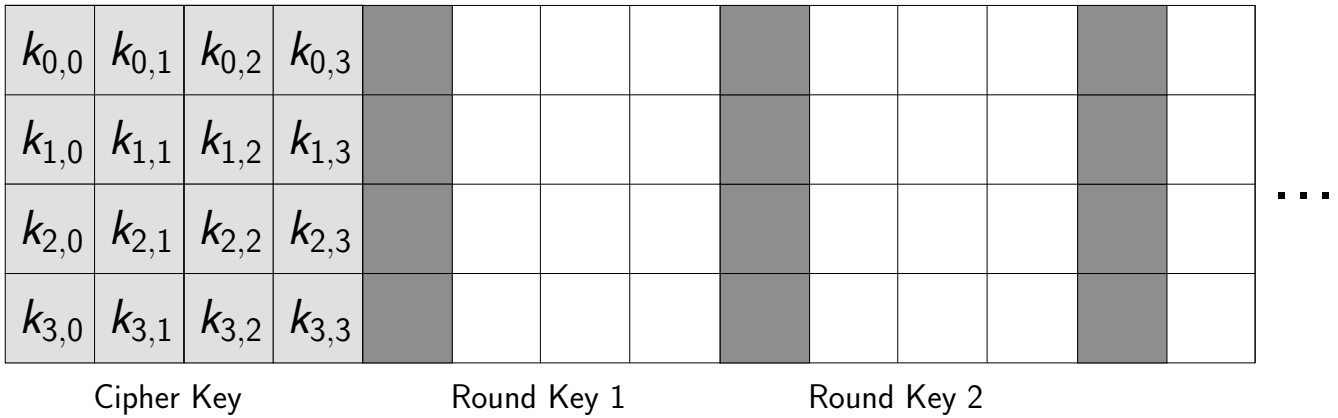
AddRoundKey



- The round key is bitwisely added.
- The four previous steps are repeated 10 times (for a 128-bits key).
- For the last round, the Mixcolumns step is not performed.

AES

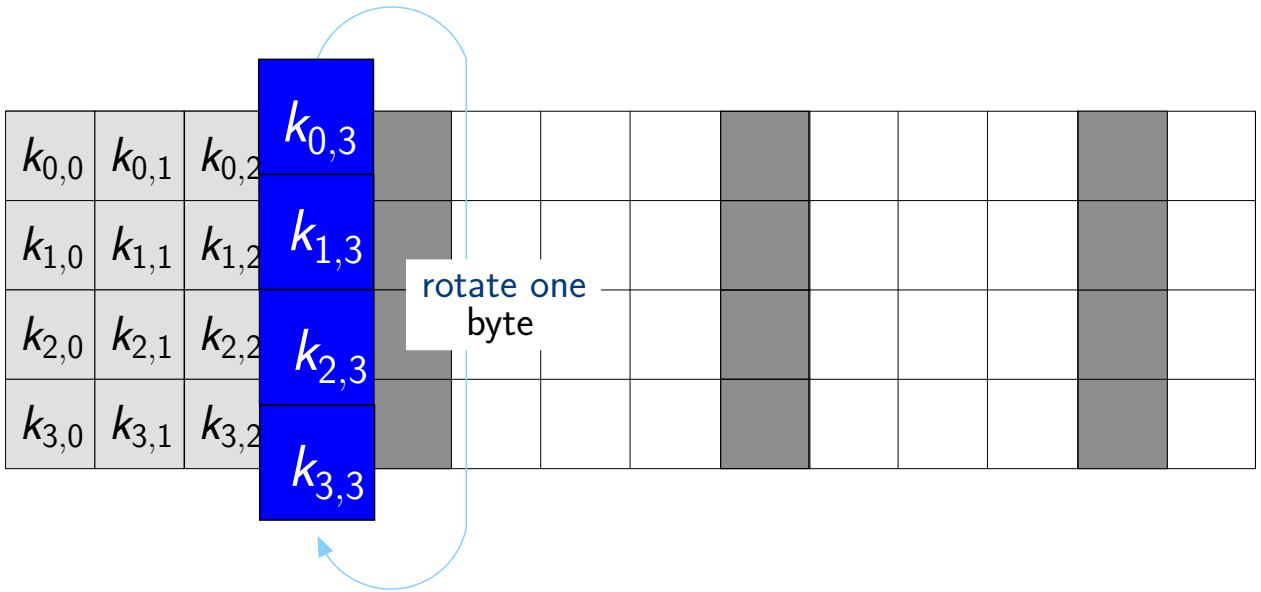
Round-key Generation



- The master key is used to derive the 10 round keys.
- The round-key generation process is built on a SPN that operates on the columns of the key state.
- Each column of index n, $n \geq 4$, is a function of the columns of indices $n-1$, ..., $n-4$.
- The first column of each round-key is the result a particular treatment.

AES

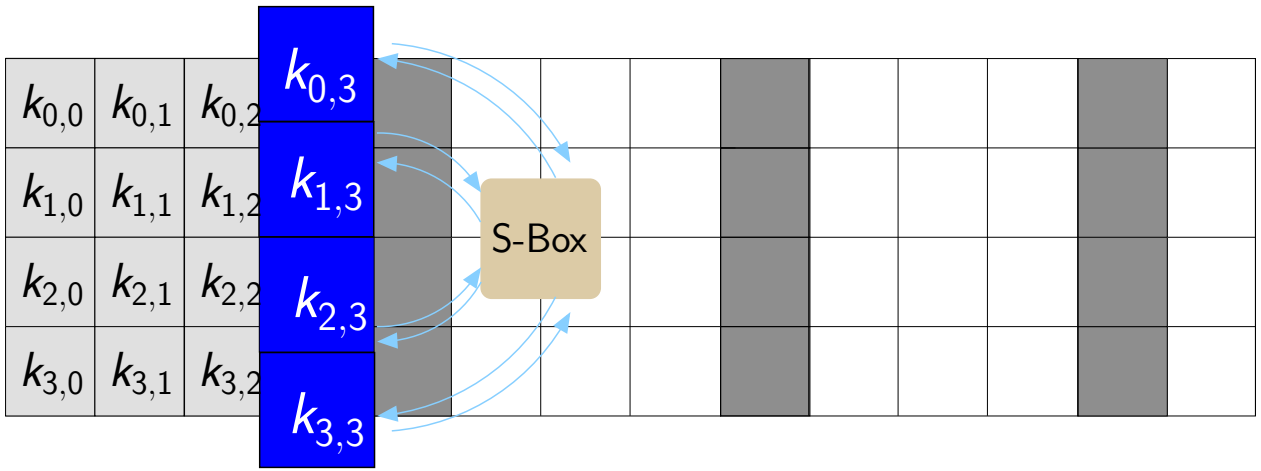
Round-key Generation



- The bytes of the last column are shifted.

AES

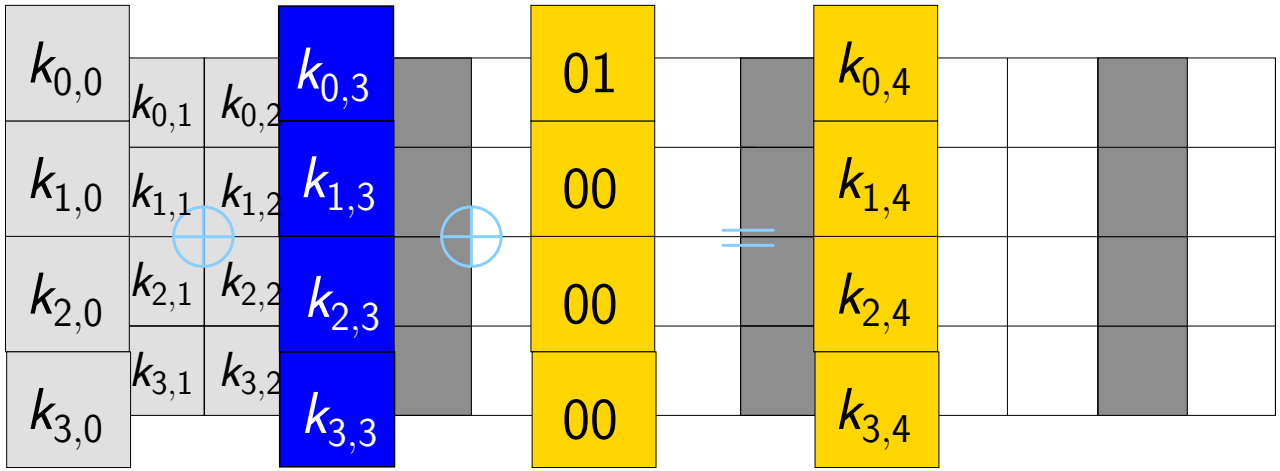
Round-key Generation

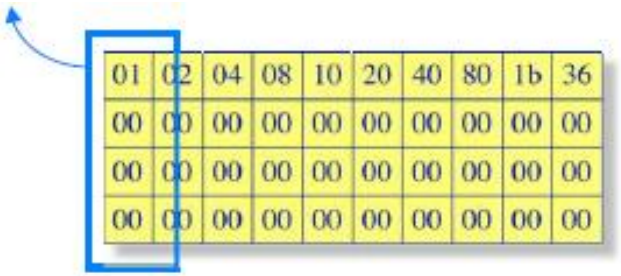


- Each byte is replaced by a value in the s-box.

AES

Round-key Generation

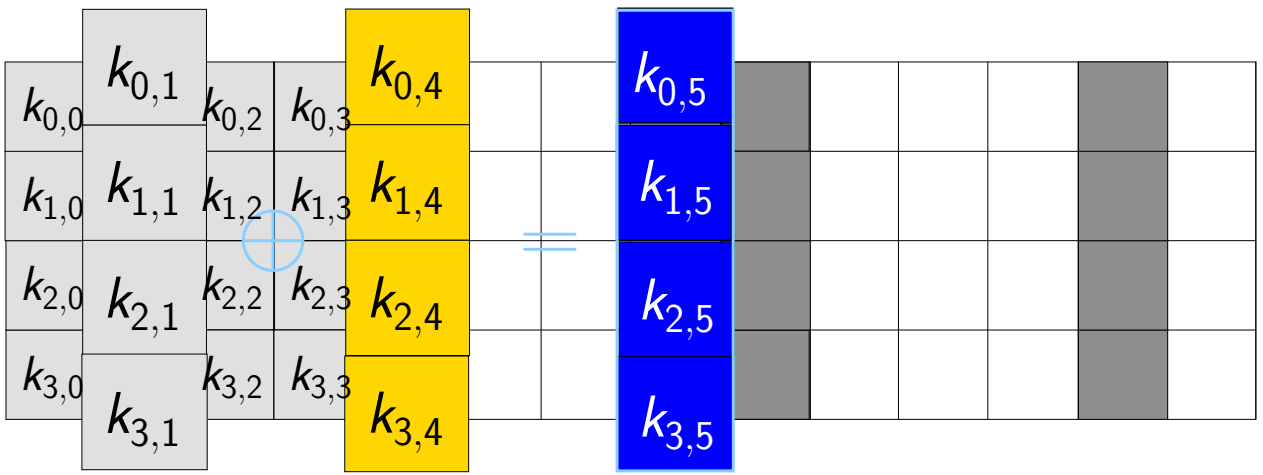




- The result is bitwisely added with the first column and ...
- ...a particular constant column is added.

AES

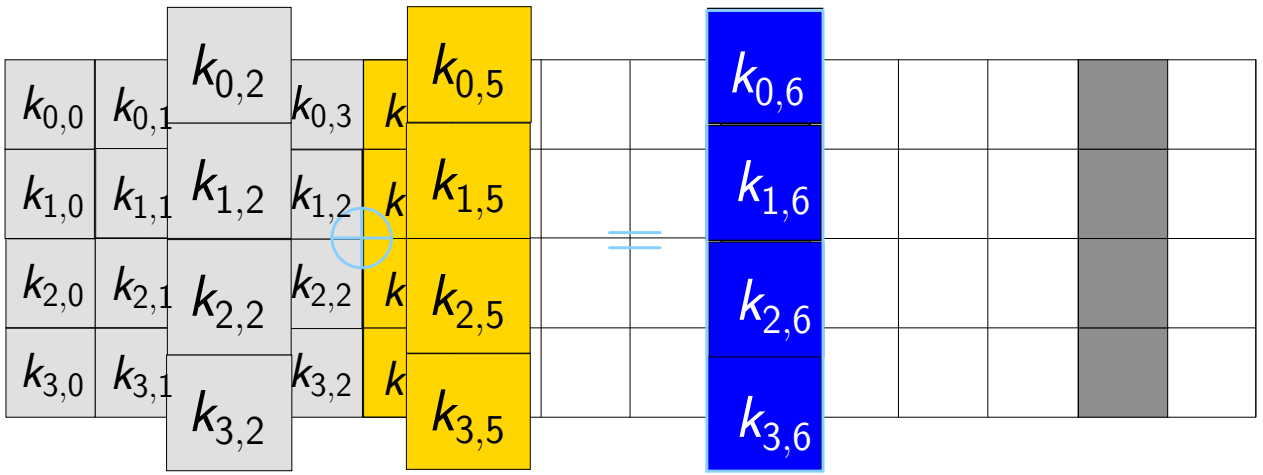
Round-key Generation



- The columns 2 to 4 are simple linear combinations of the columns of indices $n-1$ and $n-4$.

AES

Round-key Generation



- The columns 2 to 4 are simple linear combinations of the columns of indices $n-1$ and $n-4$.
- The process is repeated for each round.

AES

MixColumns Implementation

- Goal : reduce the number of field multiplications.
- Observation : on GF(256) we have $3 = 1 + 2$.
- Consequence : Mixcolumns can be processed with only additions and multiplications by 2 over $\mathrm{GF}(2)[X]/(X^8 + X^4 + X^3 + X + 1)$.

Algo Efficient Implementation of MixColumns

INPUT(S) : A column $a = (a[0], a[1], a[2], a[3])$
OUTPUT(S) : A new column $a = (a[0], a[1], a[2], a[3]) = a(X) \otimes c(X)$

1: $t = a[0] + a[1] + a[2] + a[3]$; {Pre-computation}
2: $u = a[0]$;
3: $v = a[0] + a[1]$; $v = \mathtt{xtime}(v)$; $a[0] = a[0] + v + t$;
4: $v = a[1] + a[2]$; $v = \mathtt{xtime}(v)$; $a[1] = a[1] + v + t$;
5: $v = a[2] + a[3]$; $v = \mathtt{xtime}(v)$; $a[2] = a[2] + v + t$;
6: $v = a[3] + u$; $v = \mathtt{xtime}(v)$; $a[3] = a[3] + v + t$;
7: **return** a

AES

`xtime` processing – Daemen and Rijmen 2002, DR02

Process $v \leftarrow \texttt{xtime}(v)$.

- First alternative :
  1. Represent v as a polynomial $v(X) = v_7X^7+v_6X^6+v_5X^5+v_4X^4+v_3X^3+v_2X^2+v_1X^1+v_0X^0$.
  2. $\texttt{xtime}(v) = X \otimes v(X) \mod X^8 + X^4 + X^3 + X + 1$.
  3. if $v_7 = 0$: $X \otimes v(X) \mod X^8 + X^4 + X^3 + X + 1 = v_6X^7 + v_5X^6 + v_4X^5 + v_3X^4 + v_2X^3 + v_1X^2 + v_0X^1$. A simple left-shift !
  4. if $v_7 = 1$: $X \otimes v(X) \mod X^8 + X^4 + X^3 + X + 1 = (v_6X^7 + v_5X^6 + v_4X^5 + v_3X^4 + v_2X^3 + v_1X^2 + v_0X^1) + (X^4 + X^3 + X + 1)$. A simple left-shift followed by the addition of a constant term !
- Second alternative : tabulate the function $v \mapsto \texttt{xtime}(v)$ in ROM.

AES

Inverse of MixColumns Processing

The matrix

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

is replaced by (it inverse over GF(256)) :

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

whose direct processing is more costly.

AES

Inverse of MixColumns Processing

Fortunately (P. Barreto), we have :

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} + \begin{pmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{pmatrix}$$

To process the right-hand matrix product :

Algo InvMixColumns pre-processing

INPUT(S) : A column $a = (a[0], a[1], a[2], a[3])$
OUTPUT(S) : A new column $a = (a[0], a[1], a[2], a[3])$

1: $u = \texttt{xtime}(\texttt{xtime}(a[0] + a[2]))$;
2: $v = \texttt{xtime}(\texttt{xtime}(a[1] + a[3]))$;
3: $a[0] = a[0] + u$;
4: $a[1] = a[1] + v$;
5: $a[2] = a[2] + u$;
6: $a[3] = a[3] + v$;
7: **return** a

AES

s-box Processing

Goal : processing of the function $v \in \mathrm{GF}(2)^8 \mapsto S(v)$.
Notation : the polynomial representation of a byte v is denoted by $v(X)$.
Several options are possible with different timing/memory trade-offs.

- [Best Timing Perfs] tabulate the function S in ROM.
- [Best Memory Perfs] process $S(x)$ without tables.
  - Function $v \mapsto S(v)$ is the left composition of a GF(2)-affine transformation A with the power function $Inv : v \mapsto v^{254}$ defined over $\mathrm{GF}(2)[X]/(X^8 + X^4 + X^3 + X + 1)$.
  - The affine function A is defined by

$$A(v) = v(X) \times (X^4 + X^3 + X^2 + X + 1) + (X^6 + X^5 + X + 1) \mod X^8 + 1$$

AES

s-box Processing with Tower Fields Approach

- Field additions are simple bitwise additions (XOR $\oplus$).
- Field multiplications cannot be directly tabulated : require a 256^2 bytes of ROM.
- Idea : Tower Fields Approach e.g. Paar and Rosner, PR97.
  - Every element of GF(2^8) can be mapped by a linear transformation to on an element of GF(2^4)^2, i.e. a polynomial of degree 1 with coefficients in GF(2^4).
  - To define multiplication over GF(2^4)^2, we use a polynomial of the form $X^2 + X + A$ where A is chosen to optimize the Hardware performances (e.g. $A = 0\text{x}E$).
  - If needed, one can still represent elements of GF(2^4) as elements of GF(2^2)^4, etc.
- Operations over sub-fields can be tabulated or directly processed.

AES

s-box Processing with Tower Fields Approach

Select a polynomial $P(X) = X^2 + X + p_0$ irreducible over GF(16) (e.g. choose $p_0 = 0x0E$) Rudra *et al* at CHES 2001.

Algo Inv processing

INPUT(S) : A byte v, field isomorphisms GF256ToGF16 and GF256ToGF16^{-1}
OUTPUT(S) : A byte $v \leftarrow Inv(v)$.

1: $(v_H, v_L) = \text{GF256ToGF16}(v)$; {Go to GF(16)^2}
2: $d = (v_H^2 \times p_0) + (v_H \times v_L) + v_L^2$
3: $d' = d^{-1}$; {inversion in GF(16)}
4: $v_H = v_H \times d'$
5: $v_L = (v_H + v_L) \times d'$
6: $v = \text{GF16ToGF256}(v_H, V_L)$; {Come back to GF(256)}
7: **return** a

If needed, one can still go to GF(4)^2 to process the multiplications over GF(16) (note that inversion is linear over GF(4)).

AES

s-box Processing with Addition Chain Approach

- Idea : process the exponentiation $v \mapsto v^{254}$ with multiplications and squarings only.
- Squarings can be tabulated and multiplications can be done either directly or by applying the Tower Fields Approach.

Algo Inv processing With Chain of Additions

INPUT(S) : A byte v
OUTPUT(S) : A byte $y \leftarrow Inv(v)$.

1: $z = v^2$; {squaring}
2: $y = z \times v$;
3: $w = y^4$; {squaring}
4: $y = y \times w$;
5: $y = y^{16}$; {squaring}
6: $y = y \times w$;
7: $y = y \times z$;
8: **return** y ;

Troisième partie III

Passive Side Channel Attacks

Outline

Side Channel Leakage

The behaviour of a smart card strongly depends on (1) the kind of executed operations and (2) the kind of operands that are processed. Among side channels we have :

- the execution timing,
- the power consumption,
- the electromagnetic emanations,
- the sound,
- ...

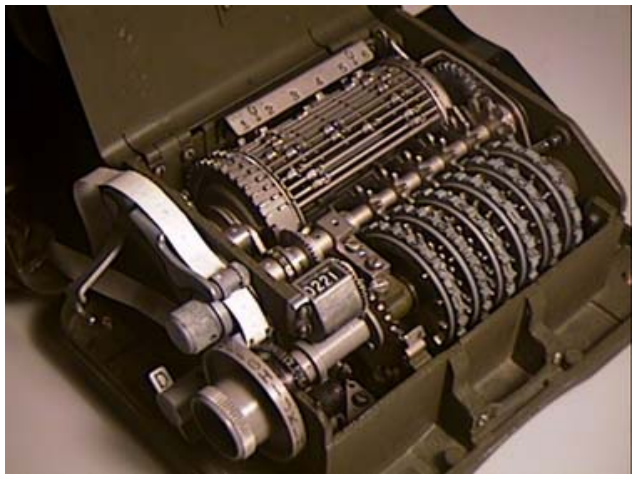
They are not taken into account by the "classical" cryptanalyses. Side channel attacks (SCA) are passives or actives. In a SCA, the adversary can :

- *observe* the behaviour of the device during the processing,
- [active SCA only] modify them.

A brief History

1956 : MI5 breaks the encryption system used by the Egyptian Embassy at London.

- Encryption system : machine with rotors from Hagelin family.
- Peter Wright suggest to de placer un micro dans la salle du chiffre.
- The sound produced by the machine when the 7 rotors are initialized indeed gives information on its initial state, then enabling to get the key and to decipher.



Exploitation : Simple Attacks (SPA)

SPA refers to attacks where the adversary focus on a single execution of an implementation (with possibility to average the observation for fixed inputs).
In some cases, this gives the adversary information about the manipulated secrets.

- The information leakage must be important.
- The secret must have a simple relationship with the leakage.

Example of SPA ; PIN verification

Algo PIN comparison

INPUT(S) : SPIN, PIN
OUTPUT(S) : `ok/nok`

1: **for** $i = 0$ to 4 **do**
2: **if** SPIN[i] $\neq$ PIN[i] **then**
3: **return** `nok`
4: **return** `ok`

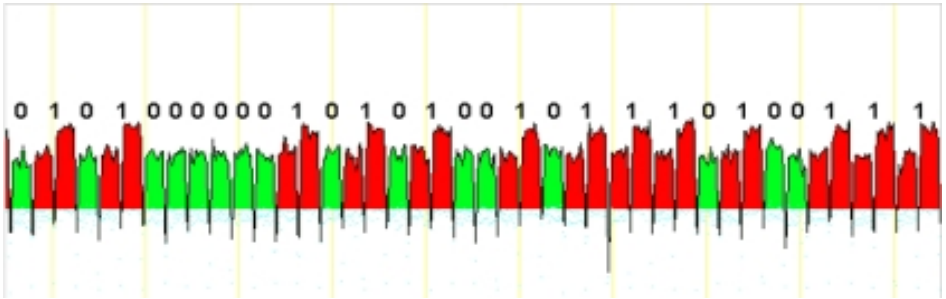
The observation of execution timing enables to retrieve PIN with 4×10 tries instead of $10^4 = 10\,000$.

Attack against the Square-and-Multiply Algorithm

Algo S&M binary from left to right

INPUT(S) :
$m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t \leftarrow 1$
2: **for** $i = n$ to 0 **do**
3: $\quad t \leftarrow t \times t$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad t \leftarrow t \times m$
6: **return** t



Remark/Assumption : squaring and multiplication have different leakage patterns.

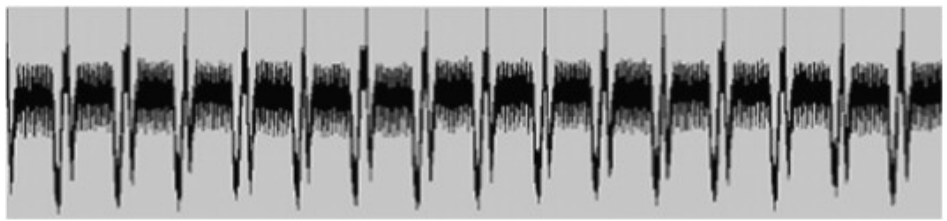
Square-and-Multiply Always

- The leakage is due to the fact that a part of the processed code depends on the secret value.
- Countermeasure : makes the code execution flaw independent of the secret parameters.

Algo S&M Always

INPUT(S) :
$m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t_0 \leftarrow 1$
2: **for** $i = n$ to 0 **do**
3: $t_0 \leftarrow t_0 \times t_0$
4: $t_1 \leftarrow t_0 \times m$
5: $t_0 \leftarrow t_{d_i}$
6: **return** t



Principle : add dummy operations. . .

The *Doubling Attack* [**?**] 1/2

- Chosen plaintexts attack against the Square-and-Multiply Always.
- Idea : exponentiations m^d and $(m^2)^d$ share intermediate results.
- Attack Assumption (realistic) : by observing the computations of A^2 and B^2, the adversary can decide whether A equals B or not.

The *Doubling Attack* [?] 2/2

Algo S&M Always

INPUT(S) :
$m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t_0 \leftarrow 1$
2: **for** $i = n$ to 0 **do**
3: $t_0 \leftarrow t_0 \times t_0$
4: $t_1 \leftarrow t_0 \times m$
5: $t_0 \leftarrow t_{d_i}$
6: **return** t

Example :
$d = 78 = (1001110)_2$.

$SQ(m, i) = SQ(m^2, i-1) \Leftrightarrow d_{i-1} = 0$

i	d_i	process m^d	process $(m^2)^d$
6	1	2×1 $1 \times m$	2×1 $1 \times m^2$
5	0	m^2 $m^2 \times m$	$(m^2)^2$ $(m^2)^2$ $m^4 \times m^2$
4	0	$(m^2)^2$ $(m^2)^2$ $m^4 \times m$	$(m^4)^2$ $(m^4)^2$ $m^8 \times m^2$
3	1	$(m^4)^2$ $(m^4)^2$ $m^8 \times m$	$(m^8)^2$ $m^{16} \times m^2$
2	1	$(m^9)^2$ $m^{18} \times m$	$(m^{18})^2$ $m^{36} \times m^2$
1	1	$(m^{19})^2$ $m^{38} \times m$	$(m^{38})^2$ $m^{76} \times m^2$
0	0	$(m^{39})^2$ $m^{78} \times m$	$(m^{78})^2$ $m^{156} \times m^2$
		return m^{78}	m^{156}

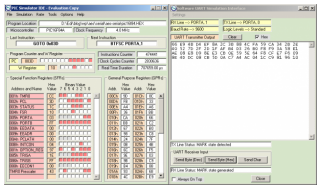
Advanced Side Channel Attacks (DPA like attacks)

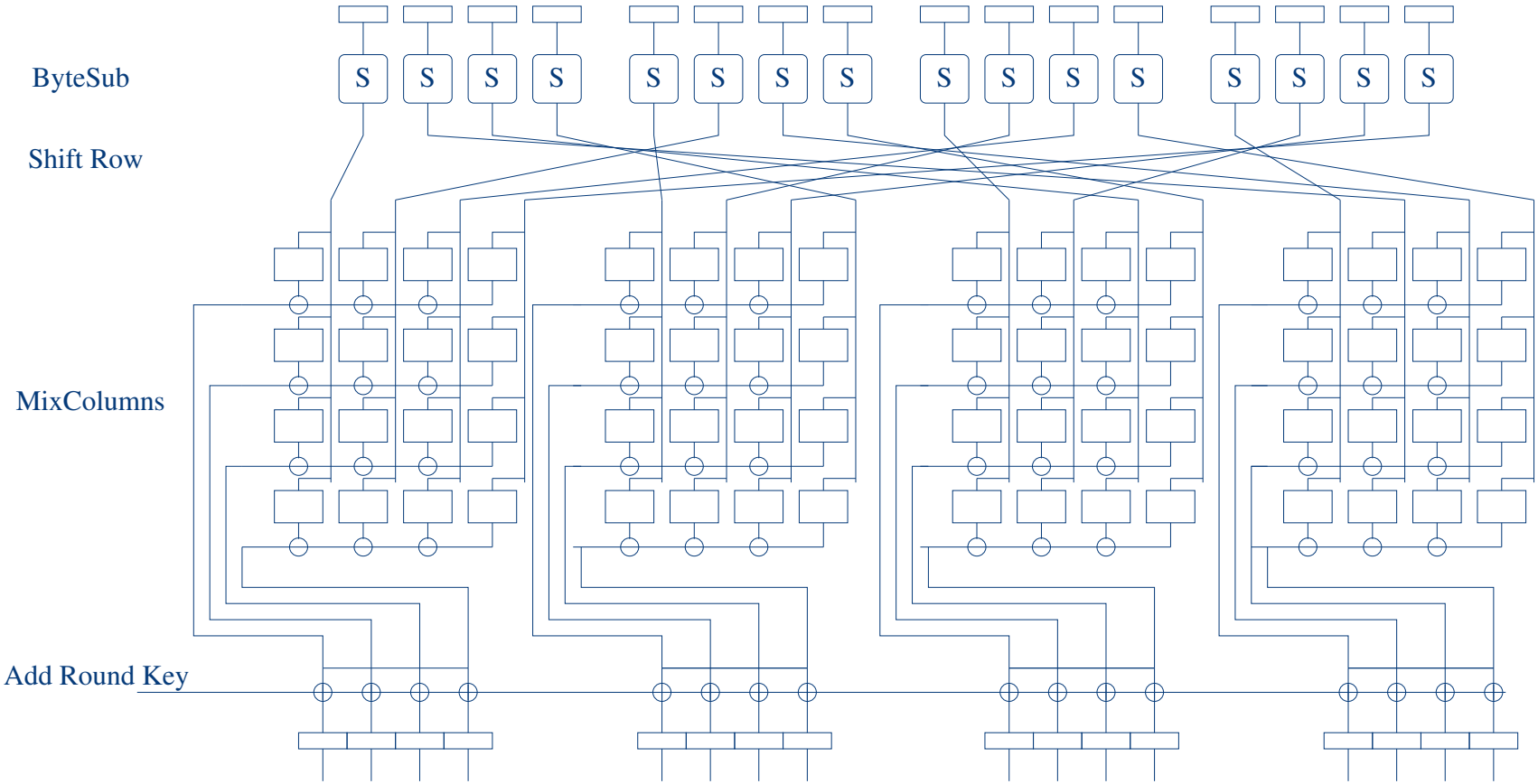
Introduction

- Advanced Side Channel Attacks can extract information from observations in contexts where SPA fails.
- They involve statistical tools (simple – difference of means tests – or sophisticated – mutual information processing –).
- They need several (between 10 and more than 10^6) traces such that :
  - the secret is constant,
  - the inputs are different and [optional] known.
  - [optional] some knowledge about the device architecture, the implementation or the noise characteristics.
- They follow a divide-and-conquer approach : the secret is rebuild piece by piece, where each piece is deduced from the behavior of an intermediate result. The size of the piece usually depends on the architecture size (*e.g.* 8, 16 or 32 bits).

Advanced Side Channel Attacks (DPA like attacks)

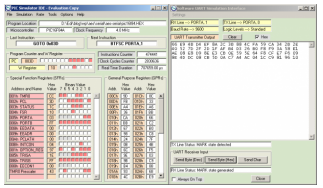
AES Round - 8-bit Software Implementation

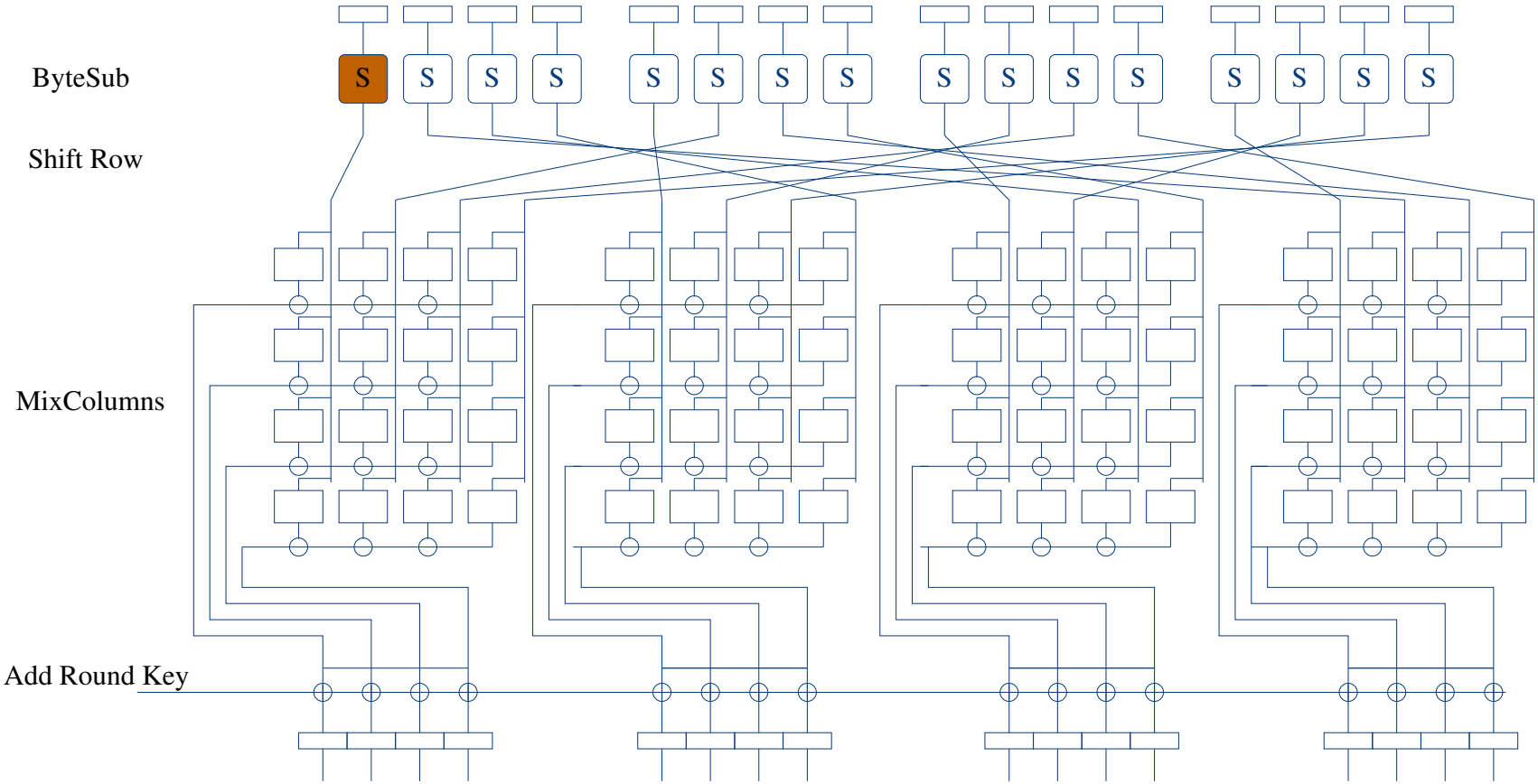




Advanced Side Channel Attacks (DPA like attacks)

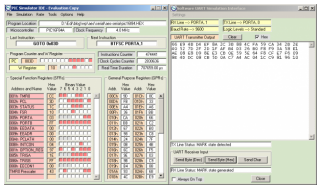
AES Round - 8-bit Software Implementation

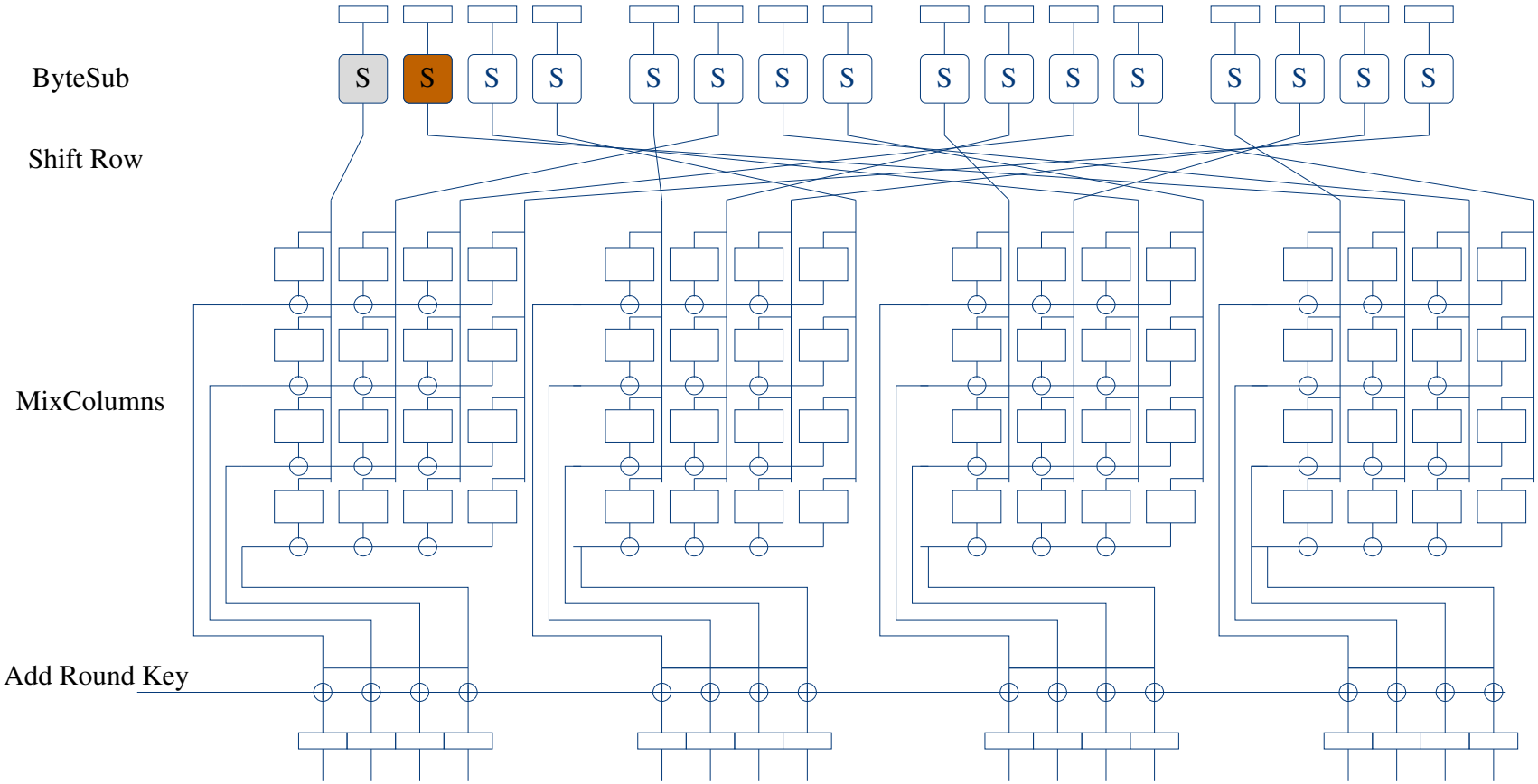




Advanced Side Channel Attacks (DPA like attacks)

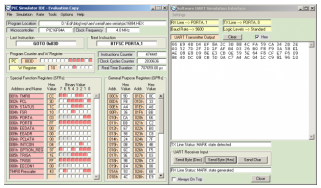
AES Round - 8-bit Software Implementation

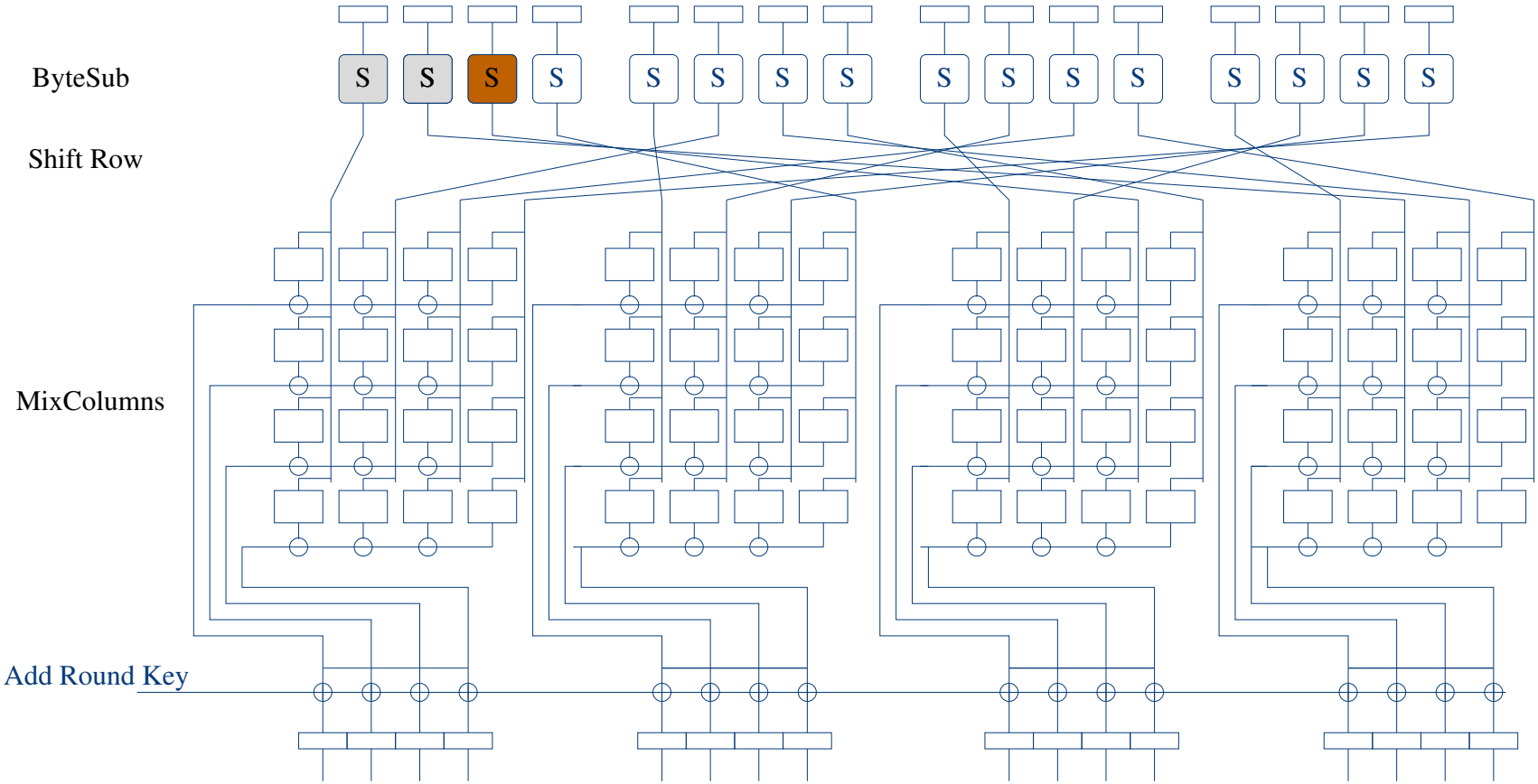




Advanced Side Channel Attacks (DPA like attacks)

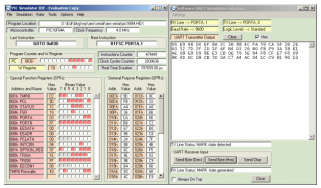
AES Round - 8-bit Software Implementation

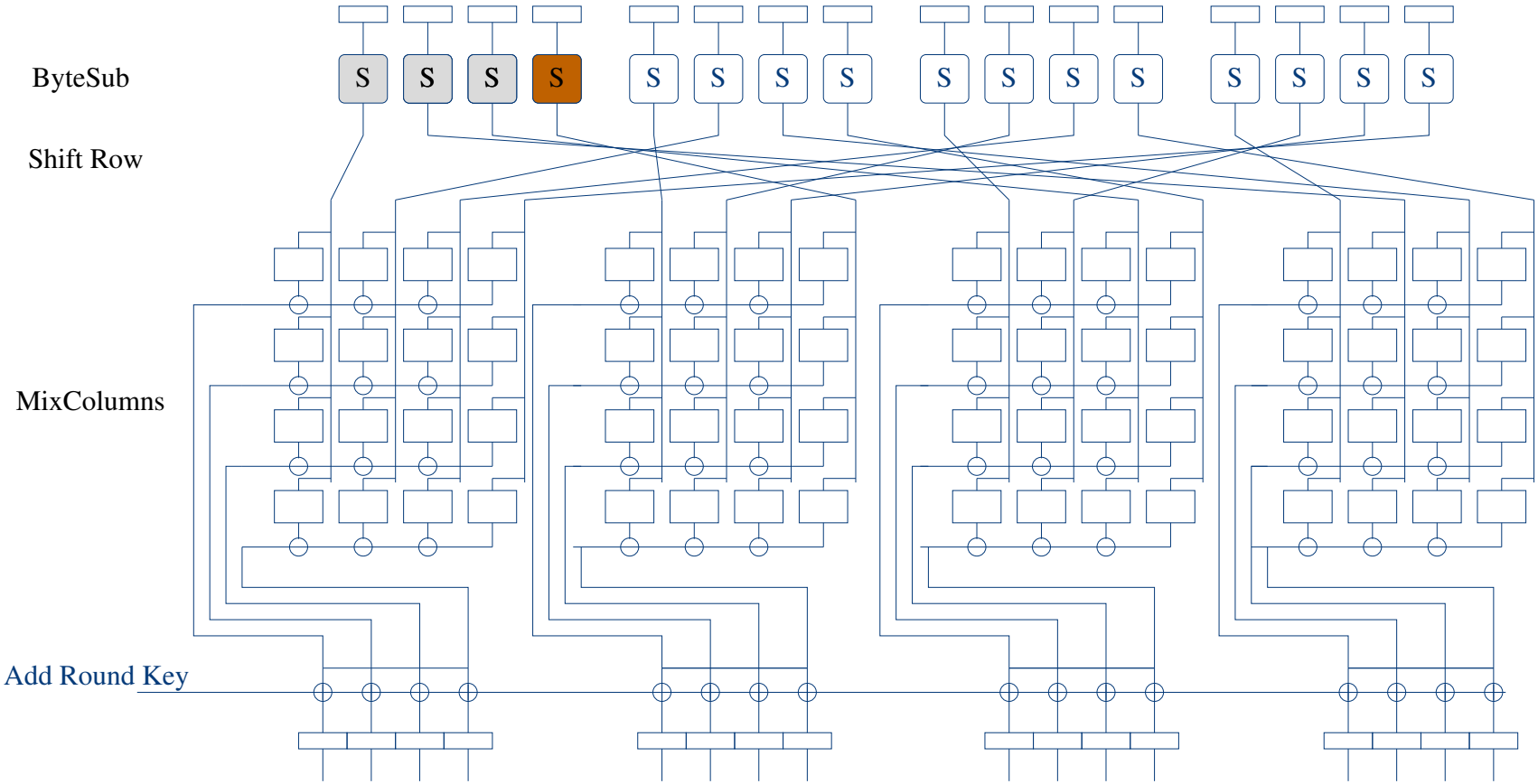




Advanced Side Channel Attacks (DPA like attacks)

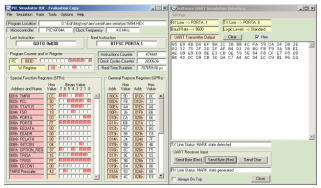
AES Round - 8-bit Software Implementation

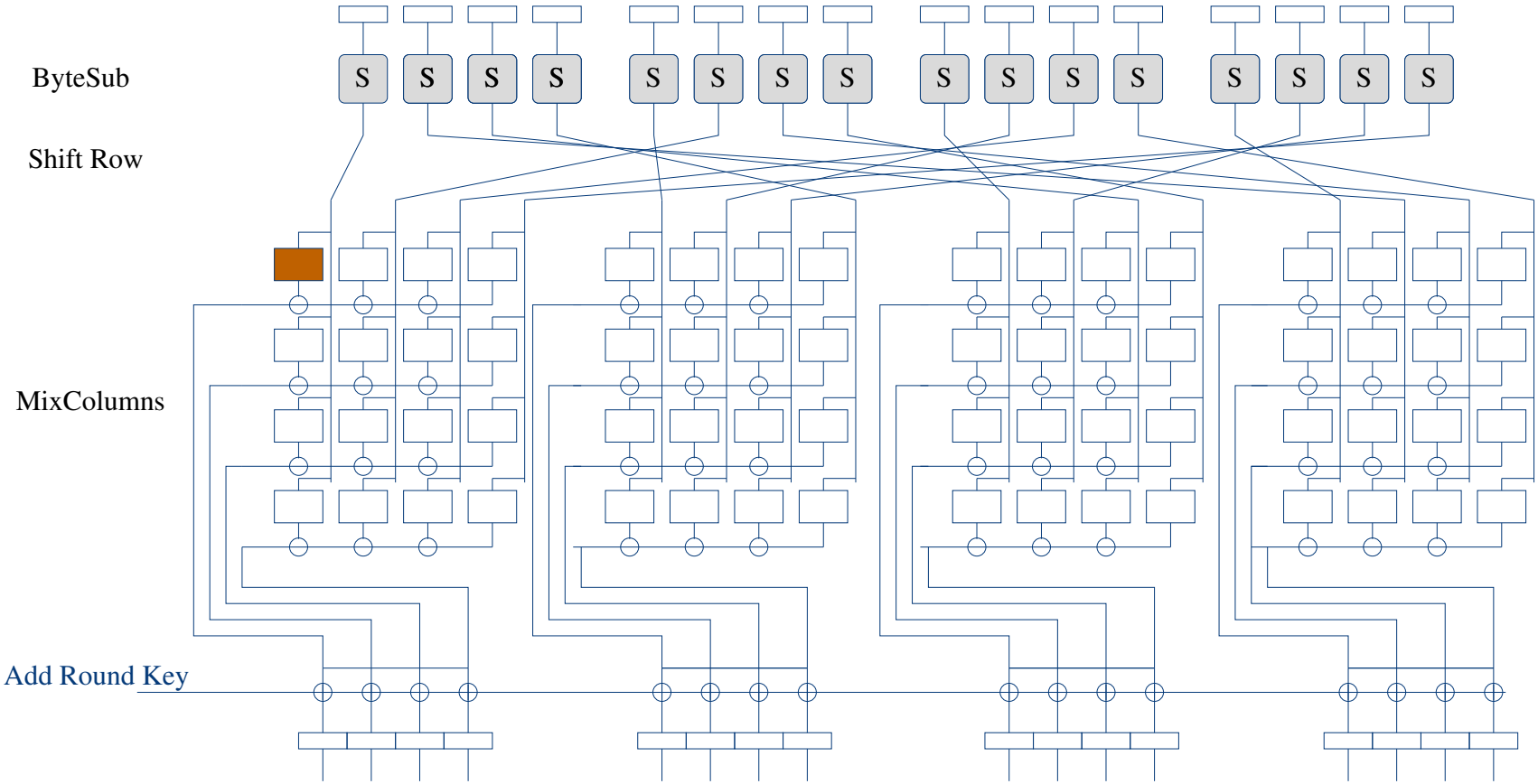




Advanced Side Channel Attacks (DPA like attacks)

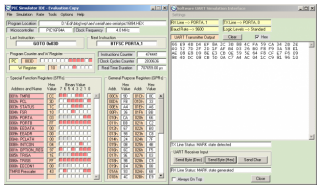
AES Round - 8-bit Software Implementation

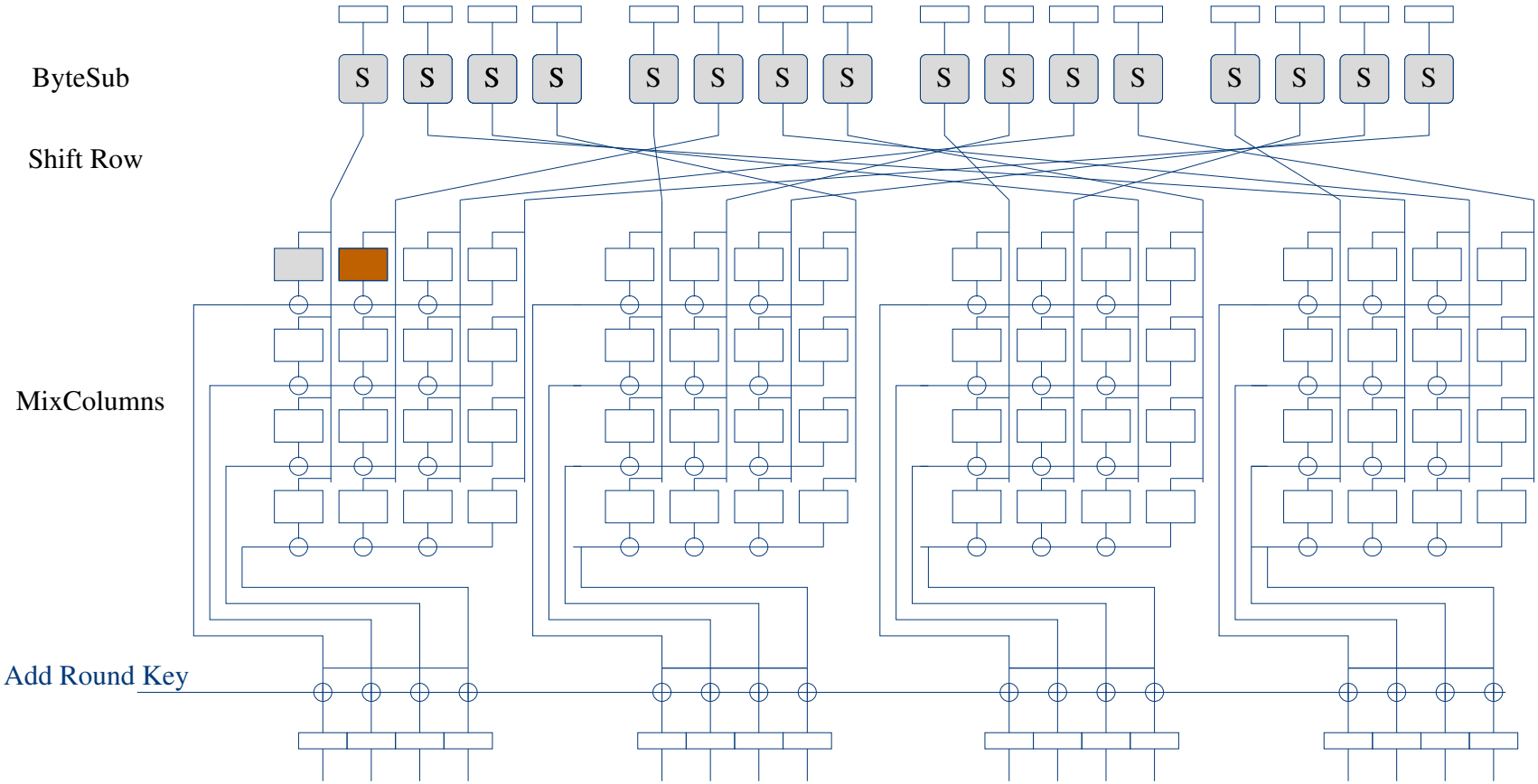




Advanced Side Channel Attacks (DPA like attacks)

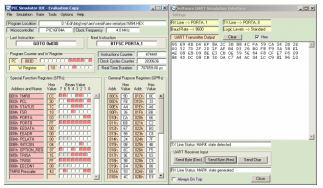
AES Round - 8-bit Software Implementation

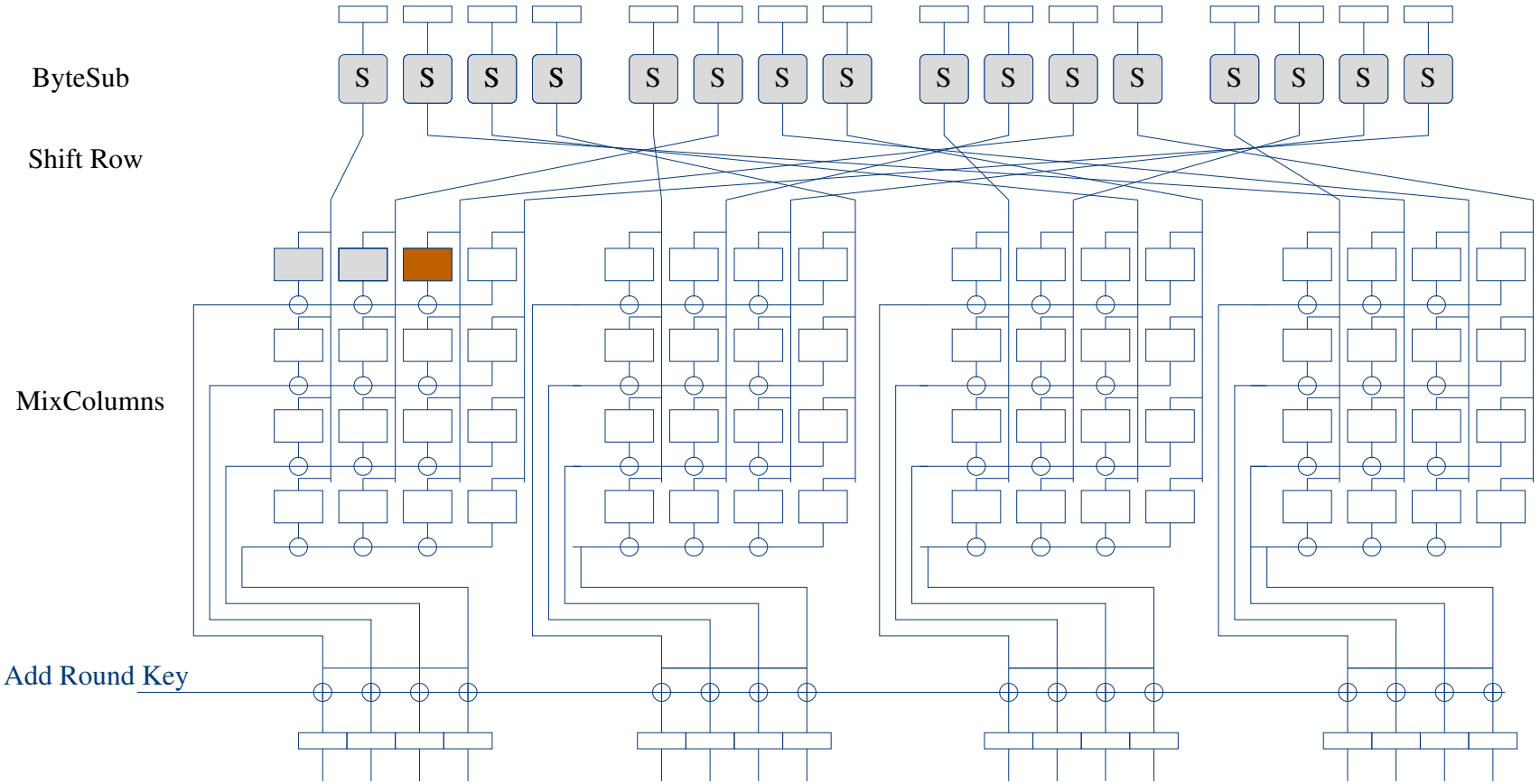




Advanced Side Channel Attacks (DPA like attacks)

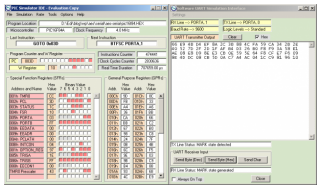
AES Round - 8-bit Software Implementation

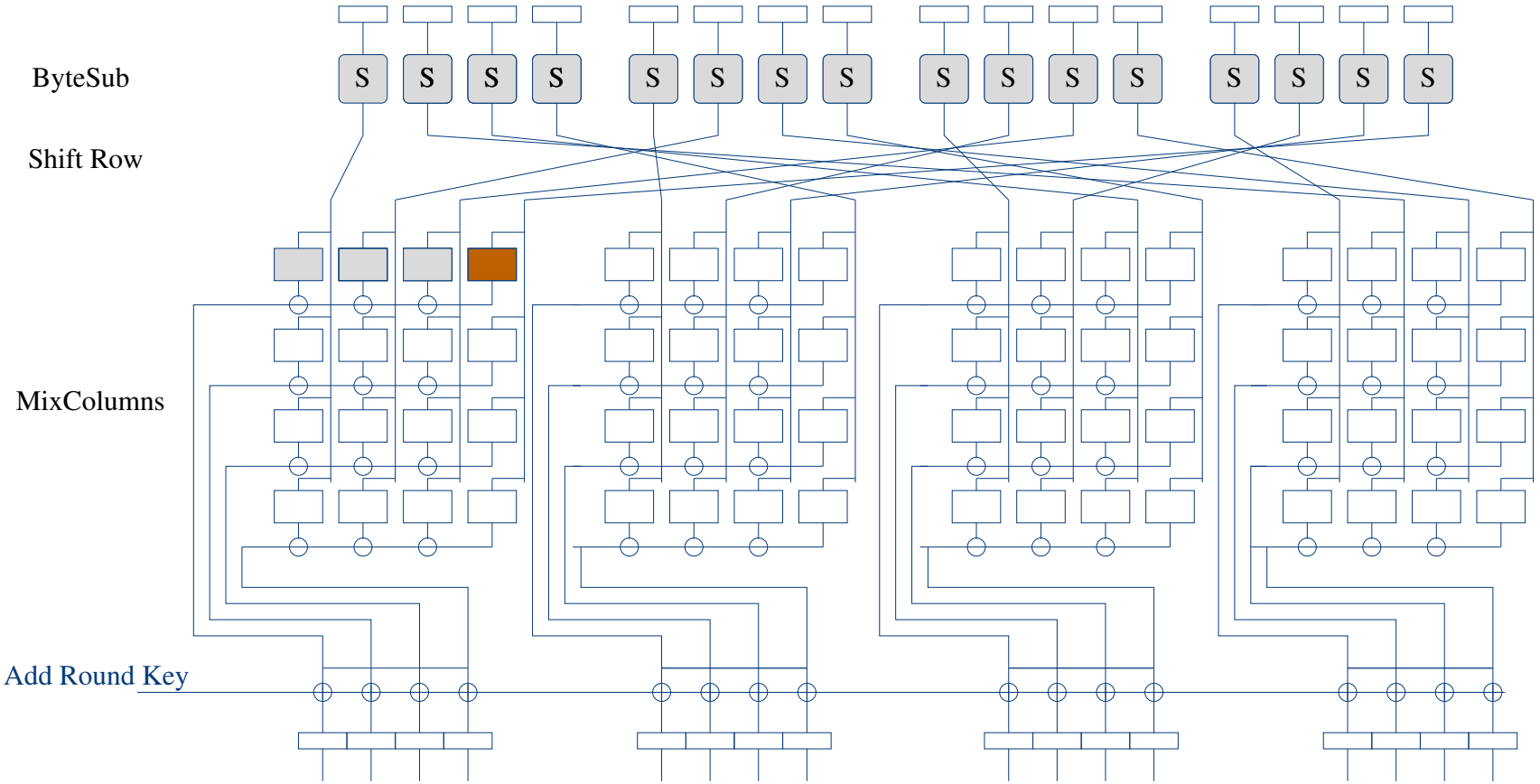




Advanced Side Channel Attacks (DPA like attacks)

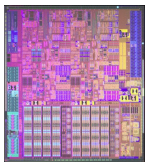
AES Round - 8-bit Software Implementation

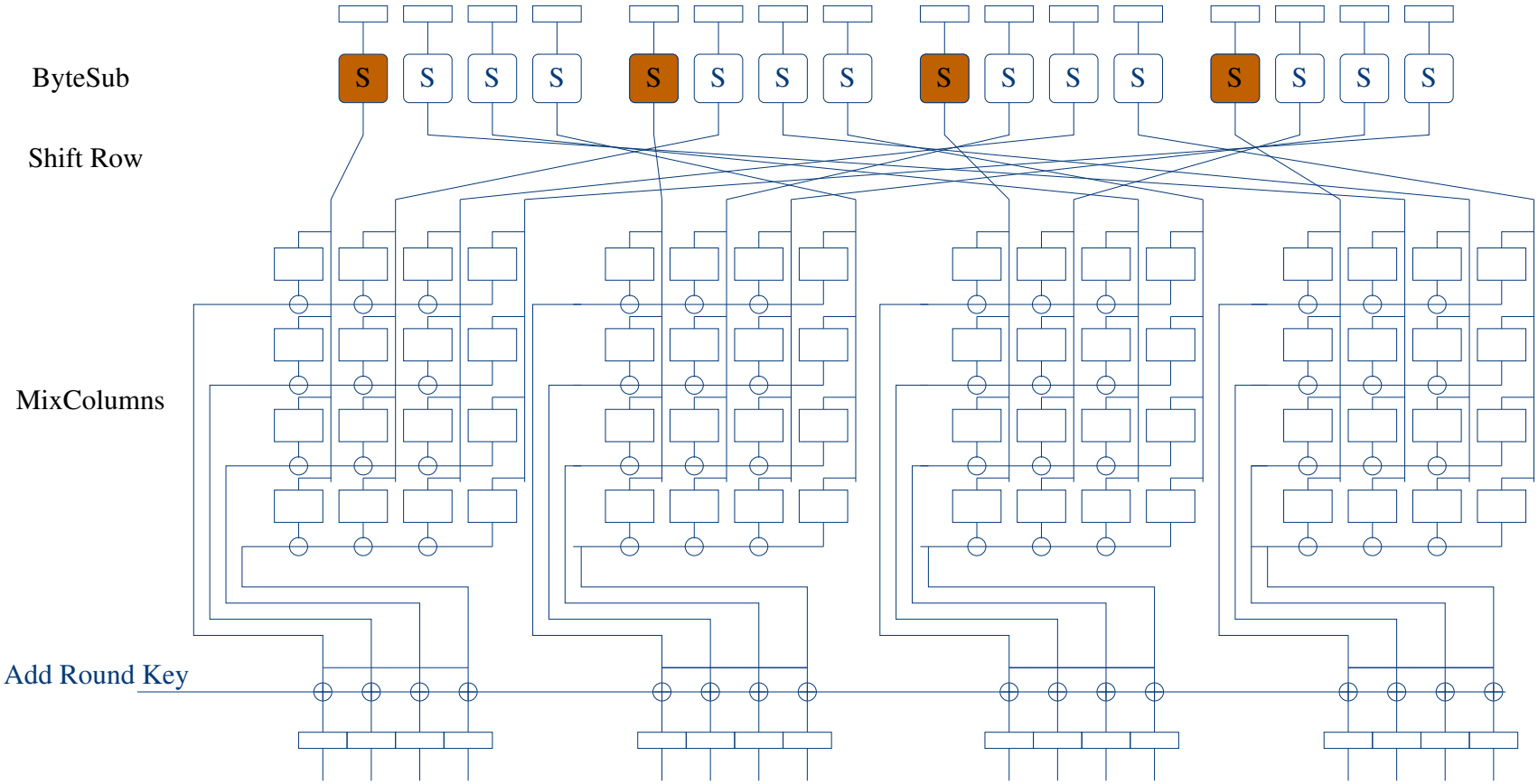




Advanced Side Channel Attacks (DPA like attacks)

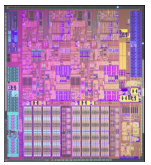
AES Round - Parallel Hardware Implementation

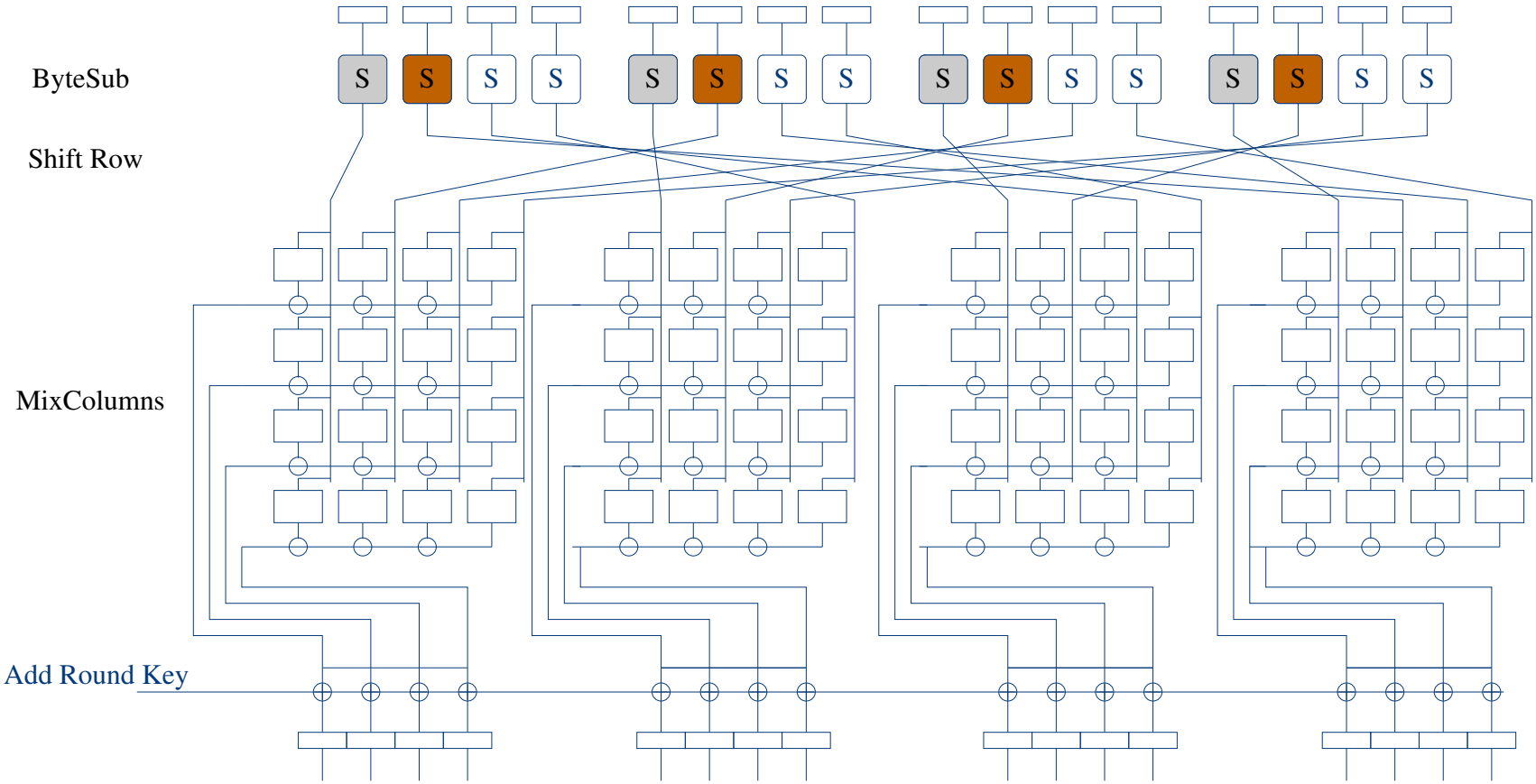




Advanced Side Channel Attacks (DPA like attacks)

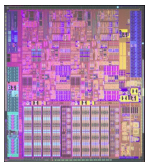
AES Round - Parallel Hardware Implementation

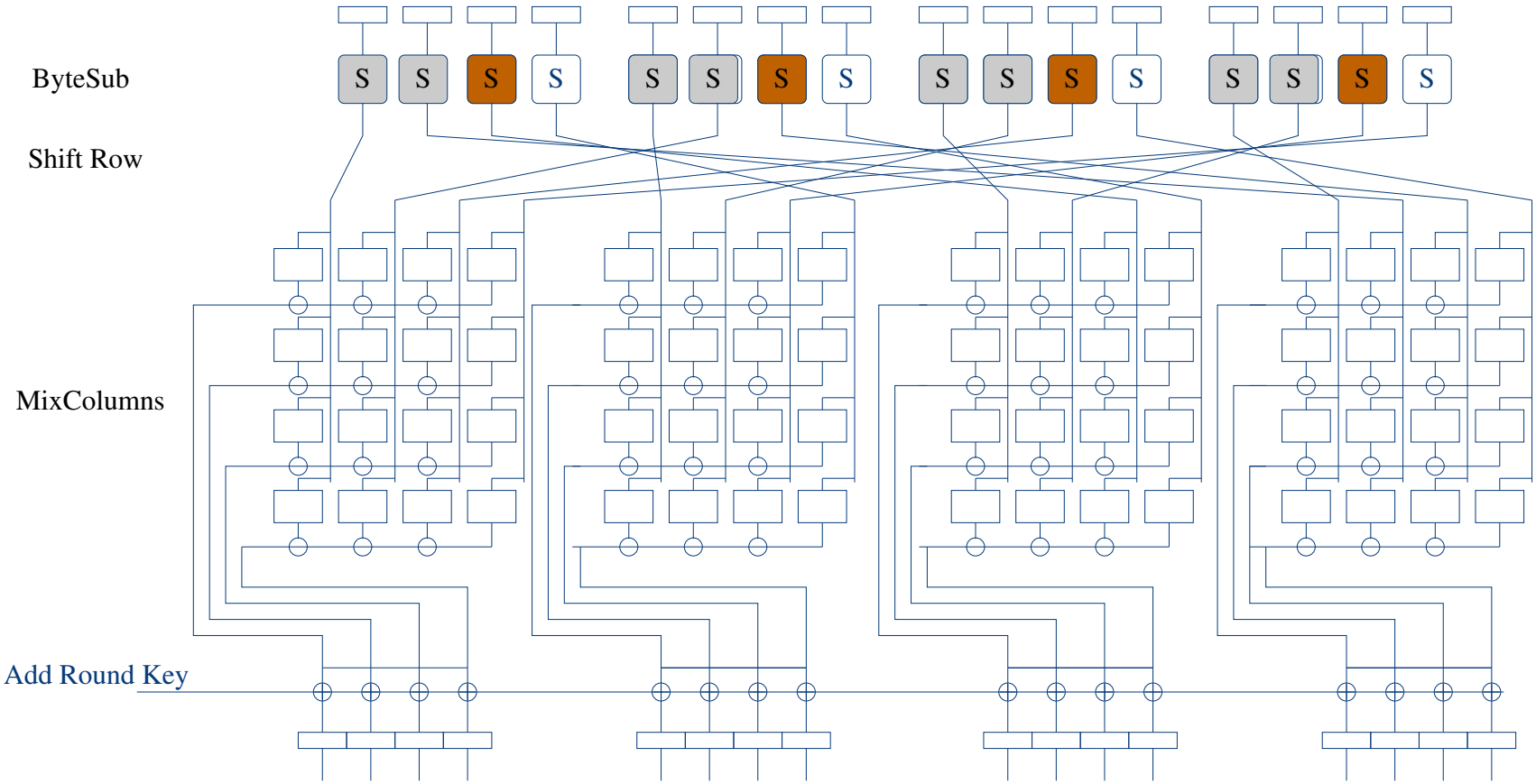




Advanced Side Channel Attacks (DPA like attacks)

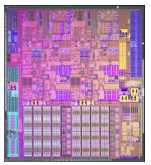
AES Round - Parallel Hardware Implementation

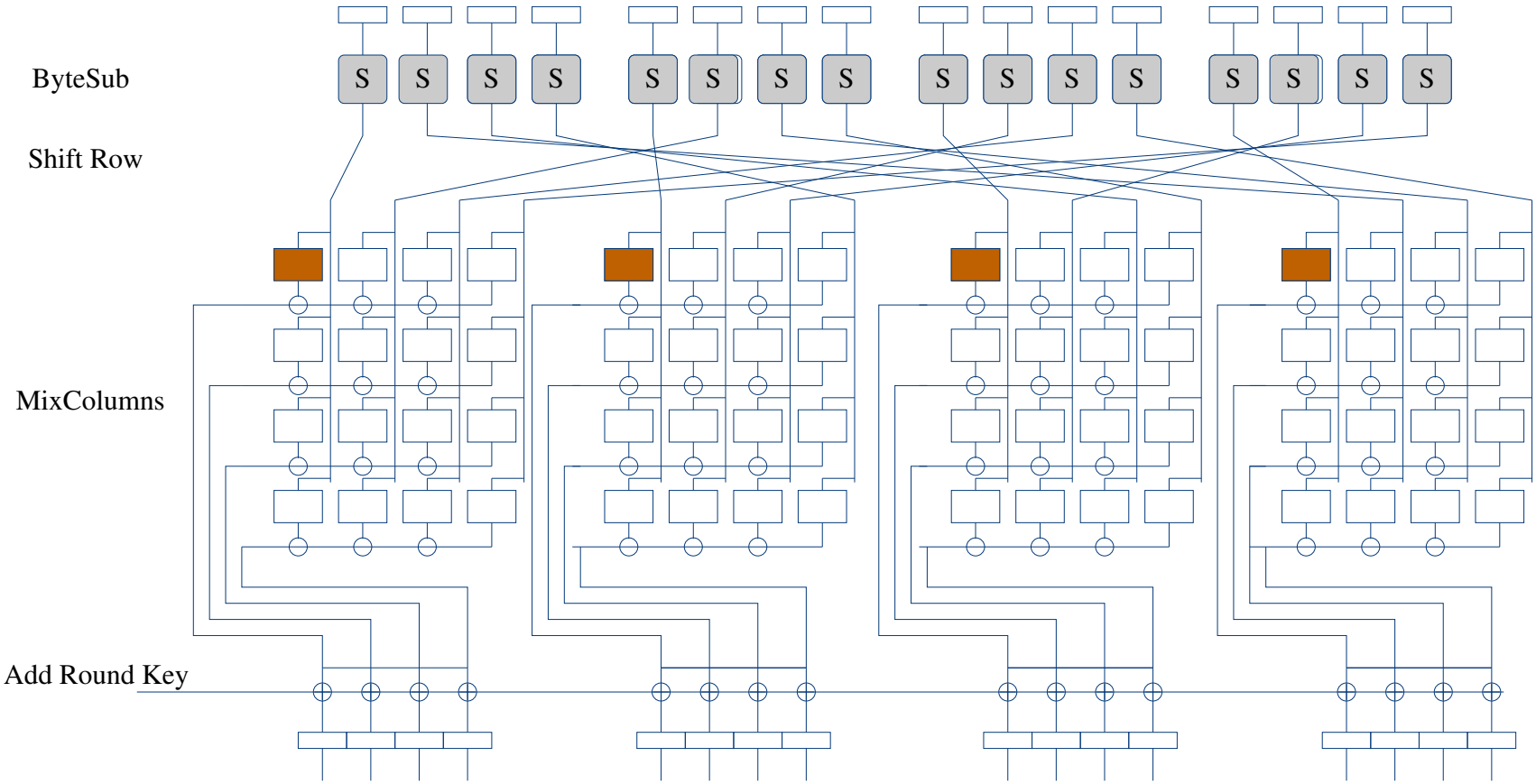




Advanced Side Channel Attacks (DPA like attacks)

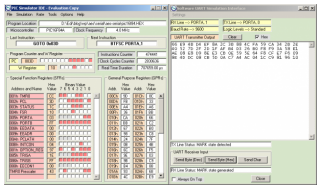
AES Round - Parallel Hardware Implementation

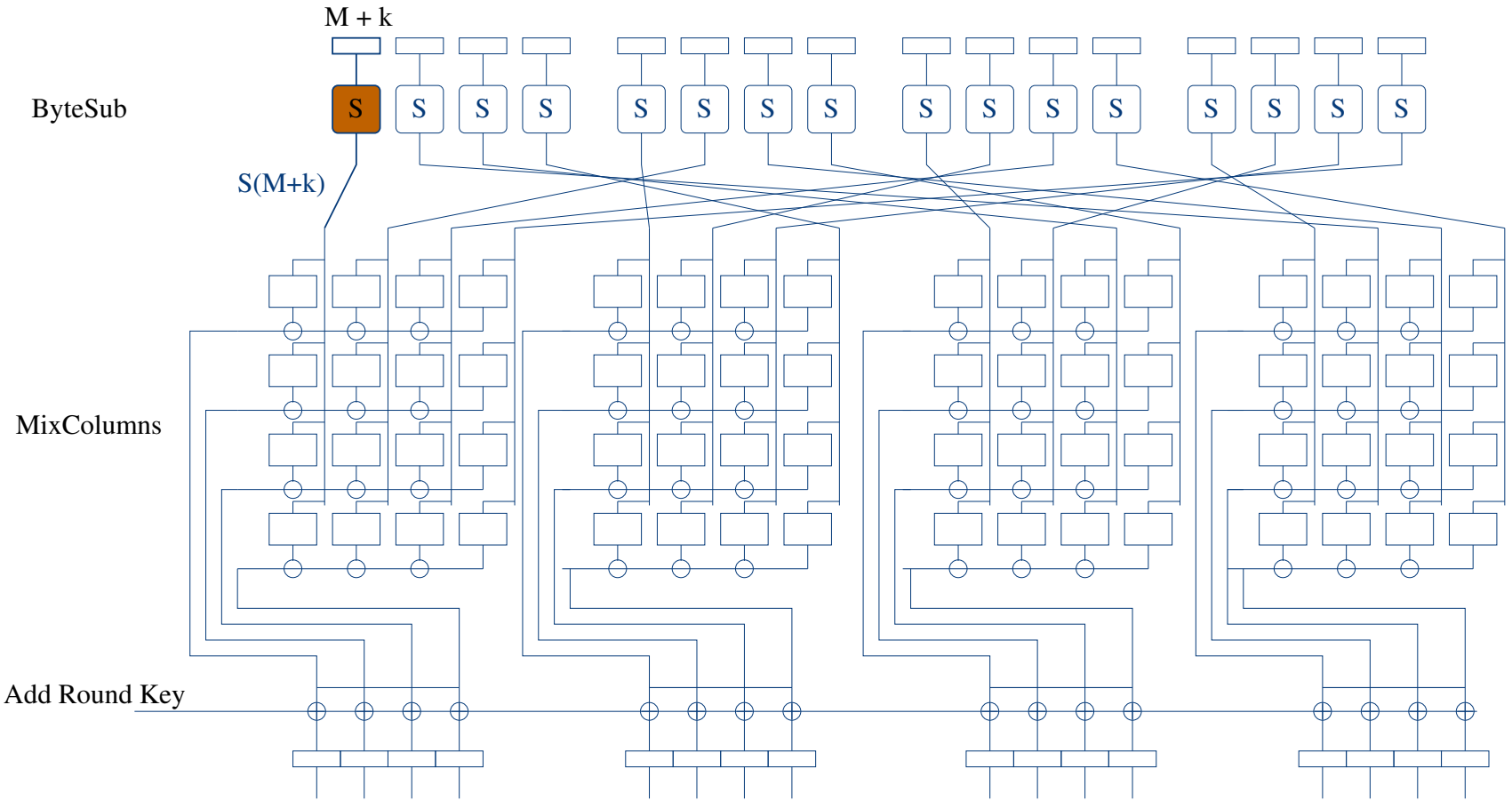




Advanced Side Channel Attacks (DPA like attacks)

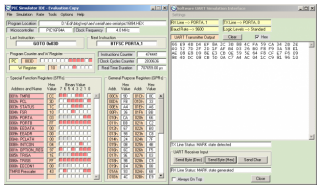
AES Round - Software Implementation – SCA attack

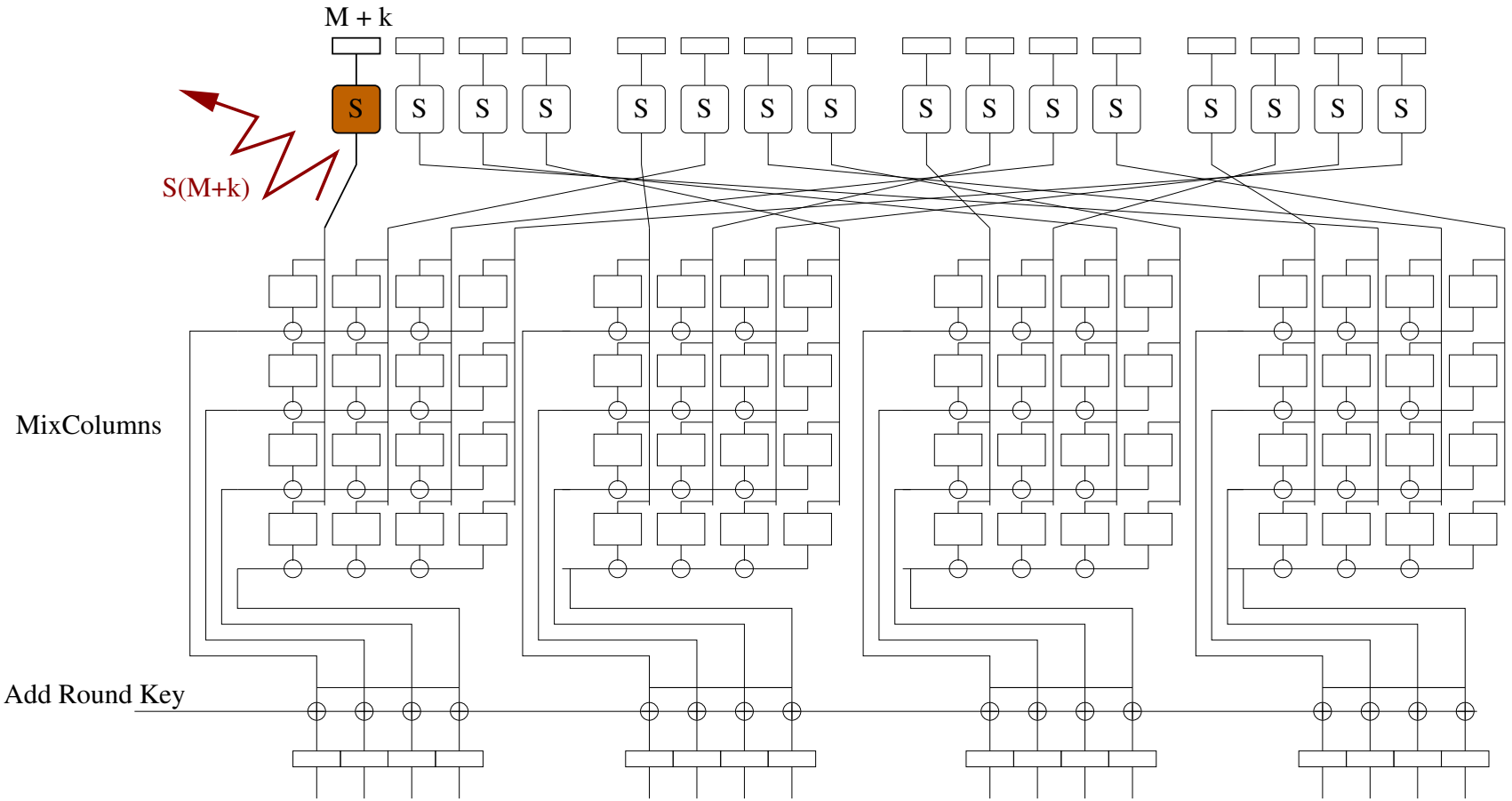




Advanced Side Channel Attacks (DPA like attacks)

AES Round - Software Implementation – SCA attack

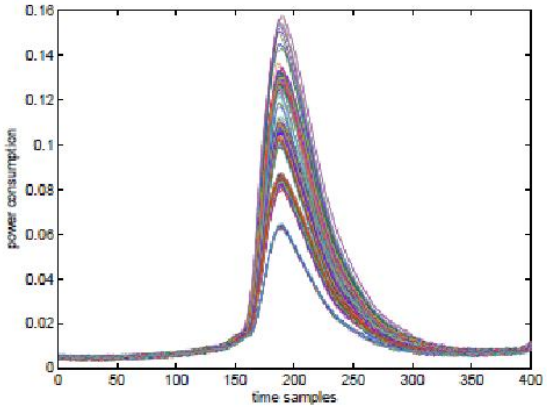


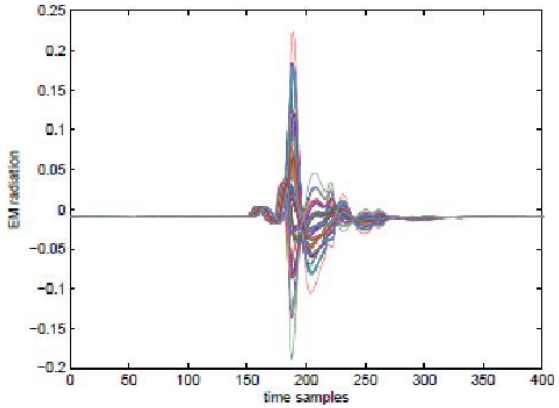


Advanced Side Channel Attacks (DPA like attacks)

Main Observation

Leakage at time t depends on the data manipulated at this time.





1. Power consumption leakage during the manipulation of a 8-bit variable by a card [Kocher, Jaffe and Jun, CRYPTO 1999].
2. Electromagnetic emanation during the same manipulation [Quisquater and Samyde, ESmart 2001].

Note 1 : traces repartition does not look random.

Note 2 : power consumptions are always positive whereas electromagnetic emanations are signed.

Advanced Side Channel Attacks (DPA like attacks)

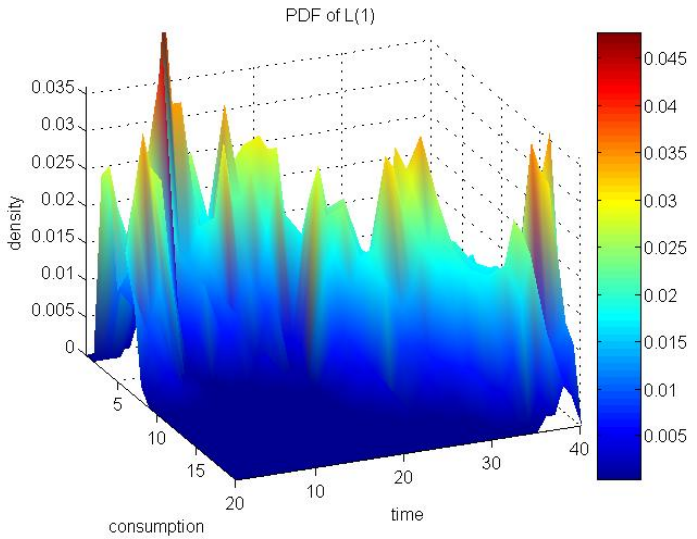
Main Observation

Example : pdf of the leakage for a device processing...

... $\texttt{AES-Sbox}(M + \mathbf{K})$ with $\mathbf{K} = 1$.

M varies uniformly
For each time (abs.) and each value x in a finite interval (ord.) we plotted in z-axis :

$$Pr[\text{leakage} = x] \sim pdf_{\text{leakage}}(x)$$



Advanced Side Channel Attacks (DPA like attacks)

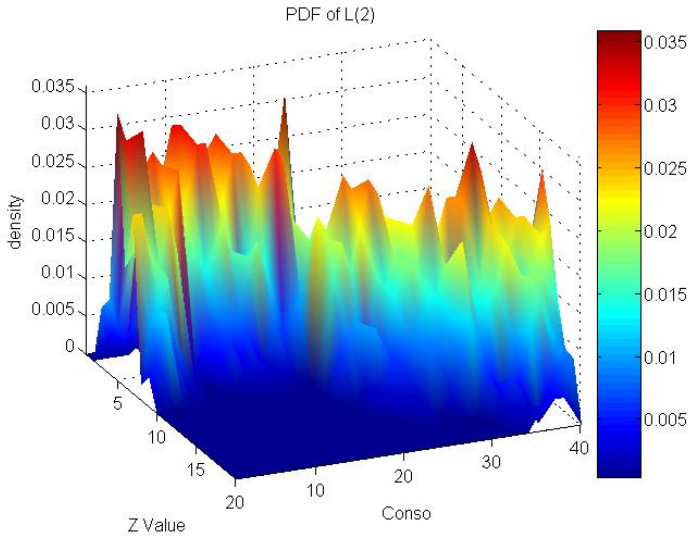
Main Observation

Example : pdf of the leakage for a device processing...

... $\texttt{AES-Sbox}(M + \mathbf{K})$ with $\mathbf{K} = 2$.

M varies uniformly
For each time (abs.) and each value x in a finite interval (ord.) we plotted in z-axis :

$$Pr[\text{leakage} = x] \sim pdf_{\text{leakage}}(x)$$



Advanced Side Channel Attacks (DPA like attacks)

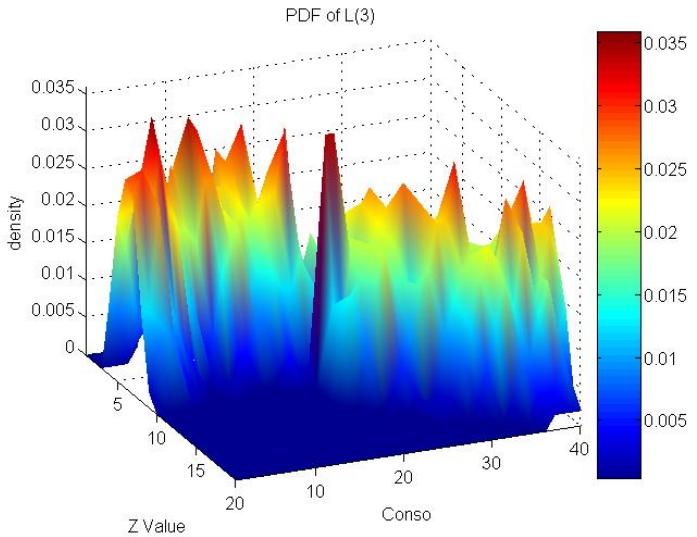
Main Observation

Example : pdf of the leakage for a device processing...

... $\texttt{AES-Sbox}(M + \mathbf{K})$ with $\mathbf{K} = 3$.

M varies uniformly
For each time (abs.) and each value x in a finite interval (ord.) we plotted in z-axis :

$$Pr[\text{leakage} = x] \sim pdf_{\text{leakage}}(x)$$



Advanced Side Channel Attacks (DPA like attacks)

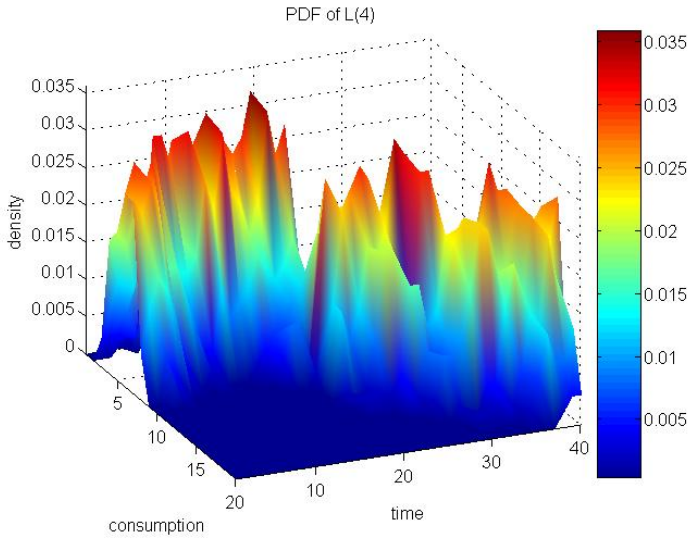
Main Observation

Example : pdf of the leakage for a device processing...

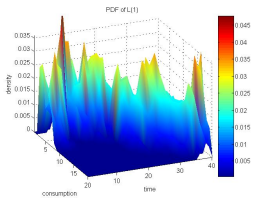
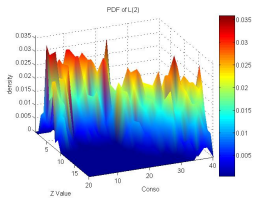
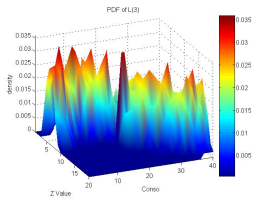
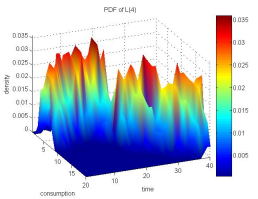
... $\texttt{AES-Sbox}(M + \mathbf{K})$ with $\mathbf{K} = 4$.

M varies uniformly
For each time (abs.) and each value x in a finite interval (ord.) we plotted in z-axis :

$$Pr[\text{leakage} = x] \sim pdf_{\text{leakage}}(x)$$



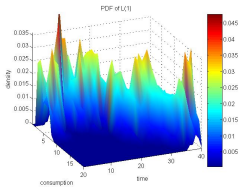
- [Pre-computation] For every possible key $k^\star$ pre-compute the pdf of the leakage **L**.

$k^\star = 1$ $k^\star = 2$ $k^\star = 3$ $k^\star = 4$ .....

  - [Necessary Condition] Have an open access to a copy of the target device and be able to choose the key value.

- [Measurement] Measure the consumption for the target device and estimate the pdf of **L** for this target.

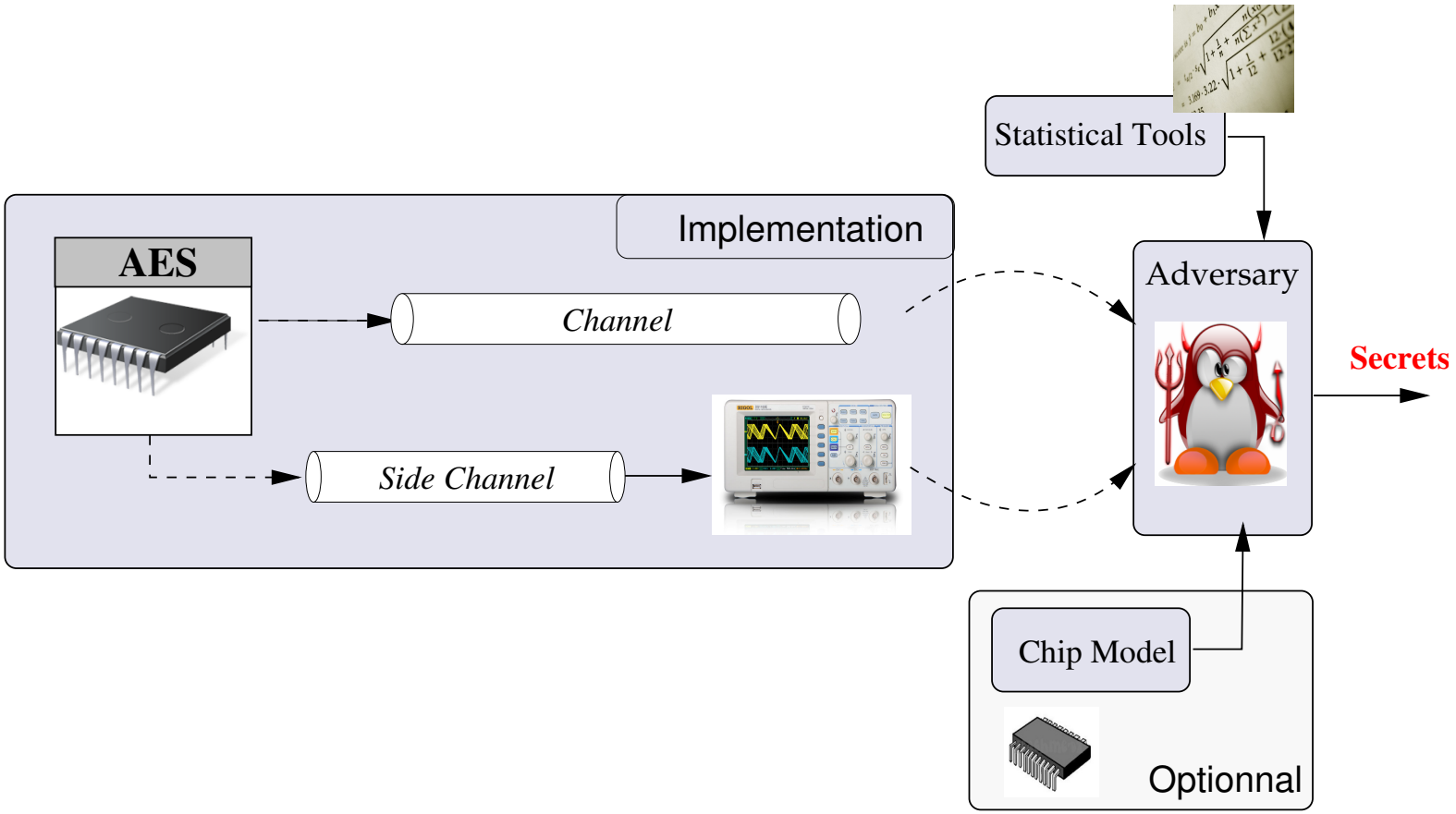


$k^\star = ?$

- [Key-recovery] Compare the pdf estimation with those pre-computed and output the most likely key candidate.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : General Framework.



Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : General Framework (Theoretical)

Context : attack during the manipulation of $S(M + k)$.

1. Measurement : get a sample $(\ell_{k,i})_i$ related to a sample $(m_i)_i$ of plaintexts.
2. Model Selection : Design/Select a function $\mathbf{m}(\cdot)$.
3. Prediction : For every $\hat{k}$, compute $m_{\hat{k},i} = \mathbf{m}(S(m_i + \hat{k}))$.
4. Distinguisher Selection : Choose a statistical distinguisher Δ.
5. Key Discrimination : For every $\hat{k}$, compute the distinguishing value $\Delta_{\hat{k}}$:

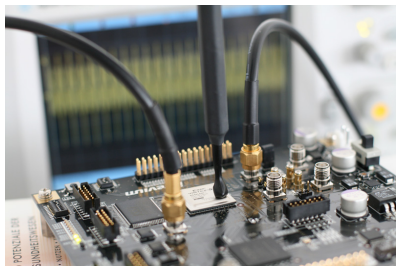
$$\Delta_{\hat{k}} = \Delta\left((\ell_{k,i})_i, (m_{\hat{k},i})_i\right) \quad .$$

6. Key Candidate Selection : Output $\hat{k}$ maximizing $\Delta_{\hat{k}}$.

Advanced Side Channel Attacks (DPA like attacks)

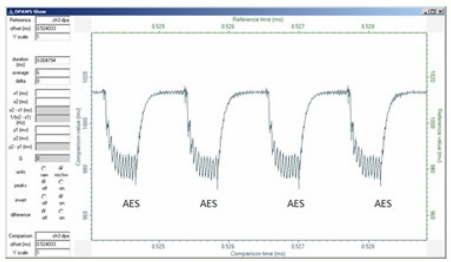
Side Channel Analysis : Measurement Step in details.

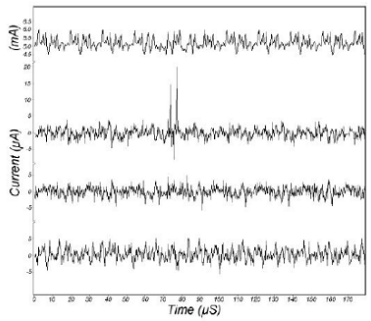
1. Which kind of measure (power consumption, electromagnetic emanations, timing, temperature) ?



Engineers Expertise. Quality of the attacks labs.

2. Which point (time) to attack in a set of measurement traces ?





Sometimes easy ... sometimes not !

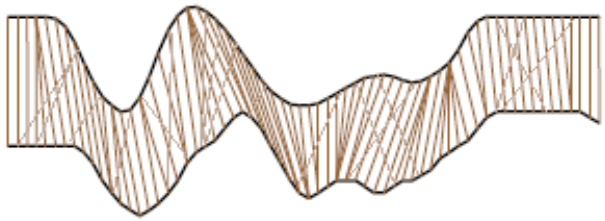
Still Engineers Expertise and use of signal processing

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : Find points of interest.

Still Engineers Expertise and use of signal processing techniques.

- First step : re-synchronization of the traces. Use of pattern matching techniques, voice synchronization techniques, etc. (*e.g.* see Muijrers et al. at CARDIS 2011).



- Second step : find the points of interest. Use of signal processing techniques (*e.g.* see Bohy et al published ESmart 2003).
- Third step : select a few points of interest and apply the attack independently on each of them.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : Find points of interest.

Identify points of interest thanks to Principal Component Analysis.

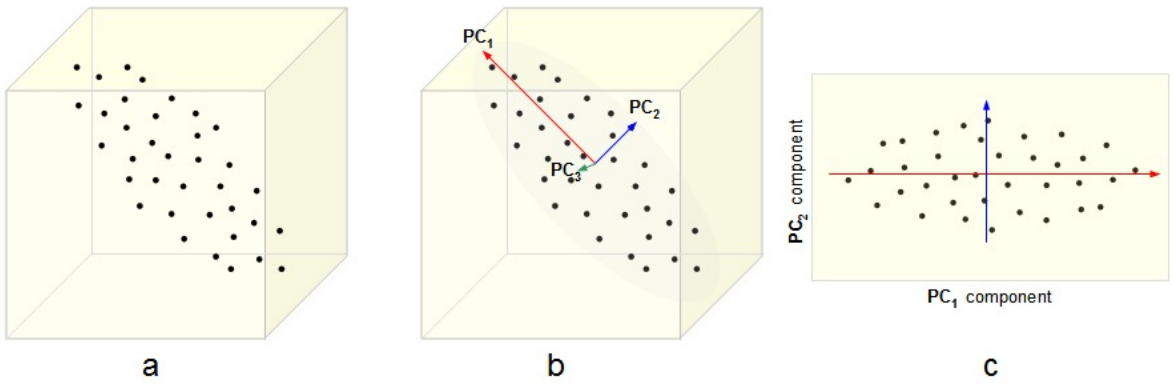
- References : "Principal Component Analysis" by I.T. Jolliffe. and "Linear Algebra and Its Applications" by D.C. Lay.
- Often use to reduce the dimension of a set of observations.
- In our context, a measurement trace for datum manipulation has size (or dimension) between 1000 to 10^6 points.
- Goal : reduce the size to only a few (< 10) points.


Karl Pearson

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : Find points of interest.

Example for 2-dimensional data sets :



1. Find a change of basis such the new coordinates of the data vectors are uncorrelated in the new basis.
2. Arranged the new vectors in order of decreasing variance.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : Find points of interest.

1. Arrange all the k-dimensional leakage measurements L_1, ..., L_N into a $k \times N$ matrix $\mathbf{X}$.
2. Compute the mean vector $u = 1/N \sum_i L_i$ and subtract it to each line of $\mathbf{X}$. The new matrix is denoted by B.
3. Compute the covariance matrix $C = B \times B^{\perp}$.
4. Compute the matric of eigenvectors V which diagonalizes the covariance matrix C. We denote by D the diagonal matrix such that $V^{-1}CV = D$.
5. Elements of the diagonal of D are the eigenvalues λ_i.
6. Rearrange the eigenvectors in decreasing order of the corresponding eigenvalues. This gives a new matrix V.
7. Select l columns in V to define a new matrix W.
8. Project B thanks to W : this gives a new matrix of reduce vectors $Y = W^{\perp} \times B$.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : General Framework (Theoretical)

Context : attack during the manipulation of $S(M + k)$.

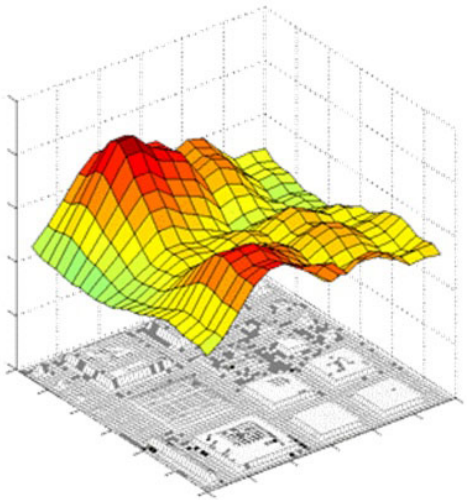
1. Measurement : get a sample $(\ell_{k,i})_i$ related to a sample $(m_i)_i$ of plaintexts.
2. Model Selection : Design/Select a function $\mathbf{m}(\cdot)$.
3. Prediction : For every $\hat{k}$, compute $m_{\hat{k},i} = \mathbf{m}(S(m_i + \hat{k}))$.
4. Distinguisher Selection : Choose a statistical distinguisher Δ.
5. Key Discrimination : For every $\hat{k}$, compute the distinguishing value $\Delta_{\hat{k}}$:
$$\Delta_{\hat{k}} = \Delta\left((\ell_{k,i})_i, (m_{\hat{k},i})_i\right) \quad .$$
6. Key Candidate Selection : Output $\hat{k}$ maximizing $\Delta_{\hat{k}}$.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : define a model for the consumption.

Defining the kind of dependency between the manipulated data and the behavior implies a modeling step.
Let L be the (univariate) observation during the manipulation of $S(M + k)$, where M is unknown and k is the secret to recover.

- First solution (template/profiled attacks principle) : use an exact copy of the attacked device and estimate the pdf of L for every possible pair (M, k). see [Chari et al at CHES 2002].
- Second solution (unprofiled attacks principle) : model the function $\mathbf{E}[L|\ M = m, K = k]$. see Messerges PhD Thesis.



Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.

This step is conducted under the following assumption

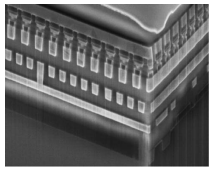
Independent Noise Assumption (INA)

The random variable L related to the manipulation of $S(Z)$ equals $Y + B$, where Y is a function of Z and B is independent of Z.

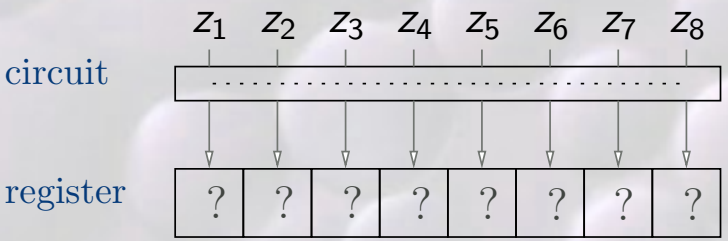
- B is usually called the noise and is viewed as a continuous random variable.
- We usually assume $B \sim \mathcal{N}(0, \sigma^2)$. (sometimes called Gaussian Noise Assumption).
- Usually, we have $Z = M + K$ where M is known and k is the secret to recover.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.



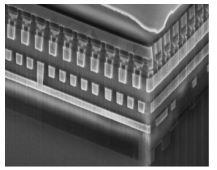
$$Z = S(M + k)$$



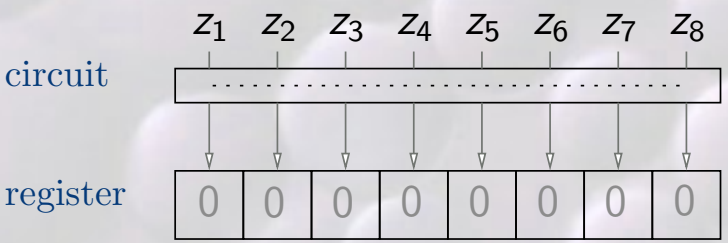
Assumption

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.



$$Z = O(M + k)$$

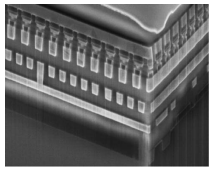


Assumption

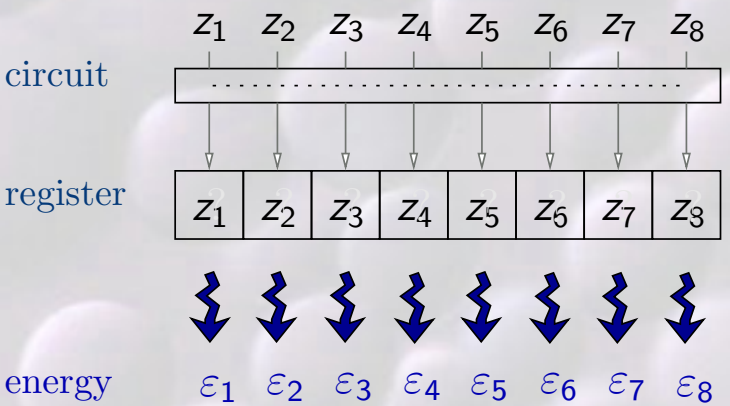
0 Precharge

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.



$$Z = O(M + k)$$



Assumption

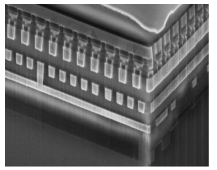
0 Precharge

Linear Regression

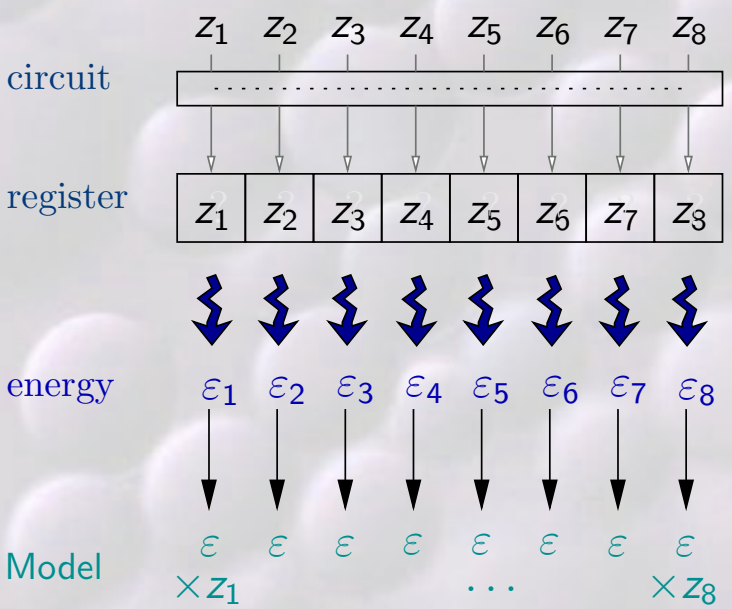
the ε_i's are indep.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.



$Z = O(M + k)$



$Y = \varepsilon \times \mathrm{HW}(Z)$

Assumption

0 Precharge

Linear Regression
the ε_i's are indep.

Hamming Weight Model

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.

$$L \leftarrow Y + B = \varphi(Z) + B$$

To sum up

- The deterministic part Y in a leakage L may be viewed as a multivariate polynomial in the bit-coordinate z_i of Z with coefficients in $\mathbb{R}$.
- Namely $\varphi(Z)$ is a polynomial in $\mathbb{R}[z_1, \cdots, z_n]$ and this polynomial is *a priori* unknown to the adversary.
- The modelling problem can hence be reduced to a problem of polynomial interpolation problem in noisy context. Namely from noisy observations of $\varphi(Y)$, we want to recover the coefficients ε_0, ε_1), ... such that :

$$\varphi(Z) = \varepsilon_0 z_0 + \varepsilon_1 z_1 + ... + \varepsilon_{0,1} z_0 z_1 + \varepsilon_{0,2} z_0 z_2 + ...$$

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : modeling for unprofiled attacks.

$$L \leftarrow Y + B = \varphi(Z) + B$$

$$\varphi(Z) = \varepsilon_0 z_0 + \varepsilon_1 z_1 + \ldots$$

To sum up

- The polynomial interpolation with noise problem is usually solved thanks to linear regression techniques. See Schindler et al. at CHES 2005 or Doget et al at JCEN 2011.
- Usually, we assume the polynomial is of degree 1.
- All the coefficients ε_i for degree-1 monomials are equal (to 1).
- The latter assumption (called Hamming Weight) is today pertinent for almost all smart card technologies.
- For recent ones (*e.g.* 65nm tech.), the nonlinear terms must be taken into account. See Veyrat-Charvillon et al's paper at CRYPTO 2011.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : General Framework (Theoretical)

Context : attack during the manipulation of $S(M + k)$.

1. Measurement : get a sample $(\ell_{k,i})_i$ related to a sample $(m_i)_i$ of plaintexts.
2. Model Selection : Design/Select a function $\mathbf{m}(\cdot)$.
3. Prediction : For every $\hat{k}$, compute $m_{\hat{k},i} = \mathbf{m}(S(m_i + \hat{k}))$.
4. Distinguisher Selection : Choose a statistical distinguisher Δ.
5. Key Discrimination : For every $\hat{k}$, compute the distinguishing value $\Delta_{\hat{k}}$:
$$\Delta_{\hat{k}} = \Delta\left((\ell_{k,i})_i, (m_{\hat{k},i})_i\right) \quad .$$
The random variable associated to $m_{\hat{k},i}$ is denoted by $M_{\hat{k}}$.
6. Key Candidate Selection : Output $\hat{k}$ maximizing $\Delta_{\hat{k}}$.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

Under INA assumption, the pdf f_L of L is a Gaussian Mixture :

$$f_L(\ell) = \sum_i \Pr[\varphi(Z) = i]\mathcal{N}(i, \sigma^2)$$

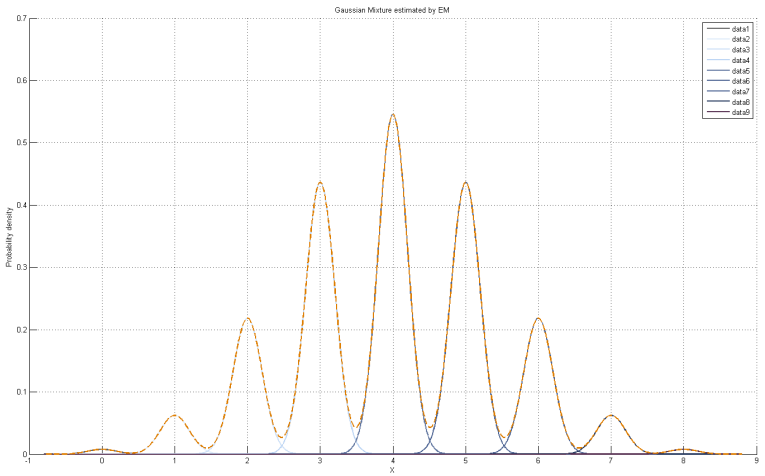


FIGURE – No noise ($\sigma = 0.2$)

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

Question : which property of this mixture depends on the the secret k ?

Note : difficult question since the adversary does not know φ but a model $\mathbf{m}$ for it !

Many proposals have been done in the literature :

- DPA Kocher et al at CRYPTO 96,
- Multi-bit DPA Messerges in his PhD Thesis,
- CPA Brier et al at CHES 2004,
- Stochastic Attacks Schindler et al at CHES 2006
- or the MIA Gierlichs et al at CHES 2008.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

DPA attack Kocher et al at CRYPTO 96.

- Model function **m** : the coordinate function $X \mapsto x_i$ for a chosen $i = 1, ..., 8$.
- Statistical Distinguisher : difference of means Test.
- Distinguishing value $\Delta_{\hat{k}}$: a statistical estimator of

$$\Delta_{\hat{k}} = \mathbf{E}(L \mid M_{\hat{k}} = 1) - \mathbf{E}(L \mid M_{\hat{k}} = 0)$$

with $M_{\hat{k}}$ equal to the ith bit of $S(M + \hat{k})$.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

DPA attack Kocher et al at CRYPTO 96. Why does it work ?

$$\begin{aligned}\Delta_{\hat{k}} &= \mathbf{E}(L \mid M_{\hat{k}} = 1) - \mathbf{E}(L \mid M_{\hat{k}} = 0) \\ &= \mathbf{E}(\varphi(Z) + B \mid M_{\hat{k}} = 1) - \mathbf{E}(\varphi(Z) + B \mid M_{\hat{k}} = 0)\end{aligned}$$

Since the noise B is independent of Z,

$$\begin{aligned}\Delta_{\hat{k}} &= \mathbf{E}(\varphi(Z) \mid M_{\hat{k}} = 1) - \mathbf{E}(\varphi(Z) \mid M_{\hat{k}} = 0) \\ &= \mathbf{E}(\varepsilon_i z_i + (\varphi(Z) - \varepsilon_i z_i) \mid M_{\hat{k}} = 1) - \mathbf{E}(\varepsilon_i z_i + (\varphi(Z) - \varepsilon_i z_i) \mid M_{\hat{k}} = 0)\end{aligned}$$

Let us assume that $(\varphi(Z) - \varepsilon_i z_i)$ is independent of z_i and $M_{\hat{k}}$ (true in practice).

$$\Delta_{\hat{k}} = \varepsilon_i \left(\mathbf{E}(z_i \mid M_{\hat{k}} = 1) - \mathbf{E}(z_i \mid M_{\hat{k}} = 0)\right)$$

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

$$\Delta_{\hat{k}} = \varepsilon_i \left(\mathbf{E}(z_i \mid M_{\hat{k}} = 1) - \mathbf{E}(z_i \mid M_{\hat{k}} = 0) \right)$$

where

- z_i is the ith bit of $S(M + k)$
- $M_{\hat{k}}$ is the ith bit of $S(M + \hat{k})$

If $k = \hat{k}$, then $z_i = M_{\hat{k}}$ and :

$$\Delta_{\hat{k}} = \varepsilon_i (1 - 0) = \varepsilon_i$$

If $k = \hat{k}$, then z_i and $M_{\hat{k}}$ are independent (due to properties of S) and

$$\Delta_{\hat{k}} = \varepsilon_i \left(\mathbf{E}(z_i) - \mathbf{E}(z_i) \right) = 0$$

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

DPA attack Kocher et al at CRYPTO 96.

- Pros : no need for assumption on the device properties, quite efficient in practice.
- Cons : does not use all the information in the trace and attack each bit of the target separately.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

Multi-bit DPA attack Messerges in his PhD Thesis.

- Model function **m** : the Hamming weight function.
- Statistical Distinguisher : difference of means Test for a parameter τ
- Distinguishing value $\Delta_{\hat{k}}$: a statistical estimator of

$$\Delta_{\hat{k}} = \mathbf{E}(L \mid M_{\hat{k}} \leq \tau) - \mathbf{E}(L \mid M_{\hat{k}} > \tau)$$

with $M_{\hat{k}}$ equal to the HW$[S(M + \hat{k})]$.

- Pros : exploit more information than the DPA.
- Cons : need assumption (Hamming weight) on the device behaviour.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

CPA attack Brier et al at CHES 2004.

- Model function **m** : possibly any function (in practice the Hamming weight function).
- Statistical Distinguisher : linear correlation coefficient.
- Distinguishing value $\Delta_{\hat{k}}$: a statistical estimator of

$$\Delta_{\hat{k}} = \rho(L, M_{\hat{k}})$$

- Pros : exploit more information than the previous ones and is more powerful
- Cons : need assumption (Hamming weight) on the device behaviour.

Advanced Side Channel Attacks (DPA like attacks)

Side Channel Analysis : the statistical distinguisher

MIA attack Gierlichs et al at CHES 2008.

- Model function **m** : possibly any function (in practice the Hamming weight function).
- Statistical Distinguisher : mutual information.
- Distinguishing value $\Delta_{\hat{k}}$: a statistical estimator of

$$\Delta_{\hat{k}} = MI(L; M_{\hat{k}}) = \mathrm{H}(L) - \mathrm{H}(L \mid M_{\hat{k}})$$

- Pros : theoretically able to detect any kind of dependency whatever the quality of the model
- Cons : need for efficient estimators of the entropy (currently less efficient than the CPA).

Advanced Side Channel Attacks (DPA like attacks)

Other attacks

- Stochastic attacks : See Schindler et al. at CHES 2005 or Doget et al at JCEN 2011.
  - Good alternative when classical models fail.
- Kolmogorov-Smirnov Based attacks : Whitnall et al. at CARDIS 2011.
  - Good alternative to the MIA.
- PPA, EPA, VPA, etc : other attacks exist but are often very ad hoc ones with no clear advantage to the "classical" ones.
- Works comparing the attacks :
  - "How to Compare Profiled Side-Channel Attacks ?" by Standaert et al at ACNS 2009.
  - "A fair evaluation framework for comparing side-channel distinguishers" by Withnall et al at JCEN 2011.
  - "Univariate Side Channel Attacks and Leakage Modeling" by Doget et al at JCEN 2011.

SCA Countermeasures

- Masking [Chari et al at CRYPTO 1999].
  - Efficient against SCA in practice.
  - Difficult to implement for non-linear transformations.
- Shuffling [Herbst et al at ACNS 2006].
  - Less efficient against SCA in practice.
  - Easy to implement for every transformation.

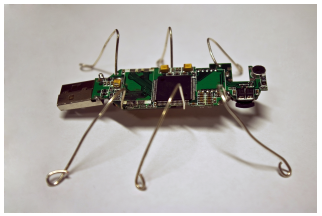
Quatrième partie IV

Active Side Channel Attacks

Outline

A card can behave in an abnormal way :

- Unexpected execution conditions :
  - Power tension out of the allowed range,
  - Clock frequency out of the allowed range.
- Light Perturbation.
- Exploitation of bugs (passive attacks).



Glitches attacks

- Induce a brief peak on one of the smart card contact lines : Vcc, Gnd, CLK $\Longrightarrow$ perturbation.

Advantages : easy to mount, non-invasive attack

Issues : the perturbation is global

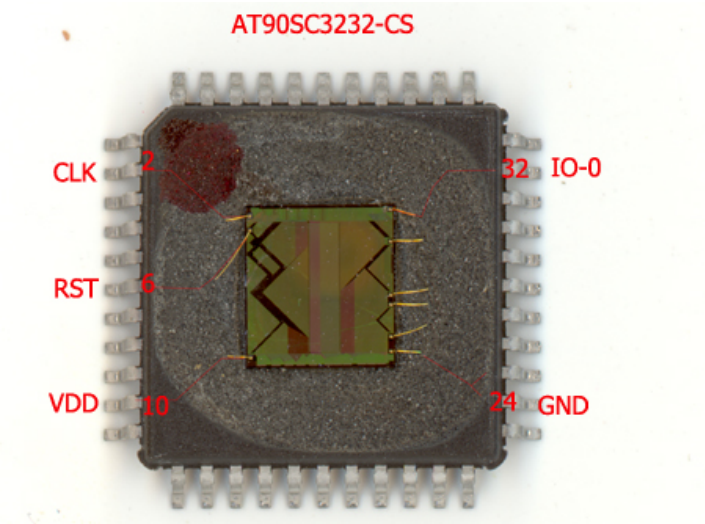
- Modern smart cards are protected against those attacks thanks to material countermeasures.

Laser Attacks

Electron of the silicium are excited by the light/laser and create an energy field that disturbs the device :

- in the logic glue,
- on the data in RAM, on the BUS,
- on the EEPROM (code or data)
- ...

Advantages : one can *choose* the fault location and timing.
Issues : invasive attacks.



Fault Models

Different models of adversary can be define according to their assumed faults ability :

- the kind of disturbed variables ;
- the timing precision of the fault ;
- the number of faulty bits ;
- the kind of fault :
  - *stuck at*
  - *bit flip*
  - random

Modification of manipulated data, of the code execution flaw,. . .

Attack against the Square-and-Multiply Always

Algo S&M Always

INPUT(S) :
$m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t_0 \leftarrow 1$
2: **for** $i = n$ to 0 **do**
3: $\quad t_0 \leftarrow t_0 \times t_0$
4: $\quad t_1 \leftarrow t_0 \times m$
5: $\quad t_0 \leftarrow t_{d_i}$
6: **return** t

- The adversary disturbs the 4th step during the ith processing of the loop :
  - If the final result is erroneous then $\Longrightarrow$ $d_i = 1$
  - Otherwise, $\Longrightarrow$ $d_i = 0$

More generally, countermeasures based on the addition of dummy operations are counteracted by this kind of attacks.

Bellcore Attack on RSA-CRT

Algo RSA-CRT

INPUT(S) : g, p, q, d_p, d_q
OUTPUT(S) : $g^d \mod m$

1: $g_p = g^{d_p} \mod p$
2: $g_q = g^{d_q} \mod q$ $g'_q = g^{d_q} \mod q$
3: $t = g_p + p\left[(g_q - g_p)(p^{-1} \mod q) \mod q\right] t' = g_p + p\left[(g'_q - g_p)(p^{-1} \mod q) \mod q\right]$
4: **return** tt'

- Correct Execution :
  - $t \equiv g_p \mod p$
  - $t \equiv g_q \mod q$
- Erroneous Execution :
  - $t' \equiv g_p \mod p$
  - $t' \not\equiv g_q \mod q$

- $t - t' \equiv 0 \mod p$
- $t - t' \not\equiv 0 \mod q$
- $\Longrightarrow \gcd(t - t', pq) = p$

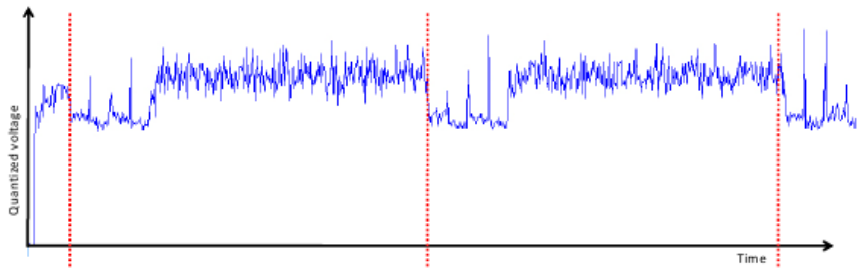
Actually, only the faulty result is needed.

Implementation

Algo RSA-CRT

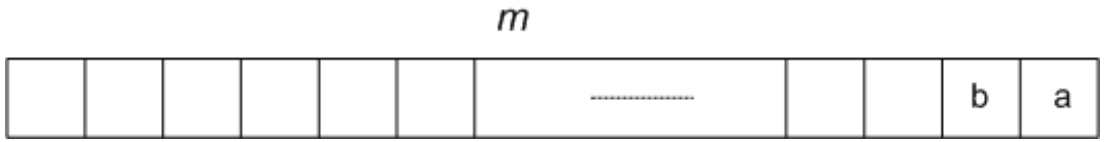
INPUT(S) : g, p, q, d_p, d_q
OUTPUT(S) : $g^d \mod m$

1: $g_p = g^{d_p} \mod p$
2: $g'_q = g^{d_q} \mod q$
3: $t' = g_p + p\left[(g'_q - g_p)(p^{-1} \mod q) \mod q\right]$
4: **return** t'



Bug Attacks [?]1/2

- Hypothesis the multiplier is bugged : $\exists\ a, b$ tq $a \boxtimes b \neq a \times b$
- After assuming $p < q$, we get $p \leq \lfloor \sqrt{n} \rfloor < q$
- We build the message m which is a close as possible to $\sqrt{n}$ and verifies :



Bug Attacks [?] 2/2

Algo RSA-CRT

INPUT(S) : m, p, q, d_p, d_q
OUTPUT(S) : $m^d \mod m$

1: $m_p = m^{d_p} \mod p$
2: $m_q = m^{d_q} \mod q$
3: $t = m_p q(q^{-1} \mod p) + m_q p(p^{-1} \mod p)$
4: **return** t

- 1st step : modular reduction of the input
  - $m \mod q = m$
  - $m \mod p$ is *random*
- 2nd step : squaring
- With large probability, only the exponentiation modulo q will be erroneous.

Countermeasures against *Bellcore*'s attack

Material (Chip manufacturer) :

- *glitches* detectors
- light detectors,
- redundancy. . .

Software (developer) :

- Check the result with the public exponent e
  - Depending on the size of e, this can multiply the timing by 2,
  - The public exponent is not always available. . .
- One usually tries to find more efficient solutions.

Montgomery Ladder Joye and Yen at CHES 2002 1/2

- Let $d = \sum_{i=0}^{n} d_i 2^i$
- Let $L_j = \sum_{i=j}^{n} d_i 2^{i-j}$ et $H_j = L_j + 1$
- We have :

$$\begin{aligned} L_j &= H_j - 1 \\ &= 2L_{j+1} + d_j \\ &= L_{j+1} + H_{j+1} + d_j - 1 \\ &= 2H_{j+1} + d_j - 2 \end{aligned}$$

- Then :

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) & \text{si } d_j = 0 \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) & \text{si } d_j = 1 \end{cases}$$

- $L_0 = d$!
-

$$(g^{L_j}, g^{H_j}) = \begin{cases} \left((g^{L_{j+1}})^2, g^{L_{j+1}} \times g^{H_{j+1}}\right) \text{si } d_j = 0 \\ \left(g^{L_{j+1}} \times g^{H_{j+(1}}, (g^{H_{j+1}})^2\right) \text{si } d_j = 1 \end{cases}$$

Montgomery Ladder Joye and Yen at CHES 2002 2/2

Algo Montgomery Ladder

INPUT(S) :
$m, d = (d_n d_{n-1} \dots d_1 d_0)_2$
OUTPUT(S) : m^d

1: $t_0 \leftarrow 1$
2: $t_1 \leftarrow m$
3: **for** $i = n$ to 0 **do**
4: **if** $d_j = 0$ **then**
5: $t_1 \leftarrow t_0 \times t_1$
6: $t_0 \leftarrow t_0^2$
7: **else**
8: $t_0 \leftarrow t_0 \times t_1$
9: $t_1 \leftarrow t_1^2$
10: **return** t_0

- $t_0 = m^{L_i}$
- $t_1 = m^{H_i}$
- $2(n+1)$ MULT
- The algorithm is regular : thwart simple attacks (e.g. SPA).
- No dummy operations.
- t_1/t_0 is invariant $(= m)$: one can use this property to test the coherence of the result.

Shamir's trick [?]

Algo RSA-CRT (recalling)

INPUT(S) : $g, p, q, d_p = d \mod (p-1), d_q = d \mod (q-1)$
OUTPUT(S) : $g^d \mod m$

1: $g_p = g^{d_p} \mod p$
2: $g_q = g^{d_q} \mod q$
3: $t = g_p + p\left[(g_q - g_p)(p^{-1} \mod q) \mod q\right]$
4: **return** t

Algo Secure RSA-CRT

INPUT(S) : $g, p, q, d_p = d \mod (p-1), d_q = d \mod (q-1), s$
OUTPUT(S) : $g^d \mod m$

1: $g_{ps} = g^{d \mod \phi(ps)} \mod ps$
2: $g_{qs} = g^{d \mod \phi(qs)} \mod qs$
3: **if** $g_{ps} \not\equiv g_{qs} \mod s$ **then**
4: **return** Error
5: $g_p = g_{ps} \mod p$
6: $g_q = g_{qs} \mod q$
7: $t = g_p + p\left[(g_q - g_p)(p^{-1} \mod q) \mod q\right]$
8: **return** t

- Don't protect the recombining
- d must be known

An example of infective countermeasure [?]

- Countermeasures that take into account all the internal computations are proposed
- Goal : minimize the cost for an error detection rate as high as possible.

ALGORITHM 2 (INFECTIVE CRT-RSA).
Input. *A message* $m \in \mathbb{Z}_N$
Output. *Sig* $:= m^d \bmod N$ *or a random number in* $\mathbb{Z}_N$
In Memory. $p \cdot t_1$, $q \cdot t_2$, N, $N \cdot t_1 \cdot t_2$, d_p, d_q, t_1, t_2, e_{t_1} *and* e_{t_2}
1 *Let* $\mathsf{S_p} := \mathsf{m}^{\mathsf{d_p}} \bmod \mathsf{p} \cdot \mathsf{t_1}$
2 *Let* $\mathsf{S_q} := \mathsf{m}^{\mathsf{d_q}} \bmod \mathsf{q} \cdot \mathsf{t_2}$
3 *Let* $\mathsf{S} := \mathsf{CRT}(\mathsf{S_p}, \mathsf{S_q}) \bmod \mathsf{N} \cdot \mathsf{t_1} \cdot \mathsf{t_2}$
4 *Let* $\mathsf{c_1} := (\mathsf{m} - \mathsf{S}^{\mathsf{e_{t_1}}} + 1) \bmod \mathsf{t_1}$
5 *Let* $\mathsf{c_2} := (\mathsf{m} - \mathsf{S}^{\mathsf{e_{t_2}}} + 1) \bmod \mathsf{t_2}$
6 *Let* $\mathsf{Sig} := \mathsf{S}^{\mathsf{c_1} \cdot \mathsf{c_2}} \bmod \mathsf{N}$
7 *output* Sig

Higher Order Fault Attacks

- The adversary induces several faults during the processing.
- For instance, a second fault can be used to circumvent a countermeasure.

Cinquième partie V

Conclusion

Conclusion

- A wide variety of attacks exist : the adversary type must hence be precisely defined to adapt the security accordingly.
- Security must be considered during the conception phase.
- Security developers must find the better compromise between performances and security according to the application context

Sixième partie VI

Annexe

Additions Chain

Idea : minimize the number of multiplications in exponentiation.

Definition

An addition chain V of size s for an integer d is a sequence of positive integers $u_0, u_1, \ldots, u_s$ associated with a second sequence $w_1, \ldots, w_s$ such that $w_i = (i_1, i_2)$, $0 \leq i_1, i_2 < i$, and :

- $u_0 = 1$ *and* $u_s = d$
- $\forall u_i, 1 \leq i \leq s, u_i = u_{i_1} + u_{i_2}$

Addition chains are not unique for a given exponentiation. The inary representation can be viewed as a particular case.

Exponentiation

Algo Additions Chain

INPUT(S) : $g, V = (u_0, u_1, \ldots, u_s)$ addition for d
OUTPUT(S) : g^d

1: $g_0 = g$
2: **for** $i = 1$ s **do**
3: $\quad g_i = g_{i_1} \times g_{i_2}$
4: **return** g_s

s MULT