MEAD Education S.A.

Advanced Engineering Course on

Cryptographic Engineering

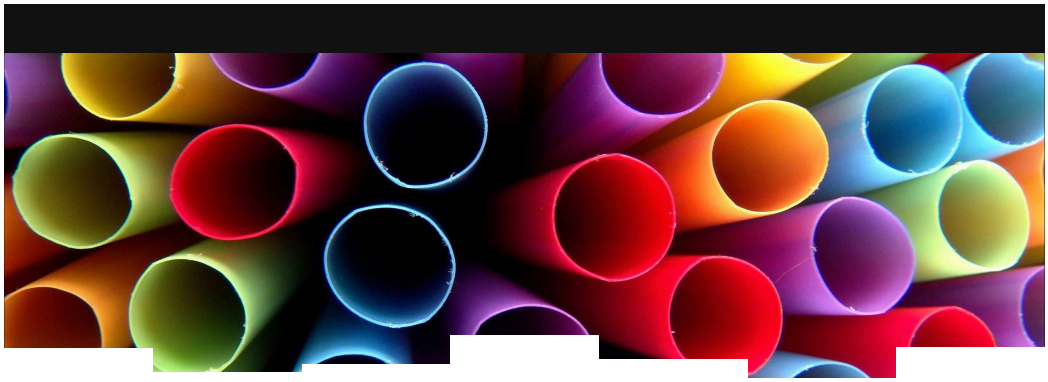
EPFL Premises, Lausanne, Switzerland
June 25-29, 2012

Thursday, June 28, 2012

Marc Joye, Technicolor, France

- **Side-Channel Attacks on Cryptographic Tokens**
- **Countermeasures for Preventing Side Channel Attacks**

Side-Channel Attacks on Cryptographic Tokens



Marc Joye

technicolor

Outline

1. Simple Power Analysis
2. Timing Attacks
3. Differential Power Analysis

Cryptographic Engineering, Lausanne, June 25-29, 2012

Part I:
Simple Power Analysis

Summary

Introduction to Power Analysis
- Experimental equipment
- Information leakage through the power

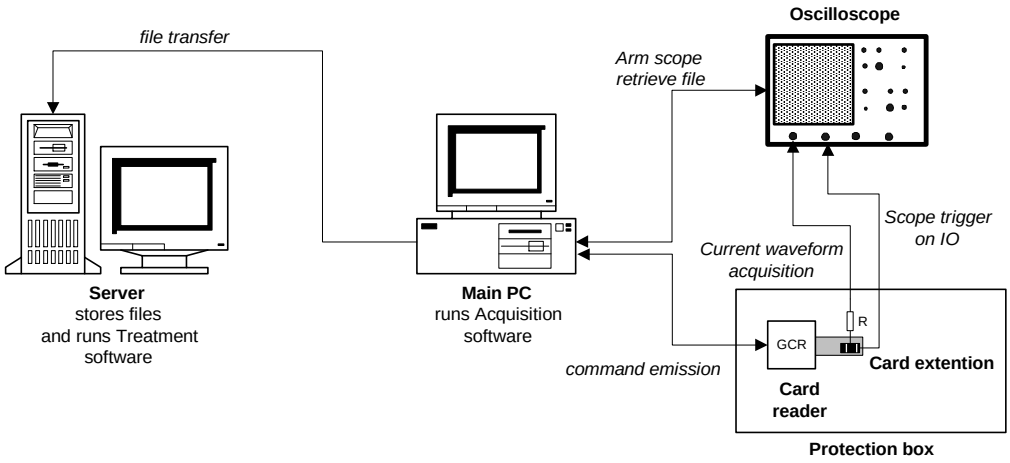
Example: reverse engineering of an algorithm
- The algorithm structure
- Electrical signatures

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

Conclusion
- Countermeasures

technicolor

Experimental equipment



Devices for monitoring the current consumption of a chip

technicolor

Information leakage

The power consumption of a chip depends on

- the manipulated data
- the executed instruction

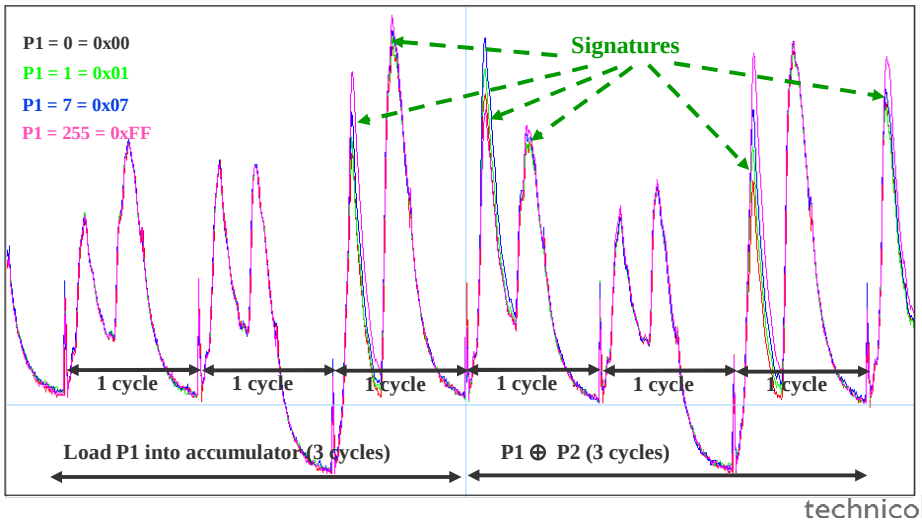
Leakage models

- Hamming Weight of the data, address, Op code
  - HW(0) = 0
  - HW(1) = HW(2) = HW(4) = HW(2^n) = 1
  - HW(3) = HW(5) = HW(6) = HW(9) = 2
  - …
  - HW(255) = HW(0xFF) = 8
- Transitions weight (flipping bits on a bus state) :
  - HW ($state_i \oplus state_{i-1}$)
- Other models, chips & technologies …

technicolor

Information leakage

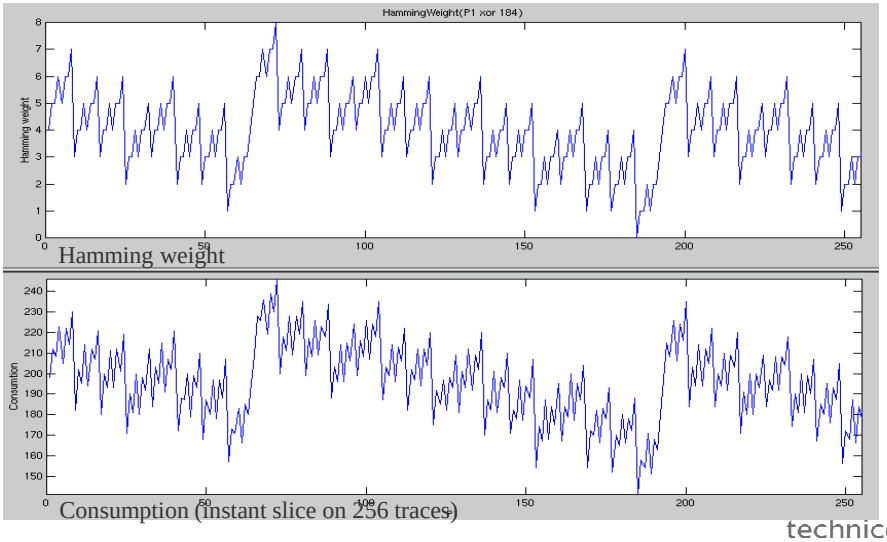
Load P1 and XOR with P2 = 0 (P1 ⊕ P2 with P1 = 0, 1, 7, 255)



technicolor

Information leakage

HW (P1 ⊕ 184), for P1 = 0, 255



technicolor

Power trace of an algorithm

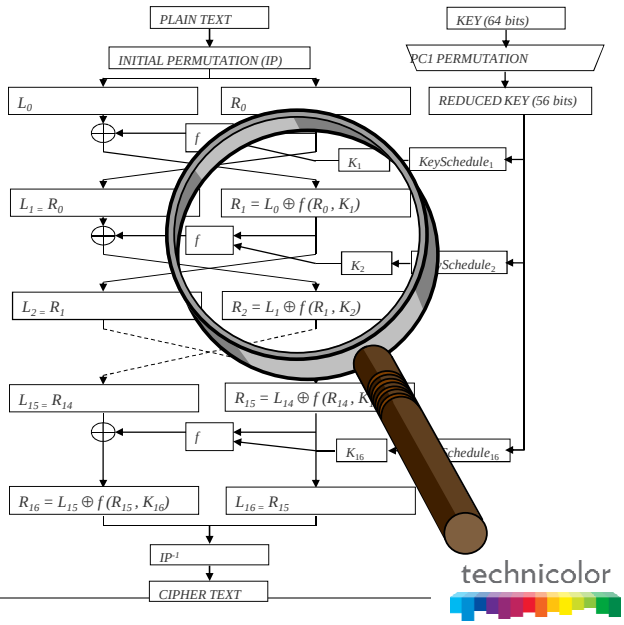
Typical SK block cipher

- key schedule
- Feistel scheme
- 16 rounds

Data

Encryption

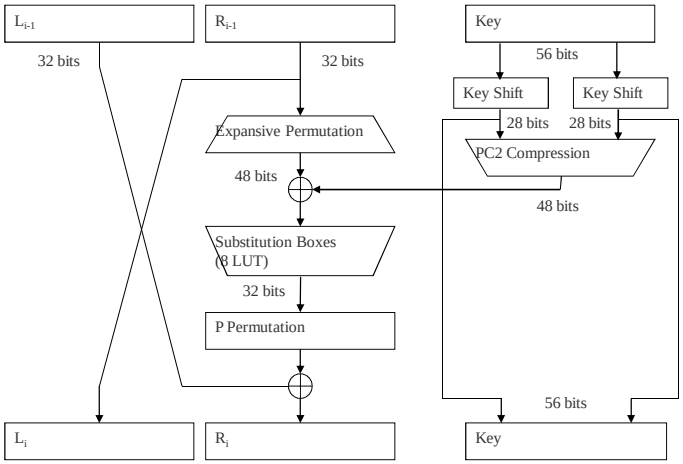
Standard



technicolor

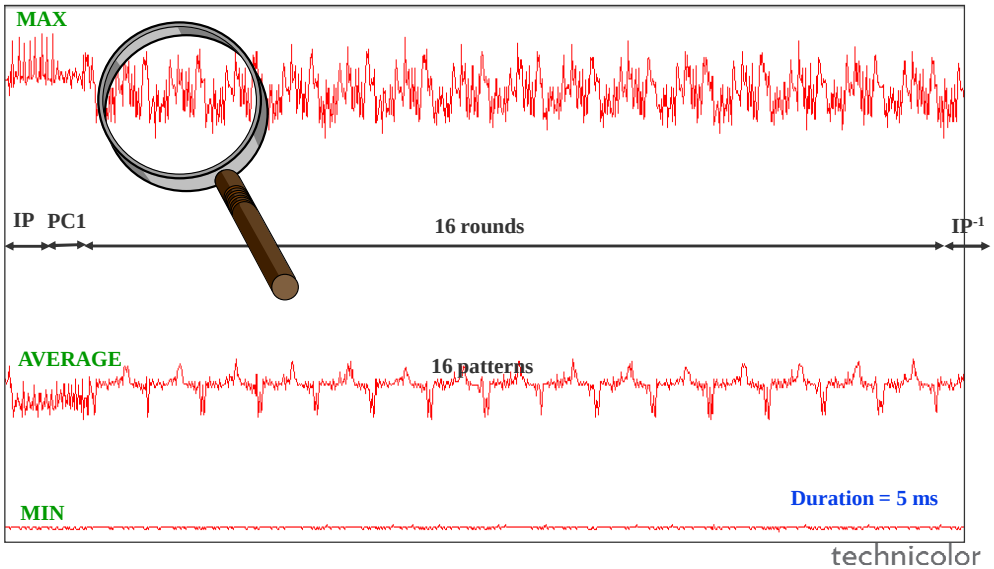
Power trace of an algorithm

Content of a DES round (with key schedule)



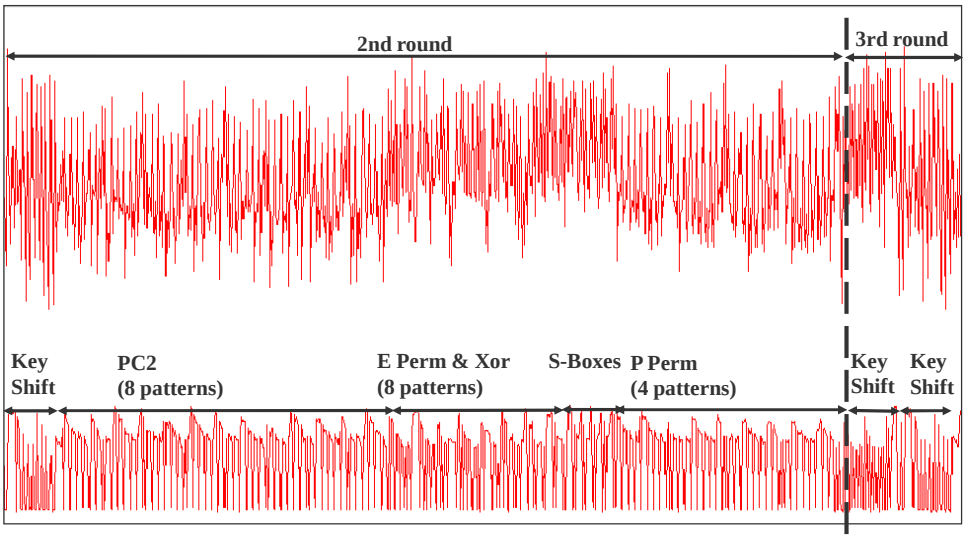
technicolor

Power trace of an algorithm: DES



technicolor

Power trace of an algorithm: DES



technicolor

SPA attack

Simple (Single) Power Analysis context

- Find out a secret or private key
- Known algorithm
- Unknown implementation (*background culture recommended*)

Conditions

- 1 card available
- Reverse engineering phase required (*signature location*)
- Key inference on a single curve (*with relevant height of view)*
- Possibly known plain or ciphertext

2 target examples:

- DES key schedule
- RSA private exponentiation

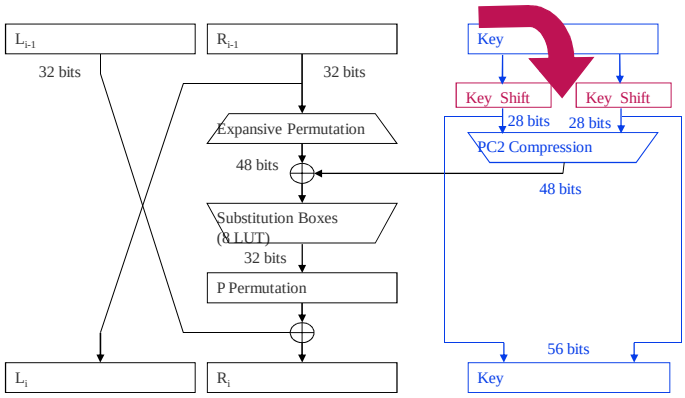
technicolor

SPA attack on DES

Goal of the attack: find the DES secret key (56 bits)

Knowledge on the implementation (assumed hereafter)

Target of the attack: key schedule

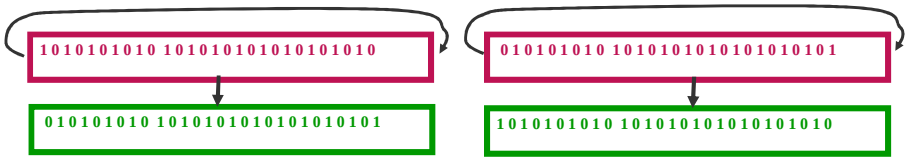


technicolor

SPA attack on DES: Key shift

The Key Shift description

- Each 28 bits half is shifted separately
- Shift to the left for DES (to the right for DES^{-1})
- 1 bit rotated at each Key Shift



- Number of rotations depends on the round

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#Shift L (DES)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
#Shift R (DES-1)	0	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

technicolor

SPA attack on DES: Key shift

The Key Shift implementation (56 bits stored in 7 bytes)

byte \ bit	7	6	5	4	3	2	1	0
des_key+0	57	49	41	33	25	17	09	01
des_key+1	58	50	42	34	26	18	10	02
des_key+2	59	51	43	35	27	19	11	03
des_key+3	60	52	44	36	**63**	55	47	39
des_key+4	31	23	15	07	62	54	46	38
des_key+5	30	22	14	06	61	53	45	37
des_key+6	29	21	13	05	28	20	12	04

- Set the carry with bit n°3 of des_key+3: *Carry* = bit 63
- Left rotate des_key+6 (input carry to the right): *Carry* = bit 29
- Left rotate des_key+5 (input carry to the right): *Carry* = bit 30
- … down to des_key+0: *Carry* = bit 57

byte \ bit	7	6	5	4	3	2	1	0
des_key+0	49	41	33	25	17	09	01	58
des_key+1	50	42	34	26	18	10	02	59
des_key+2	51	43	35	27	19	11	03	60
des_key+3	52	44	36	63	55	47	39	31
des_key+4	23	15	07	62	54	46	38	30
des_key+5	22	14	06	61	53	45	37	29
des_key+6	21	13	05	28	20	12	04	63

technicolor

SPA attack on DES: Key shift

The Key Shift implementation (continued)

- Clear bit n°4 in des_key+3 (forced to 0)

byte \ bit	7	6	5	4	3	2	1	0
des_key+0	49	41	33	25	17	09	01	58
des_key+1	50	42	34	26	18	10	02	59
des_key+2	51	43	35	27	19	11	03	60
des_key+3	52	44	36	**'0'**	55	47	39	31
des_key+4	23	15	07	62	54	46	38	30
des_key+5	22	14	06	61	53	45	37	29
des_key+6	21	13	05	28	20	12	04	63

***Carry* = bit 57**

- If Carry is set (= 1) set bit n°4 in des_key+3 (forced to 1)

byte \ bit	7	6	5	4	3	2	1	0
des_key+0	49	41	33	25	17	09	01	58
des_key+1	50	42	34	26	18	10	02	59
des_key+2	51	43	35	27	19	11	03	60
des_key+3	52	44	36	**'1'**	55	47	39	31
des_key+4	23	15	07	62	54	46	38	30
des_key+5	22	14	06	61	53	45	37	29
des_key+6	21	13	05	28	20	12	04	63

SPA attack on DES: Key shift

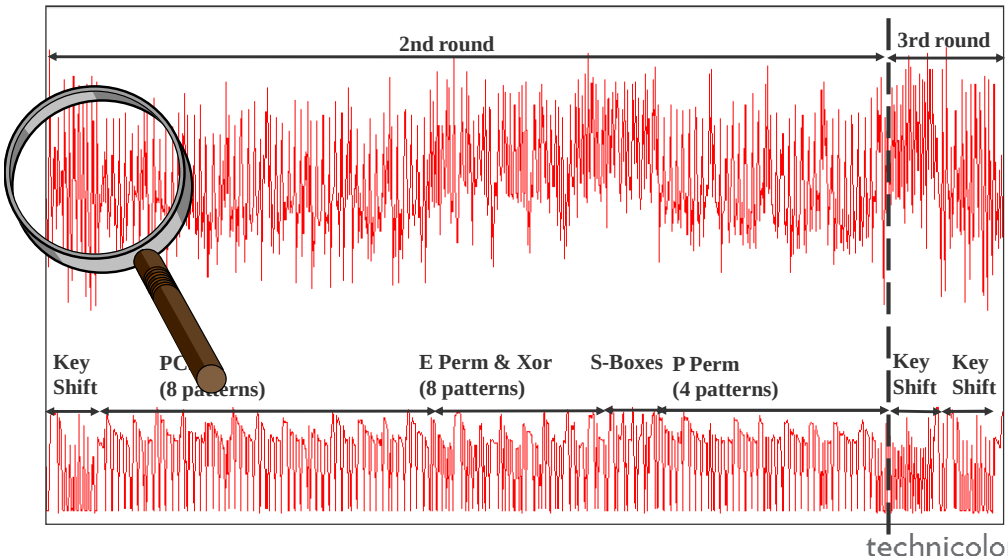
After 16 rounds, 28 key bits have gone through the carry…

… and have been tested each time!

If a successful test (with related bit set) is electrically different from an unsuccessful test…

… then it is possible to read the 28 bit values!

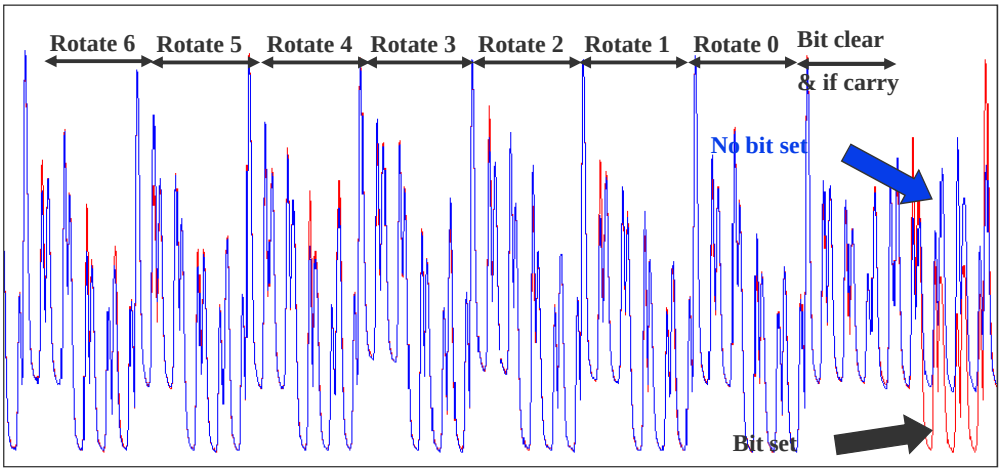
SPA attack on DES: Key shift



technicolor

SPA attack on DES: Key shift

Consumption: single “Key Shift” and conditional “bit set”



technicolor

SPA attack on DES: Conclusion

1 or 2 key bits can be read per round

28 remaining bits can be retrieved by brute force…

… or 27 can be found by doing the same on DES^{-1}

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#Shift L (DES)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
#Shift R (DES-1)	0	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

- **BEWARE OF NAIVE PROGRAMMING!**

SPA attack on RSA

SPA against RSA private exponentiation

$$s = \mu(m)^d \bmod N$$

- N large modulus, say 1024 bits ($N = pq$, with p & q large primes)
- m message and μ is a padding function (e.g., PSS)
- s signature
- d private exponent such that : $ed \equiv 1 \bmod (p-1)(q-1)$, with e public exponent

The attacker aims at retrieving d

SPA attack on RSA

- **Implementation points (assumed known hereafter)**
  - N, $\mu(m)$, s and d are 128-byte buffers
  - basic “square and multiply” algorithm
  - exponent bits scanned from MSB to LSB (left to right)

k = bitsize(d)
s = 1
For i = k-1 down to 0
  $s = s*s \bmod N$ *(SQUARE)*
  If ($d[i]$=1) then
    $s = s*m \bmod N$ *(MULTIPLY)*
  End if
End for

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

init (MSB 1) $s = m$

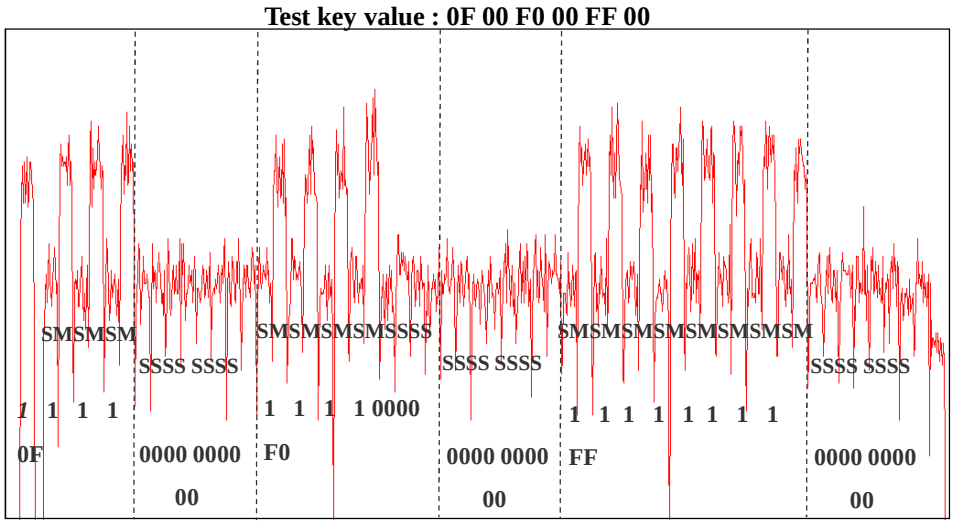
round 2 (bit 0) $s = m^2$

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

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

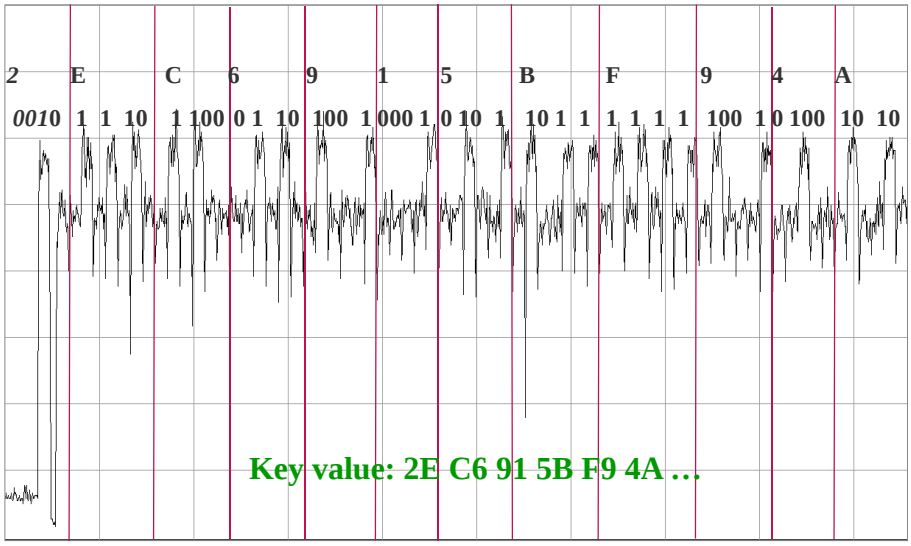
technicolor

SPA attack on RSA



technicolor

SPA attack on RSA



technicolor

Conclusion

SPA uses implementation related patterns

SPA strategy

- algorithm knowledge
- reverse engineering phase (*signature location*)
- representation tuning (*height of view*, *zoom*, *visualisation*)
- then play with implementation assumptions...

SPA is always specific due to

- the algorithm implementation
- the application constraints
- the chip's technology (*electrical properties*)
- possible counter-measures...

technicolor

Conclusion: Countermeasures

Counter-measure: anything that foils the attack !

Trivial countermeasure

- prohibit code branches conditioned by the secret bits

Advanced counter-measures

- algorithm specification refinement
  - code structure
  - data whitening (a.k.a. blinding)
- implementation design based on the chip's resources
  - play with instructions set
  - hardware electrical behaviour (current scrambler, desynchronisation, cryptoprocessor…)

Part II:
Timing Analysis

Summary

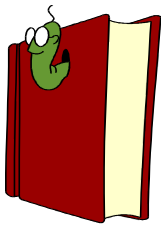
What are timing attacks?

Attack on a pin code verification

- Non constant time execution
- Randomised execution

Attack on an RSA computation

Is there a future for 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

GEMPLUS put theory into practice in early `98

Timing attacks belong to the large family of "side channel" attacks

What are timing attacks?

Principle of Timing Attacks:

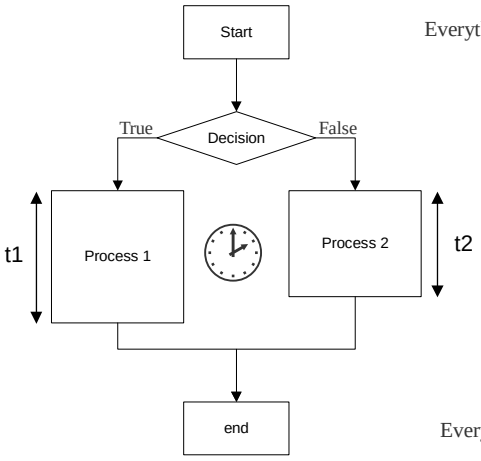
- Secret data are processed in the card
- Processing time
  - depends on the value of the secret data
  - leaks information about the secret data
  - can be measured (or at least their differences)

Practical attack conditions

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

technicolor

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

technicolor

PIN code verification

Secret data are stored in the smart card

- Example: a PIN code, 8 bytes long

Like passwords on a PC, authentication is based on this secret

- A dedicated function exists in the smart card software :

The 'VerifySecret' command which:

- Receives the challenge (proposed value for the PIN code)
- Compares the challenge with the stored PIN
- Grants access rights if the comparison is successful

PIN code verification

Level 1

Pseudo-code for the "VerifySecret" command

- IN
  - *P* = PIN code value stored in the card
  - *C* = Challenge (proposed value for the PIN)
- OUT
  - 'KO' or 'OK'
- VERIFY SECRET
  - For b = 0 to 7
    - *If* C[b] != P[b] *then return 'KO'*
  - Return 'OK'

PIN code verification

Level 1

Attack implementation

- Propose the n possible values of $C[0]$ (256 values)
- Measure $\tau[n]$ the corresponding command duration
- Compute the maximum command duration τ , $\tau[n_0]$
  - $\tau[n_0] = \max(\tau[n])$, $n \in \{0, \ldots, 255\}$
  - n_0 is the solution $P[0]$ for the first byte of the PIN code
- $C[0]$ being known, iterate successively for all $C[i]$

Complexity

- Number of comparisons: 8 * 256 = 2048 (instead of 256^8)

PIN code verification

Level 2

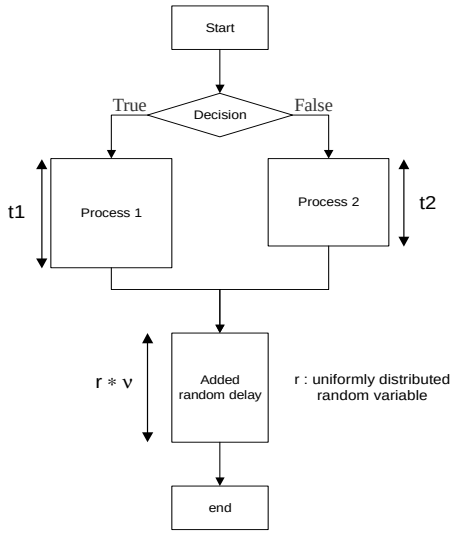
Possible countermeasure

To defeat this attack one may think to add a random delay during the execution:

- Generate a random delay τ_a uniformly distributed
  - $\tau_a \in \{0, \nu, 2\nu, 3\nu, \ldots, r\nu\}$ with $0 \leq r \leq 255$
  - ν is an elementary time unit
- Wait τ_a whatever the command status 'KO' or 'OK'
- Follow the same implementation as the previous one

PIN code verification

Level 2



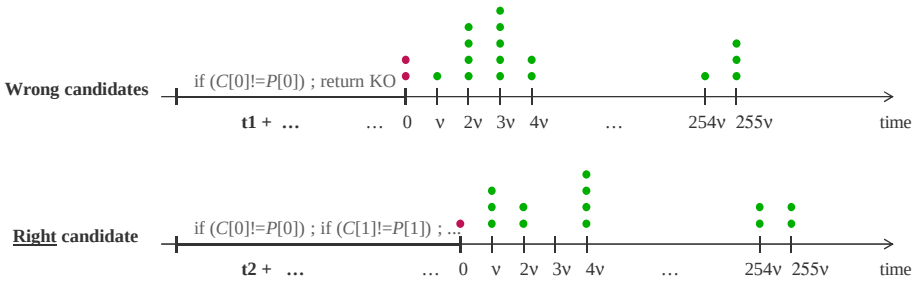
technicolor

PIN code verification

Level 2

Attack idea

- It is possible to know what would be the duration for processing a challenge as if there were no delay



technicolor

PIN code verification

Level 2

Random delay elimination

- For each *n* (*n* is the candidate *C*[0] for the first PIN code byte)
  - Acquire a series of *N* command execution durations $\tau_i[n]$
  - The minimum duration corresponds to a $\tau_a = 0$ random delay (with high probability, if *N* is chosen large enough)
- Consider the corresponding $\tau_{min}[n]$ run time value

Attack implementation

- Get rid of the random delay for each candidate ($\tau_{min}[n]$)
- Apply the previous attack scheme

Complexity

- Number of comparisons: 2048 * *N* (still feasible)

PIN code verification

Level n

More complicated counter-measure may be thought of...

- Add a binomial (rather than uniform) random delay
- …

…but they also may be defeated by more clever attacks!!

PIN code verification: Conclusion

A typical example of unsecure smart card software

- Can happen in any routine processing secret data
  - Secret values comparison
  - Memory scanning and loading
  - Checksum computation

Counter-measures evaluation

- Add a delay is definitely not the good alternative
- An inspection of the assembly code for correct implementation may be a warranty

TIME-CONSTANT CODE (for sensitive data)

IS THE SOLUTION

Attack on RSA: Introduction

First known practical attacks

- During the rump session of CRYPTO'97 by Lenoir
- In the “Université Catholique de Louvain” (UCL), for the research project Cascade (multi-application smart card)
  - *A practical implementation of the timing attack* (J.F. Dhem, J.L. Willems, F. Koeune & J.J. Quisquater)

RSA is not an exception, all cryptosystems may be threatened

- Basic mathematical operations
- Modular exponentiation
- Cryptographic algorithms

Attack on RSA: Principle

All the requisites

- A minimum of knowledge on the RSA algorithm
- Knowledge and variability of the message are needed
- Time measurements must be accurate to within few clock cycles

Targeted RSA algorithm

- A standard RSA exponentiation ($s = m^d \bmod N$)
  - Montgomery method for the modular multiplication on large numbers shows computation time variations
  - The classic square & multiply exponentiation routine allows these variations to be exploited

Attack on RSA: Square-and-Multiply

Straightforward implementation for $s = m^d \bmod N$

- Input: m, (d, N)
  - m = message (k bits)
  - (d, N) = RSA private key (k bits)
- Output: $s = m^d \bmod N$
  - s = signature (k bits)
- Square & Multiply
  - $s = 1$
  - for $i = k$-1 down to 0
    - $s = s^2 \bmod n$
    - *If* (d[i] = 1) *then* s = s*m *mod* N
  - return s

Attack on RSA: Montgomery multiplication

Montgomery modular multiplication ($\otimes$) is dedicated to modular exponentiation

- It enhances its efficiency
- The result of each multiplication lies in [0, 2**N*[
  - A subtraction may be needed to fully reduce mod *N*

- if (*d*[*i*]= 1) then $s = s \otimes m \bmod N$
  - Step 1: modular multiplication by *m*
  - Step 2: optional subtraction by *N*

Attack on RSA: Description

Working hypothesis

- Bits *d*[*k*-1] to *d*[*k*-*i*+1] are already known
  - Knowing the message, the intermediate value of *s* after the square at iteration *k*-*i* is computed
  - Whether the subtraction in $s \otimes m \bmod N$ is required may be stated

Attack on RSA: Description

The attack is based on an oracle

- Sign with same (d, N) for many random messages
- Make the assumption that $d[k\text{-}i] = 1$
- Construct 2 sets of messages depending on the fact that the subtraction happens or not during the multiplication
  - A = {m : $s \otimes m$ mod N implies a subtraction}
  - B = {m : $s \otimes m$ mod N implies no subtraction}

The time for the subtraction will be discriminatory

Attack on RSA: Description

Case ($d[k\text{-}i] = 0$)

- Global times for sets A and B are not statistically distinguishable (the split is based on a multiplication which does not occur)

Case ($d[k\text{-}i] = 1$)

- Global times for sets A and B show a statistical difference related to the optional subtraction (the multiplication does occur)

Attack on RSA: Description

Time measurements validate or invalidate the oracle

- Compute the mean of the global duration for each subset
  - <A>: mean global duration for messages in A
  - <B>: mean global duration for messages in B
- The oracle criterion is the following
  - <A> - <B> >> 0 $\Rightarrow$ oracle was right ($d[k\text{-}i] = 1$)
  - <A> - <B> $\approx$ 0 $\Rightarrow$ oracle was wrong ($d[k\text{-}i] = 0$)

Attack on RSA: Conclusion

Results (on a Pentium 200)

- For 128 bits, recovers 2bits / s with 10.000 messages
- For 512 bits, recovers 1bit / 20s with 100 k messages

Conclusion

- Time-constant code is a solution
- Data blinding (randomization) may also be possible

Is there a future for timing attacks?

Associated with other side-channels, it becomes far more efficient

- Global measurements are replaced by local ones

Timing attacks are still an important threat

- Against existing devices applied to secret management
- Not only a smart cards issue
- Designers have to think about it
- Software has still to circumvent hardware flaws

Solutions do exist !

Part III:
Differential Power Analysis

Summary

DPA Statistical Principle
- Acquisition procedure
- Selection & prediction
- Differential operator and curves
- Reverse engineering using the DPA indicator

Attacking the DES with DPA
- Classical target
- Hypothesis testing (Guesses management)

Generalisation of DPA
- Other targets
- Other algorithms (RSA, AES…)

Conclusion: anti-DPA counter-measures

DPA statistical principle

Published on the web by Paul KOCHER (1998)

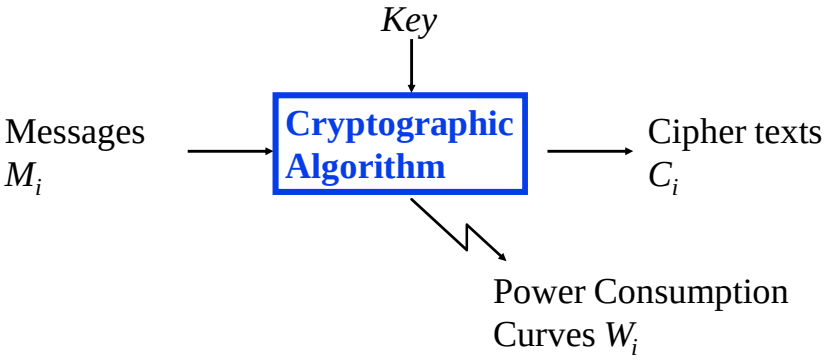
Powerful & generic Power Attack
- statistical & signal processing
- known random messages
- targeting a known algorithm
- running on a single smartcard

Big noise in the cryptographic community

Big fear in the smartcard industry !

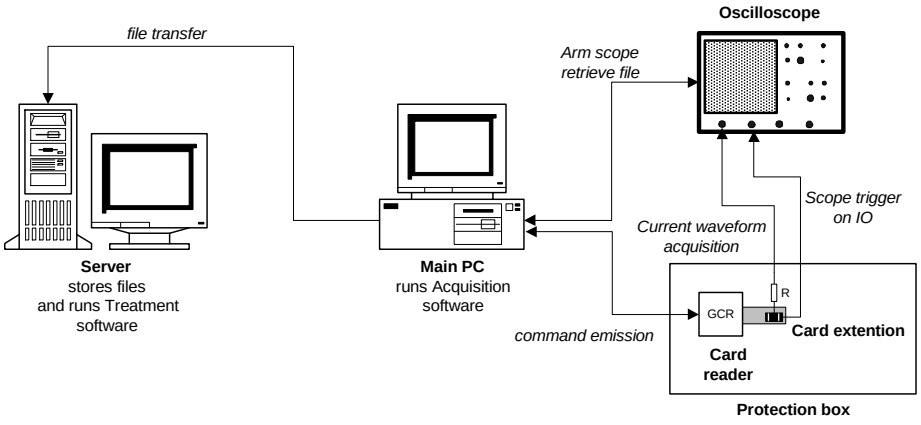
Acquisition procedure

Play the algorithm N times
(100 < N < 100000)



technicolor

Acquisition procedure



Monitoring equipment for iterated acquisitions

technicolor

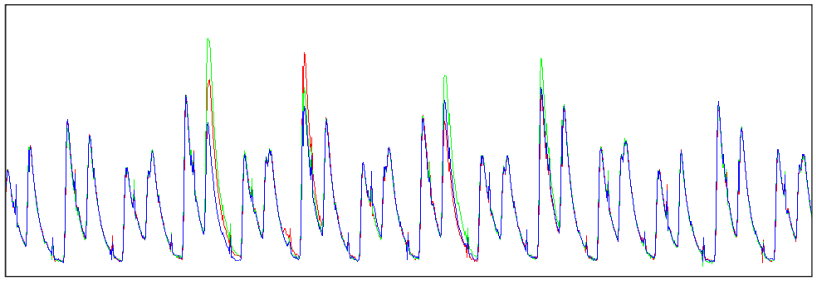
Acquisition procedure

After data collection, what is available?

- *N* plain or cipher random texts

  00 B688EE57BB63E03E
  01 185D04D77509F36F
  02 C031A0392DC881E6 …

- *N* corresponding power consumption waveforms



technicolor

Selection & prediction

Assume the message is processed by a known deterministic function *f* (transfer, permutation...)

Knowing the message, one can recompute off line its image through *f*

$$M_i \longrightarrow \boxed{f} \longrightarrow M'_i = f[M_i]$$

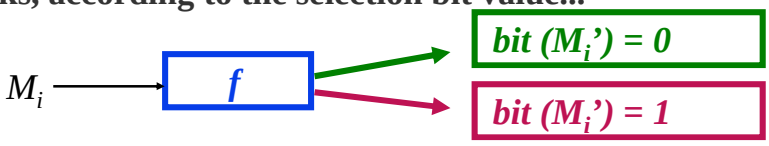
- **Now select a single bit among *M'* bits (in M' buffer)**
- **One can predict the true story of its variations**

i	Message		bit	
0	B688EE57BB63E03E	1		
1	185D04D77509F36F	0		
2	C031A0392DC881E6	1	….	*for i = 0, N-1*

technicolor

DPA operator & curve

- **Partition the messages and related curves into two packs, according to the selection bit value...**



- **… and assign -1 to pack 0 and +1 to pack 1**

0	B688EE57BB63E03E	1	+1	
1	185D04D77509F36F	0	-1	
2	C031A0392DC881E6	1	+1	… *for i = 0, N-1*

- **Sum the signed consumption curves and normalize**
- **<=> Difference of averages**

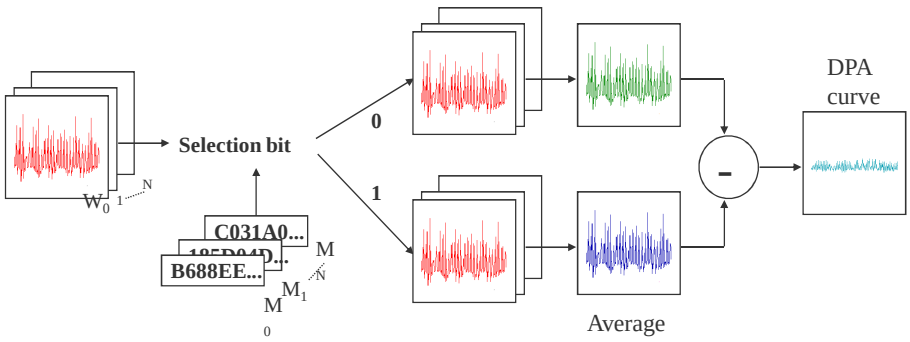
$(N_0 + N_1 = N)$

$$DPA = \frac{\sum W_1}{N_1} - \frac{\sum W_0}{N_0}$$

technicolor

DPA operator & curve

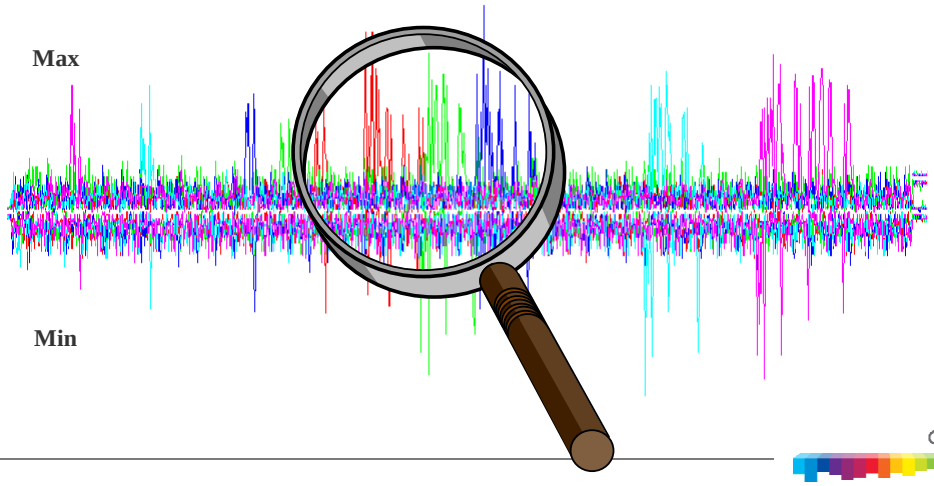
DPA curve construction



technicolor

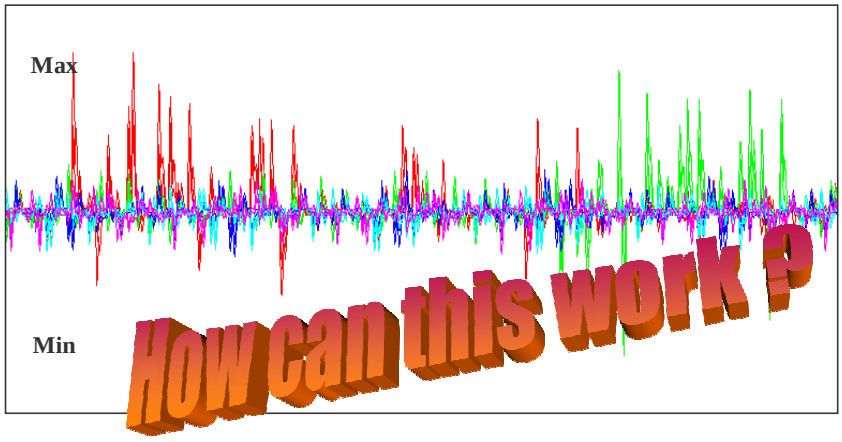
DPA operator & curve

DPA curves for different selection bits



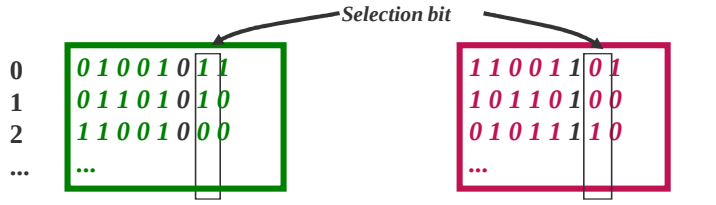
DPA operator & curve

Peaks are rising when selection bits are handled



DPA operator & curve

- **Spikes explanation : Hamming Weight of the bit's byte**



Average = E [HW_0] = 0 + 3.5 *Average = E [HW_1] = 1 + 3.5*

$\Delta = E[HW_1] - E[HW_0] = 1$

- **Contrast (peak height) proportional to $N^{1/2}$ (evaluation criterion)**
- **If prediction was wrong: selection bit would be random**

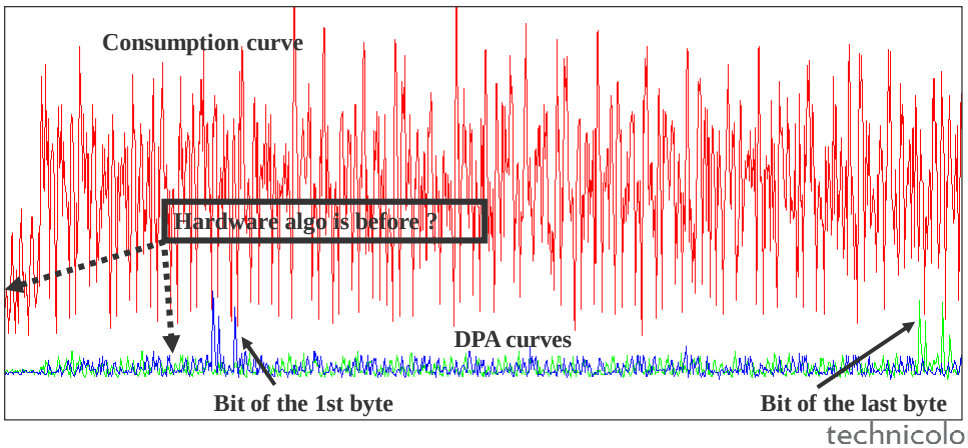
E [HW0] = E [HW1] = 4 => $\Delta = 0$

technicolor

Reverse engineering using DPA

Use DPA to locate when predictible things occur

- DPA and power curves superposition
- Example : hardware algo & ciphertext transfer to RAM

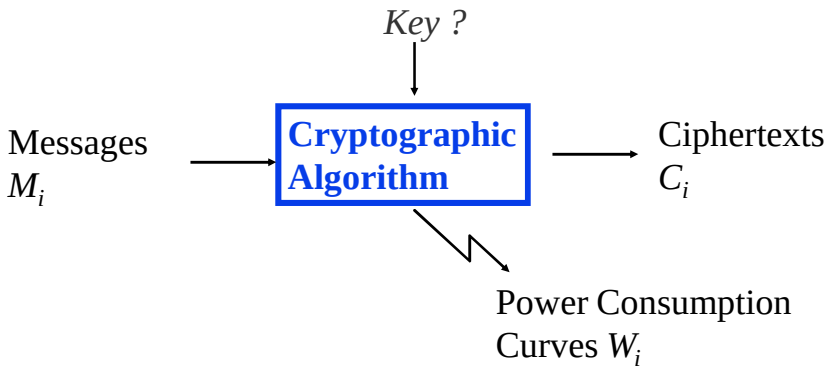


technicolor

Attacking DES with DPA

DPA works thanks to the perfect prediction of the selection bit

How to break a key?



Attacking DES with DPA

Try different keys a valid them with DPA

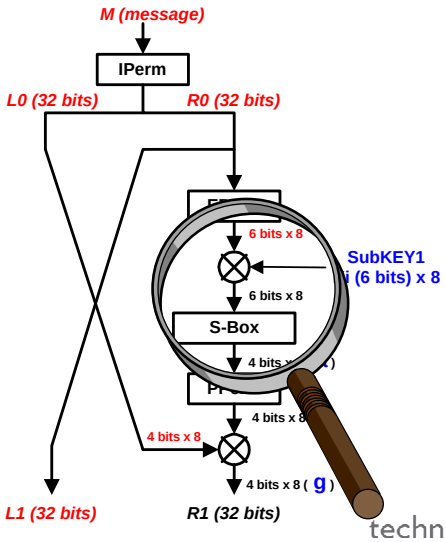
Isn't it like cryptographic exhaustive search ?

Not exactly …

… because the research space is drastically reduced!

Classical target

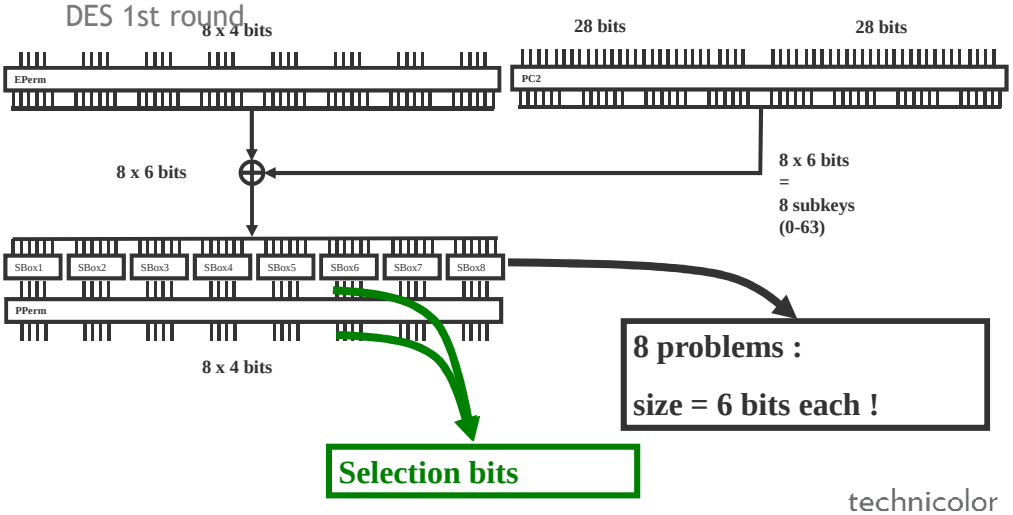
DES 1st round



technicolor

Classical target

DES 1st round



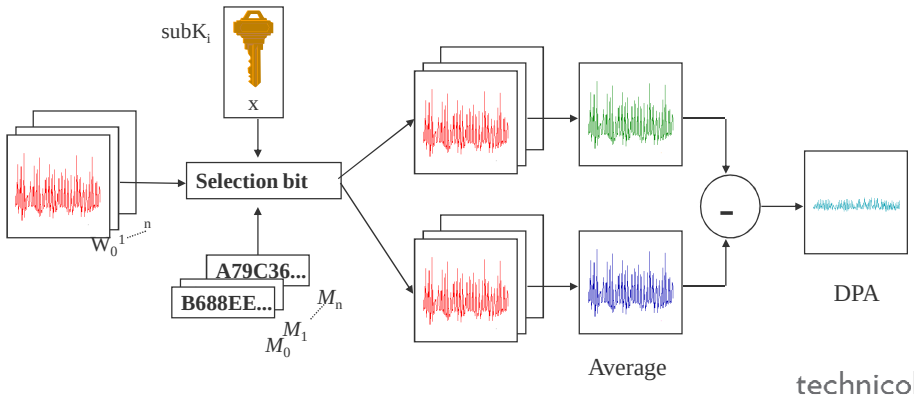
technicolor

Hypothesis testing (guess)

GUESS = value (0 to 63) of $subK_i$ (i = 1 to 8)

Try 64 guesses for each subkey: test them with 64 DPA

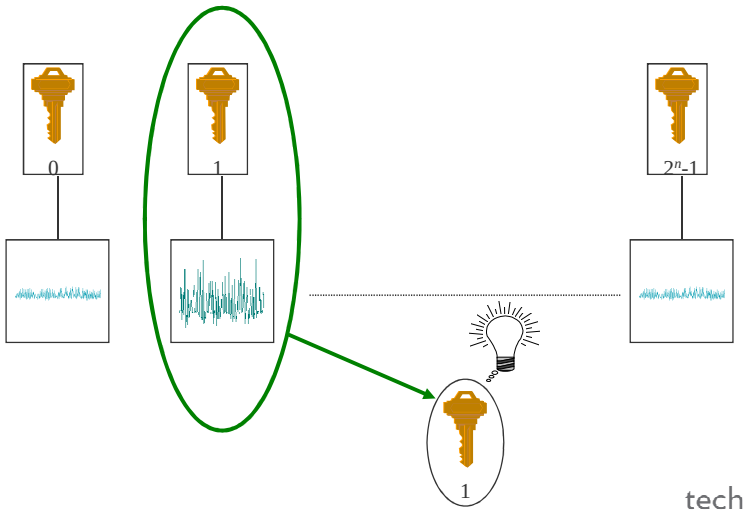
48 key bits disclosed with 8 x 64 DPA (<< 2^{48} !)



technicolor

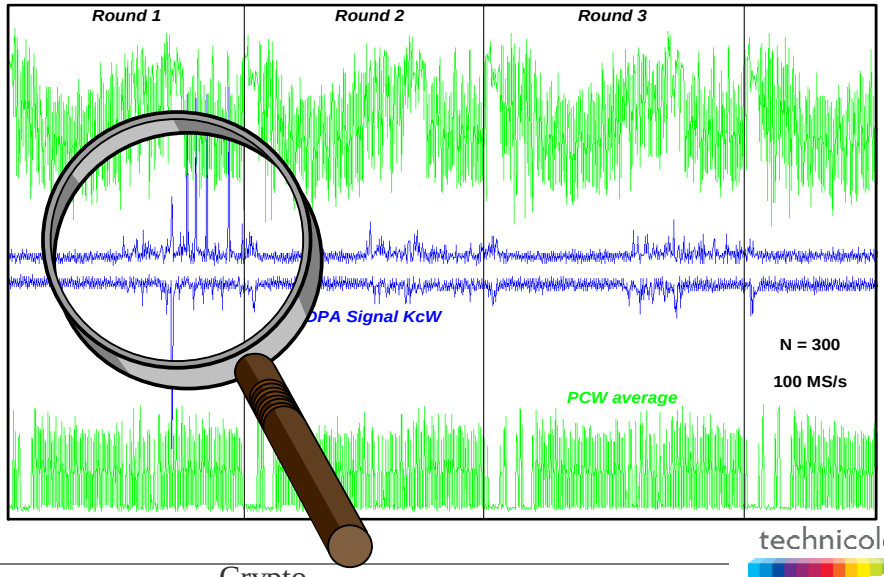
Hypothesis testing (guess)

The right guess provides the highest spikes !

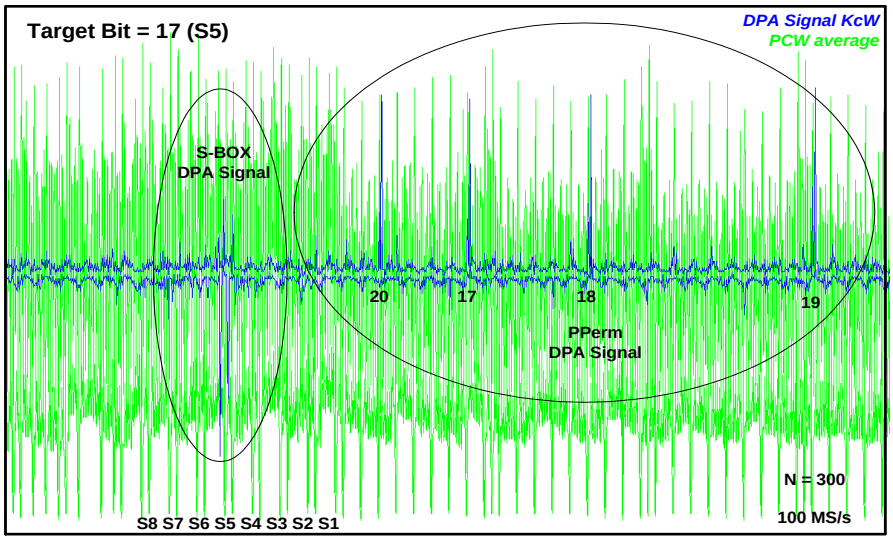


technicolor

DES DPA signal type a



DES DPA signal type a



Hypothesis testing (right guess)

Why does the right guess provide the highest spikes!

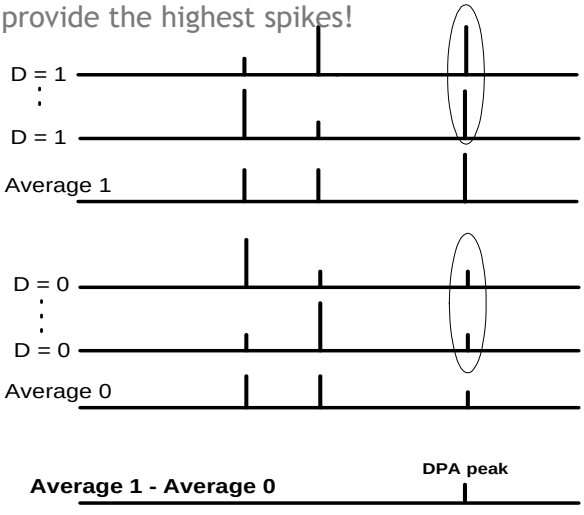
Right guess

Exact prediction of the selection bit

0	B688EE57BB63E03E	1	1
1	185D04D77509F36F	0	0
2	C031A0392DC881E6	1	1
…			

Real

Predicted



technicolor

Hypothesis testing (wrong guess)

Why do the wrong guesses provide no spike?

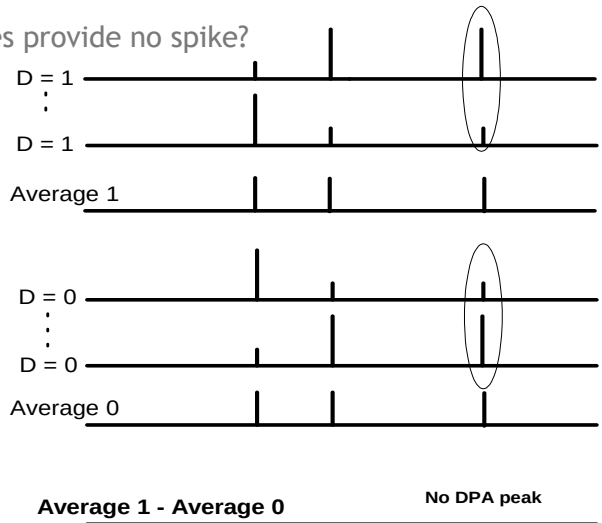
Wrong guess

Wrong prediction of the selection bit

0	B688EE57BB63E03E	1	0
1	185D04D77509F36F	0	1
2	C031A0392DC881E6	1	1
…			

Real

Predicted



technicolor

Hypothesis testing

Reality is not so easy because of

- low contrast between the guesses
- wrong guesses leading to higher spikes (wrong model)

Decision consolidation : compare different equivalent selection bits (4 by SBox)

- they do not agree necessarily !

In the best case: 48 subkey bits are broken!

- Inverse key schedule to recover the “plain” key bits

8 significant bits remain to be found

- by exhaustive search (256 combinations)
- or by DPA on 2nd round

Generalisation of DPA: Other targets

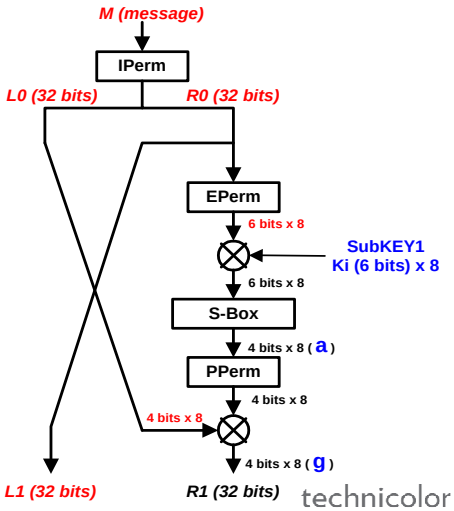
SBoxes output : 1 selection bit “sensitive” to 6 key bits!

PPerm output is equivalent

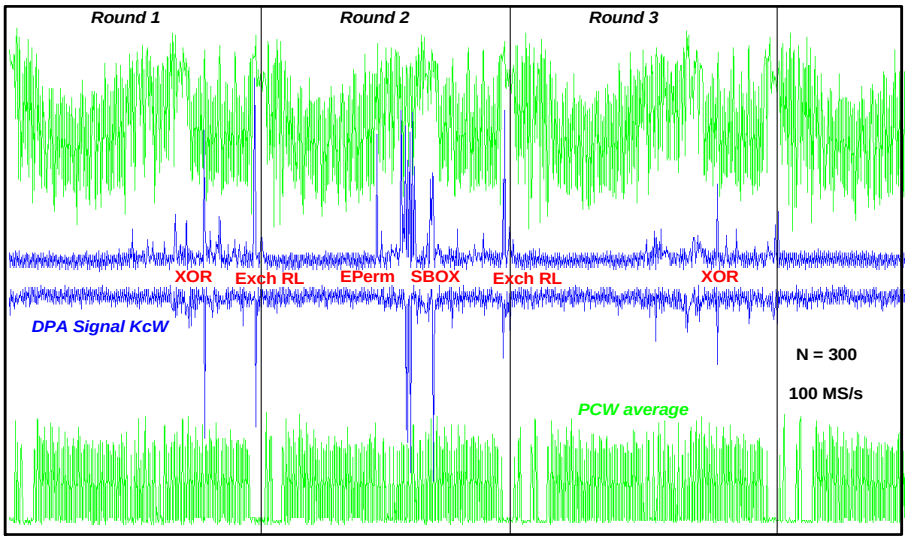
- More calculation

SBoxes intput ?

- 1 selection bit “sensitive”…
- … to 1 key bit only ! (low yield)



Other targets: DPA type g



technicolor

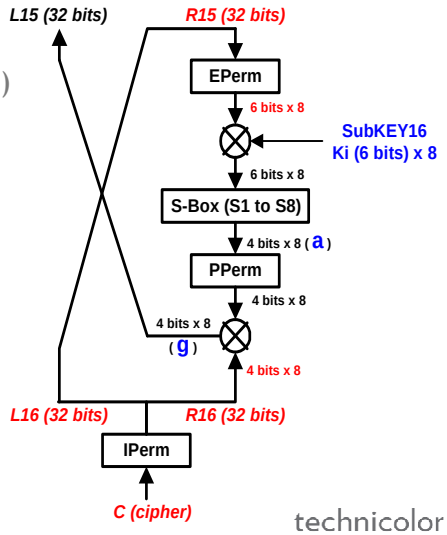
Generalisation of DPA: Other targets

If only cipher text is available...

(3DES, application constraints)

... do last round DPA !

- 16th round is symmetric for DPA
- key schedule inversion is more complicated



technicolor

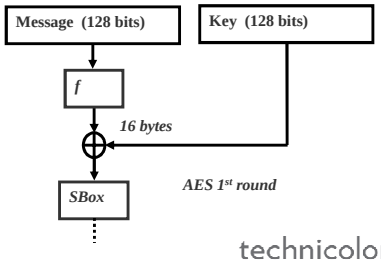
Generalisation to other algorithms

DPA on RSA

- The key is not entirely handled from the beginning, but progressively introduced
  - Prediction is to be done by time slices: next bit inference requires the previous bit to be broken

DPA on AES (Advanced Encryption Standard)

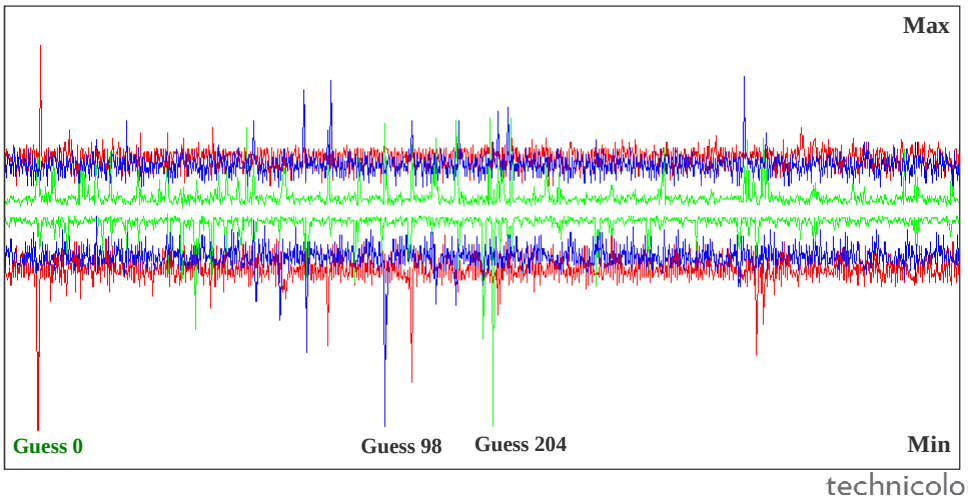
- Easier than on DES
- But larger : 16 x 8 bits subkeys
- => 16 x 256 guesses



technicolor

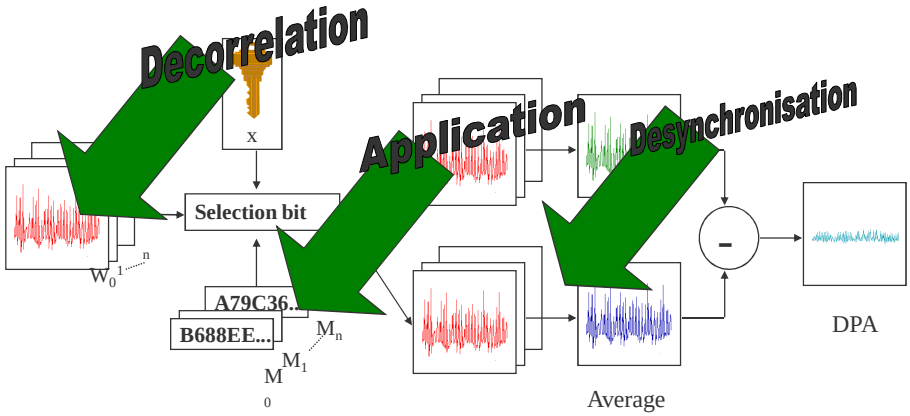
Generalisation to other algorithms

- **DPA on AES : 1st round and 1st byte (right guess = 0)**



technicolor

Conclusion: Countermeasures



DPA is powerful, generic (to many algorithms) and robust (to model errors)…

… but there are countermeasures!

technicolor

DPA countermeasures

Application countermeasures: make message free randomization impossible !

- Fix some message bytes
- Constrain the variable bytes (ex: transaction counter)

Decorrelate power curves from data

- by hardware : current scramblers (additive noise)
- by software : data whitening

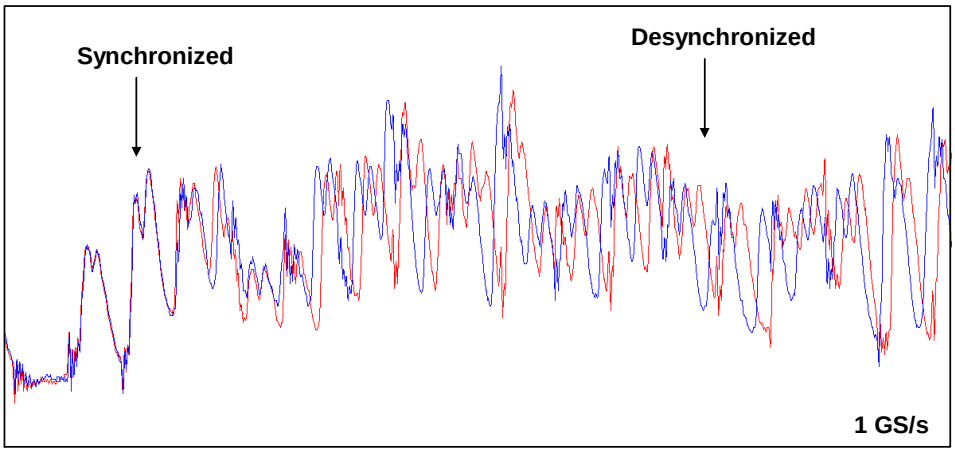
Desynchronise the *N* executions

- software random delays
- software random orders (ex: SBoxes in random order)
- hardware wait states (dummy cycles randomly added by the CPU)
- hardware unstable internal clock (phase shift)

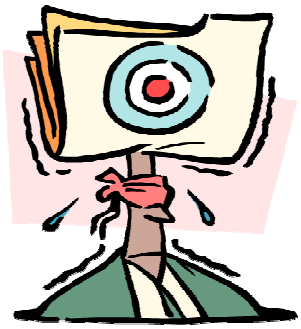
technicolor

DPA countermeasures

Internal clock effects (phase shift)



Comments/Questions?



`http://www.thlab.net/~joyem/`

Preventing Side-Channel Attacks

Application to RSA

Marc Joye

technicolor

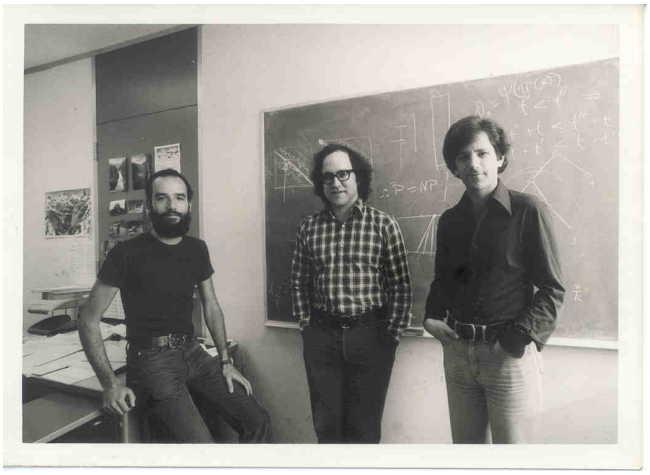
Outline

1. **RSA Cryptosystem**
2. **Basic Algorithms**
3. **Side-Channel Atomicity**
4. **Implementing the RSA**

Cryptographic Engineering, Lausanne, June 25-29, 2012

RSA Cryptosystem

- Invented by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977



- Useful for [public-key] encryption and digital signature

RSA Primitive (1/3)

- $(\mathbb{Z}/N\mathbb{Z})^* = \{x \in [0, N[\mid \gcd(x, N) = 1\}$
  - [multiplicative] group
  - Euler totient function $\phi(N) := \#(\mathbb{Z}/N\mathbb{Z})^*$

Example

$(\mathbb{Z}/10\mathbb{Z})^* = \{1, 3, 7, 9\}$ and $\phi(10) = 4$

- Modular exponentiation:

$$(\mathbb{Z}/N\mathbb{Z})^* \times \mathbb{Z} \to (\mathbb{Z}/N\mathbb{Z})^*, \quad (x, e) \mapsto y = x^e \bmod N$$

  - permutation if $\gcd(e, \phi(N)) = 1$

Example

In $(\mathbb{Z}/10\mathbb{Z})^*$, $\{1, 3, 7, 9\} \mapsto \{1, 7, 3, 9\}$ for $e = 3$, and
$\{1, 3, 7, 9\} \mapsto \{1, 9, 9, 1\}$ for $e = 2$ $(\gcd(2, 4) \neq 1)$

RSA Primitive (2/3)

- RSA primitive = modular exponentiation

$$(\mathbb{Z}/N\mathbb{Z})^* \times \mathbb{Z} \to (\mathbb{Z}/N\mathbb{Z})^*, \quad (x, e) \mapsto y = x^e \bmod N$$

  - one-way, trapdoor function for an RSA modulus $N = pq$ where p, q are 512-bit primes

Definition (RSA Problem)

Given an RSA modulus N, $y \in (\mathbb{Z}/N\mathbb{Z})^*$ and an integer $e > 1$ with $\gcd(e, \phi(N)) = 1$, compute $x = y^{1/e} \bmod N$

Solution

$d = e^{-1} \bmod \phi(N)$ with $\phi(N) = (p-1)(q-1)$

$\Longrightarrow \; x = y^d \bmod N$

RSA Primitive (3/3)

- Key generation

  Input keylength k and e
  Output $N = pq$ such that $|N|_2 = k$ and $\gcd(e, \phi(N)) = 1$
  $d = e^{-1} \bmod \phi(N)$

  $pk = \{e, N\}$ and $sk = \{d\}$

- [Plain] RSA encryption

  Input message m and public key pk
  Output ciphertext $c = m^e \bmod N$

- [Plain] RSA decryption

  Input ciphertext c and private key sk
  Output message $m = c^d \bmod N$

RSA Encryption in Practice

- Plain RSA encryption is insecure
  - encryption should be probabilistic
  - plain RSA is homomorphic
    $\Rightarrow$ e.g., "garbage-man-in-the-middle" attack
- RSA-OAEP
  - Optimal Asymmetric Encryption Padding

$$c = \mu_{\text{OAEP}}(m, r)^e \bmod N \text{ for a random } r$$

  - proposed by Mihir Bellare and Phillip Rogaway in 1994
  - included in PKCS #1
  - highest security level (IND-CCA2) in the ROM

technicolor

RSA Signature in Practice (1/4)

- Plain RSA signature is universally forgeable
  $\Rightarrow$ messages should be "appropriately" padded
- RSA signature (with appendix)
  - setup: $N = pq$ with p, q prime
    (e, d) satisfying $e\,d \equiv 1 \pmod{\phi(N)}$
  - public parameters: $\{e, N\}$
  - private parameters: $\{d, N\}$

Signature on message m

$S = \dot{m}^d \bmod N$ where $\dot{m} = \mu(m)$

Verification

$S^e \stackrel{?}{\equiv} \mu(m) \pmod{N}$

technicolor

RSA Signature in Practice (2/4)

- Deterministic paddings
  - RSA-FDH [Bellare and Rogaway, 1993]
    - Full Domain Hash

$$\mu(m) = H(m) \quad \text{with } H : \{0,1\}^* \to \mathbb{Z}/N\mathbb{Z}$$

    - highest security level (EUF-CMA) in the ROM
  - PKCS #1 v1.5 [RSA Labs]
- Probabilistic paddings
  - RSA-PSS [Bellare and Rogaway, 1996]
    - Probabilistic Signature Scheme

$$\mu(m) = \mu_{\text{PSS}}(m, r) \text{ for a random } r$$

    - highest security level (EUF-CMA) in the ROM
    - tight security proof and can be with message recovery
  - PKCS #1 v2.1 [RSA Labs]

RSA Signature in Practice (3/4)

FDH padding (example)

Signature $S = \mu_{\text{FDH}}(m)^d \bmod N$ with $\mu_{\text{FDH}} : \{0,1\}^* \to \mathbb{Z}/N\mathbb{Z}$,

$$m \mapsto \mu_{\text{FDH}}(m) = \underbrace{\overbrace{h(1\|m)}^{160 \text{ bits}} \| \overbrace{h(2\|m)}^{160 \text{ bits}} \| \ldots}_{1024 \text{ bits}} \pmod N$$

where h is a cryptographic hash function (e.g., SHA-1)

RSA Signature in Practice (4/4)

PSS padding

Signature $S = \mu_{\text{PSS}}(m, r)^d \bmod N$ where

$$\mu_{\text{PSS}}(m, r) = 0 \| w \| r^* \| g_2(w)$$

with $w = h(m, r)$ and $r^* = g_1(w) \oplus r$

- Signing
  1. pick a random r
  2. compute $w = h(m, r)$ and $r^* = g_1(w) \oplus r$
  3. return $S = [0 \| w \| r^* \| g_2(w)]^d \bmod N$
- Verification
  1. compute $S^e \bmod N = b \| \bar{w} \| \rho \| \omega$
  2. compute $\bar{r} = g_1(\bar{w}) \oplus \rho$
  3. check whether (i) $b = 0$, (ii) $h(m, \bar{r}) = \bar{w}$, and (iii) $g_2(\bar{w}) = \omega$

Square-and-Multiply Algorithm (1/3)

- Square-and-multiply algorithm

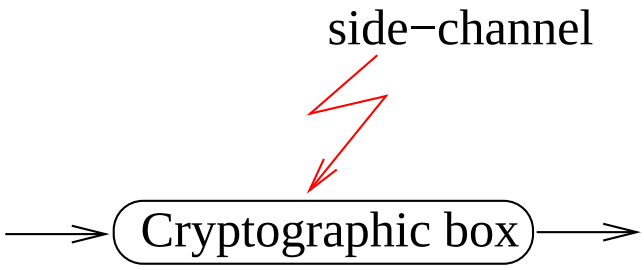
Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$
2. For $i = k - 1$ downto 0 do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - If $(d_i = 1)$ then $R_0 \leftarrow R_0 \, \dot{m} \pmod N$
3. Return R_0

- left-to-right exponentiation
- $1 + 1$ temporary variables (R_0 and $\dot{m}$)
- . . . subject to SPA-type attacks

Square-and-Multiply Algorithm (2/3)

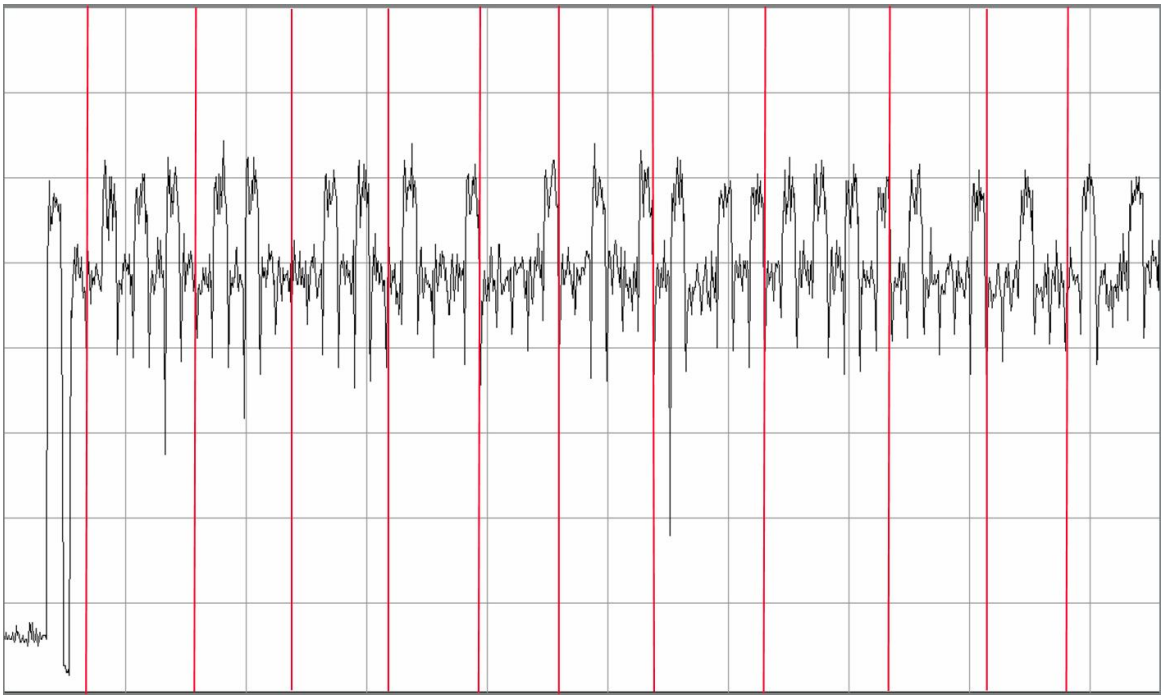
- Simple side-channel analysis



  - side-channel = timing, power consumption, . . .

technicolor

Square-and-Multiply Algorithm (3/3)



Key: $d = \texttt{2E C6 91 5B FE 4A}$

technicolor

Square-and-Multiply-*Always* Algorithm

- Square-and-multiply-*always* algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow 1$
2. For $i = k - 1$ downto 0 do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - $b \leftarrow 1 - d_i$; $R_b \leftarrow R_b \dot{m} \pmod N$
3. Return R_0

- when $b = 1$ (i.e., $d_i = 0$), there is a dummy multiplication
- the power trace now appears as a regular succession of squares and multiplies
- $2 + 1$ temporary variables (R_0, R_1 and $\dot{m}$)
- . . . subject to safe-error attacks

Safe-Error Attacks

- Timely induce a fault into the ALU during the multiply operation at iteration i
- Check the output
  - if the result is incorrect (invalid signature or error notification) then the error was effective
    $\Rightarrow d_i = 1$
  - if the result is correct then the multiplication was dummy [safe error]
    $\Rightarrow d_i = 0$
- Re-iterate the attack for another value of i

Lesson

Protection against certain implementation attacks (e.g., SPA) may introduce new vulnerabilities

Montgomery Powering Ladder

- Montgomery exponentiation algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$
2. For $i = k - 1$ downto 0 do
   - $b \leftarrow 1 - d_i$; $R_b \leftarrow R_0 R_1 \pmod N$
   - $R_{d_i} \leftarrow {R_{d_i}}^2 \pmod N$
3. Return R_0

  - behaves regularly without dummy operations
  - only 2 temporary variables (R_0, R_1)

Summary

- Comparison

Algorithm	Temp. var.	# mult.
Square-and-multiply	$1 + 1$	$k + k/2$
Square-and-multiply-*always*	$2 + 1$	$k + k$
Montgomery ladder	2	$k + k$

- Side-channel atomicity
  - converts any crypto-algorithm into a protected algorithm with (virtually) no penalty

Bibliography

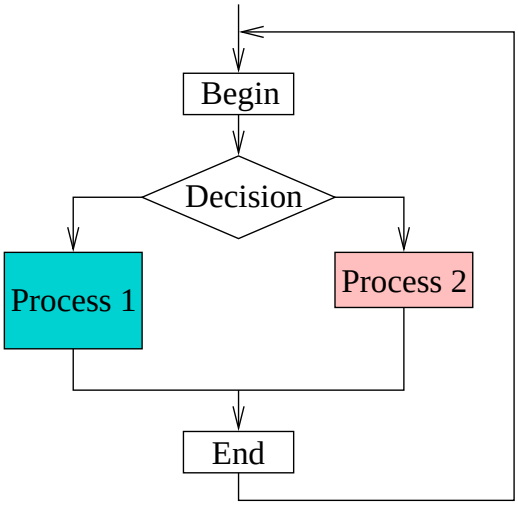
A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone
Handbook of Applied Cryptography
Chapter 14, CRC Press, 1997

S.-M. Yen and M. Joye
Checking before output may not be enough against fault-based cryptanalysis
IEEE Trans. Computers, **49**(9):967–970, 2000

P.L. Montgomery
Speeding the Pollard and elliptic curve methods of factorization
Math. Comp. **48**(177):243–264, 1987

General Principle

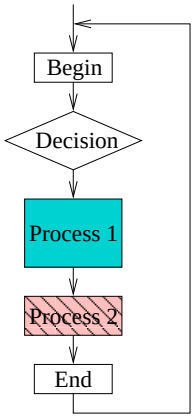
- Example of a crypto-algorithm



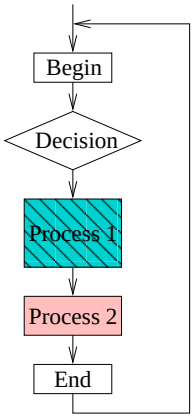
- Side-channel information
  - timing, power consumption, etc. . .

General Principle (1/3)

- Straightforward solution



. . . for executing Process 1



. . . for executing Process 2

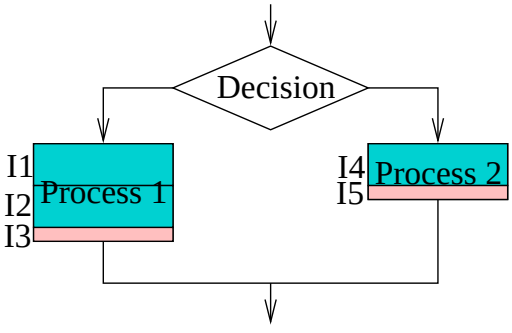
provided that a fake execution is indistinguishable from a true execution!

technicolor

General Principle (2/3)

Side-channel atomicity

Refinement of the straightforward solution so that the running time is not (too much) penalized



Process 1 = $\mathrm{I1}||\mathrm{I2}||\mathrm{I3}$ and Process 2 = $\mathrm{I4}||\mathrm{I5}$

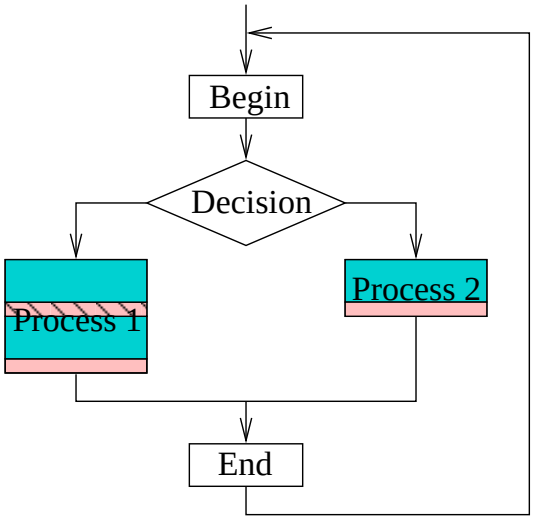
Common atomic block

: $\mathrm{I1}||\mathrm{I3}^{(\mathrm{fake})} \sim \mathrm{I2}||\mathrm{I3} \sim \mathrm{I4}||\mathrm{I5}$

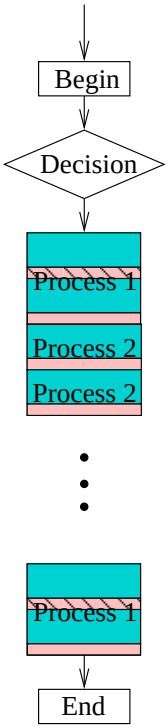
technicolor

General Principle (3/3)

- The whole crypto-algorithm



with chained blocks →

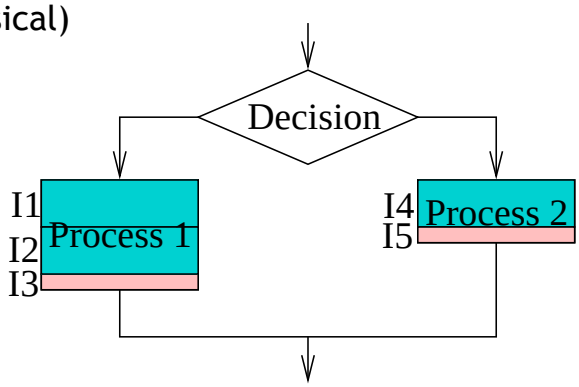


technicolor

Atomic Square-and-Multiply (1/3)

- Application of the 'General Principle'
  - square-and-multiply algorithm (classical)

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - If $(d_i = 1)$ then $R_0 \leftarrow R_0 R_1 \pmod N$
   - $i \leftarrow i - 1$
3. Return R_0



Assumptions

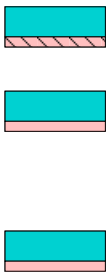
- $[R_0 \leftarrow R_0 R_0 \pmod N] \sim [R_0 \leftarrow R_0 R_1 \pmod N]$
- $[i \leftarrow i - 1] \sim [i \leftarrow i - 0]$

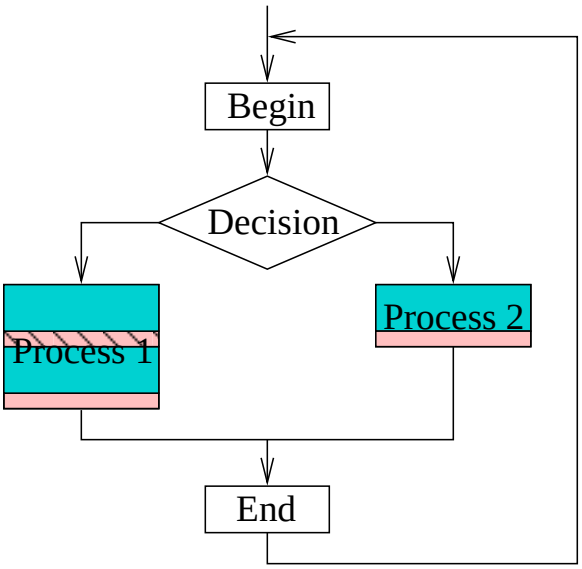
technicolor

Atomic Square-and-Multiply (2/3)

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   Case
   $(d_i = 1)$: /* Process 1 */
   - $R_0 \leftarrow R_0 R_0 \pmod N$
   - $i \leftarrow i - 0$
   - $R_0 \leftarrow R_0 R_1 \pmod N$
   - $i \leftarrow i - 1$

   $(d_i = 0)$: /* Process 2 */
   - $R_0 \leftarrow R_0 R_0 \pmod N$
   - $i \leftarrow i - 1$
3. Return R_0



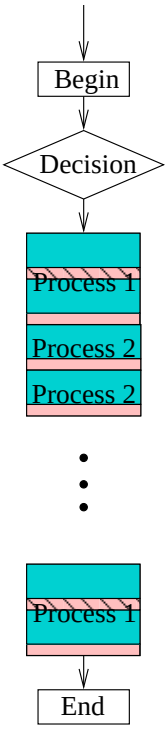


technicolor

Atomic Square-and-Multiply (3/3)

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   Case
   $(d_i = 1)$: /* Process 1 */
   - $R_0 \leftarrow R_0 R_0 \pmod N$
   - $i \leftarrow i - 0$
   - $R_0 \leftarrow R_0 R_1 \pmod N$
   - $i \leftarrow i - 1$

   $(d_i = 0)$: /* Process 2 */
   - $R_0 \leftarrow R_0 R_0 \pmod N$
   - $i \leftarrow i - 1$
3. Return R_0



Chaining the blocks

technicolor

Chaining the Blocks (1/4)

Methodology

1. Each process is divided into common atomic blocks
2. Each block inside a process receives a number r (in chronological order)
3. A bit s is used to keep track whether there remain blocks to be executed in the current process

The trick:

$$r \leftarrow (\neg s) \cdot (r+1) + s \cdot f(\text{input values})$$

- $s = 0 \implies r \leftarrow r + 1$
- $s = 1 \implies r \leftarrow f(\text{input values})$

Chaining the Blocks (2/4)

- Square-and-multiply algorithm

		r	s
$(d_i = 1)$	$R_0 \leftarrow R_0 R_0 \pmod N;\ i \leftarrow i - 0$	0	0
	$R_0 \leftarrow R_0 R_1 \pmod N;\ i \leftarrow i - 1$	1	1
$(d_i = 0)$	$R_0 \leftarrow R_0 R_0 \pmod N;\ i \leftarrow i - 1$	2	1

- We can choose for the common atomic block ()

$$R_0 \leftarrow R_0 R_{(r \bmod 2)} \pmod N; i \leftarrow i - s$$

and for the update

$$\begin{cases} r = (\neg s) \cdot (r+1) + s \cdot f(d_i) & \text{with } f(d_i) = 2 \cdot (\neg d_i) \\ s = (r \bmod 2) + (r \operatorname{div} 2) \end{cases}$$

Chaining the Blocks (3/4)

- Resulting algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$; $s \leftarrow 1$
2. While $(i \geqslant 0)$ do
   - $r \leftarrow (\neg s) \cdot (r+1) + s \cdot 2(\neg d_i)$
   - $s \leftarrow (r \bmod 2) + (r \operatorname{div} 2)$
   - $R_0 \leftarrow R_0 \cdot R_{(r \bmod 2)}$; $i \leftarrow i - s$
3. Return R_0

- after simplification. . .

technicolor

Chaining the Blocks (4/4)

- Atomic square-and-multiply algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$; $b \leftarrow 0$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow R_0 R_b \pmod N$
   - $b \leftarrow b \oplus d_i$; $i \leftarrow i - \neg b$
3. Return R_0

- behaves regularly without dummy operations
- only 2 temporary variables (R_0 and R_1)

technicolor

Further Atomic Algorithms (1/7)

- Right-to-left binary algorithm (classical)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow 0$
2. While $(i \leqslant k-1)$ do
   - If $(d_i = 1)$ then $R_0 \leftarrow R_0 R_1 \pmod N$
   - $R_1 \leftarrow {R_1}^2 \pmod N$; $i \leftarrow i + 1$
3. Return R_0

Further Atomic Algorithms (2/7)

- Right-to-left binary algorithm (atomic)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow 0$; $b \leftarrow 1$
2. While $(i \leqslant k-1)$ do
   - $b \leftarrow b \oplus d_i$
   - $R_b \leftarrow R_b R_1 \pmod N$; $i \leftarrow i + b$
3. Return R_0

Further Atomic Algorithms (3/7)

- ω-bit sliding window algorithm (classical)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N, \omega > 1$
Output: $S = \dot{m}^d \bmod N$
Pre-comp.: $R_{j+1} \leftarrow \dot{m}^{2j+1} \pmod N$, $1 \leqslant j \leqslant 2^{\omega-1} - 1$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k - 1$
2. While $(i \geqslant 0)$ do
   - If $(d_i = 0)$ then $R_0 \leftarrow {R_0}^2 \pmod N$; $i \leftarrow i - 1$
   - Otherwise $(d_i \neq 0)$,
     1. find the longest string $(d_i, d_{i-1}, \ldots, d_\ell)_2$ s.t.
        (a) $i - \ell + 1 \leqslant \omega$ and (b) $e_\ell = 1$
     2. $j \leftarrow (d_i, d_{i-1}, \ldots, d_\ell)_2$
     3. $R_0 \leftarrow {R_0}^{2^{i-\ell+1}} \pmod N$; $i \leftarrow \ell - 1$
3. Return R_0

technicolor

Further Atomic Algorithms (4/7)

- ω-bit sliding window algorithm (atomic)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N, \omega > 1$
Output: $S = \dot{m}^d \bmod N$
Pre-comp.: $R_{j+1} \leftarrow \dot{m}^{2j+1} \pmod N$, $1 \leqslant j \leqslant 2^{\omega-1} - 1$

1. For $j = 1$ to $\omega - 1$ do $d_{-j} \leftarrow 0$
2. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k - 1$; $s \leftarrow 1$
3. While $(i \geqslant 0)$ do
   - $r \leftarrow (\neg s) \cdot (r + 1)$; $b \leftarrow 0$; $t \leftarrow 1$; $l \leftarrow \omega$; $u \leftarrow 0$
   - For $j = 1$ to ω do
     $b \leftarrow b \vee d_{i-\omega+j}$; $l \leftarrow l - (\neg b)$
     $u \leftarrow u + t \cdot d_{i-\omega+j}$; $t \leftarrow b \cdot (2t) + \neg b$
   - $l \leftarrow l \cdot d_i$; $u \leftarrow [(u + 1) \operatorname{div} 2] \cdot d_i$; $s \leftarrow (r = l)$
   - $R_0 \leftarrow R_0 R_{u \cdot s} \pmod N$; $i \leftarrow i - u \cdot s - \neg d_i$
4. Return R_0

technicolor

Further Atomic Algorithms (5/7)

- (M, M^3) algorithm (classical)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $R_2 \leftarrow \dot{m}^3 \pmod N$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow {R_0}^2 \pmod N$; $i \leftarrow i-1$
   - If $(d_i = 1)$ then
     if $(d_{i-1} = 0)$ then $R_0 \leftarrow R_0 R_1 \pmod N$
     else $(d_{i-1} = 1)$
     $R_0 \leftarrow {R_0}^2 \pmod N$; $R_0 \leftarrow R_0 R_2 \pmod N$
     $i \leftarrow i-1$
3. Return R_0

Further Atomic Algorithms (6/7)

- (M, M^3) algorithm (atomic)

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $R_2 \leftarrow \dot{m}^3 \pmod N$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   - $r \leftarrow (\neg s) \cdot (r+1)$; $s \leftarrow s \oplus d_i \oplus [d_{i-1} \wedge (r \bmod 2)]$
   - $R_0 \leftarrow R_0 R_{r \cdot s} \pmod N$; $i \leftarrow i - r \cdot s - \neg d_i$
3. Return R_0

Further Atomic Algorithms (7/7)

- More involved algorithms
  - e.g., point multiplication on elliptic curves over $\mathbb{F}_p$
  - table-based methods apply

Summary

- Side-channel atomicity
  - generic method to convert an algorithm into a SPA-protected algorithm
  - applies to a large variety of crypto-algorithms
  - can be combined with other techniques for preventing other classes of attacks
- Ex: atomic square-and-multiply algorithm
  - behaves regularly
  - complexity: $3k/2$ multiplications
    - as in the classical (i.e., unprotected) algorithm!

Bibliography


M. Joye
Recovering lost efficiency of exponentiation algorithms on smart cards

Electronics Letters, **38**(19):1095–1097, 2002

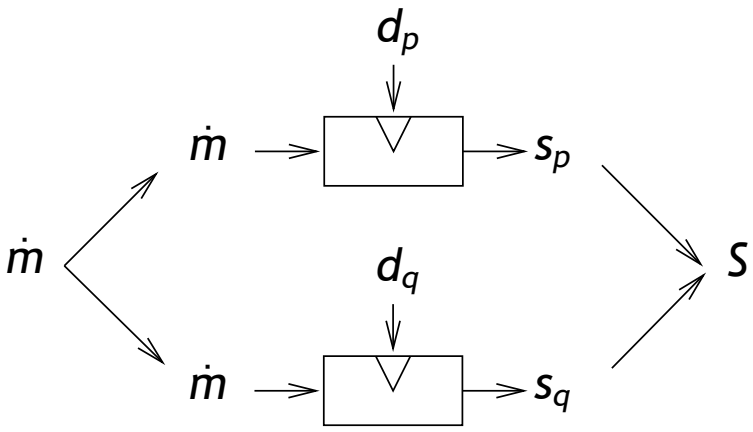

B. Chevallier-Mames, M. Ciet, and M. Joye
Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity
IEEE Trans. Computers, **53**(6):760–768, 2004

Modes of Computation

- Setup:
  - $N = pq$ with p, q prime
  - (e, d) satisfying $e \cdot d \equiv 1 \pmod{\phi(N)}$
- Public parameters: $\{e, N\}$
- Private parameters:
  - standard mode: $\{d, N\}$
  - CRT mode: $\{p, q, d_p, d_q, i_q\}$ where $\begin{cases} d_p = d \bmod (p-1) \\ d_q = d \bmod (q-1) \\ i_q = q^{-1} \bmod p \end{cases}$

Chinese Remaindering

- Computation of a signature $S = \dot{m}^d \bmod N$ using CRT
  1. $s_p = \dot{m}^{d_p} \bmod p$
  2. $s_q = \dot{m}^{d_q} \bmod q$
  3. $S = \mathrm{CRT}(s_p, s_q) = s_q + q[i_q(s_p - s_q) \bmod p]$



technicolor

SPA-type Attacks

- Preventing SPA-like attacks
  - atomic algorithms
  - Montgomery ladder
  - ...
- Protections against SPA-type attacks are not enough to thwart DPA-like attacks...

technicolor

DPA-type Attacks (1/3)

Recover secret d in the computation of $S = \dot{m}^d \bmod N$

- e.g., in the atomic square-and-multiply algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$; $b \leftarrow 0$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow R_0 R_b \pmod N$
   - $b \leftarrow b \oplus d_i$; $i \leftarrow i - \neg b$
3. Return R_0

DPA-type Attacks (2/3)

- Let $d = (d_{k-1}, \ldots, d_0)_2$
- At step j, the attacker
  - already knows bits $d_{k-1}, d_{k-2}, \ldots, d_{j+1}$
  - guesses that next bit $d_j = 1$
  - chooses t [padded] messages $\dot{m}_1, \ldots, \dot{m}_t$ and computes
$$X_i = \dot{m}_i^{(d_{k-1}, d_{k-2}, \ldots, d_{j+1}, d_j)_2} \bmod N \quad \text{for } 1 \leqslant i \leqslant t$$
  - prepares two sets
$$\mathcal{S}_0 = \{\dot{m}_i \mid g(X_i) = 0\} \text{ and } \mathcal{S}_1 = \{\dot{m}_i \mid g(X_i) = 1\}$$
  - if $\langle \mathcal{C}(i) \rangle_{\substack{1 \leqslant i \leqslant t \\ X_i \in \mathcal{S}_0}} - \langle \mathcal{C}(i) \rangle_{\substack{1 \leqslant i \leqslant t \\ X_i \in \mathcal{S}_1}} \begin{cases} \approx 0 & \text{then } d_j = 0 \\ \not\approx 0 & \text{then } d_j = 1 \end{cases}$
- Iterate the attack to find $d_{j-1}, \ldots$

DPA-type Attacks (3/3)

- The previous attack requires that
  1. the crypto device computes $S = \dot{m}^d \bmod N$ for known [padded] messages $\dot{m}$
     - does not apply to PSS-R
  2. the attacker can evaluate

$$g(X_i) \quad \text{with } X_i = \dot{m}_i^{(d_{k-1},\ldots,d_j)_2} \bmod N$$

Countermeasure

Randomize $\dot{m}$, d or N in the computation of $S = \dot{m}^d \bmod N$

DPA-type Countermeasures (1/3)

Randomizing $\dot{m}$ (useless for probabilistic paddings)

- For a random r, compute
  1. $\dot{m}^* = r^e \dot{m} \bmod N$
  2. $S^* = (\dot{m}^*)^d \bmod N$
  3. $S = S^* r^{-1} \bmod N$
- If e is unknown, compute
  1. $\dot{m}^* = r \dot{m} \bmod N$
  2. $S^* = (\dot{m}^*)^d \bmod N$
  3. $S = S^* r^{-d} \bmod N$
- For a [short] random $r < 2^\ell$, compute
  1. $\dot{m}^* = \dot{m} + r N$ and $N^* = 2^\ell N$
  2. $S^* = (\dot{m}^*)^d \bmod N^*$
  3. $S = S^* \bmod N$

DPA-type Countermeasures (2/3)

Randomizing d

- For a [short] random r, compute
  1. $d^* = d + r\,\phi(N)$
  2. $S = \dot{m}^{d^*} \bmod N$
- If $\phi(N)$ is unknown, compute
  1. $d^* = d + r(e\,d - 1)$
  2. $S = \dot{m}^{d^*} \bmod N$
- If e is unknown, for a random $r \in [0, d]$, compute
  1. $d^* = d - r$
  2. $S_1^* = \dot{m}^{d^*} \bmod N$ and $S_2^* = \dot{m}^r \bmod N$
  3. $S = S_1^*\,S_2^* \bmod N$

DPA-type Countermeasures (3/3)

Randomizing N (combination with previous technique)

- For [short] randoms r_1 and $r_2 > r_1$, compute
  1. $\dot{m}^* = \dot{m} + r_1\,N$ and $N^* = r_2\,N$
  2. $S^* = (\dot{m}^*)^d \bmod N^*$
  3. $S = S^* \bmod N$

Fault Attacks

- Randomizing N also protects against fault attacks (when e is unknown)
  - For [short] randoms r_1 and $r_2 > r_1$, compute
    1. $\dot{m}^* = \dot{m} + r_1 N$ and $N^* = r_2 N$
    2. $S^* = (\dot{m}^*)^d \bmod N^*$
    3. $Y = (\dot{m}^*)^{d \bmod \phi(r_2)} \bmod r_2$
    4. $c = (S^* - Y + 1) \bmod r_2$
    5. $S = (S^*)^c \bmod N$
  - (when e is known, the correctness of the computation of S can be checked by verifying the validity of S)

technicolor

Summary

How to implement the RSA

- Use a regular exponentiation algorithm to prevent SPA-like attacks
- Randomize the inputs to prevent DPA-like attacks
- Check the computations to prevent fault attacks

technicolor

Bibliography

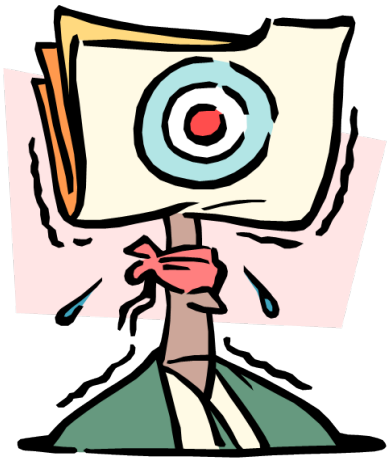
RSA Laboratories
PKCS #1 v2.1: RSA Cryptography Standard
June 14, 2002

D. Boneh, R.A. DeMillo, and R.J. Lipton
On the importance of eliminating errors in cryptographic computations
J. Cryptology, **14**(2):101–119, 2001

Final Recommendations

- Consider side-channel attacks when implementing cryptographic routines
  - check that the countermeasures do not introduce new vulnerabilities
- Avoid decisional tests
- Randomize the execution
- Combine hardware and software protections
- Always prefer cryptographic standards

Comments/Questions?



http://www.thlab.net/~joyem/

technicolor