Secure computations using smart cards

Louis Goubin
December 19, 2006
Trento (Italy)

✦ An example: RSA

RSA Cryptosystem (1977)

de facto standard of public-key cryptosystems

p, q: primes, $n = pq$, $ed = 1 \bmod (p-1)(q-1)$,

e, n: public key, d: secret key, (factoring, n: 1024 bits)
M: message, $M \in \{0,1,2,\ldots,n-1\}$.

$$\boxed{\textbf{Encryption: } C = M^{e} \bmod n}$$

e: small ($2^{16}+1$)

$$\boxed{\textbf{Decryption: } M = C^{d} \bmod n}$$

d: large ($d>n^{1/2}$)

Fast Exponentiation

The binary representation of d =$d[k-1]2^{k-1} + d[k-2]2^{k-2} + \ldots + d[1]2^{1} + d[0]2^{0}$, where d[k-1]=1.

Left-to-right binary method

Input C, n, d

Output C^{d} **mod n**

```
X= C;
For i=k-2 to 0
   X = X^2 mod n;
  if d[i]=1, then X=X*C mod n;
Return X
```

cubic complexity $O((\log n)^3)$.
- we need about 1500 modular multiplications for 1024-bit n,d on average.

d = 179769313486231590772930519078902473361797697894230657273430081157732639445209167262771634937140456477800995856
4863673560357494227785840418926558467439899258695049140360821770965996851973903412635215659390188627764072341203
1668285970266526289737711820513944871376325649575655785893257302729658745304709432808

RSA Decryption using Chinese Remainder Theorem

$\mathbf{M = C^d \bmod n}$ n

$2^3 = 8$ time faster

dp = d mod (p-1)

$\mathbf{C_p = C \bmod p}$

$\mathbf{M_p = C_p^{dp} \bmod p}$ p

dq = d mod (q-1)

$\mathbf{C_q = C \bmod q}$

$\mathbf{M_q = C_q^{dq} \bmod q}$ q

$\mathbf{M = M_p + pv \bmod n, \ \ v = (M_q - M_p)\, p^{-1} \bmod q}$ n

Garner's algorithm

RSA decryption using the CRT can be computed about 4 times faster than the original decryption.

RSA with CRT

Algorithm RSA_Decryption_CRT (n=pq)
Input C,n,p,q,dp,dq,p_inv_q
Output M

1: $Mp = C^{dp} \bmod p$;
2: $Mq = C^{dq} \bmod q$;
3: $v = (Mq - Mp)\, p\_inv\_q \bmod q$;
4: $M = Mp + pv$;
5: Return M

Pre-computation avoiding inversion

PKCS #1, http://www.rsasecurity.com/rsalabs/pkcs/

✦ Timing Attacks

What are Timing Attacks ?

- The term “Timing Attack” was first introduced at CRYPTO'96 in Paul Kocher's paper
- Few other theoretical approaches without practical experiments up to the end of 97’
- Theory was put into practice in early 98’
- Timing attacks belong to the large family of "side channel" attacks

What are Timing Attacks ?

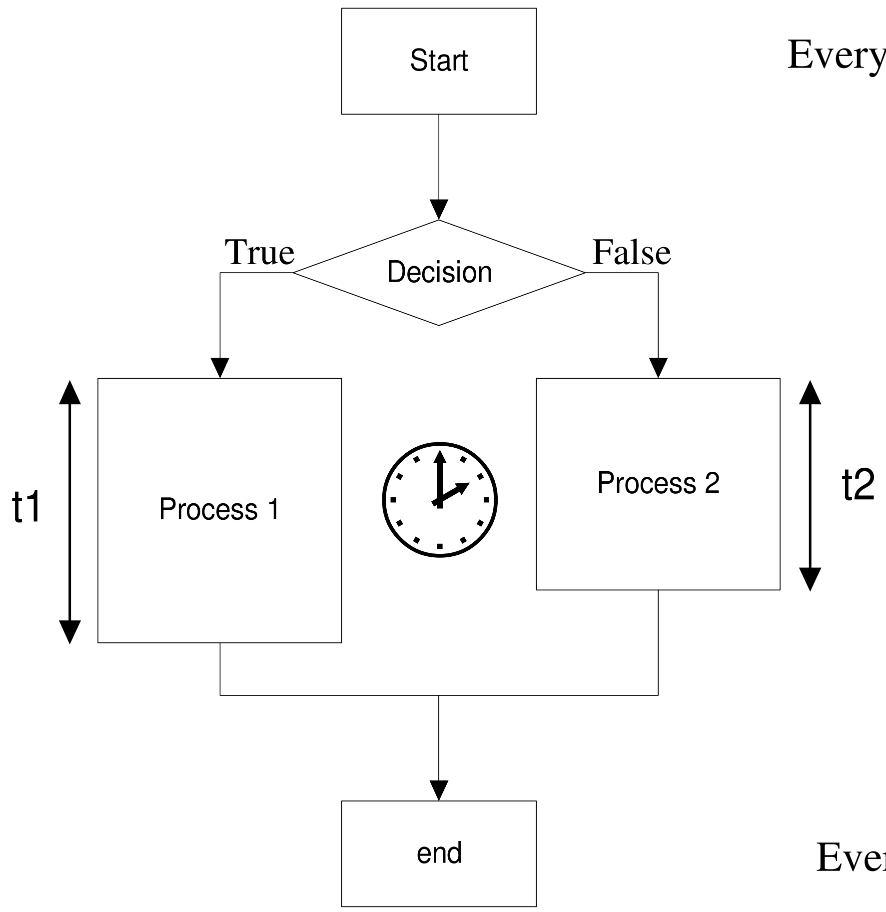
✦Principle of Timing Attacks :

- Secret data are processed in the card
- Processing time
  - depends on the value of the secret data
  - leaks information about the secret data
  - can be measured (or at least their differences)

✦Practical attack conditions

- Possibility to monitor the processing of the secret data
- Have a way to record processing times
- Have basic computational & statistical tools
- Have some knowledge of the implementation

What are Timing Attacks ?



Everything performed unconditionally before the test

A test based on secret data is performed
that leads to a boolean decision

Depending on the boolean condition,
the process may be long (t1) or short (t2)

Everything performed unconditionally after the test

Timing attack on RSA

- Timing Attacks: by precisely measuring the time it takes the smartcard to perform an decryption, Marvin can discover *d*.
- "repeated squaring algorithm", compute $C=M^d \bmod N$. $d=d_n d_{n-1} \ldots d_0$
  - Set z equals to M and $C=1$. For i=0,...n do:
  - if $d_i=1$ set $C=C*z \bmod N$
  - set z equal to $z^2 \bmod N$

  At the end, C has the value $M^d \bmod N$
- To mount attack, Marvin asks the smartcard to generate signatures on a large number of random messages $M_1, M_2, \ldots M_k \in Z_N^*$ and measure the time T_i it takes to generate each signature.

Timing attack on RSA

- Timing Attack
- If d_1=1, smartcard computes $Cz=MM^2 \bmod N$ and, Otherwise it does not. Let t_i be the time it takes the smart card to compute $M_iM_i^2 \bmod N$. The t_i's differ from each other and depends on M_i. Marvin measures them offline.
- When d_1=1, the two ensembles $\{t_i\}$ and $\{T_i\}$ are correlated. when d_1=0, they behave as independent random variables. By measuring the correlation, Marvin can determine d_1=1 or 0.
- Continuing in this way, he can discover d_2,d_3... and so on.
- Solutions: 1) add appropriate delay s.t. modular exponentiation always takes a fixed amount of time. 2) Rivest's blinding trick.
- Kocher's Power cryptanalysis?

Power Analysis Attacks

Power Analysis: Basic Principles

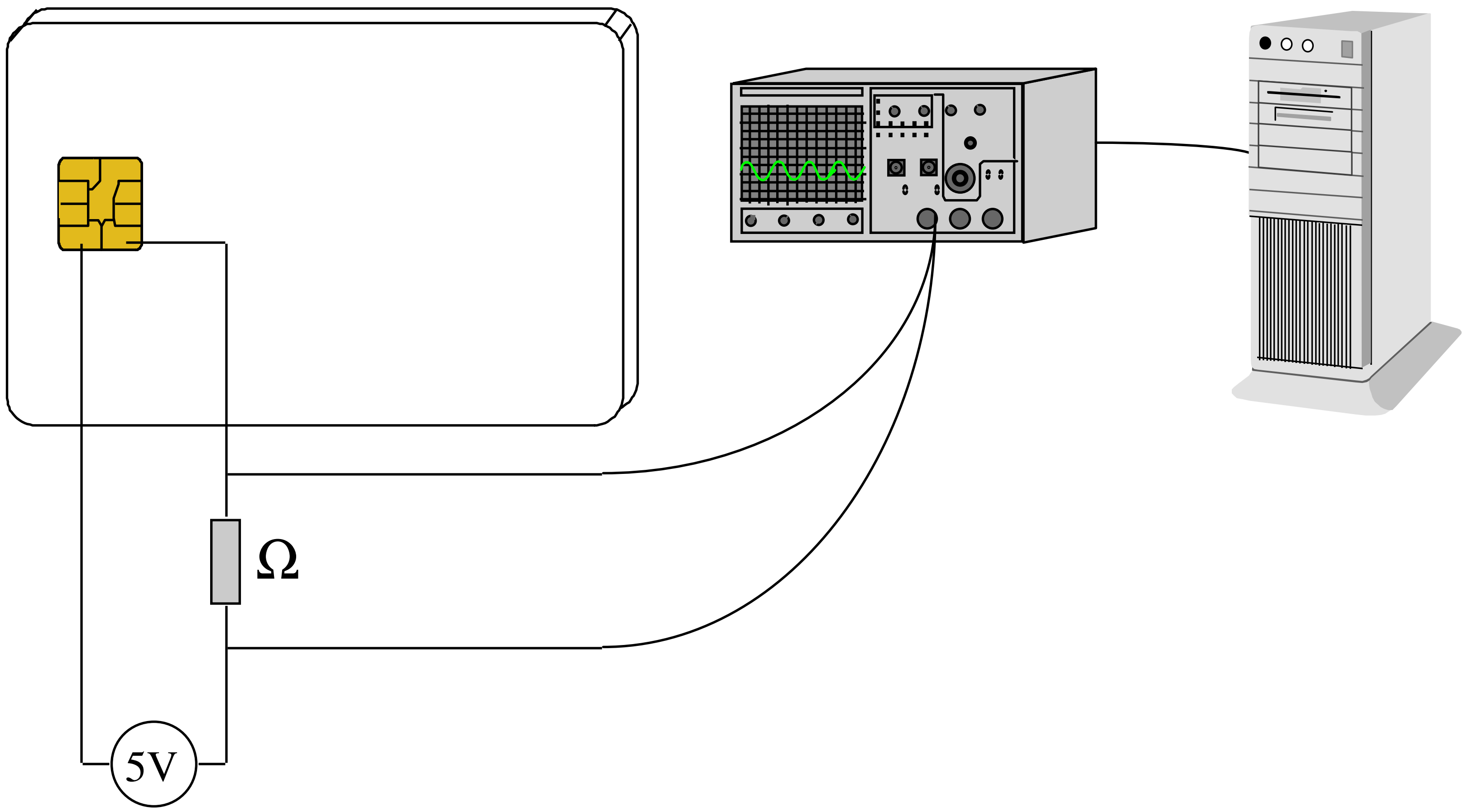
- ICC's Power Consumption leaks information about data processing
  - Power Consumption = f(processing, data)

- Deduce information about secret data and processing
  - empirical methods
  - statistical treatment

- Example : reverse engineering of an algorithm
  - The algorithm structure
  - Electrical signatures

- Single Power Analysis (SPA)
  - Attack against the DES key schedule
  - Attack against RSA

Power Analysis Tools



Side Channel Attacks

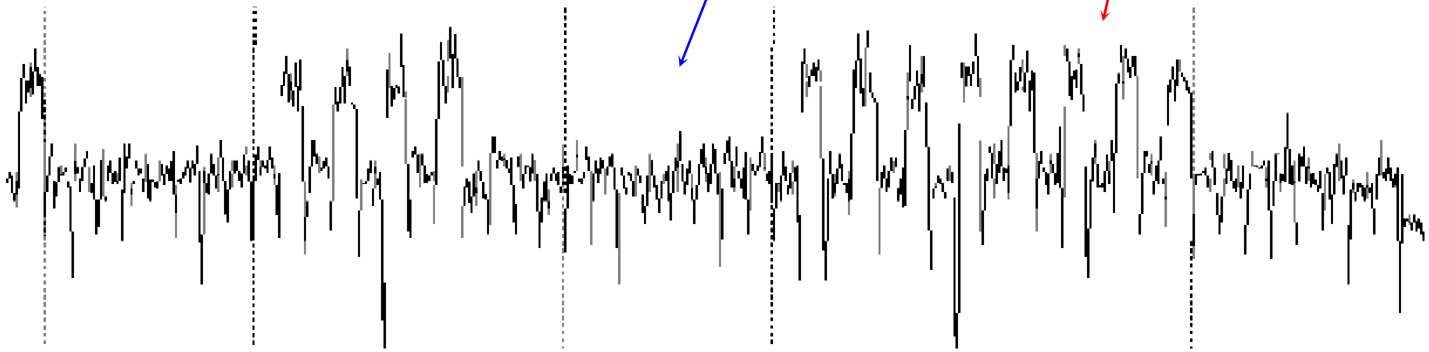
Left-to-right binary method

Input **M, n, d**

Output **M^d mod n**

```
X= M;
For i=k-2 to 0
   X = X*X mod n;
  if d[i]=1, then X=X*M mod n;
Return X
```

The time or the power to execute Squaring and Multiplication are different (side-channel information).



Cited from Clavier et. al, Universal exponentiation algorithm: A first step towards provable SPA-resistance, *CHES 2001*, LNCS 2162, pp. 300-308, 2001

SPA attack on RSA

- basic “square and multiply” algorithm
- exponent bits scanned from MSB to LSB (left to right)

```
Let k = bitsize of d (say 1024)

Let s = m

For i = k-2 down to 0

  Let s = s*s mod n (SQUARE)

  If (bit i of d) is 1  then
   Let s = s*m mod n (MULTIPLY)
  End if

End for
```

Example : $s = m^9 = m^{1001b}$

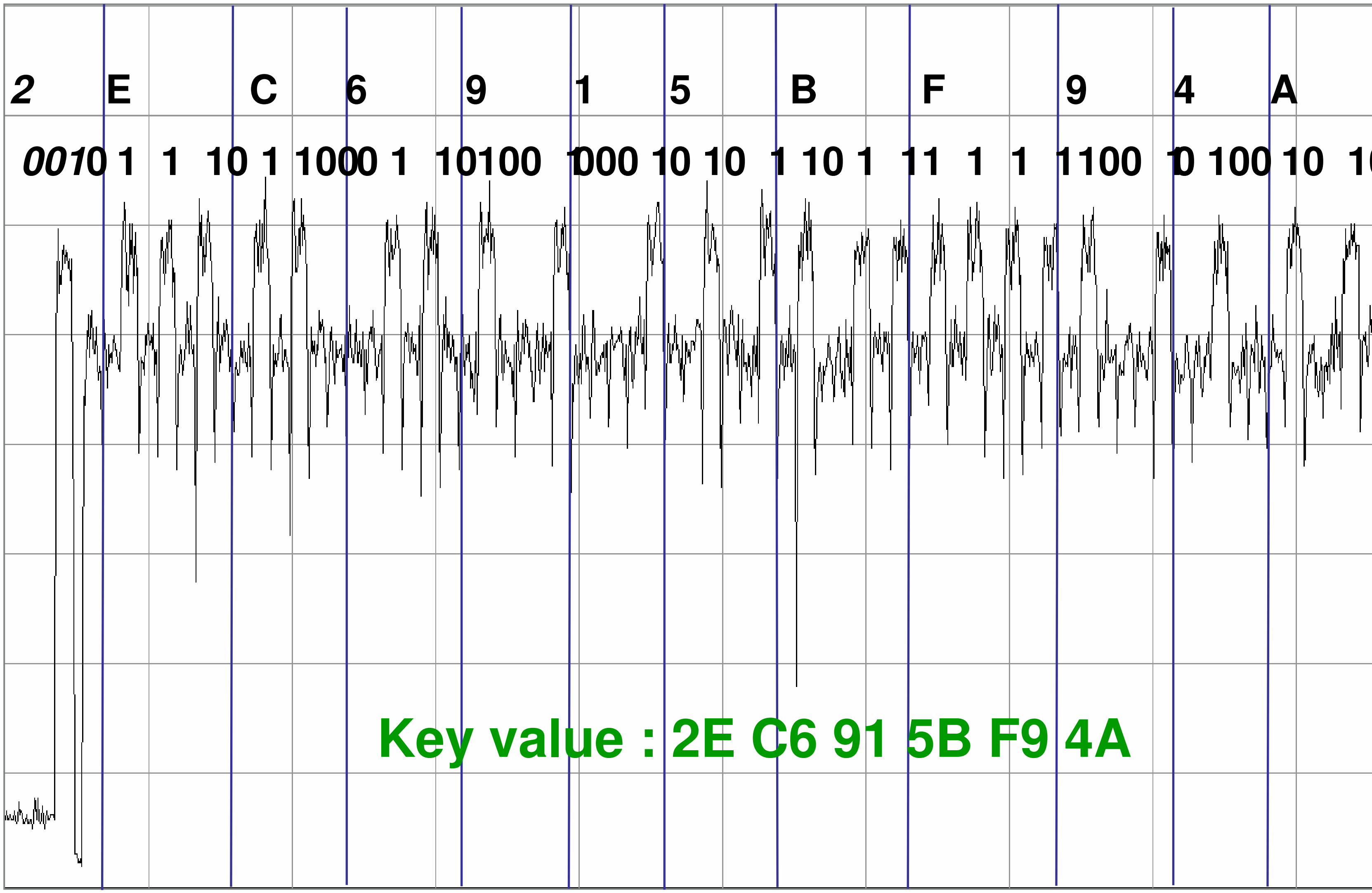
init (MSB 1) $s = m$

round 2 (bit 0) $s = m^2$

round 1 (bit 0) $s = (m^2)^2 = m^4$

round 0 (bit 1) $s = (m^4)^2 * m = m^9$

SPA attack on RSA

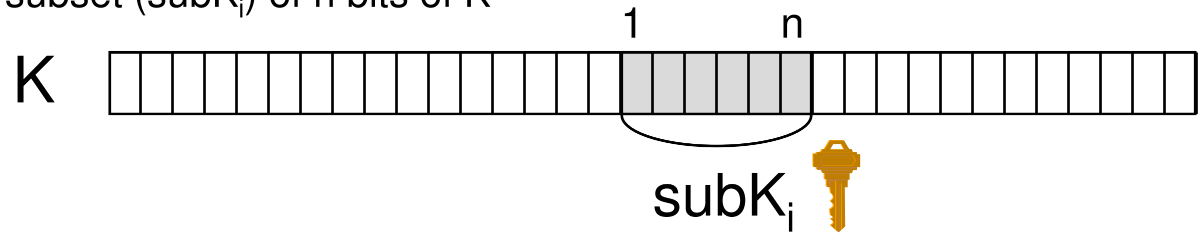


Differential Power Analysis

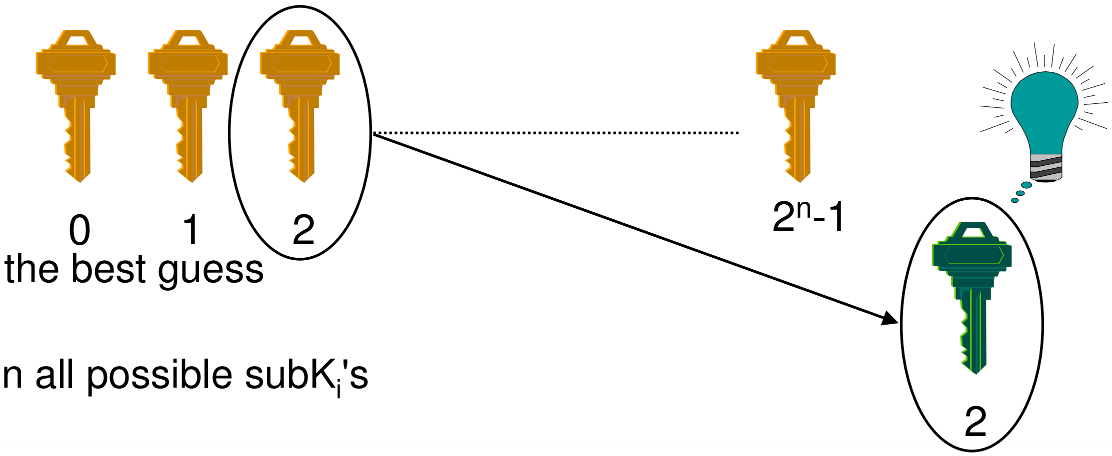
- required number of acquisitions : 500 to 10,000
- prerequisite
  - physical access to the card under attack
  - access to either plaintext M or ciphertext C
  - varying plaintext and constant key
  - algorithm specifications (MANDATORY)
- cost
  - A few dollars (to a few thousands)
  - A few days training
  - Average good level of expertise
  - Chip and implementation independent

Differential Power Analysis

- description :
  - choose a subset ($subK_i$) of n bits of K



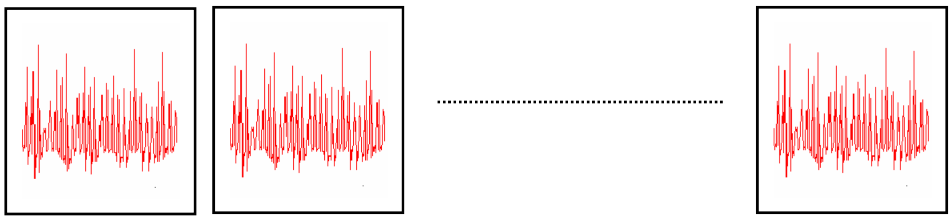
  - perform a statistical test for each possible value of a $subK_i$



  - Choose the best guess
  - Iterate on all possible $subK_i$'s

Differential Power Analysis

- DPA statistical test :
  - a batch of data acquisitions for various messages M_k

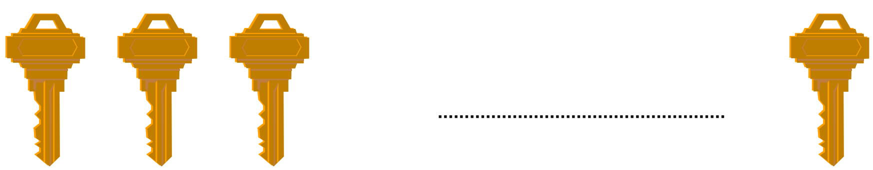


0 1 ...... k

  - the corresponding plaintext M_k or the cipher text C_k

dfdsffb	fdgcxv	......	lklkjlsdq
M_0	M_1		M_k

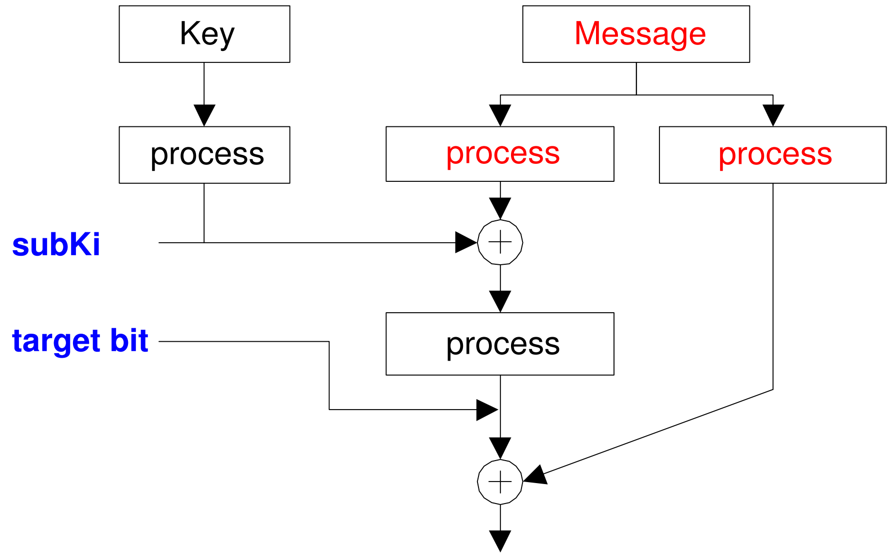
  - the values of the subK_i



0 1 2 ...... 2^n-1

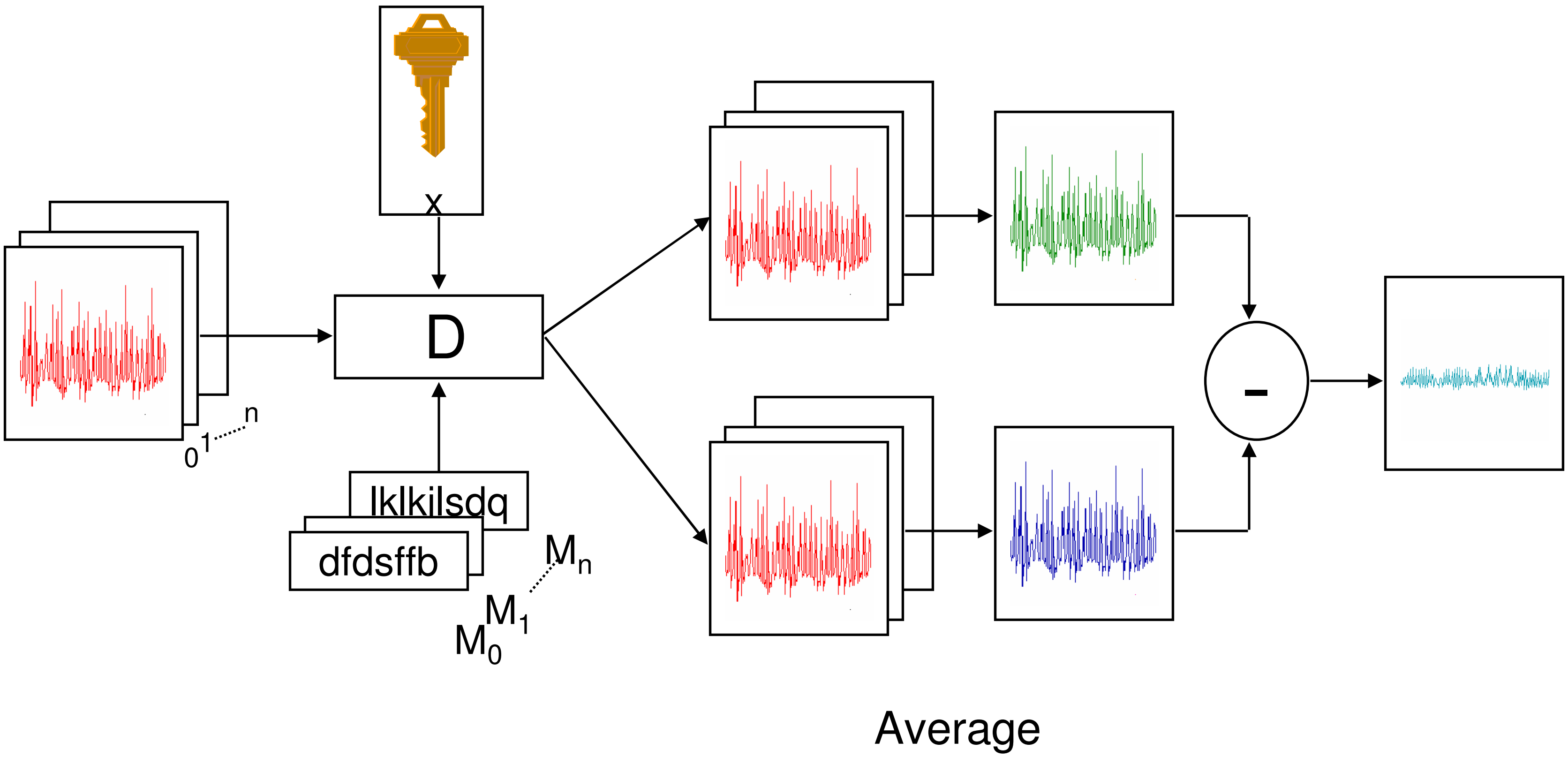
Differential Power Analysis

- DPA statistical test :
  - selection function D :
    - sort curves according to M_k or C_k for each value of a $subK_i$
    - output = image of a target bit of the algorithm



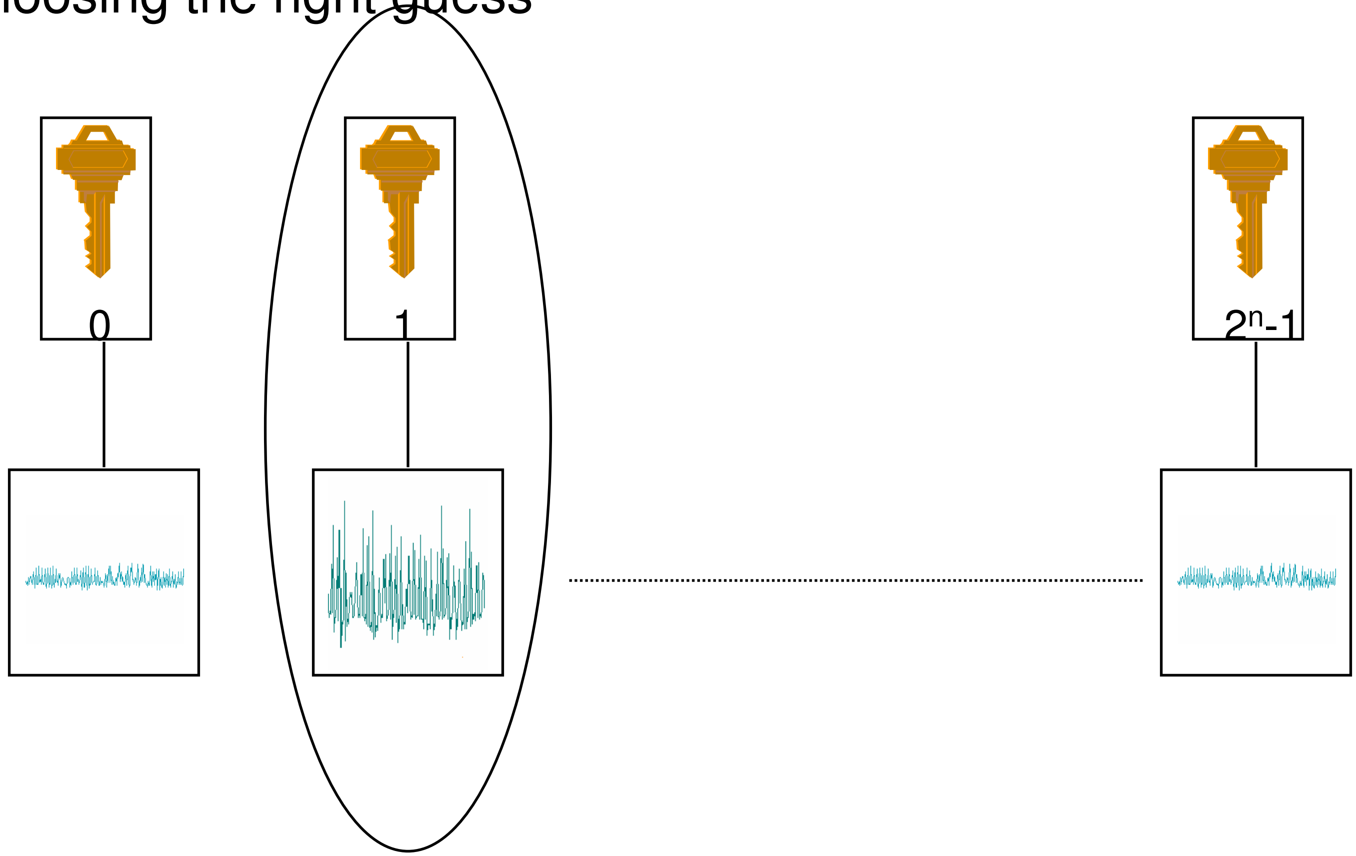
Differential Power Analysis

- data processing for a value x of a $subK_i$:



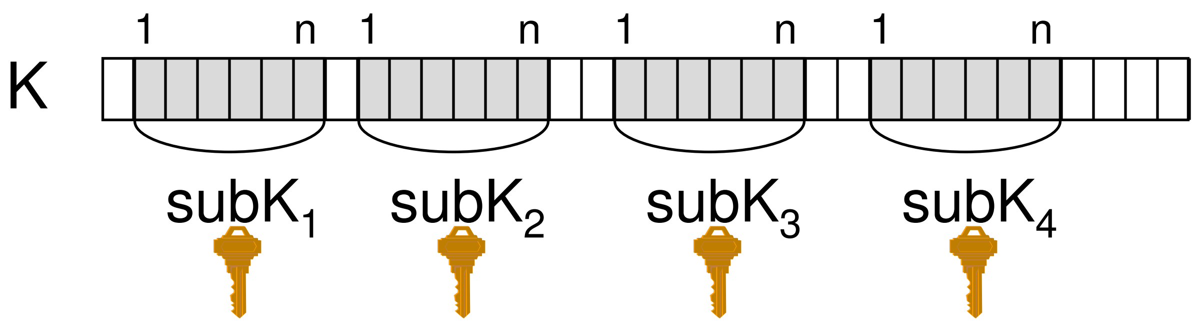
Differential Power Analysis

- Choosing the right guess

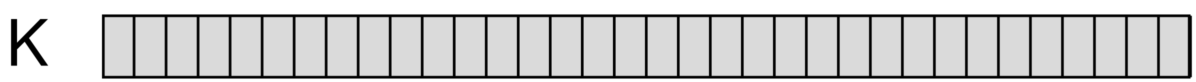


Differential Power Analysis

- iterate on all possible sub-keys :

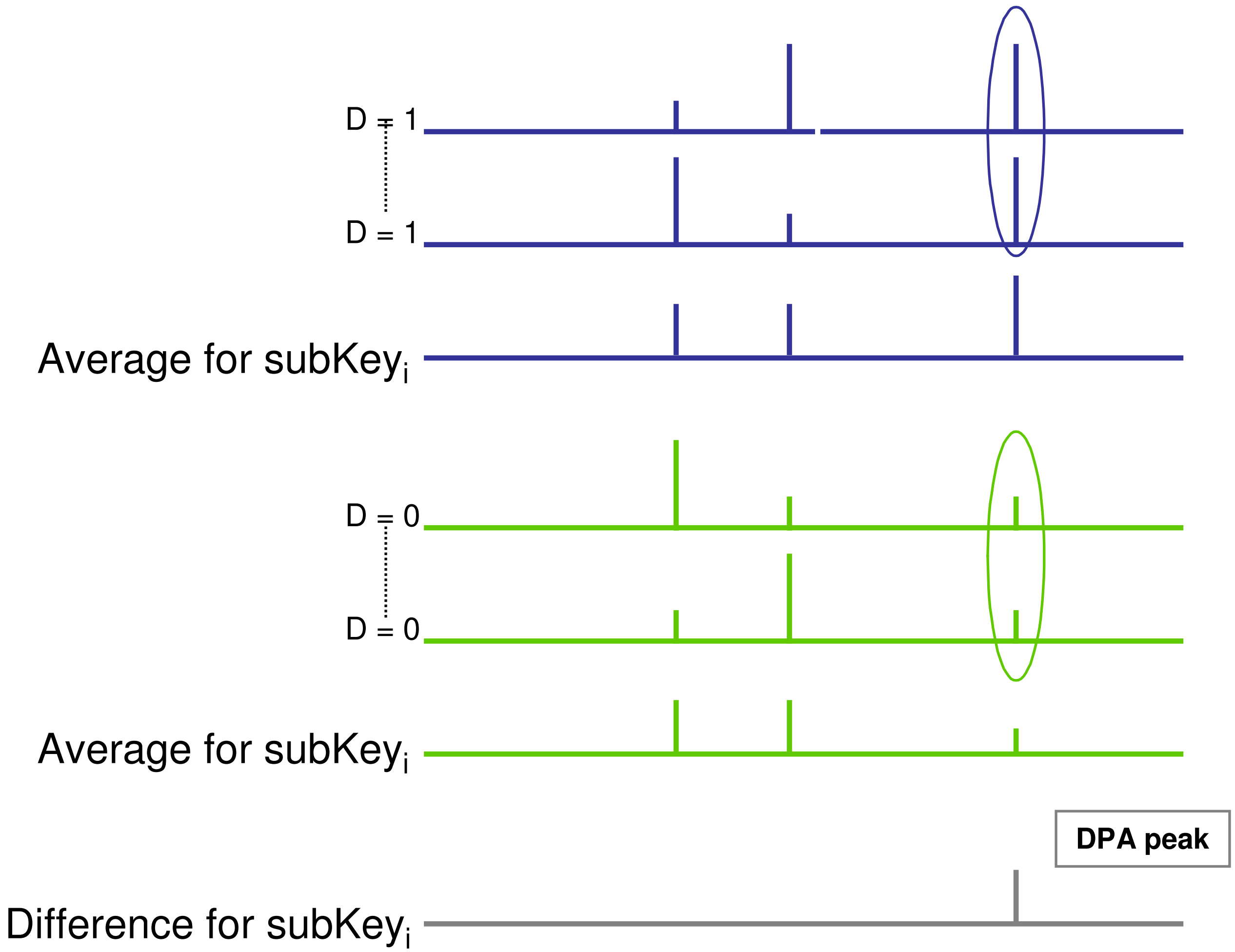


- find the remaining bits through exhaustive search



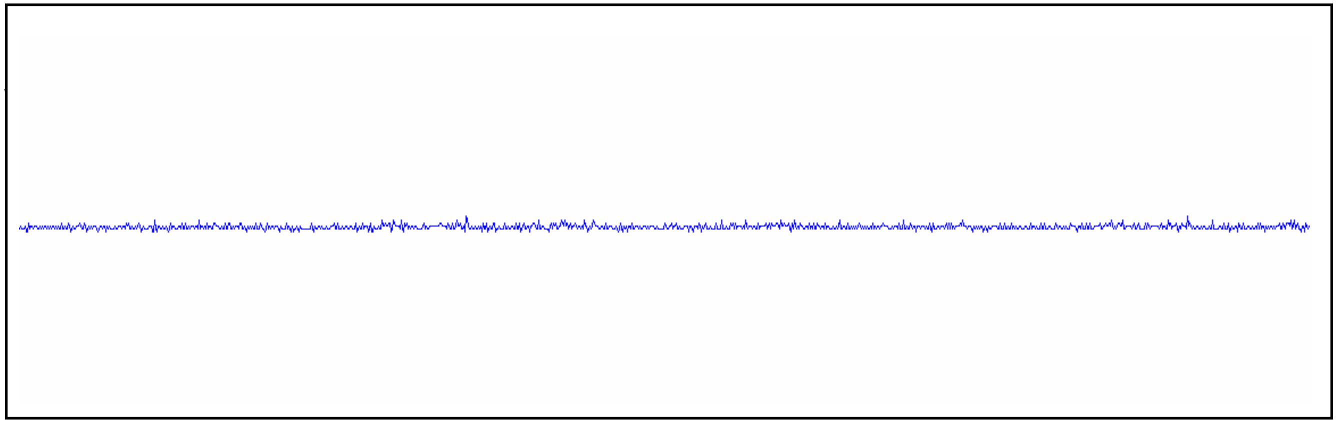
Differential Power Analysis

- How does it work ?

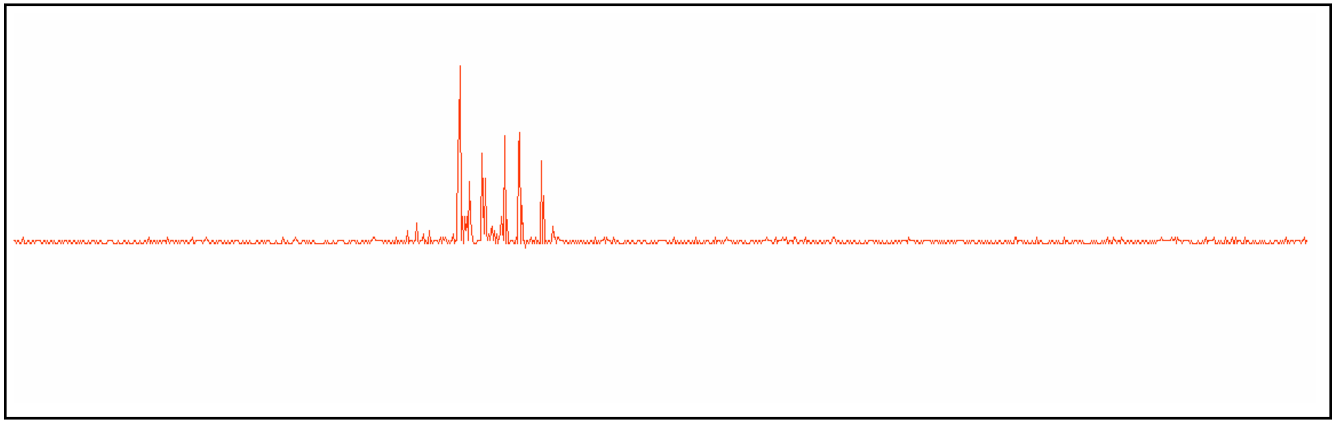


gemalto

Differential Power Analysis



- right $subK_i$



Counter-measures

- Add noise
- Scramble power consumption or stabilize it
- Randomize all sensitive data variables with a fresh mask for every execution of an algorithm

- Randomize, randomize, randomize …
  - Secret keys
  - Messages
  - Private exponents
  - Bases
  - Moduli

Conclusion on Power Analysis Attacks

- Naïve smartcard implementations of cryptosystems can leak secret data.
- Power Analysis Attacks
  - – target symmetric and asymmetric cryptosystems
  - – practical, ‘fast’ and cheap
  - – difficult to circumvent
  - – countermeasures may impact efficiency.

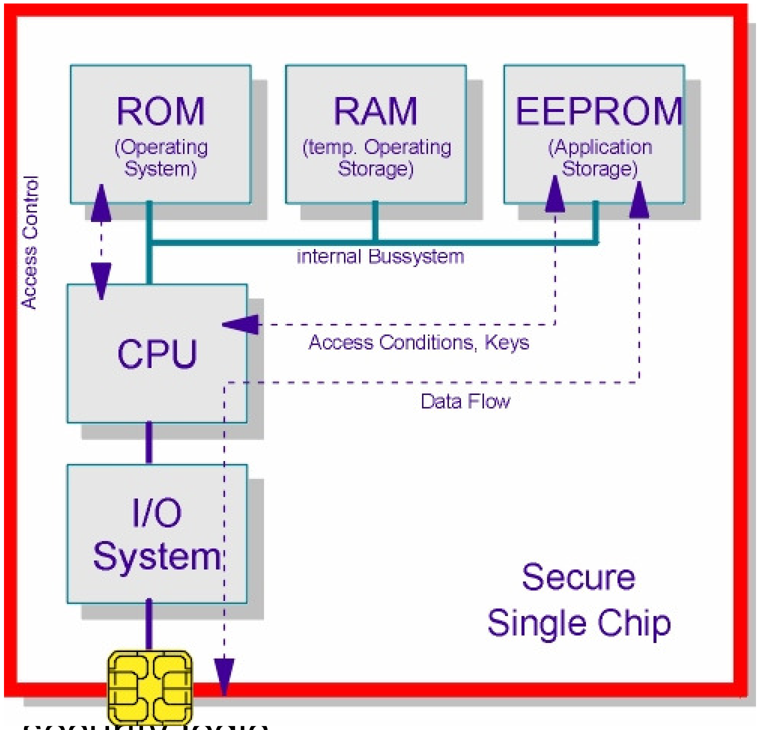
✦ Executing external code

Need for a Tamper-Proof Environment

- Many emerging applications require physical security as well as conventional security against software attacks
  - Digital Rights Management (DRM): illegal copies of protected digital content
  - Mobile agent applications: sensitive electronic transactions are performed on untrusted hosts
- Conventional approach: build processing systems containing processor and memory elements
  - Within a private and tamper-proof environmnent
  - Typically implemented using active intrusion detectors

Conventional Smart Cards

- 8, 16 or 32-bit CPU
  - Typically 10 MHz
- RAM: 2-8 Kbytes
- ROM: 100-200 Kbytes
  - Contains the code
- E^2PROM: 32-128 Kbytes
  - Contains the data
- Optional:
  - Random Noise Generation, sensors, security logic
  - Modular Exponentiations Unit or Co-processor
  - Random Generator



gemalto

Limitations of Conventional Solutions

- Providing high-grade tamper-resistance can be quite expensive
- System computation power is limited by the components that can be enclosed in a small tamper-proof package
  - The applications of these systems are limited to perform a small number of security critical operations
- These processors are not flexible
  - E.g. their memory or I/O cannot be upgraded easily

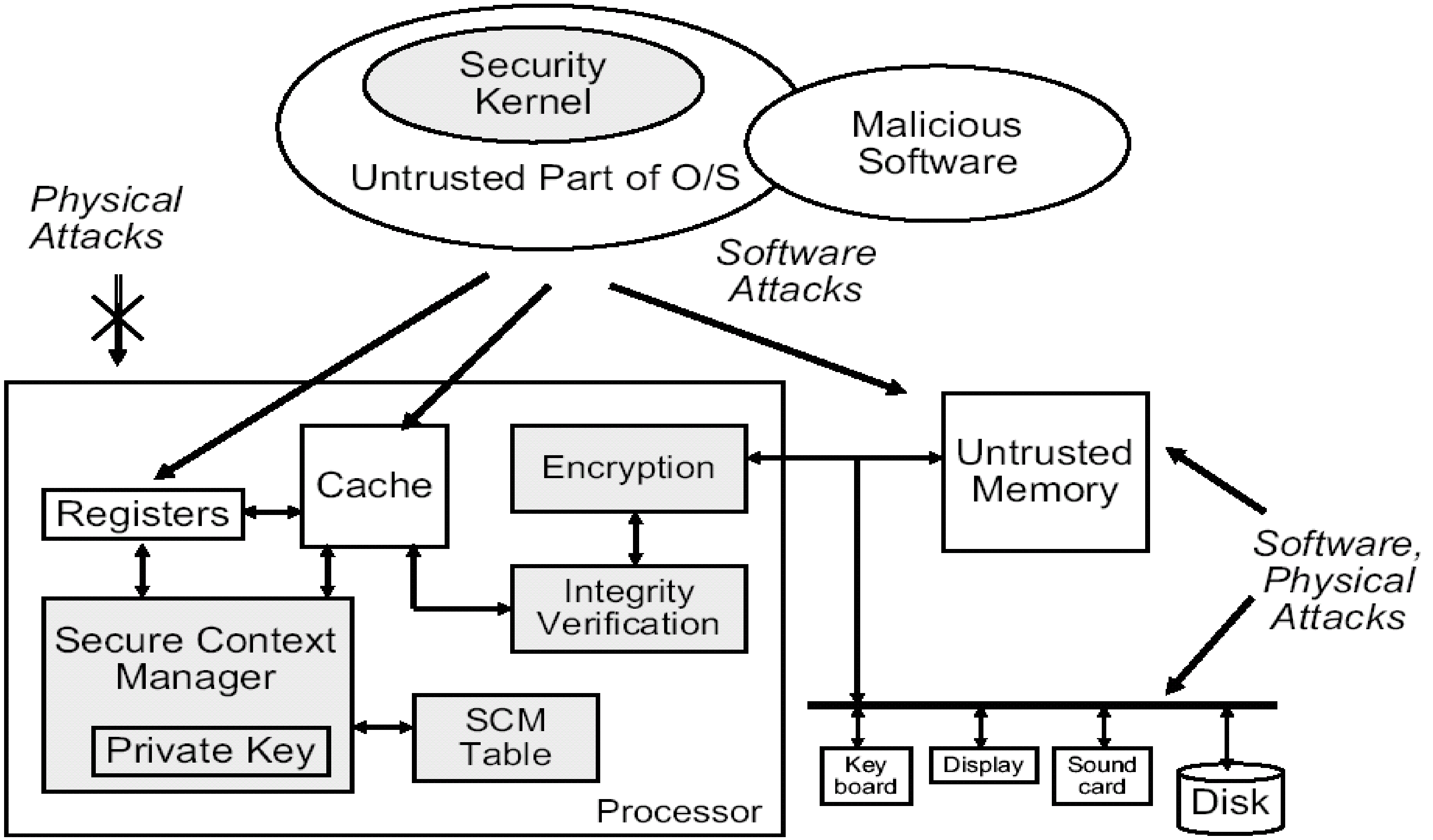
Emerging New Solutions

- Just requiring tamper-resistance for a single processor chip would significantly enhance the amount of secure computing power
- Makes possible applications with heavier computation requirements
- Secure processors have been recently proposed, where
  - only a single processor chip is trusted
  - the operations of all other components including off-chip memory are verified by the processor

Private Tamper Resistant (PTR) Environment

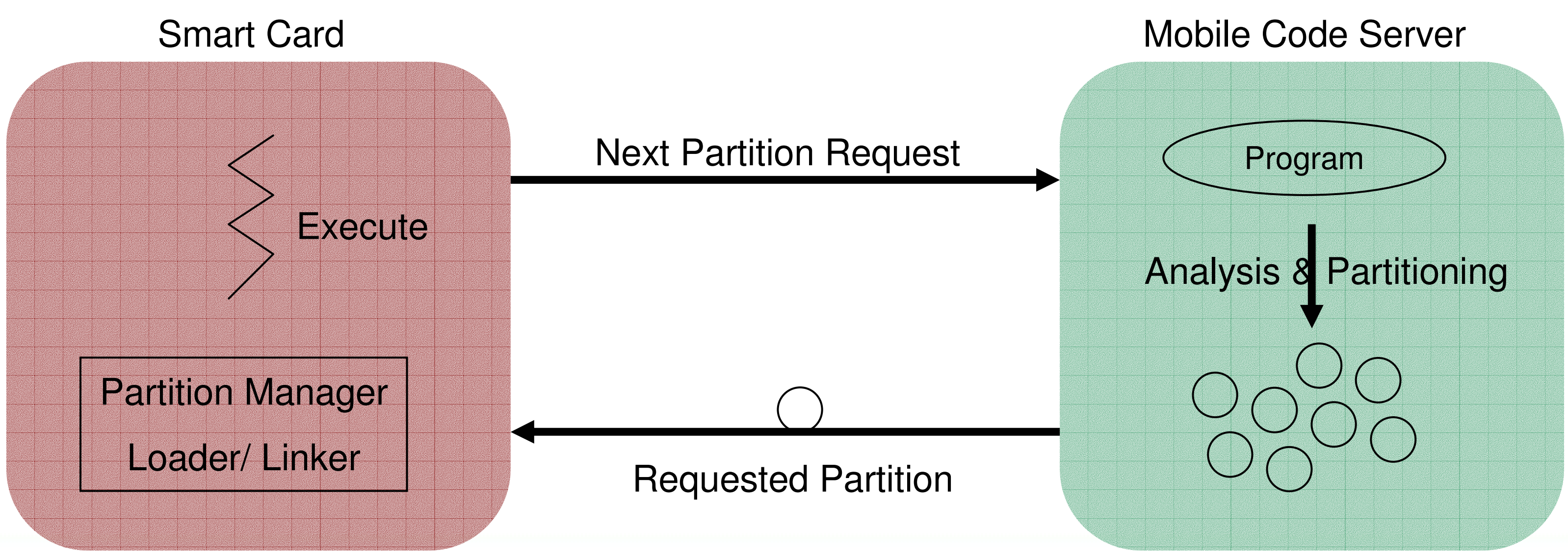
- To prevent an attacker from tampering with the off-chip untrusted memory, two main primitives have to be developed.
- Memory integrity verification
  - The processor monitors the memory from any form of corruption
  - If any is detected, then the processor aborts the tasks that were tampered with to avoid producing incorrect results
- Memory Encryption
  - Ensures the privacy of data stored in the off-chip memory

Secure Computing Model



Secure code execution

- Taken from *"Tamper-Resistant Whole Program Partitioning"* (Zhang, Pande & Valverde, 2003 ACM SIGPLAN conference on Language, compiler, and tool support for embedded systems)
  - Download application via network from server
  - A program is **partitioned**.
  - Why partition?



gemalto

Problem: Program Partitioning

- Easiest way to partitioning
  - Partitioning by basic blocks

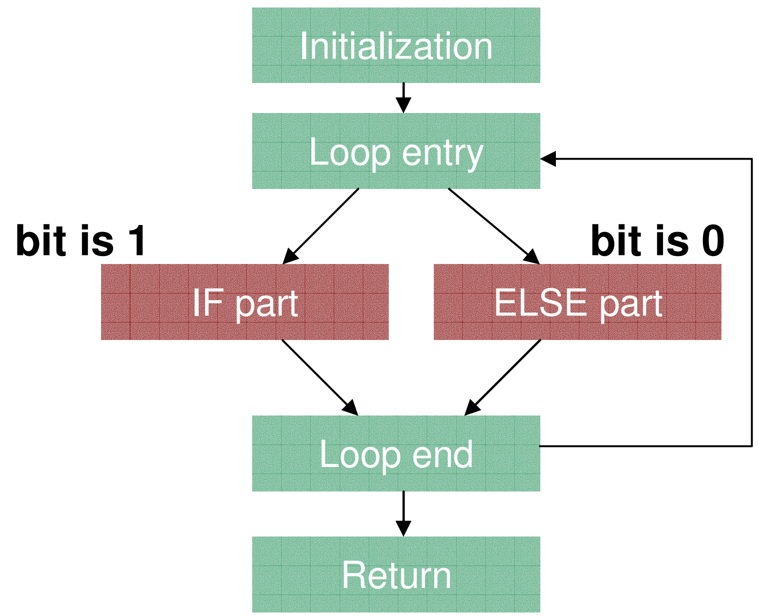
- However, simple partitioning may reveal control flow, which is dangerous for security

Problem: Program Partitioning

- RSA private-key operation

computing $R = y^x \bmod n$

```
let s_0 = 1
for k=0 upto k-1
    if (bit k of x) is 1 then
        let r_k = (s_k * y) mod n
    else
        let r_k = s_k
    let s_{k+1} = r_k^2 mod n
end for
return r_{w-1}
```



Partition transmission sequence: IF → ELSE → IF → IF → ELSE

We can guess key x as 10110 or 01001

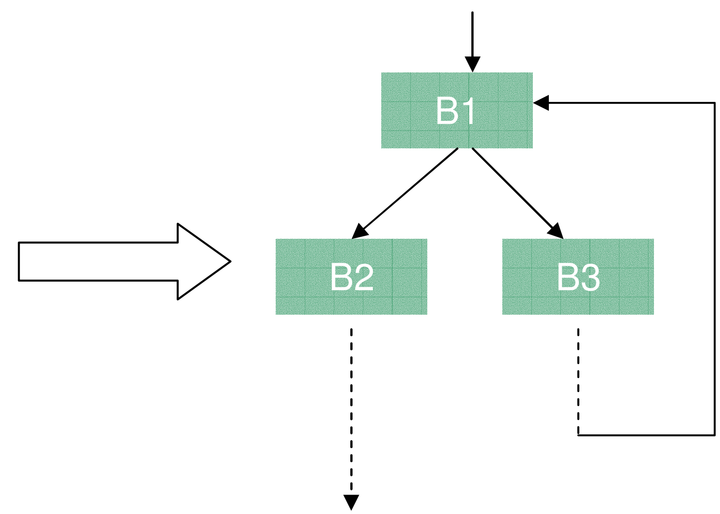
Safe Partitioning

- Safe partitioning
  - Partitioning that does not reveal control flow.
  - How to partition?
  - How to transmit through network?
  - How to manage partitions on smart card?

Partition sequence through network

…, B1, B3, …., B1, B2, …

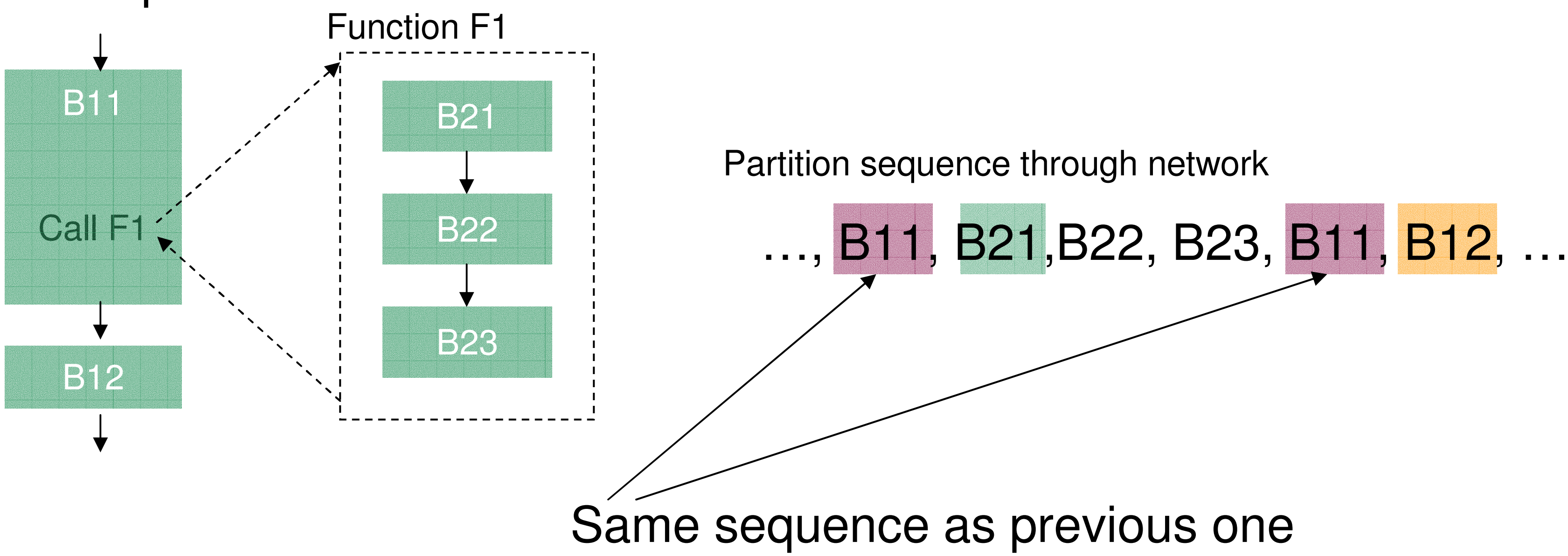
B1 can be guessed as control block



This is not a safe partitioning.

Safe Partitioning

- Exception: a function call



But, this partitioning is not dangerous.

Safe Partitioning

Same

…, B11, B21,B22, B23, B11, B12, …

Different

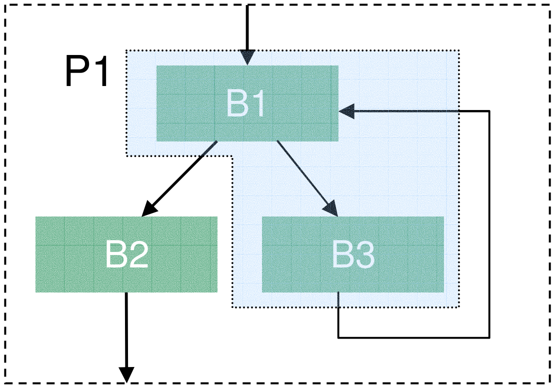
- Safe partitioning
  - Do not generate this kind of sequence through network
  - Except previous case

Code Partitioning

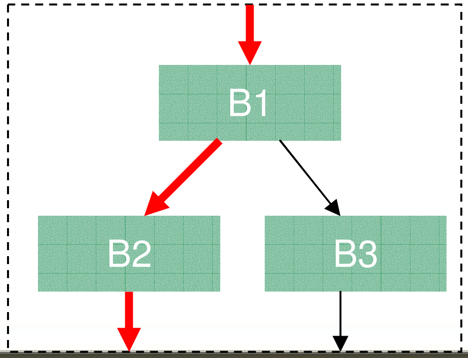
- Partition management policy in smart card
  - Keep nothing received policy
  - Discard partition after its execution (does not cached)
  - To avoid problem when there is a long function call chain
    - If we cache partitions that are to be executed when a function returns, …
- Code partitioning algorithm
  - For non-recurring function
    - Executed at most once (e.g., main function, initialization func.,)
  - For recurring function
    - Executed multiple times

Code Partitioning

A non-recurring function



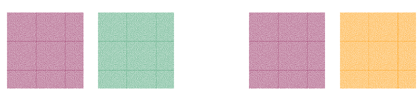
A non-recurring function



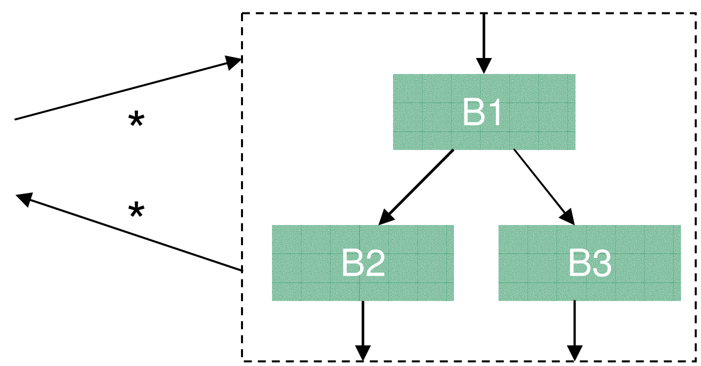
Code Partitioning

- For **recurring** functions
  - Step 1: If there is loop → merge loop body
    - Same reason for non-recurring function
    - Then CFG becomes acyclic
  - Step 2: If CFG does not contain loop (= acyclic)
    - **Not safe**
    - Since they are executed multiple times
    - Possible sequence

      B1, B2, …, B1, B3, …

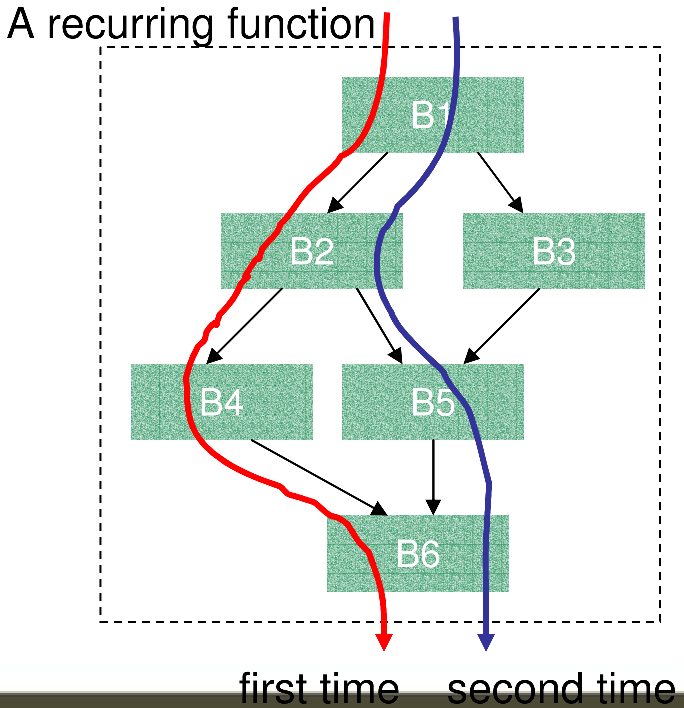


A recurring function



Code Partitioning

- If CFG in recurring function is acyclic
  - Server transmits whole partitions in one of the topological order (regardless of control flow)
  - Smart card discards the unnecessary partitions



Horizontal Ordering

$$\underbrace{B1, B2, \cancel{B3}, B4, \cancel{B5}, B6}_{\text{Same when seen by an attacker}}, \ldots, \underbrace{B1, B2, \cancel{B3}, \cancel{B4}, B5, B6}$$

Same when seen by an attacker

Improving Performance

- Too many communication overhead
- Solutions
  - Merging adjacent partitions
    - Greedy algorithm
    - Merge until sum of partitions exceed predefined limit
  - Function caching
    - Do not discard function after it is executed
    - Dynamic cache eviction (e.g., LRU) is not acceptable (high overhead)
    - → Static caching by compiler