Embedding Crypto in SoCs: Threats and Protections

Arnaud Tisserand

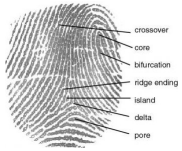
CNRS, Lab-STICC laboratory

GDR SoC'17, Bordeaux

Summary

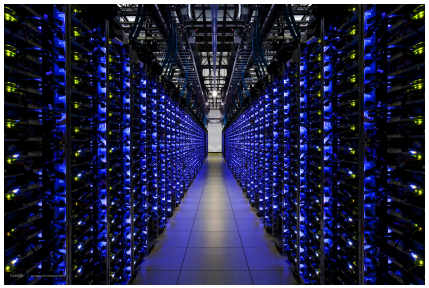
- Introduction & Cryptographic Background
- Side Channel Attacks
- Fault Injection Attacks
- Protections Examples
- Conclusion and References

Applications with Security Needs













Applications: smart cards, computers, Internet, telecommunications, set-top boxes, data storage, RFID tags, WSN, smart grids...

Cryptographic Features

Objectives:

- Confidentiality
- Integrity
- Authenticity
- Non-repudiation
- ...

Cryptographic Features

Objectives:

- Confidentiality
- Integrity
- Authenticity
- Non-repudiation
- ...

Cryptographic primitives:

- Encryption
- Digital signature
- Hash function
- Random numbers generation
- ...

Cryptographic Features

Objectives:

- Confidentiality
- Integrity
- Authenticity
- Non-repudiation
- ...

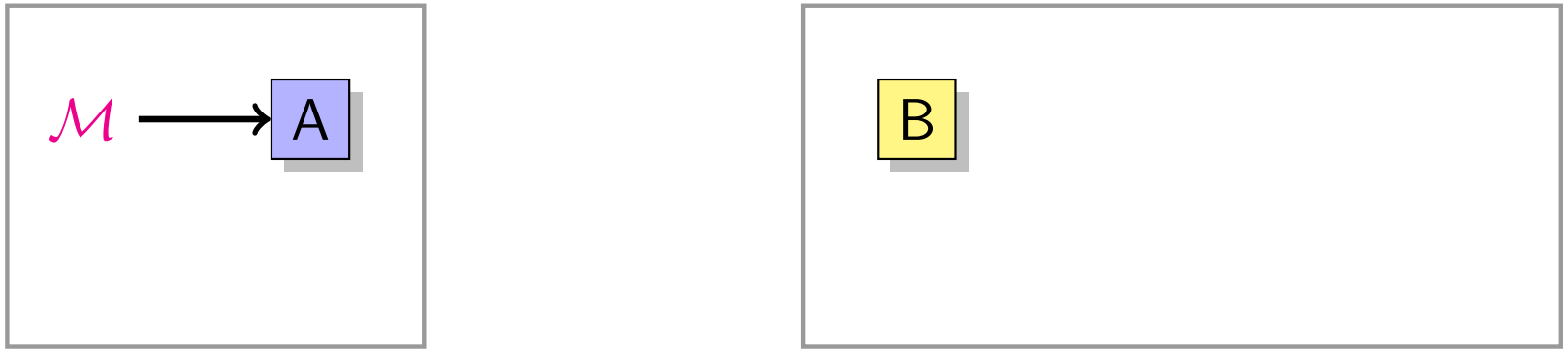
Cryptographic primitives:

- Encryption
- Digital signature
- Hash function
- Random numbers generation
- ...

Implementation issues:

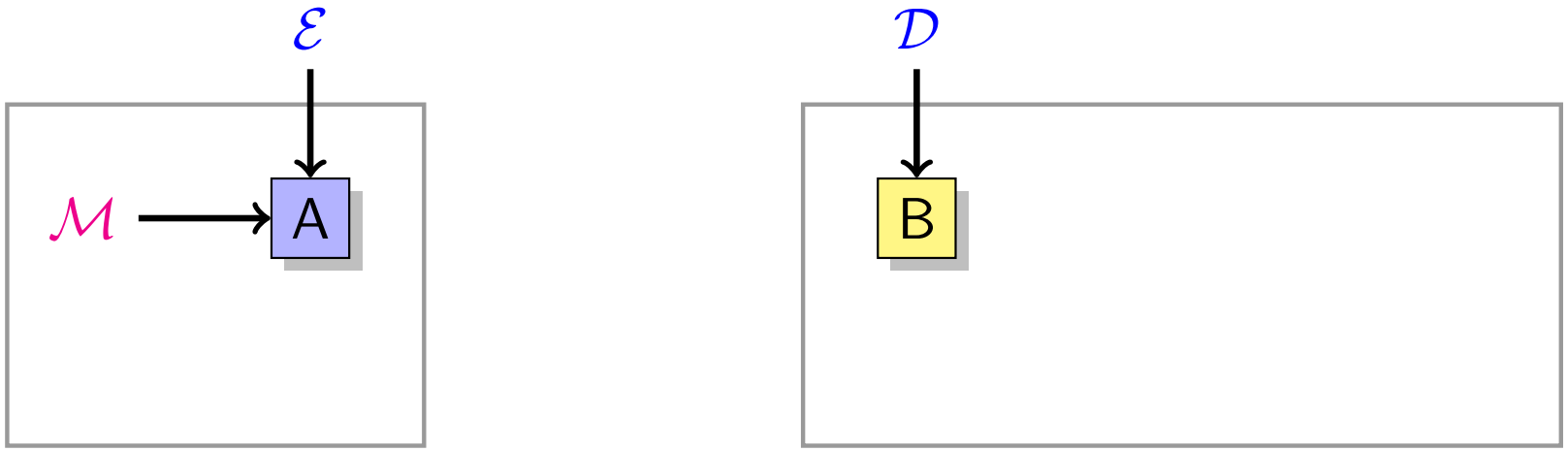
- Performances: speed, delay, throughput, latency
- Cost: device (memory, size, weight), low power/energy consumption, design
- Security: protection against attacks

Symmetric / Private-Key Cryptography



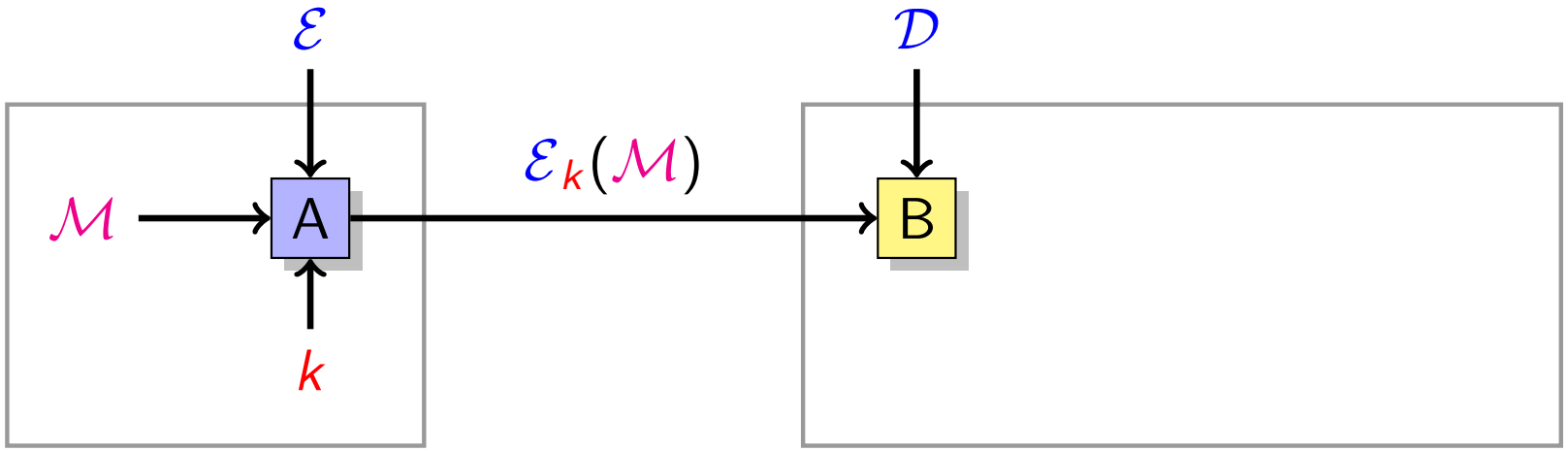
- A : Alice, B : Bob
- $\mathcal{M}$: plain text/message

Symmetric / Private-Key Cryptography



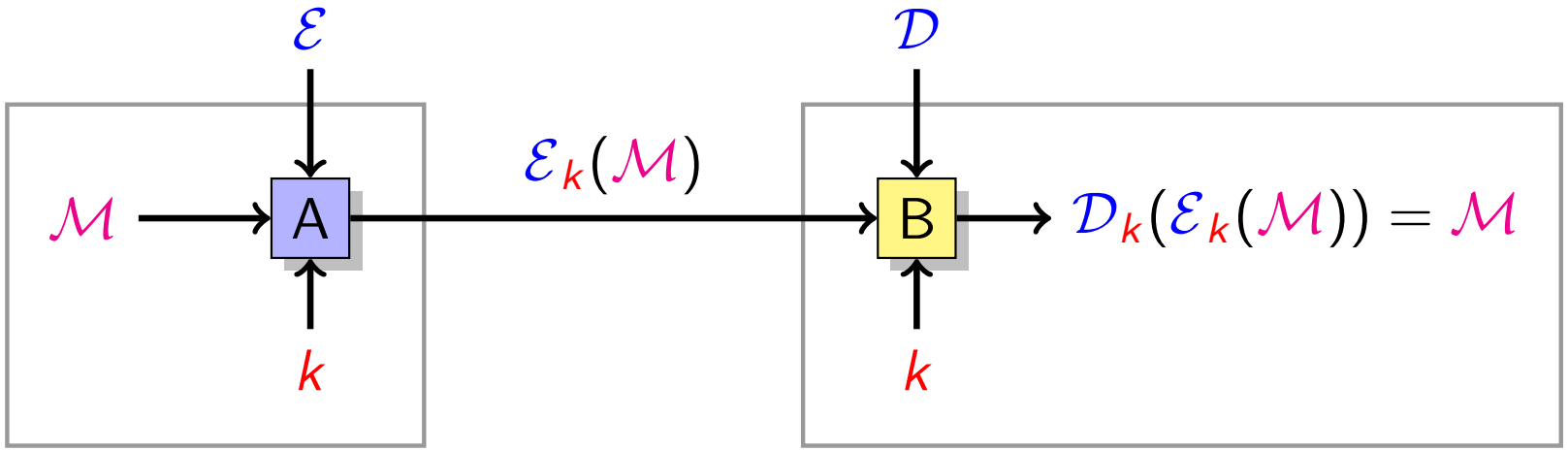
- A : Alice, B : Bob
- $\mathcal{M}$: plain text/message
- $\mathcal{E}$: encryption/ciphering algorithm, $\mathcal{D}$: decryption/deciphering algorithm

Symmetric / Private-Key Cryptography



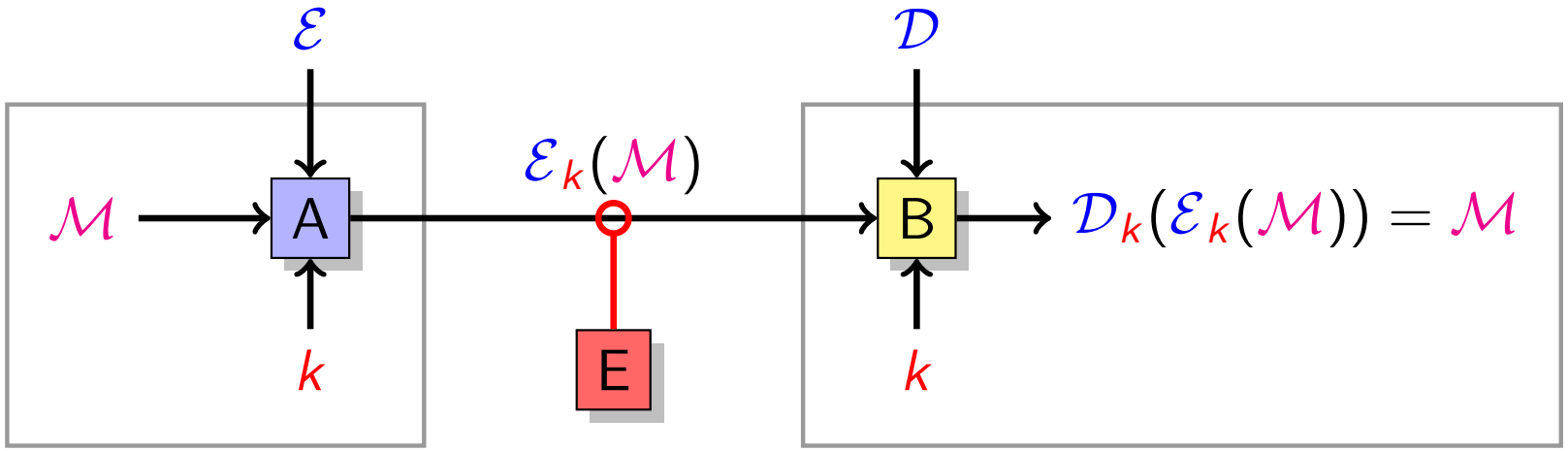
- A : Alice, B : Bob
- $\mathcal{M}$: plain text/message
- $\mathcal{E}$: encryption/ciphering algorithm, $\mathcal{D}$: decryption/deciphering algorithm
- k: secret key
- $\mathcal{E}_k(\mathcal{M})$: encrypted text

Symmetric / Private-Key Cryptography



- A : Alice, B : Bob
- $\mathcal{M}$: plain text/message
- $\mathcal{E}$: encryption/ciphering algorithm, $\mathcal{D}$: decryption/deciphering algorithm
- k: secret key to be shared by A and B
- $\mathcal{E}_k(\mathcal{M})$: encrypted text
- $\mathcal{D}_k(\mathcal{E}_k(\mathcal{M}))$: decrypted text

Symmetric / Private-Key Cryptography



- A : Alice, B : Bob
- $\mathcal{M}$: plain text/message
- $\mathcal{E}$: encryption/ciphering algorithm, $\mathcal{D}$: decryption/deciphering algorithm
- k: secret key to be shared by A and B
- $\mathcal{E}_k(\mathcal{M})$: encrypted text
- $\mathcal{D}_k(\mathcal{E}_k(\mathcal{M}))$: decrypted text
- E : eavesdropper/spy

Advanced Encryption Standard (AES)

Established by NIST in 2001

Symmetric encryption

Block size: 128 bits

key length	#round
128	10
192	12
256	14

Based on substitution-permutation network

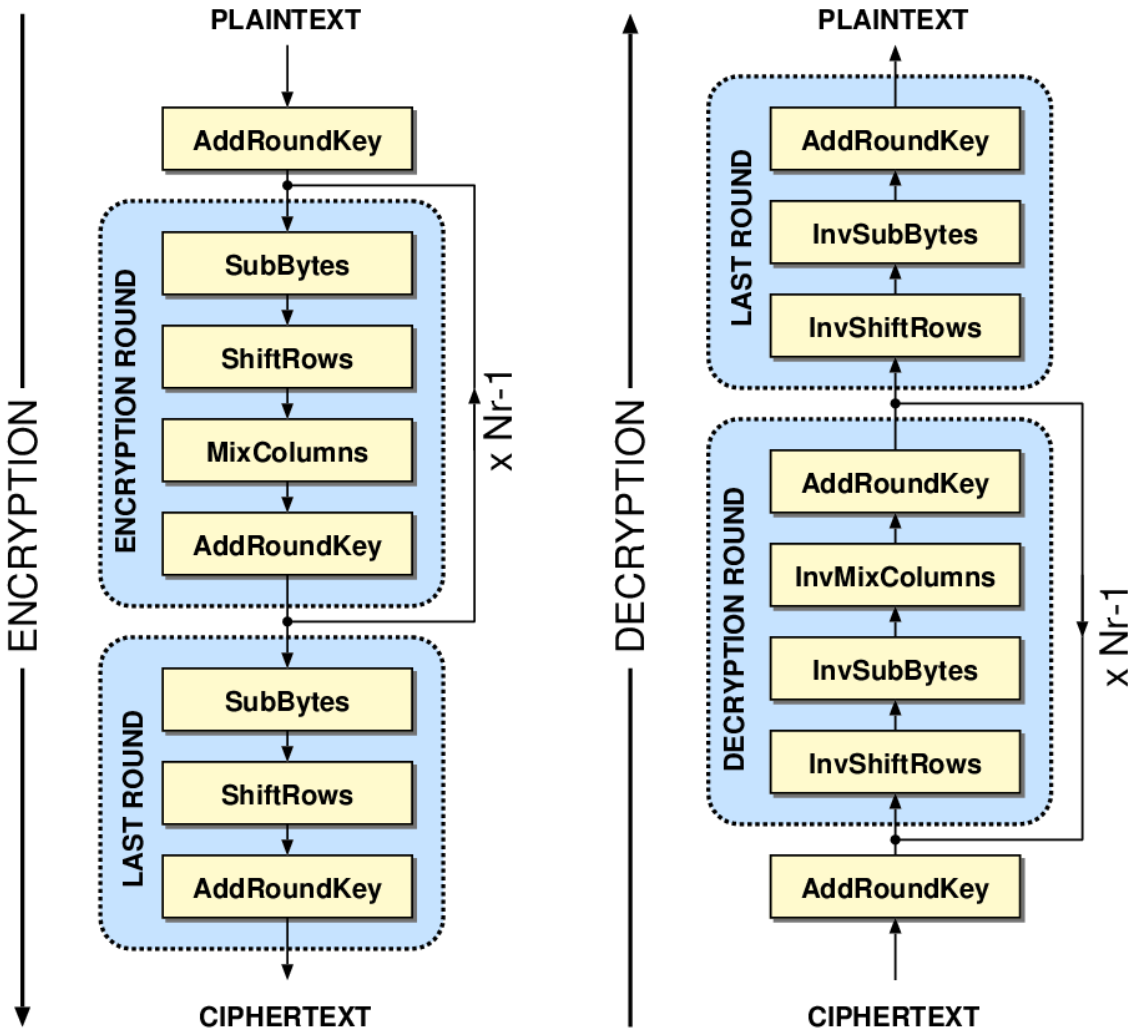
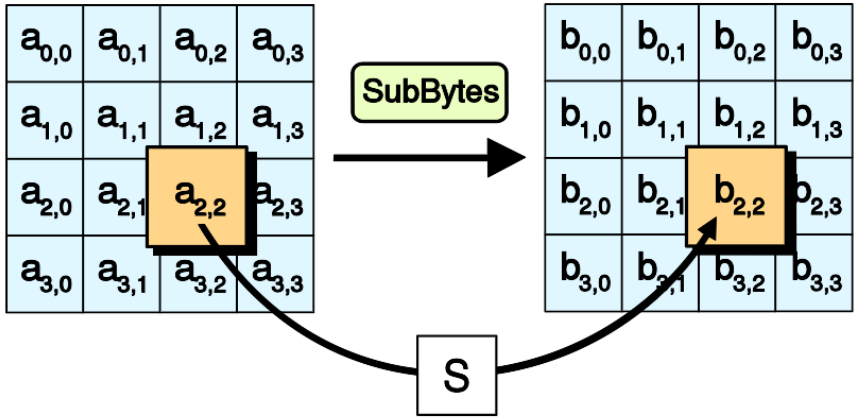
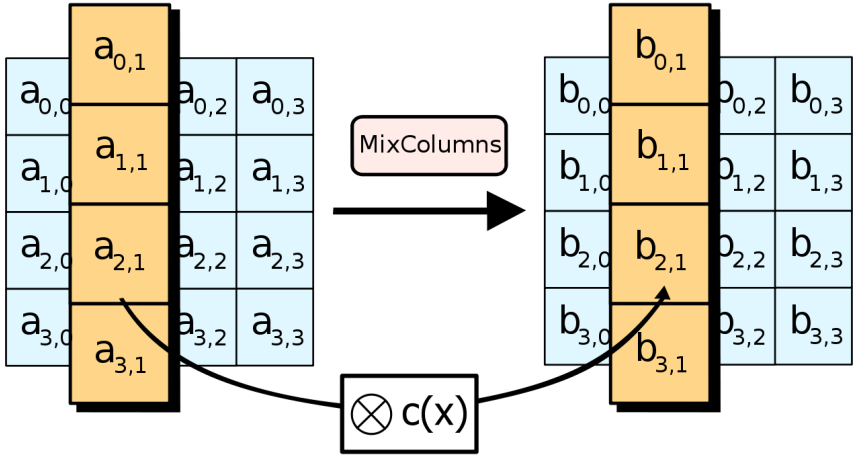


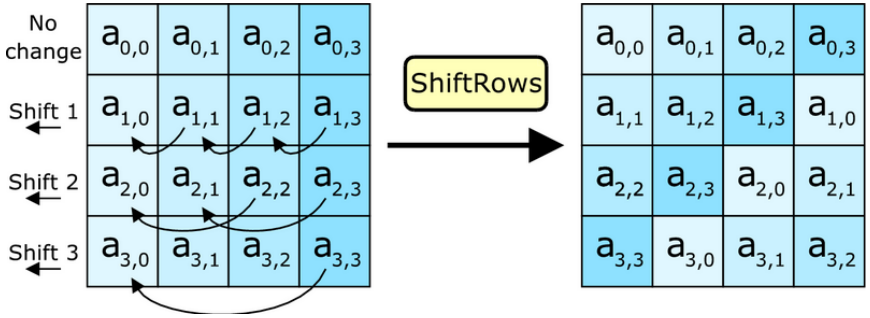
Image source: `http://fr.wikipedia.org/`

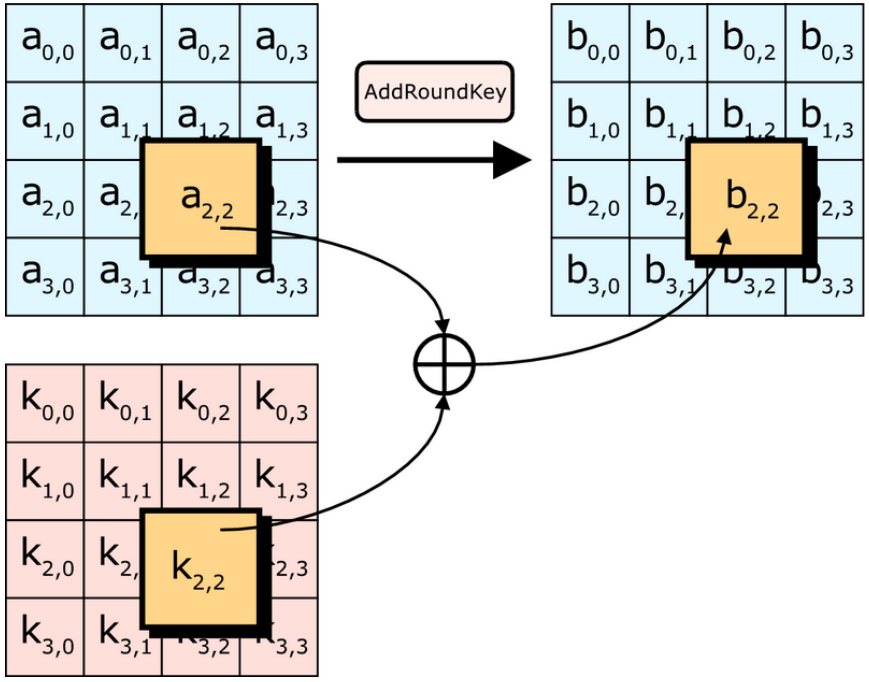
NIST: National Institute of Standards and Technology

AES Round Operations



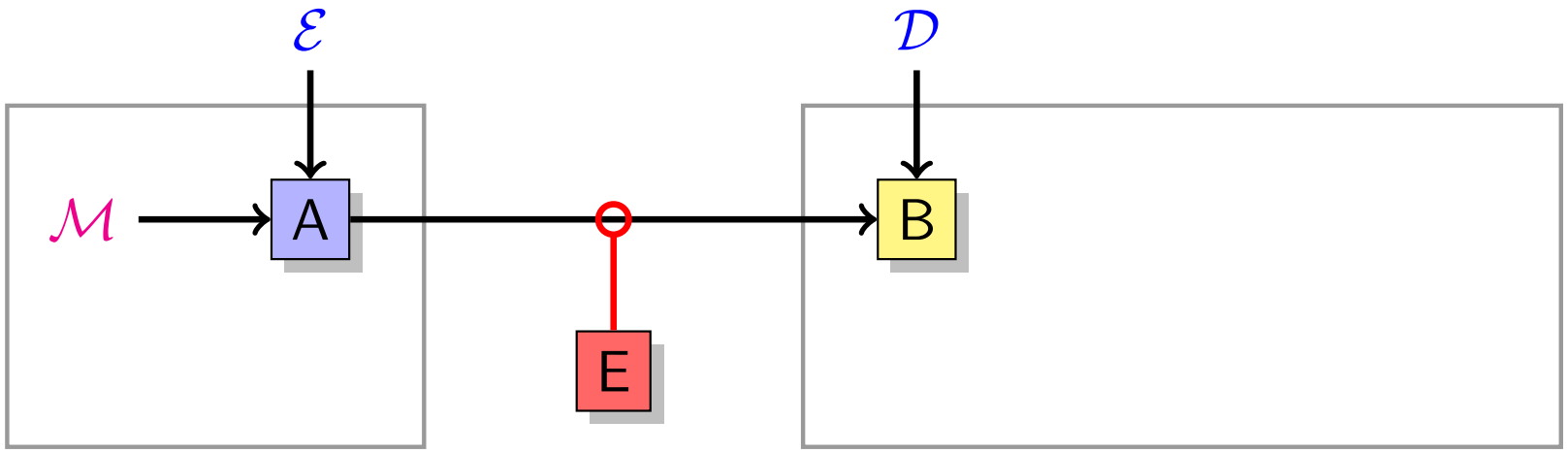




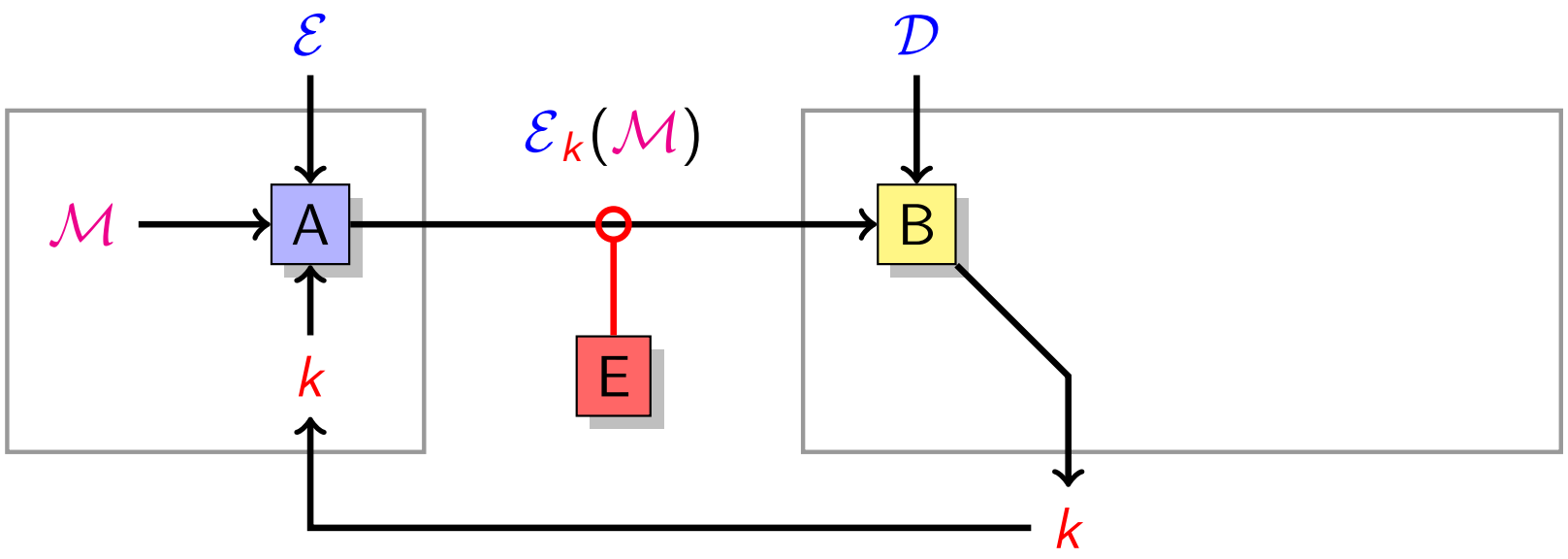


Images source: `http://fr.wikipedia.org/`

Asymmetric / Public-Key Cryptography

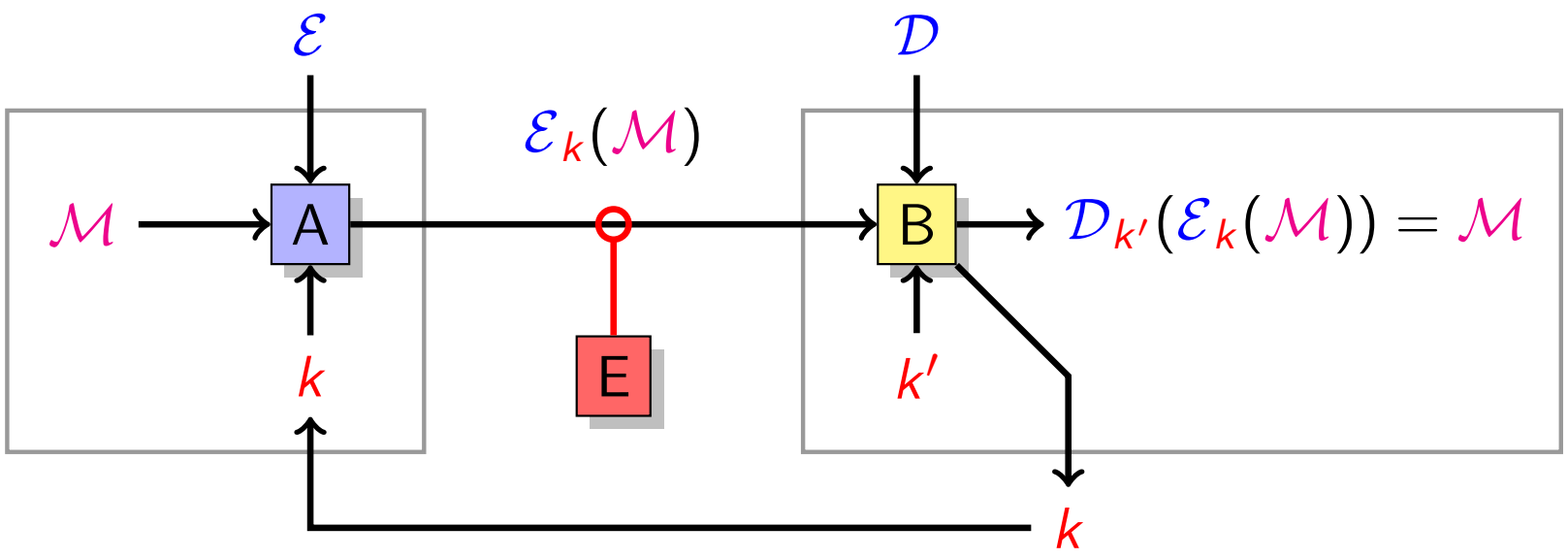


Asymmetric / Public-Key Cryptography



- k: B's public key (known to everyone including E)
- $\mathcal{E}_k(\mathcal{M})$: ciphered text

Asymmetric / Public-Key Cryptography



- k: B's public key (known to everyone including E)
- $\mathcal{E}_k(\mathcal{M})$: ciphered text
- k': B's private key (must be kept secret)
- $\mathcal{D}_{k'}(\mathcal{E}_k(\mathcal{M}))$: deciphered text

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q
- Compute the modulus $n = pq$

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q
- Compute the modulus $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q
- Compute the modulus $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q
- Compute the modulus $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$
- Compute $d = e^{-1} \bmod \varphi(n)$

RSA Asymmetric Cryptosystem (1/2)

Published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman [17]

Key generation (Alice side)

- Choose two large prime integers p and q
- Compute the modulus $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$
- Compute $d = e^{-1} \bmod \varphi(n)$

- Private key (kept secret by Alice): d and also $p, q, \varphi(n)$
- Public key (published): (n, e)

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d

Public key (all): (n, e)

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d

Public key (all): (n, e)

Encryption (Bob side):

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message $\mathtt{M}$ to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message M to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$
- compute the cipher text $c = m^e \bmod n$

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message M to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$
- compute the cipher text $c = m^e \bmod n$

Decryption (Alice side):

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message M to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$
- compute the cipher text $c = m^e \bmod n$

Decryption (Alice side):

- compute $m = c^d \bmod n$

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message M to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$
- compute the cipher text $c = m^e \bmod n$

Decryption (Alice side):

- compute $m = c^d \bmod n$
- convert the integer m to the message M

RSA Asymmetric Cryptosystem (2/2)

Private key (Alice): d Public key (all): (n, e)

Encryption (Bob side):

- convert the message M to an integer m $(1 < m < n \text{ and } \gcd(m, n) = 1)$
- compute the cipher text $c = m^e \bmod n$

Decryption (Alice side):

- compute $m = c^d \bmod n$
- convert the integer m to the message M

Theoretical security: integer factorization, *i.e.* computing (p, q) knowing n, is not possible when n is large enough

Modular Exponentiation

Computation of operations such as : $a^b \bmod n$

$$a^b = \underbrace{a \times a \times a \times a \times \ldots \times a \times a \times a}_{a \text{ appears } b \text{ times}}$$

Modular Exponentiation

Computation of operations such as : $a^b \bmod n$

$$a^b = \underbrace{a \times a \times a \times a \times \ldots \times a \times a \times a}_{a \text{ appears } b \text{ times}}$$

Order of magnitude of exponents: $2^{\text{size of exponent}} \rightsquigarrow 2^{1024} \ldots 2^{2048} \ldots 2^{4096}$

Modular Exponentiation

Computation of operations such as : $a^b \bmod n$

$$a^b = \underbrace{a \times a \times a \times a \times \ldots \times a \times a \times a}_{a \text{ appears } b \text{ times}}$$

Order of magnitude of exponents: $2^{\text{size of exponent}} \rightsquigarrow 2^{1024} \ldots 2^{2048} \ldots 2^{4096}$

Fast exponentiation principle:

$$\begin{aligned} a^b &= (a^2)^{\frac{b}{2}} && \text{when } b \text{ is even} \\ &= a \times (a^2)^{\frac{b-1}{2}} && \text{when } b \text{ is odd} \end{aligned}$$

Least significant bit of the exponent: $\texttt{bit} = 0 \rightsquigarrow$ even and $\texttt{bit} = 1 \rightsquigarrow$ odd

Square and Multiply Algorithm

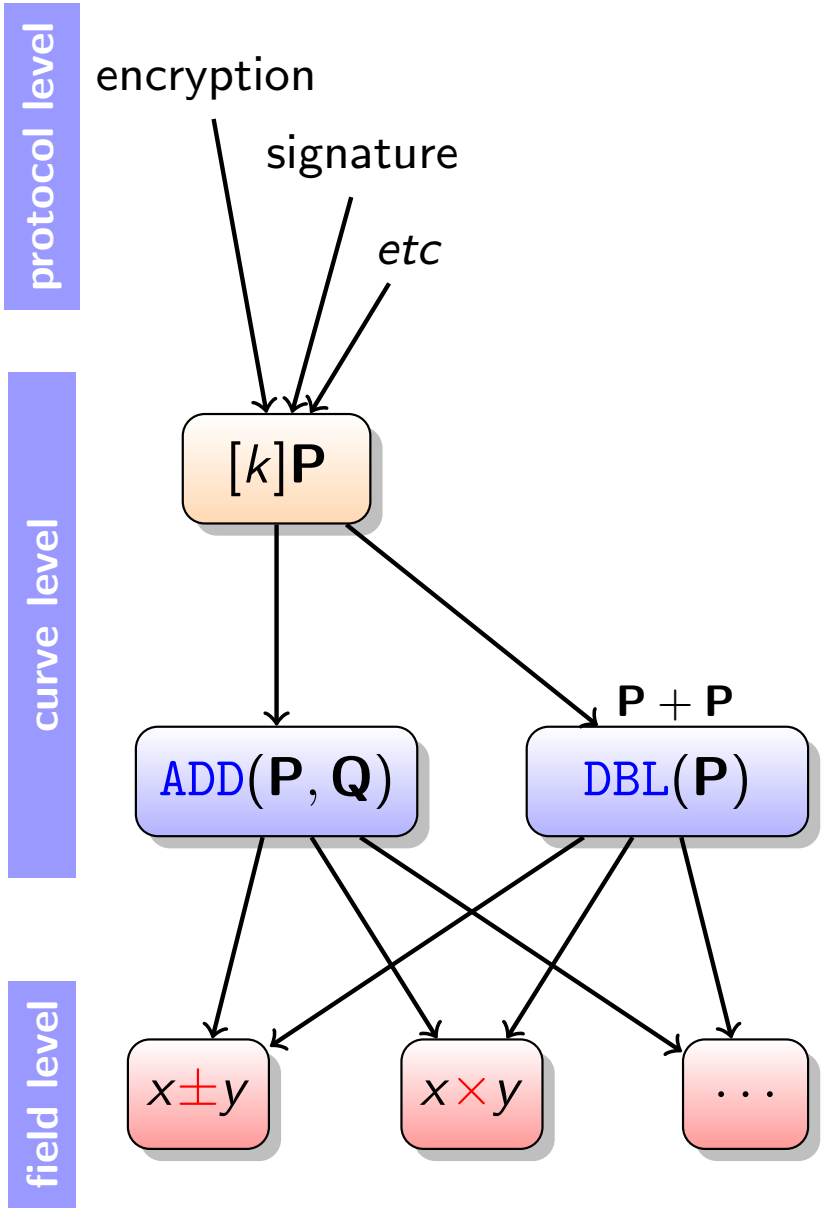
input: a, b, n where $b = (b_{t-1} b_{t-2} \ldots b_1 b_0)_2$

output: $a^b \bmod n$

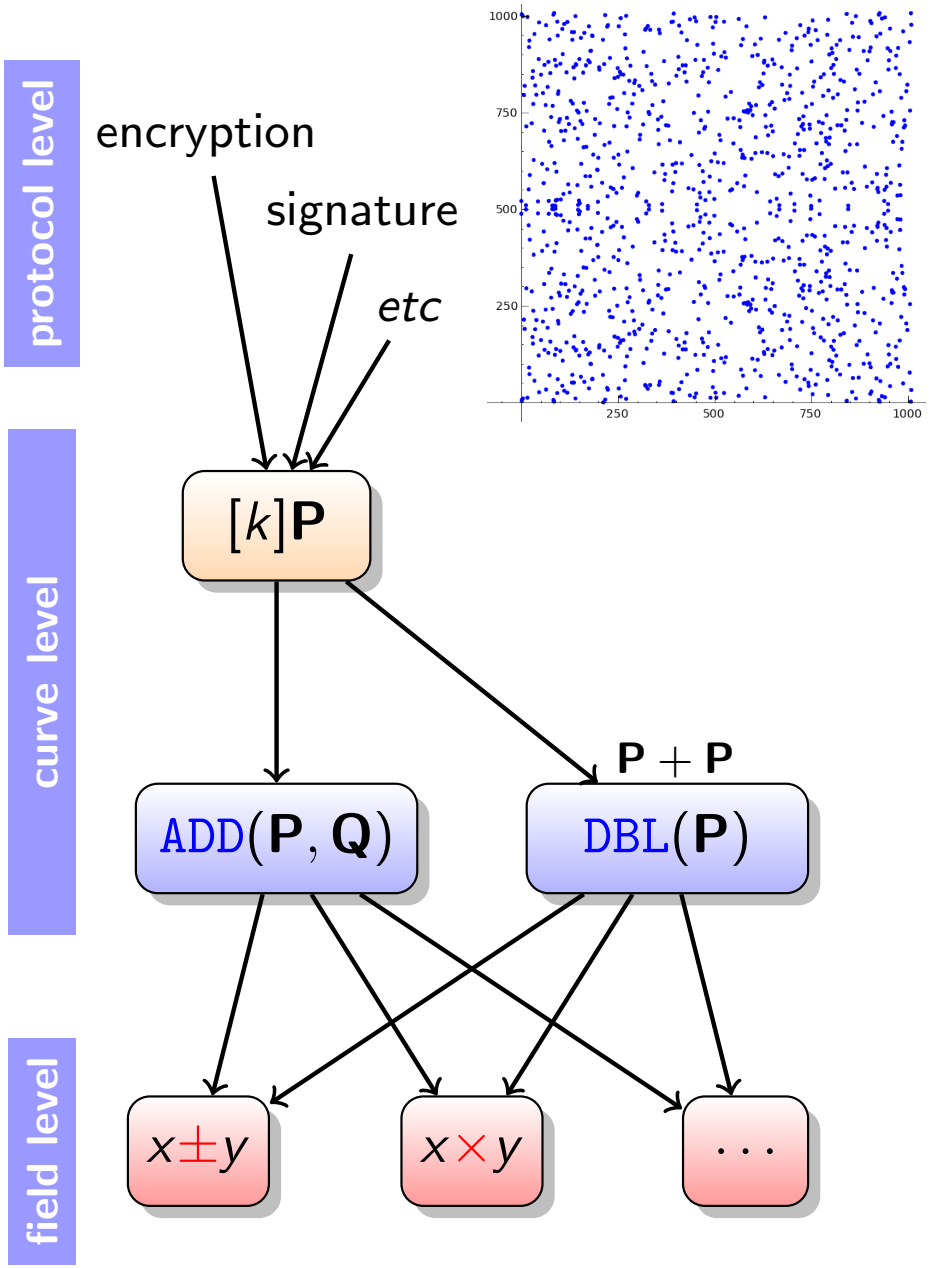
$r = 1$
for i **from** 0 **to** $t-1$ **do**
  if $b_i = 1$ **then**
    $r = r \cdot a \mod n$
  endif
  $a = a^2 \mod n$
endfor
return r

This is the right to left version (there exists a left to right one)

Hardware Accelerators for Elliptic Curve Crypto.



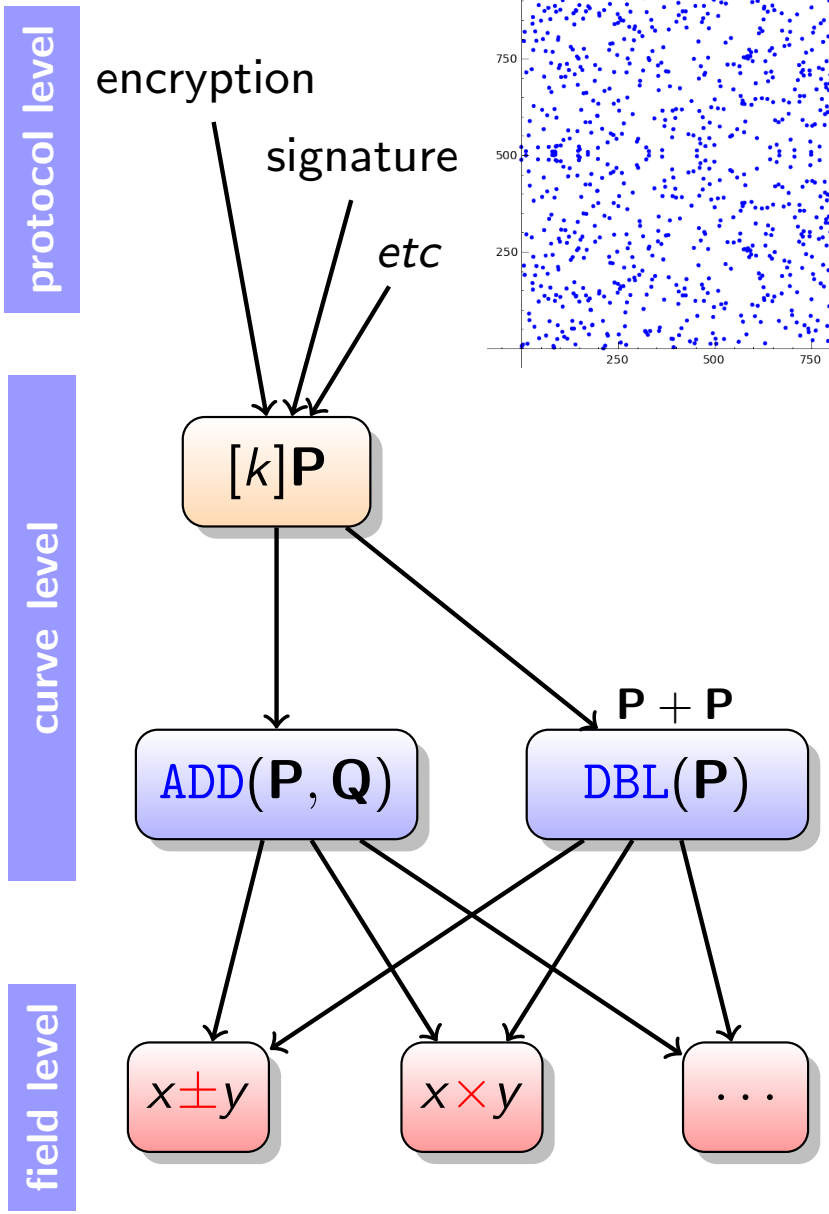
Hardware Accelerators for Elliptic Curve Crypto.

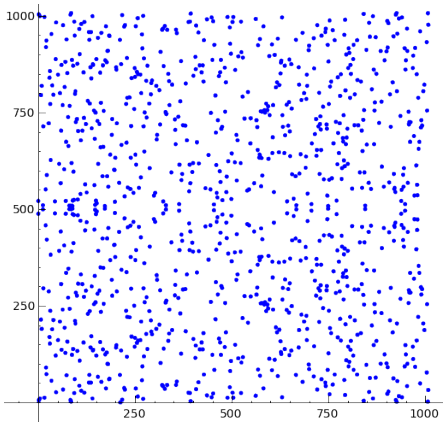


$E : y^2 = x^3 + 4x + 20$ over GF(1009)

points: $\mathbf{P}$, $\mathbf{Q} = (x, y)$ or (x, y, z) or ...

Hardware Accelerators for Elliptic Curve Crypto.





$E : y^2 = x^3 + 4x + 20$ over GF(1009)

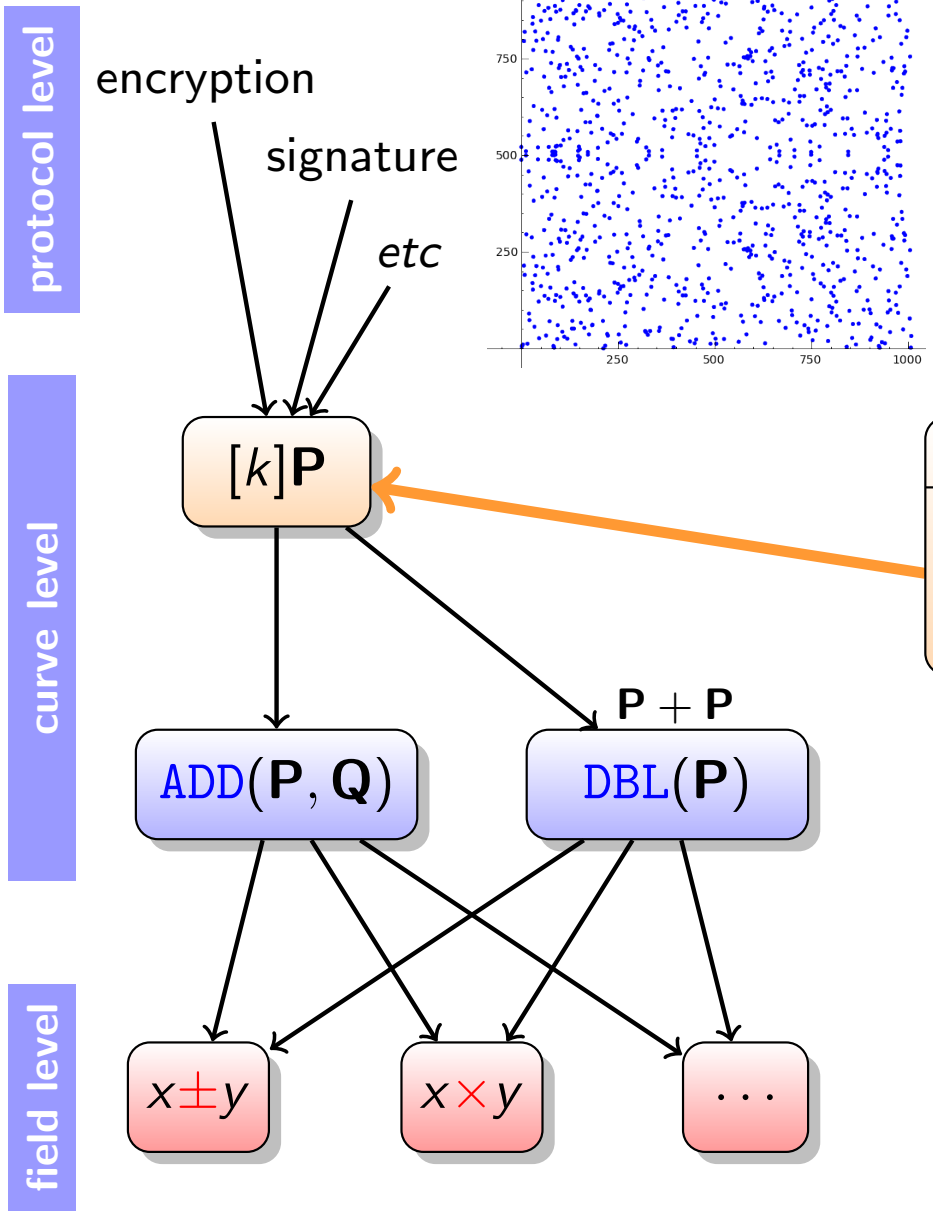
points: $\mathbf{P}$, $\mathbf{Q} = (x, y)$ or (x, y, z) or ...

coordinates: $x, y, z \in \mathrm{GF}(\cdot)$

$\mathrm{GF}(p)$, $\mathrm{GF}(2^m)$, t : 200–600 bits

$k = (k_{t-1}k_{t-2}\ldots k_1k_0)_2 \in \mathbb{N}$

Hardware Accelerators for Elliptic Curve Crypto.



$E : y^2 = x^3 + 4x + 20$ over GF(1009)

points: $\mathbf{P}$, $\mathbf{Q} = (x, y)$ or (x, y, z) or ...

coordinates: $x, y, z \in \mathrm{GF}(\cdot)$

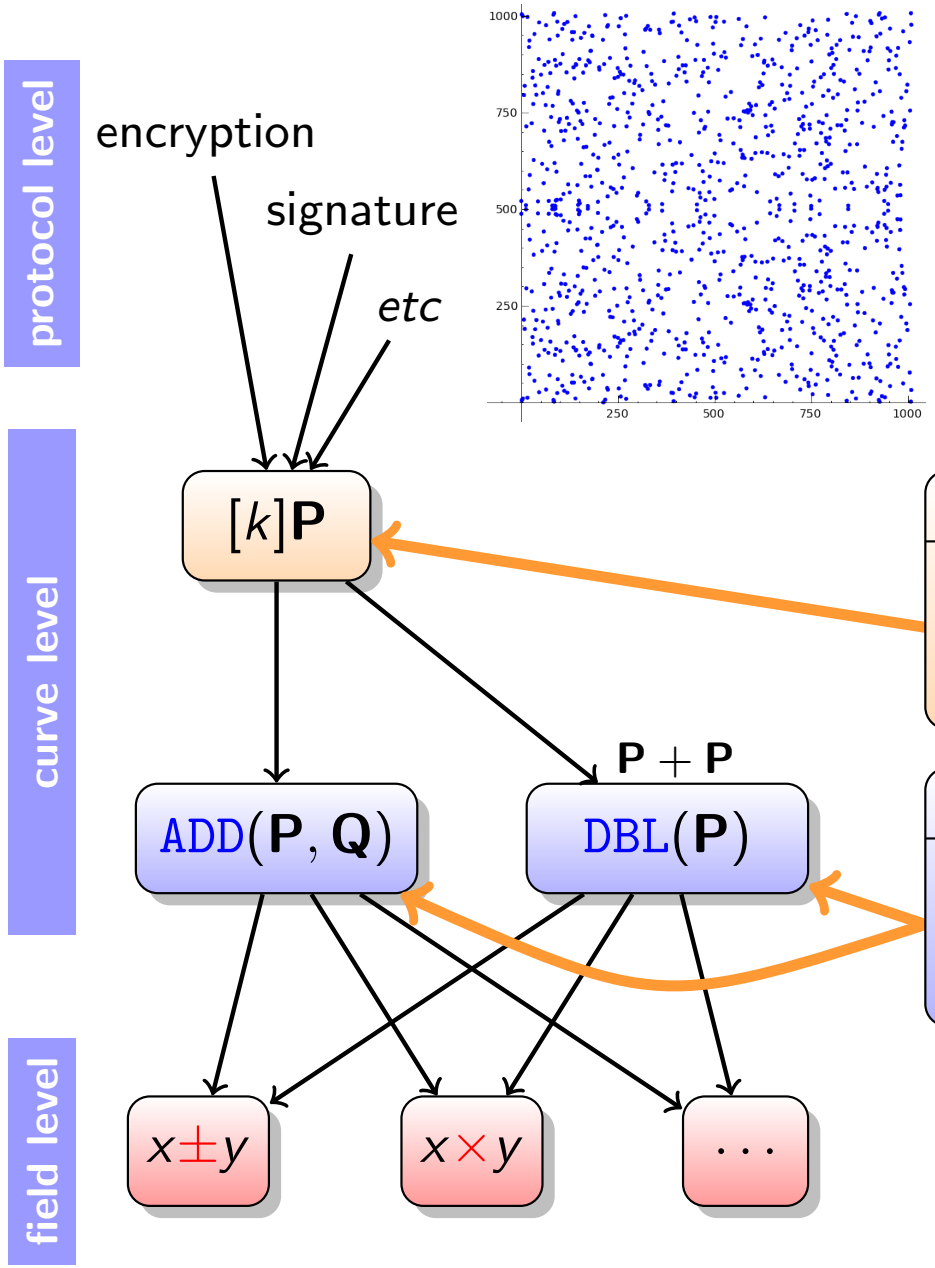
$\mathrm{GF}(p)$, $\mathrm{GF}(2^m)$, t : 200–600 bits

$k = (k_{t-1}k_{t-2}\ldots k_1k_0)_2 \in \mathbb{N}$

Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Hardware Accelerators for Elliptic Curve Crypto.



$E : y^2 = x^3 + 4x + 20$ over GF(1009)

points: $\mathbf{P}$, $\mathbf{Q} = (x, y)$ or (x, y, z) or ...

coordinates: $x, y, z \in \mathrm{GF}(\cdot)$

$\mathrm{GF}(p)$, $\mathrm{GF}(2^m)$, t : 200–600 bits

$k = (k_{t-1}k_{t-2}\ldots k_1k_0)_2 \in \mathbb{N}$

Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

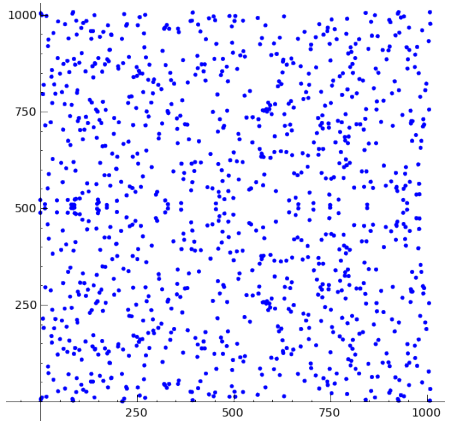
Point addition/doubling operations

sequence of finite field operations

DBL: $v_1 = z_1^2, v_2 = x_1 - v_1, \ldots$

ADD: $w_1 = z_1^2, w_2 = z_1 \times w_1, \ldots$

Hardware Accelerators for Elliptic Curve Crypto.



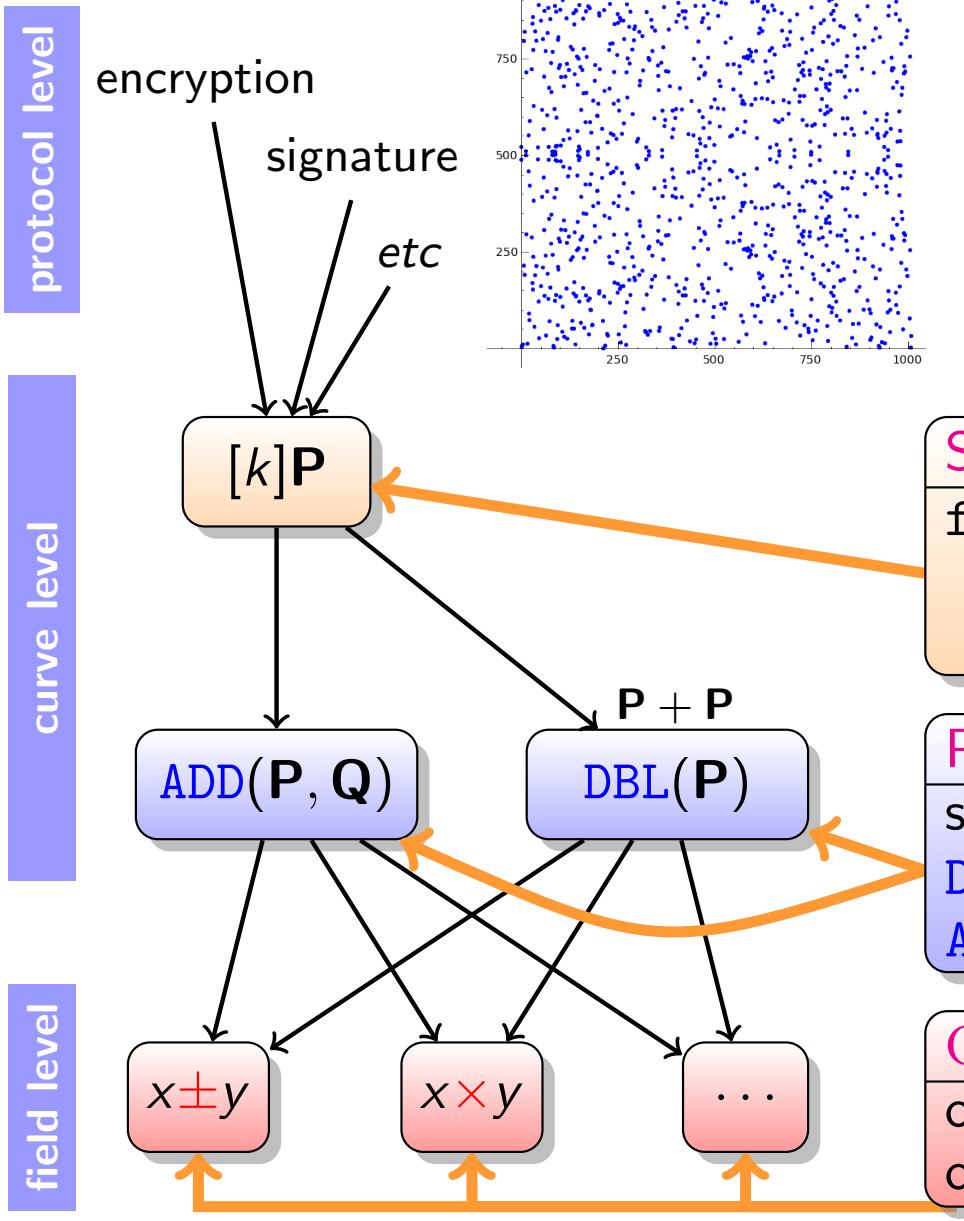
$E : y^2 = x^3 + 4x + 20$ over GF(1009)

points: $\mathbf{P}$, $\mathbf{Q} = (x, y)$ or (x, y, z) or ...

coordinates: $x, y, z \in \mathrm{GF}(\cdot)$

$\mathrm{GF}(p)$, $\mathrm{GF}(2^m)$, t : 200–600 bits

$k = (k_{t-1}k_{t-2}\ldots k_1k_0)_2 \in \mathbb{N}$



Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Point addition/doubling operations

sequence of finite field operations

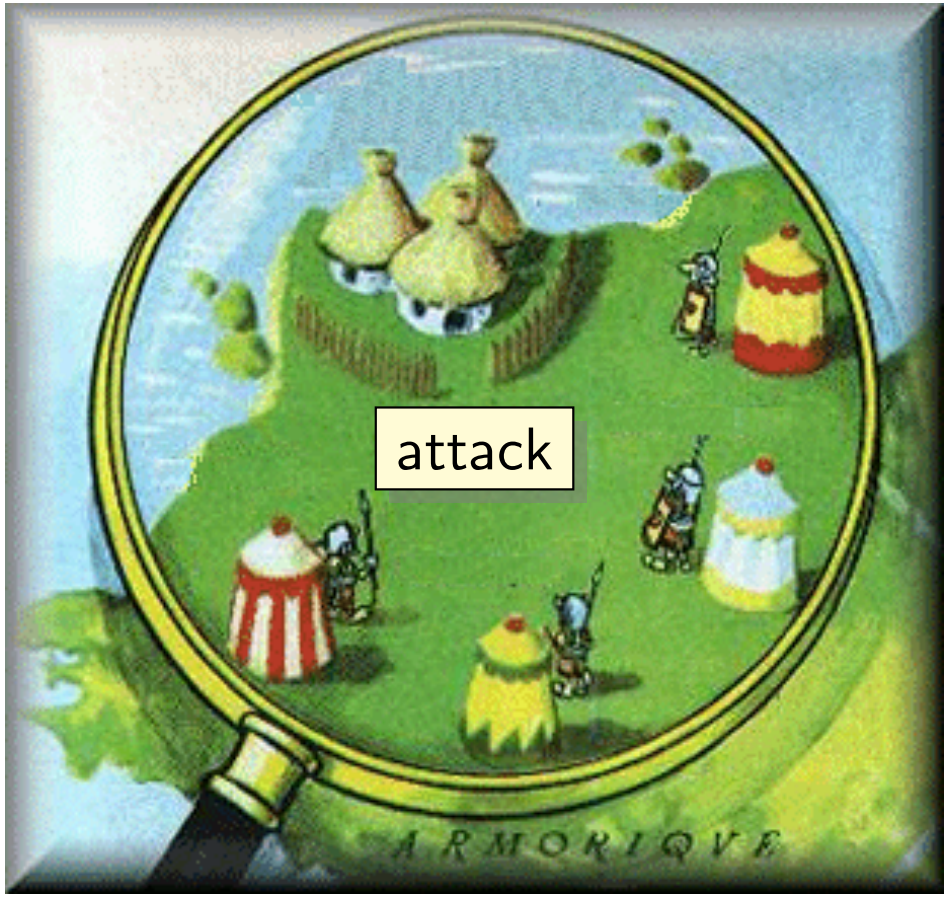
DBL: $v_1 = z_1^2, v_2 = x_1 - v_1, \ldots$

ADD: $w_1 = z_1^2, w_2 = z_1 \times w_1, \ldots$

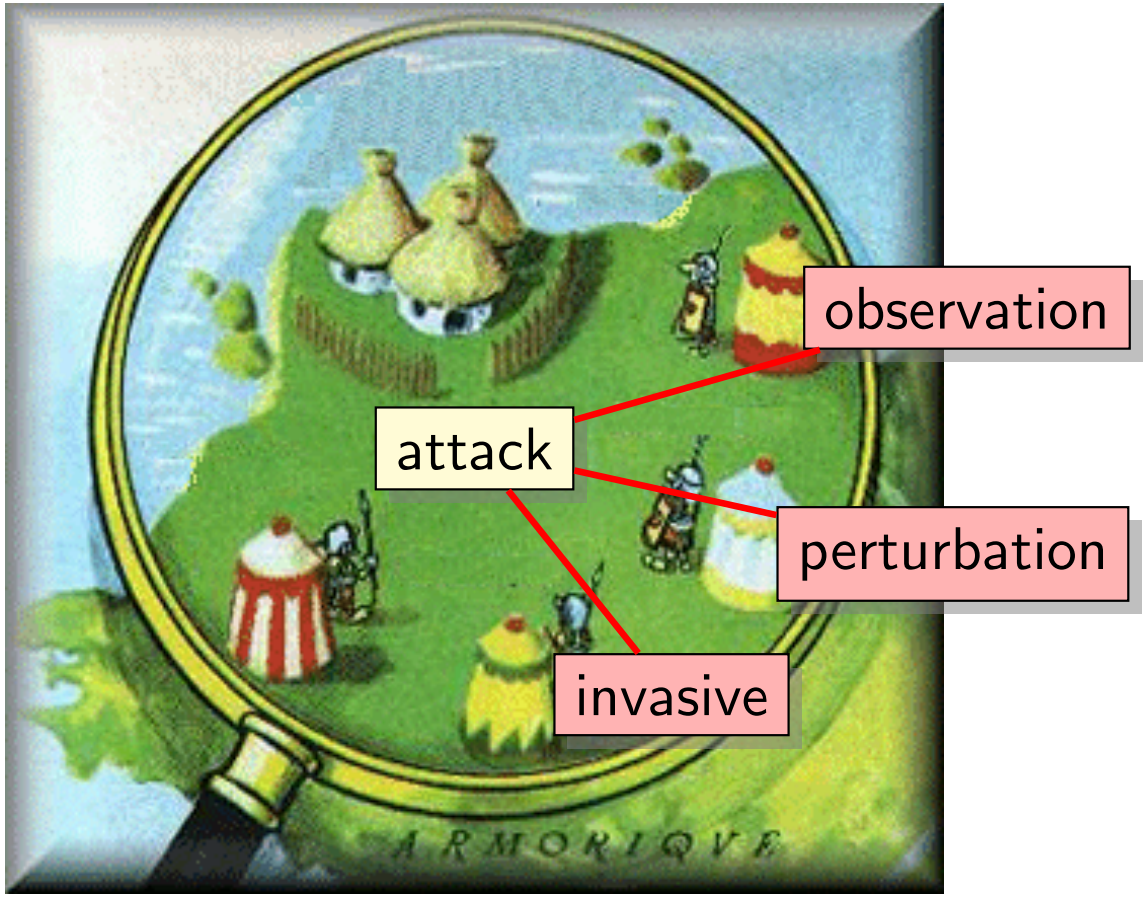
$\mathrm{GF}(p)$ or $\mathrm{GF}(2^m)$ operations

operation modulo large prime ($\mathrm{GF}(p)$) or irreducible polynomial ($\mathrm{GF}(2^m)$)

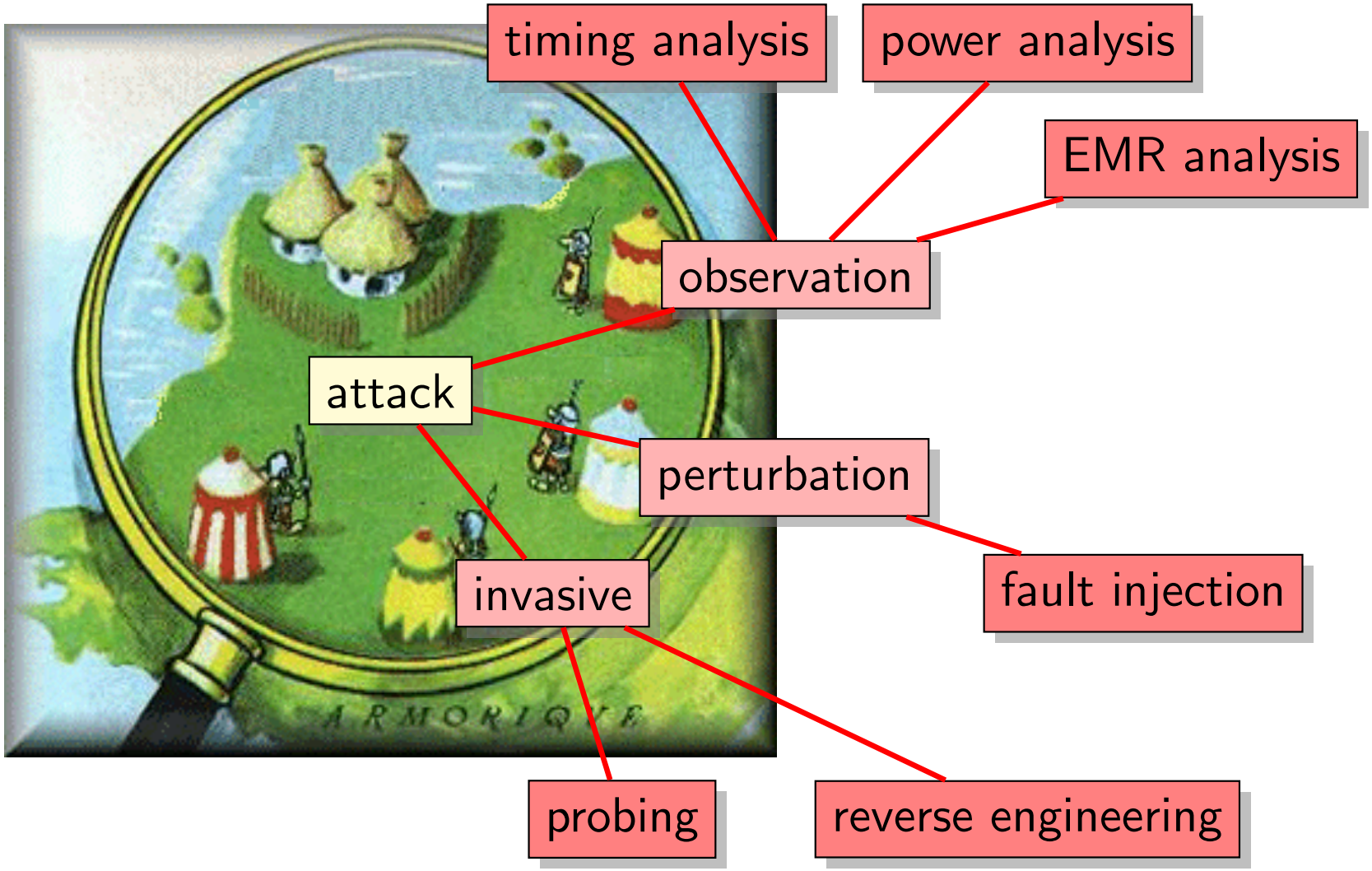
Attacks



Attacks

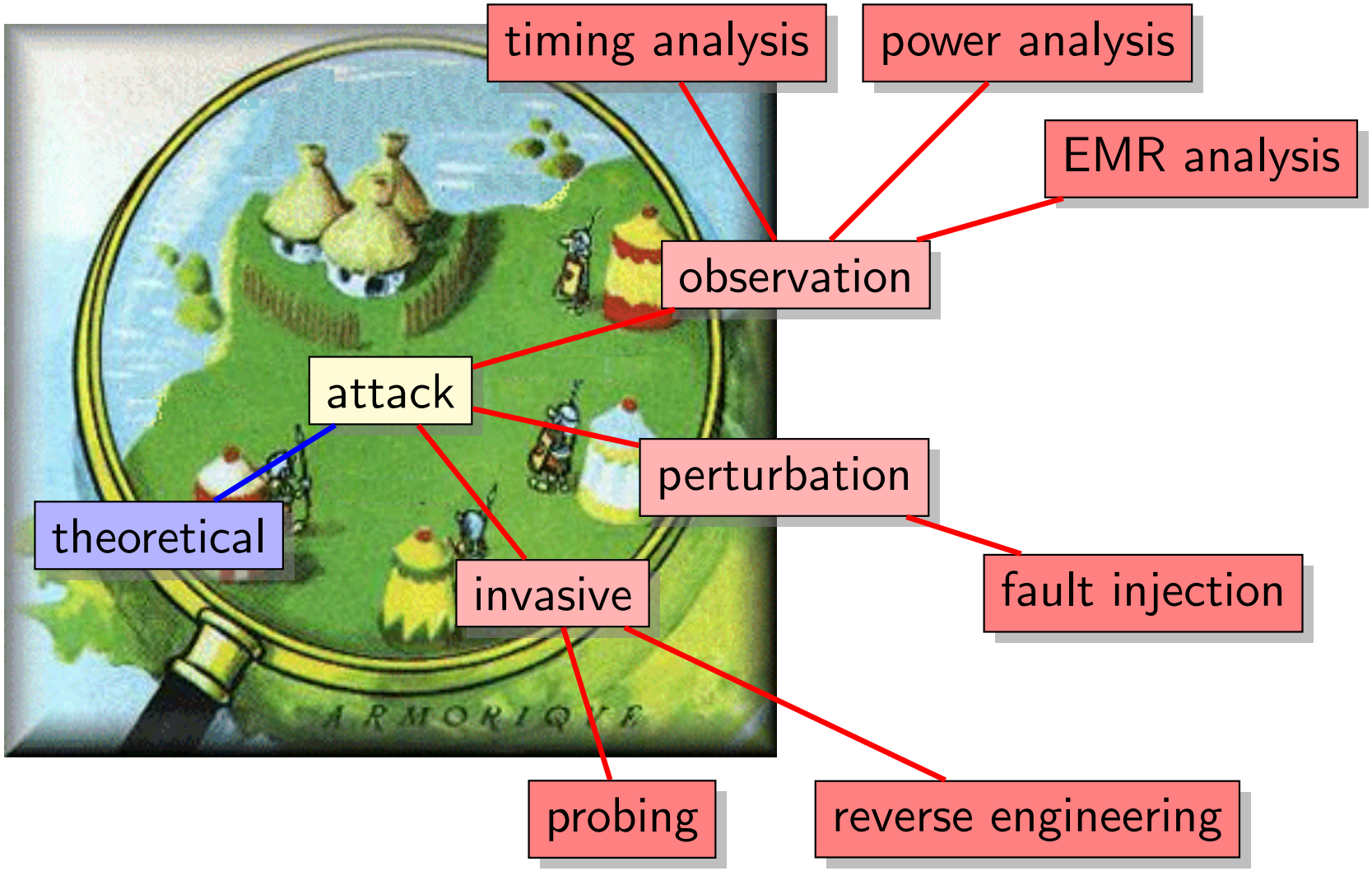


Attacks



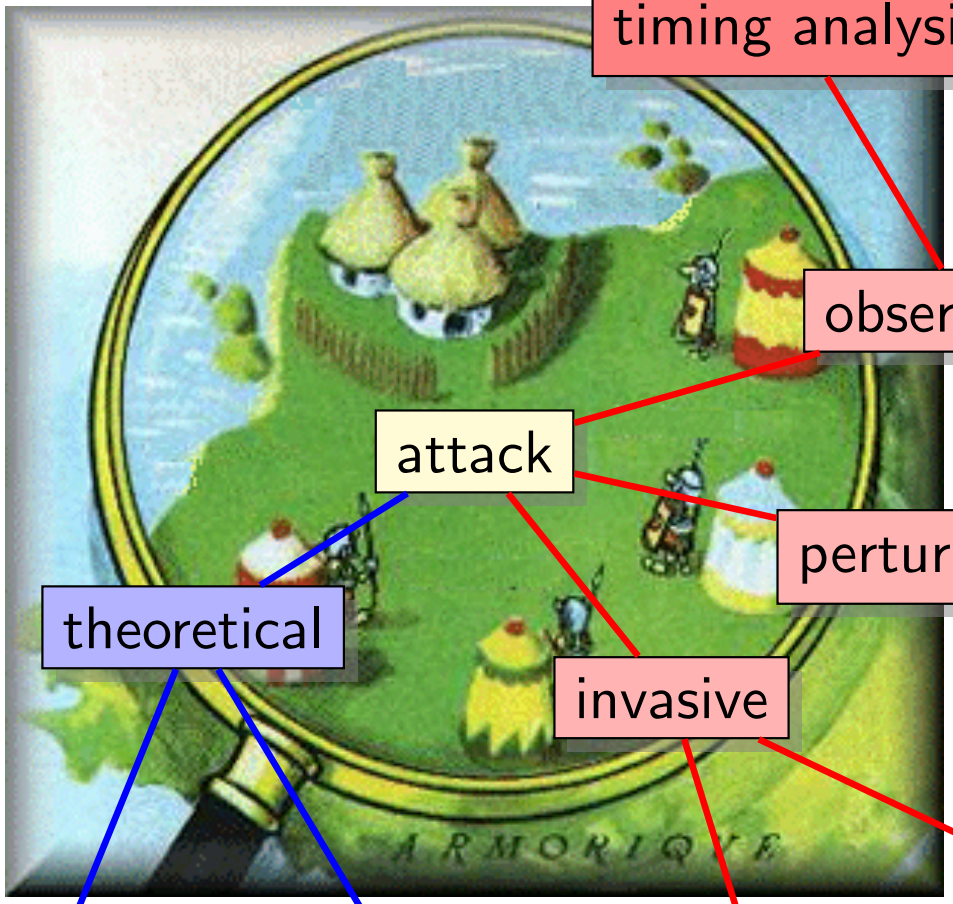
EMR = Electromagnetic radiation

Attacks



EMR = Electromagnetic radiation

Attacks



timing analysis

power analysis

EMR analysis

observation

attack

perturbation

theoretical

fault injection

invasive

advanced algorithms

probing

reverse engineering

optimized programming

EMR = Electromagnetic radiation

Side Channel Attacks (SCAs) (1/2)

Attack: attempt to find, without any knowledge about the secret:

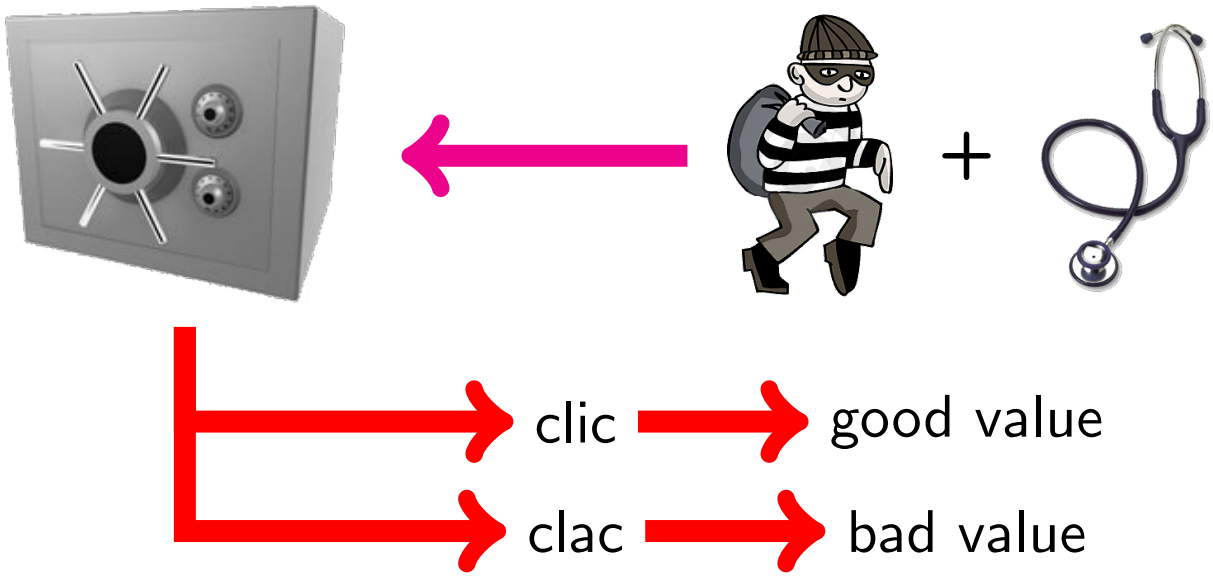
- the message (or parts of the message)
- informations on the message
- the secret (or parts of the secret)

Side Channel Attacks (SCAs) (1/2)

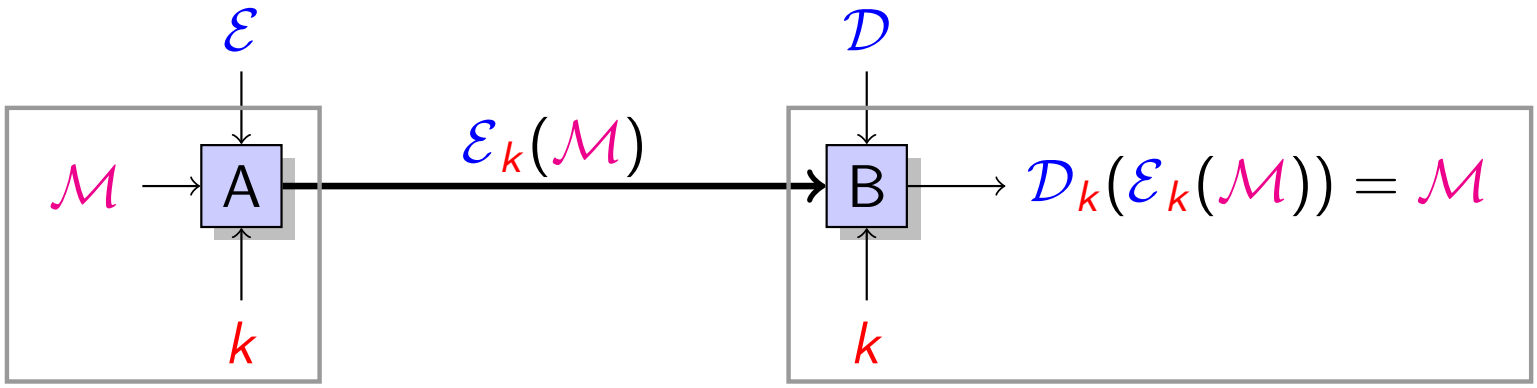
Attack: attempt to find, without any knowledge about the secret:

- the message (or parts of the message)
- informations on the message
- the secret (or parts of the secret)

"Old style" side channel attacks:

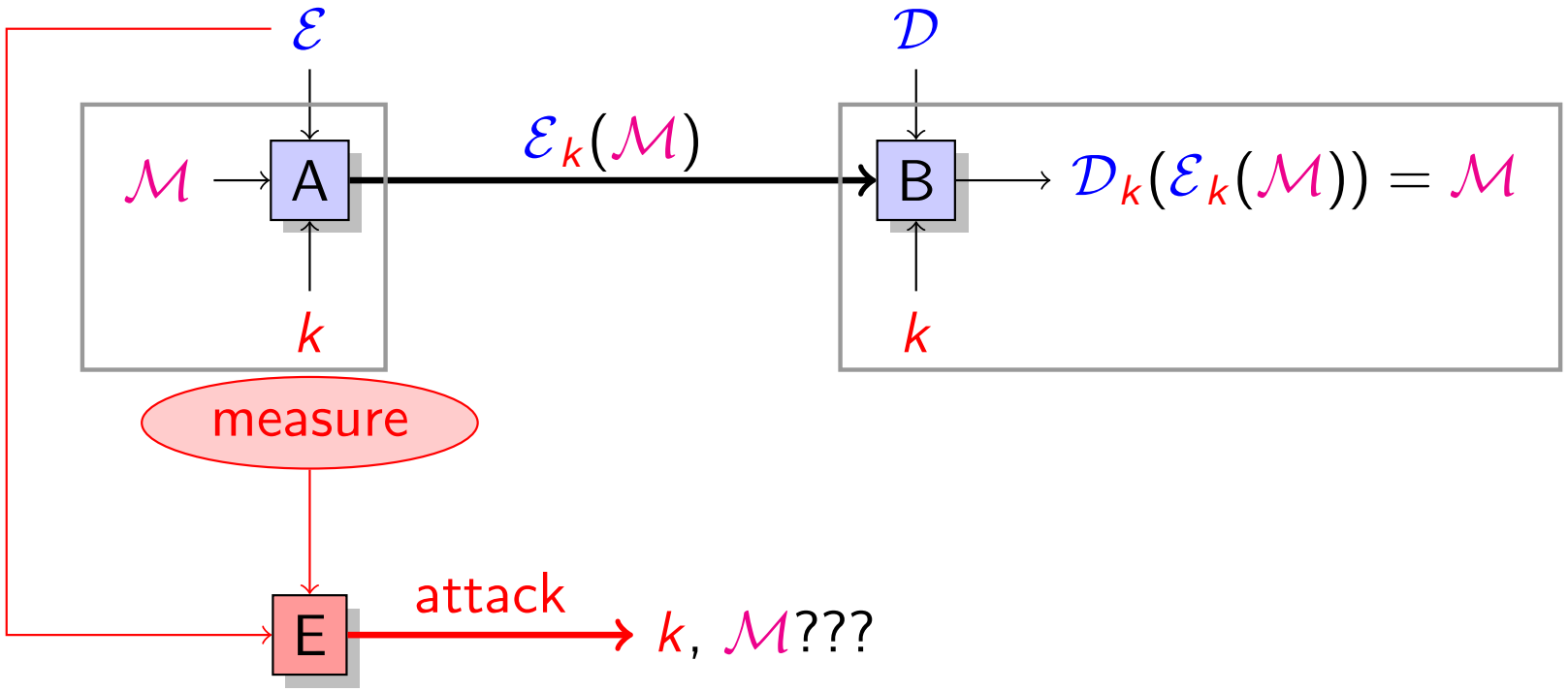


Side Channel Attacks (SCAs) (2/2)



General principle: measure external parameter(s) on running device in order to deduce internal informations

Side Channel Attacks (SCAs) (2/2)



General principle: measure external parameter(s) on running device in order to deduce internal informations

What Should be Measured?

Answer: everything that can "enter" and/or "get out" in/from the device

- power consumption
- electromagnetic radiation
- temperature
- sound
- computation time
- number of cache misses
- number and type of error messages
- ...

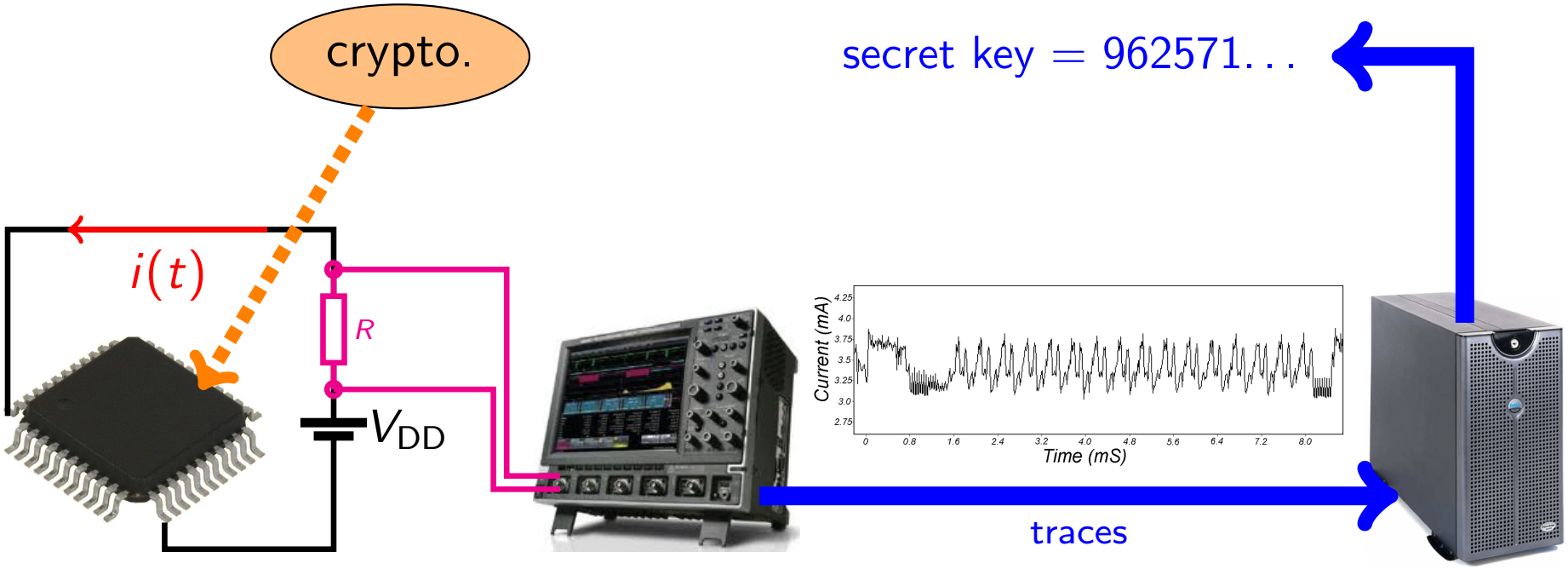
The measured parameters may provide informations on:

- global behavior (temperature, power, sound...)
- local behavior (EMR, # cache misses...)

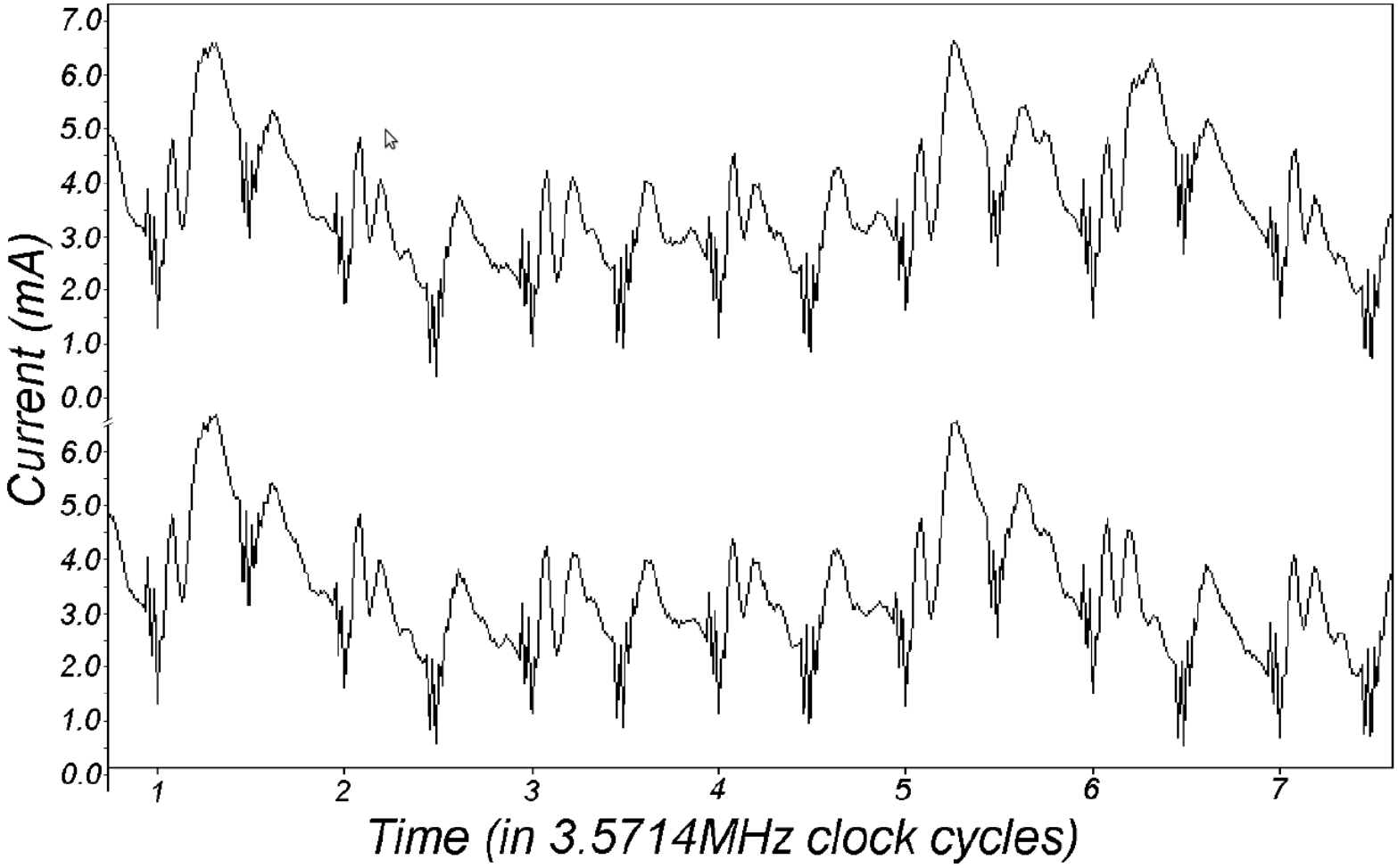
Power Consumption Analysis

General principle:

1. measure the current $i(t)$ in the cryptosystem
2. use those measurements to "deduce" secret informations

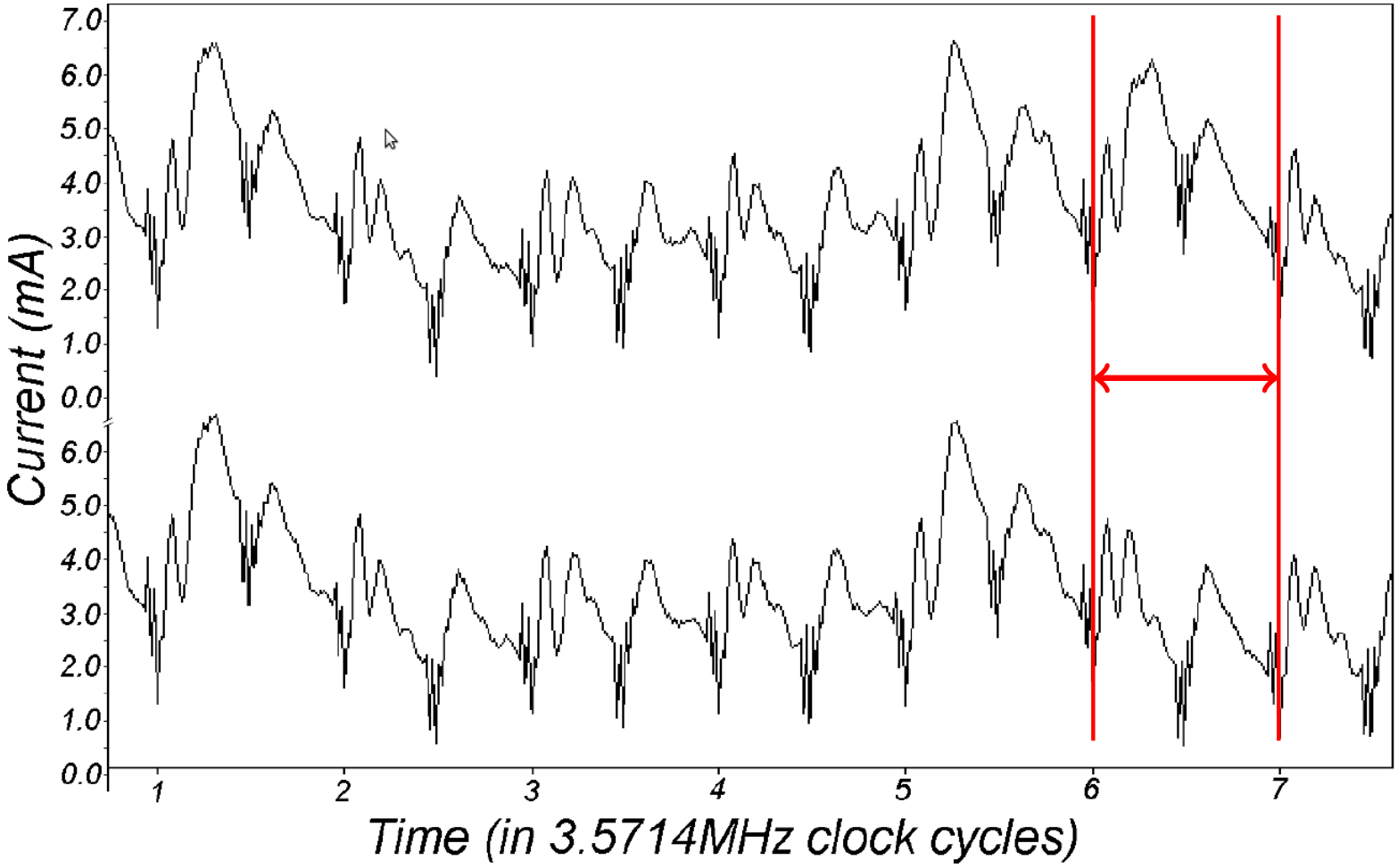


Simple Power Analysis (SPA)



Source: [11]

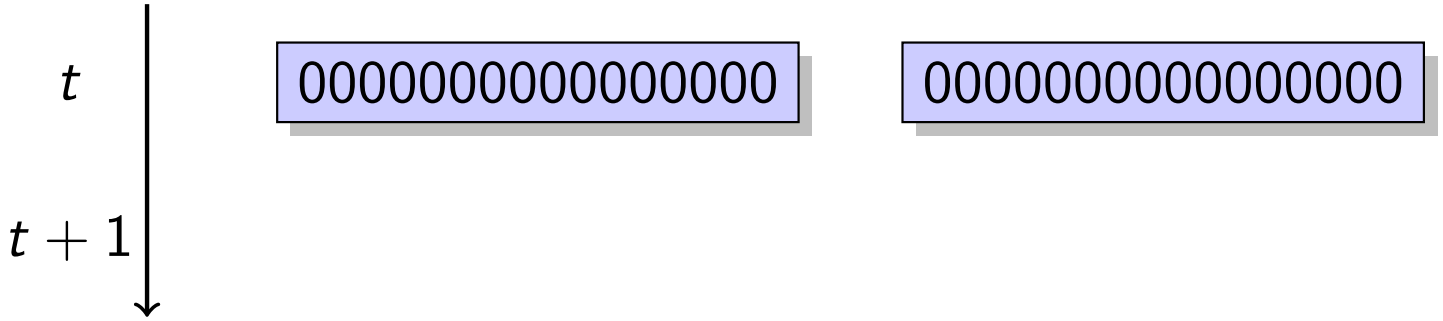
Simple Power Analysis (SPA)



Source: [11]

Limits of the SPA

Example of behavior difference: (activity into a register)



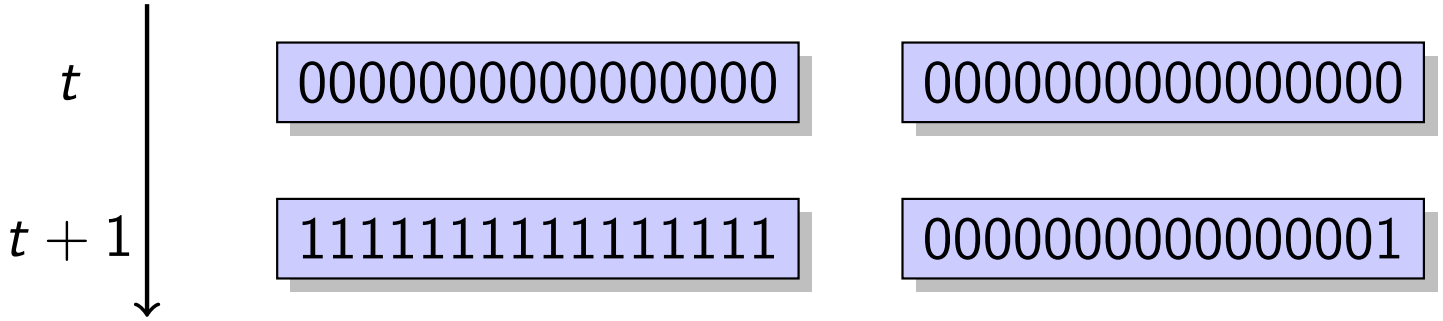
Limits of the SPA

Example of behavior difference: (activity into a register)

t	0000000000000000	0000000000000000
$t+1$	1111111111111111	0000000000000001

Limits of the SPA

Example of behavior difference: (activity into a register)



Important: a small difference may be evaluated has a noise during the measurement → traces cannot be distinguished

Question: what can be done when differences are too small?

Limits of the SPA

Example of behavior difference: (activity into a register)

t	0000000000000000	0000000000000000
$t+1$	1111111111111111	0000000000000001

Important: a small difference may be evaluated has a noise during the measurement → traces cannot be distinguished

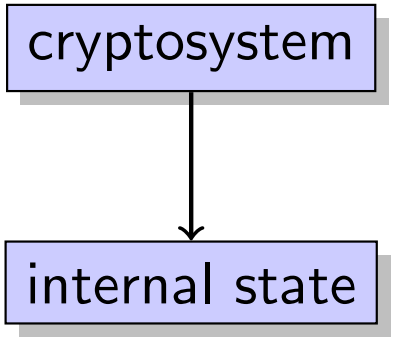
Question: what can be done when differences are too small?

Answer: use statistics over several traces

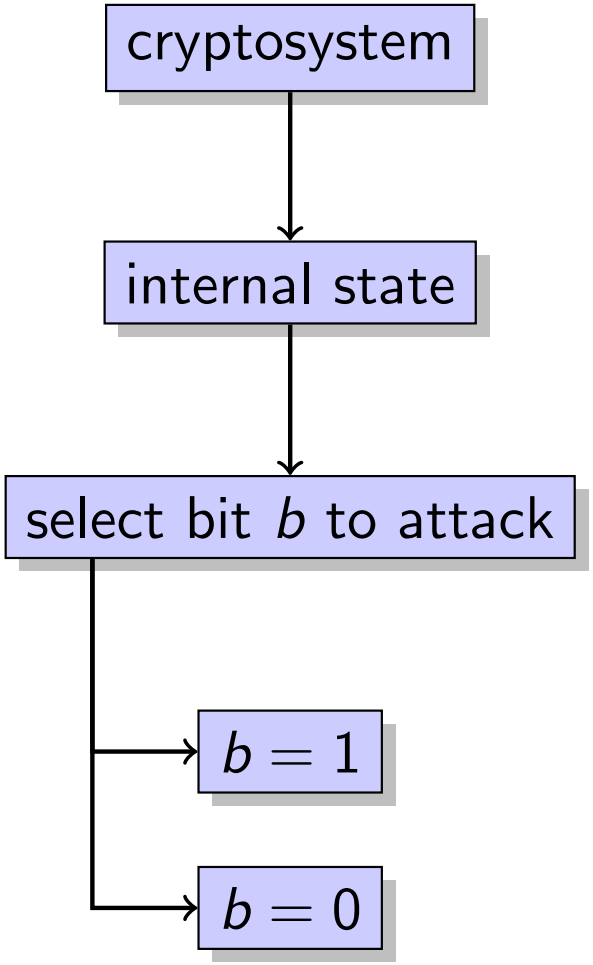
Differential Power Analysis (DPA)

cryptosystem

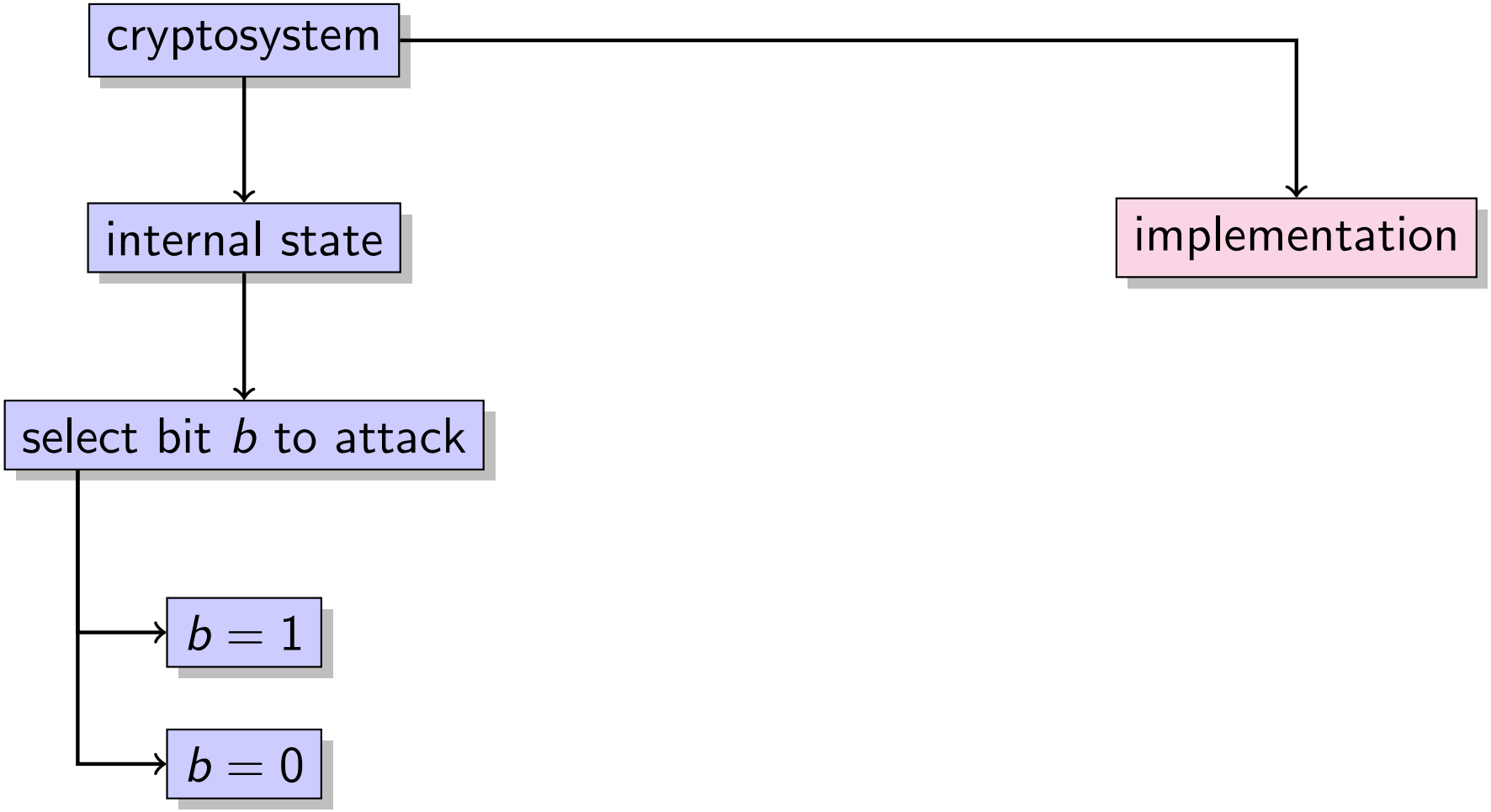
Differential Power Analysis (DPA)



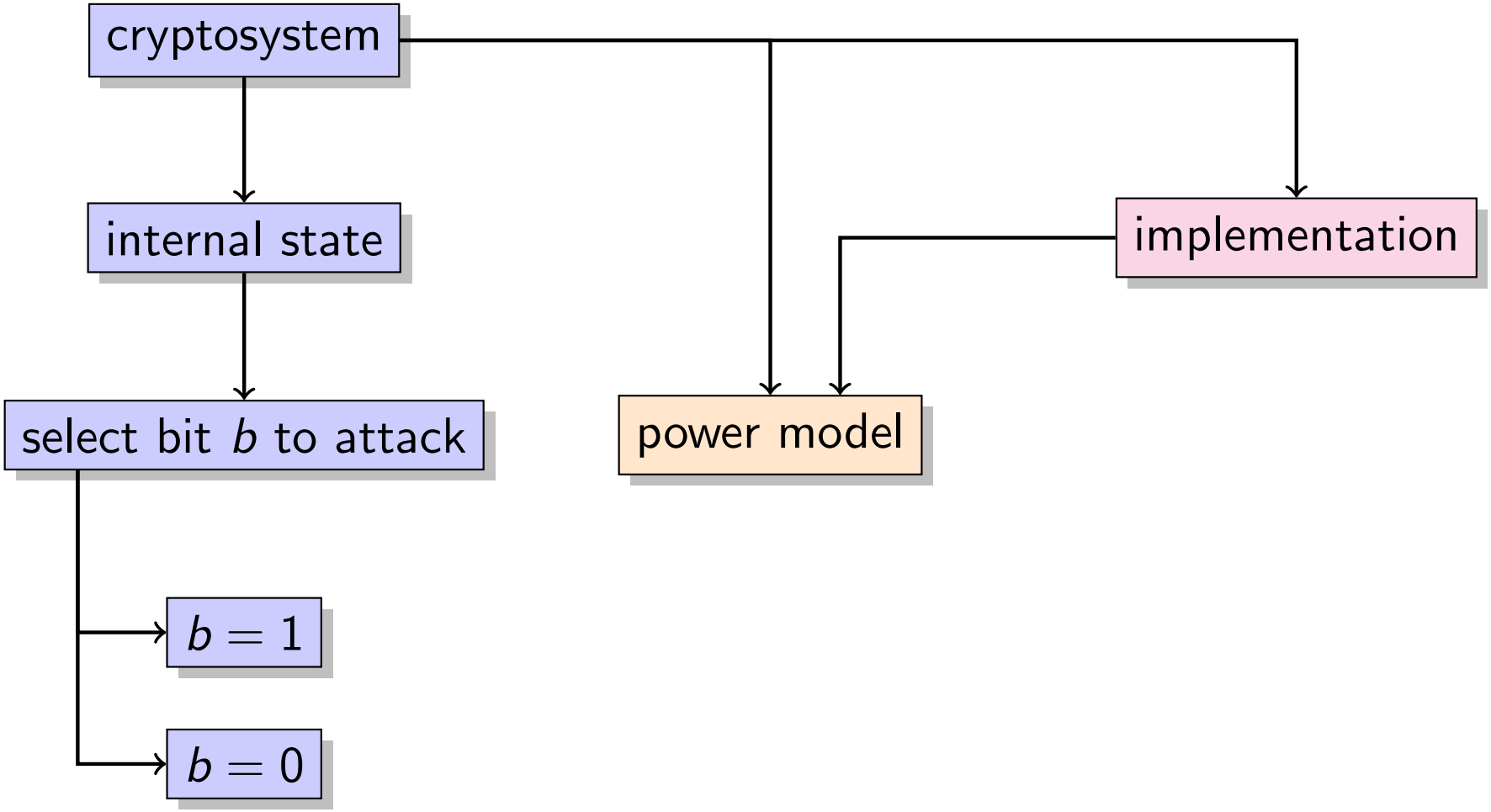
Differential Power Analysis (DPA)



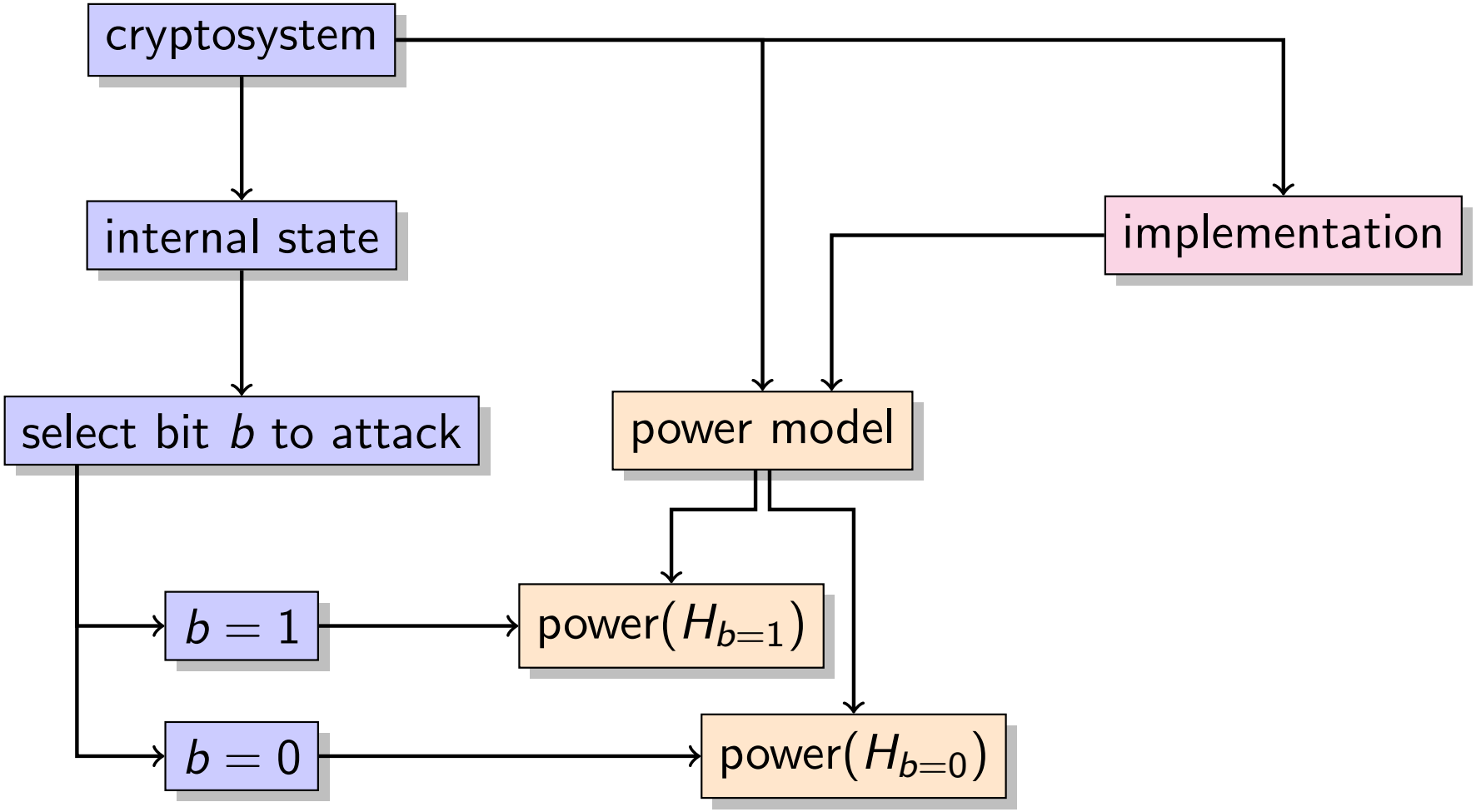
Differential Power Analysis (DPA)



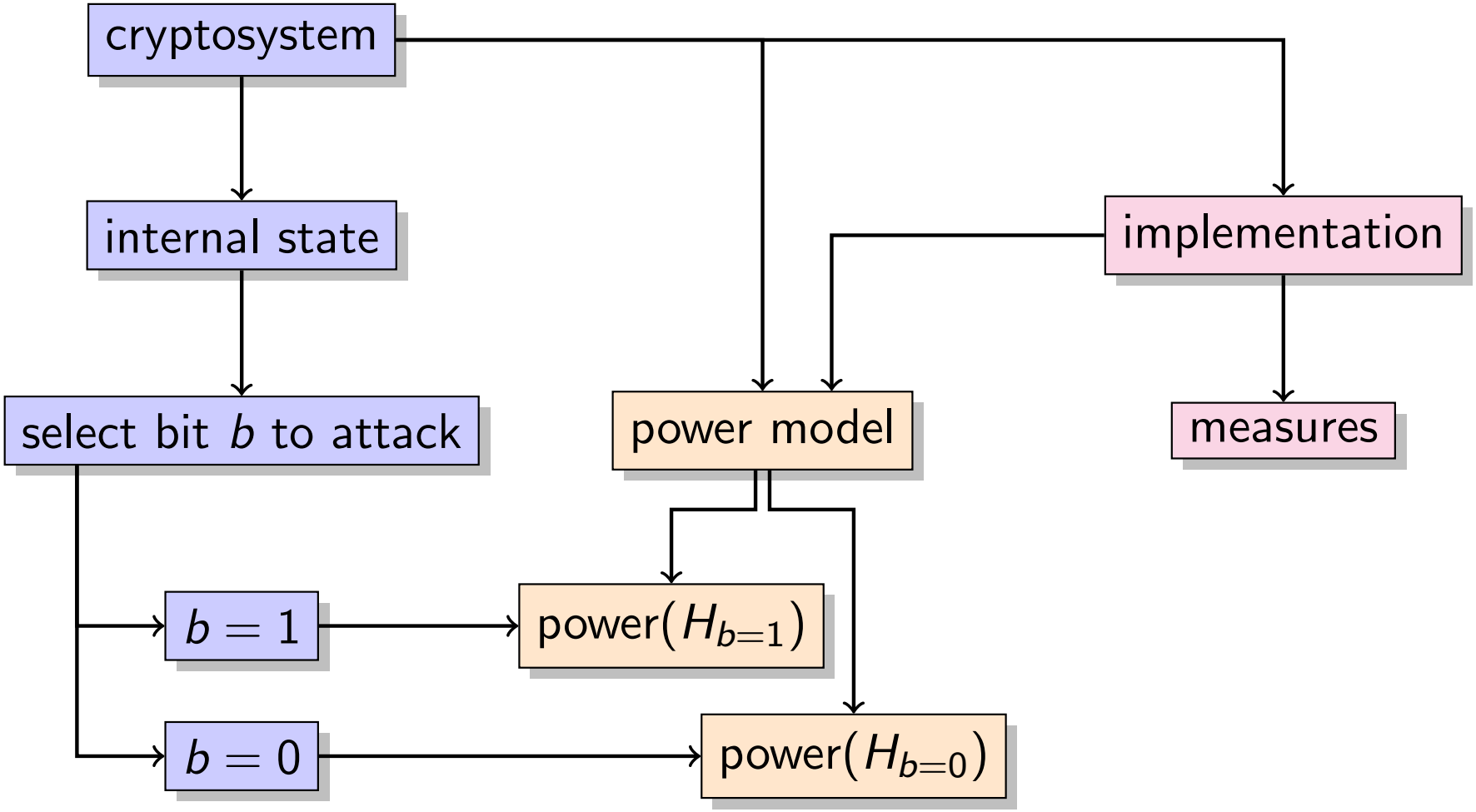
Differential Power Analysis (DPA)



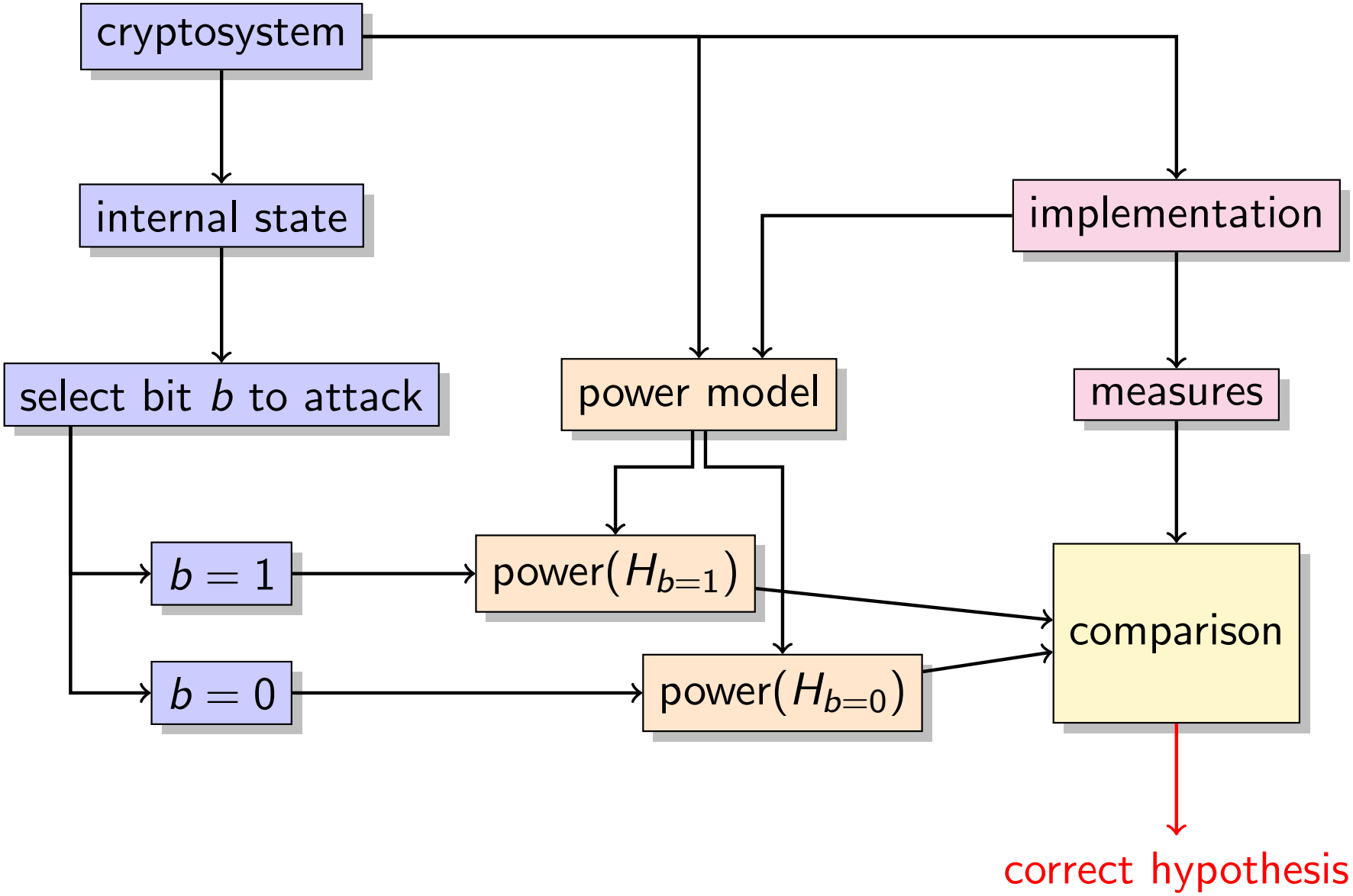
Differential Power Analysis (DPA)



Differential Power Analysis (DPA)



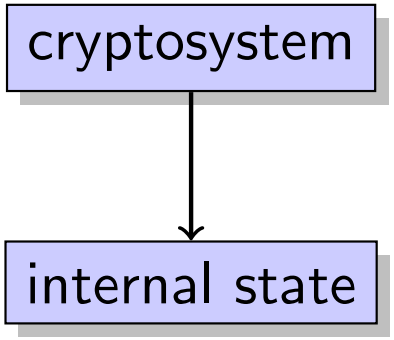
Differential Power Analysis (DPA)



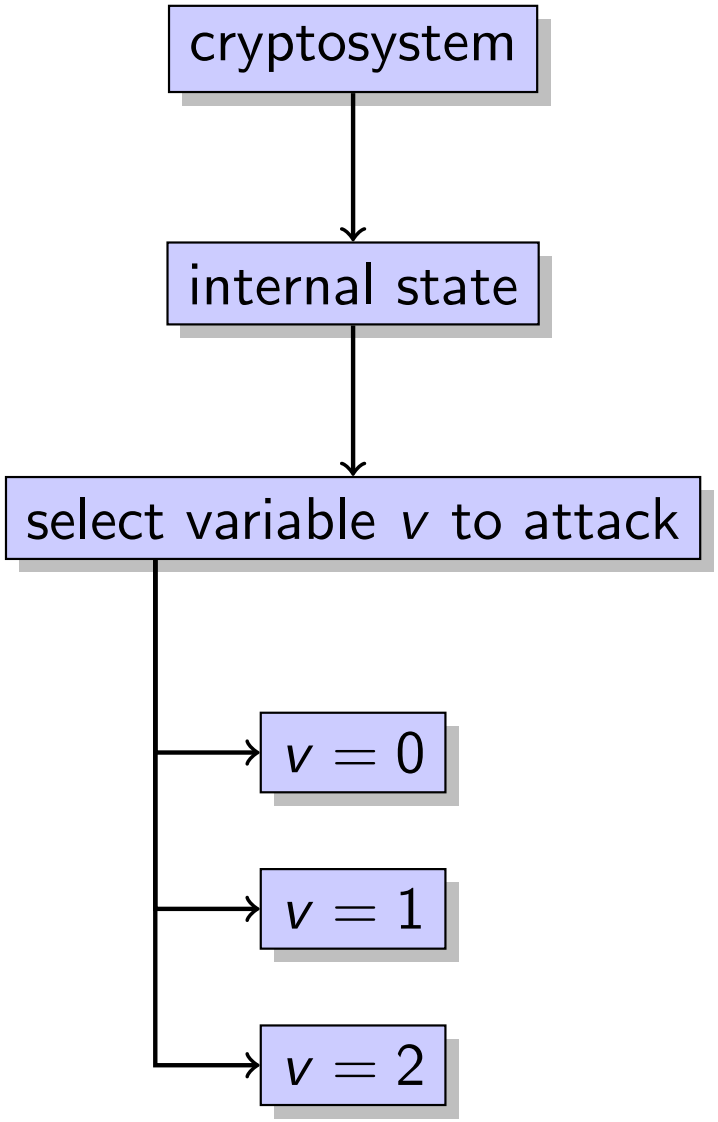
Template Attack

cryptosystem

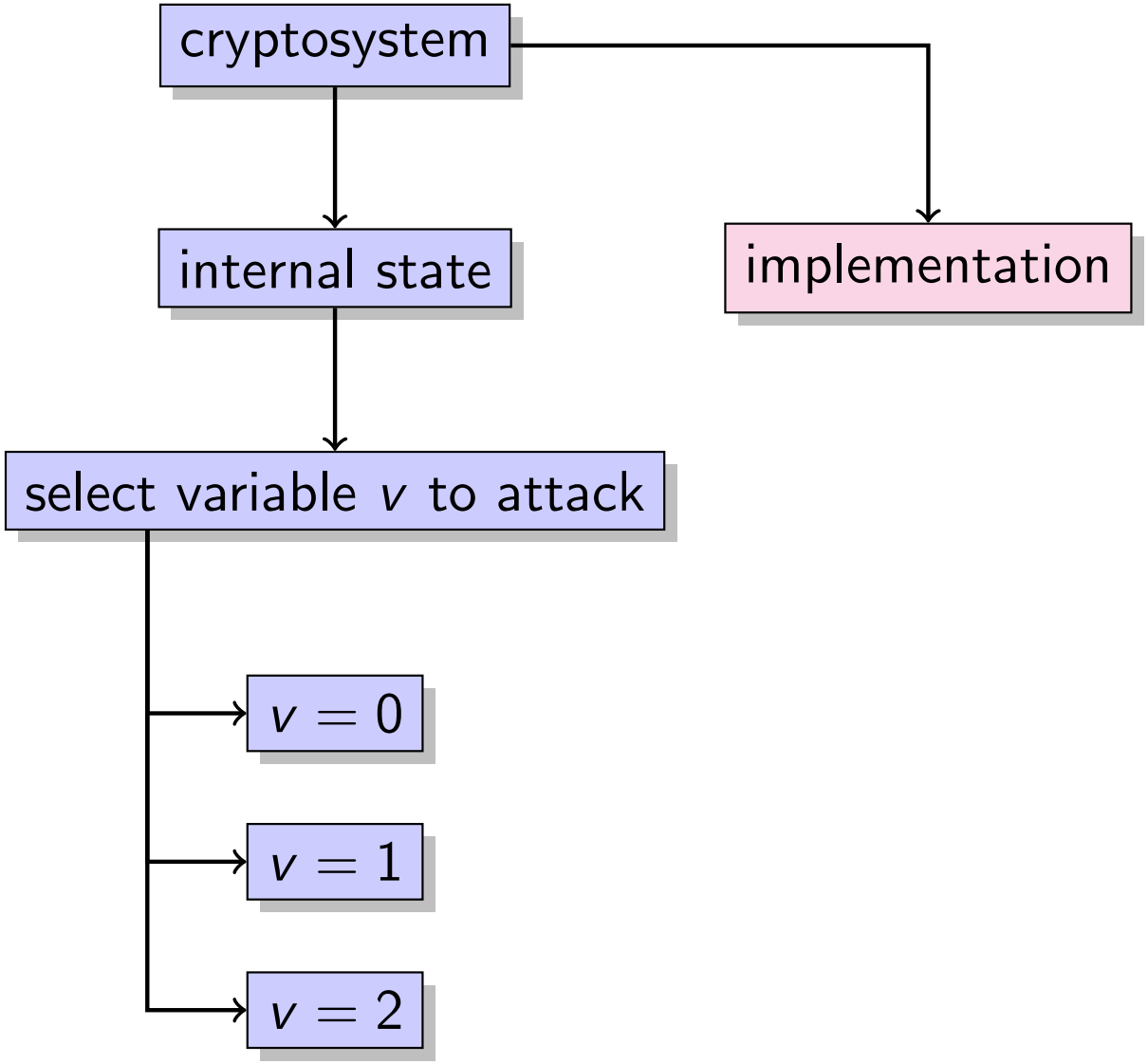
Template Attack



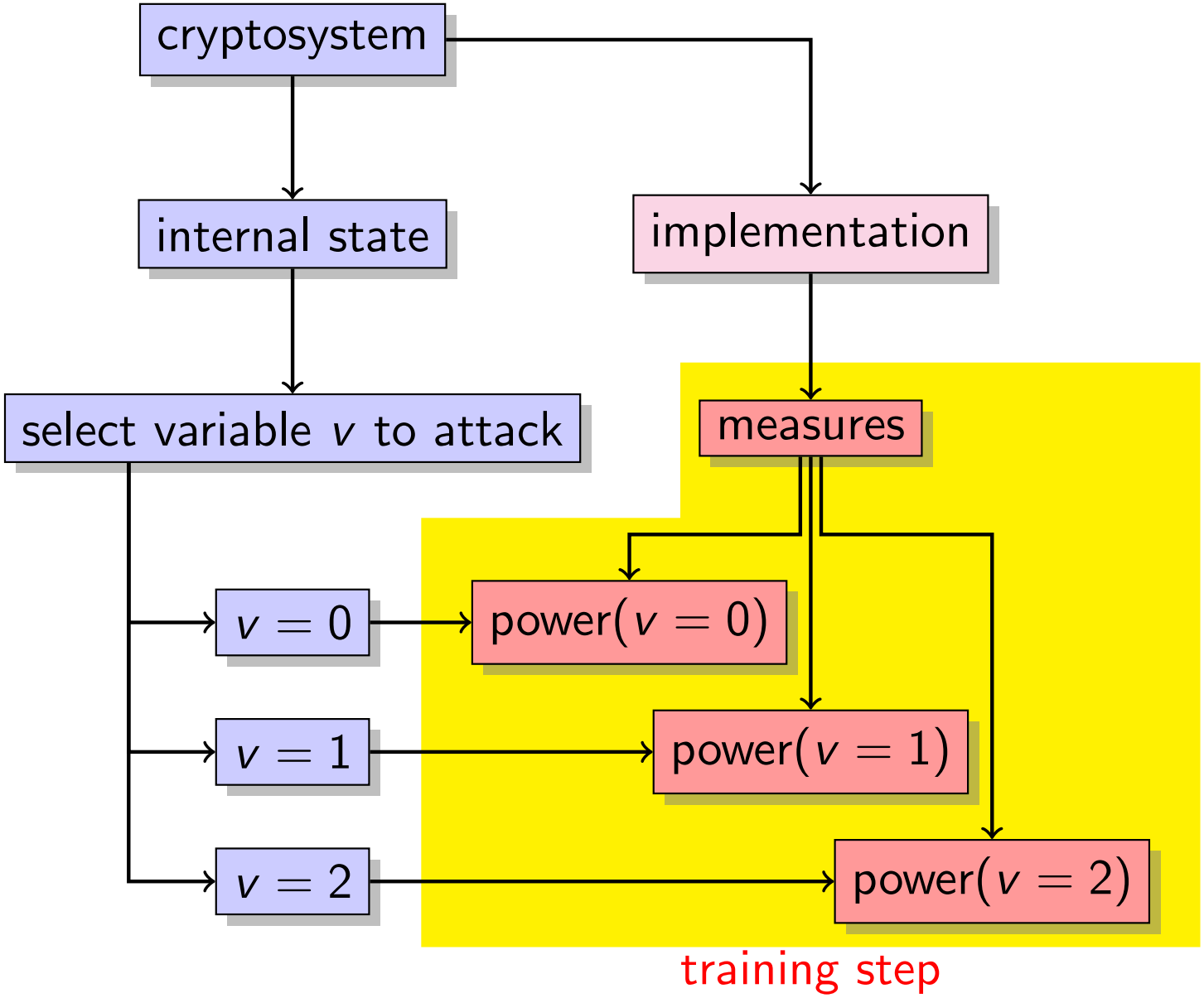
Template Attack



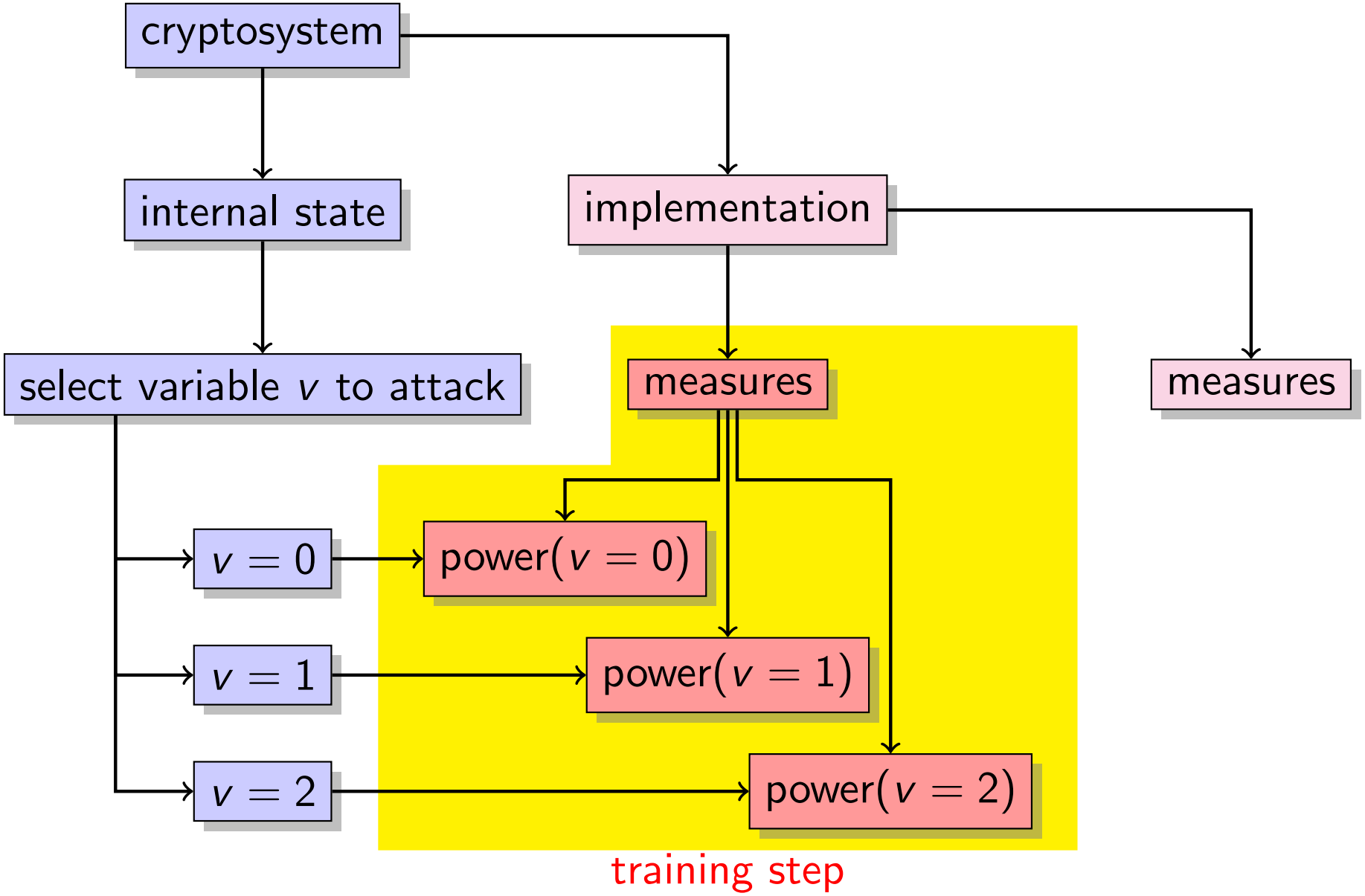
Template Attack



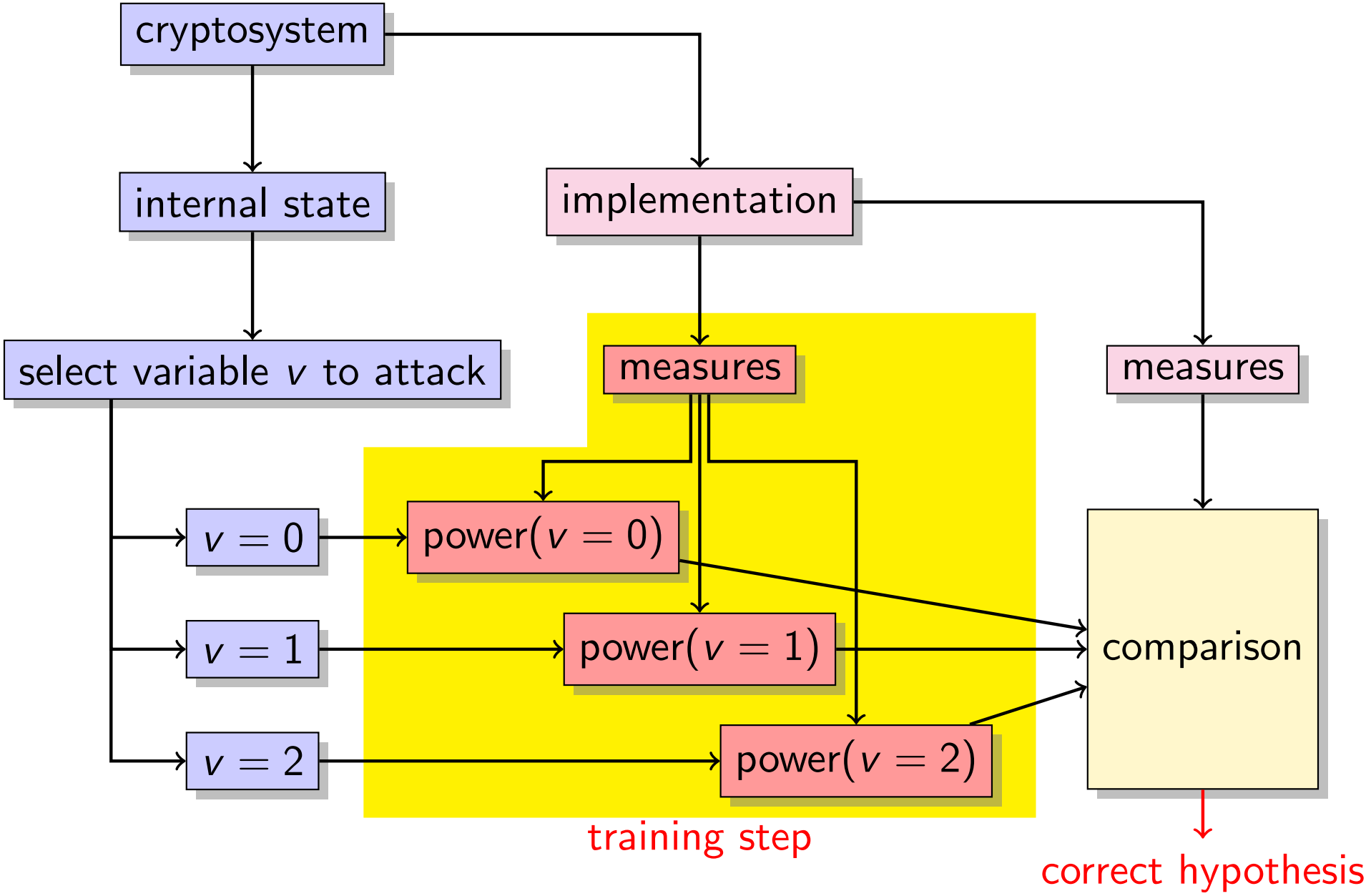
Template Attack



Template Attack

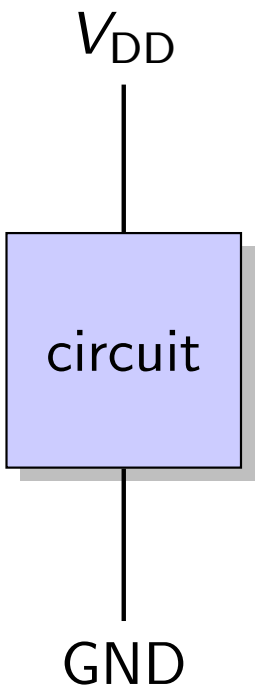


Template Attack



Electromagnetic Radiation Analysis

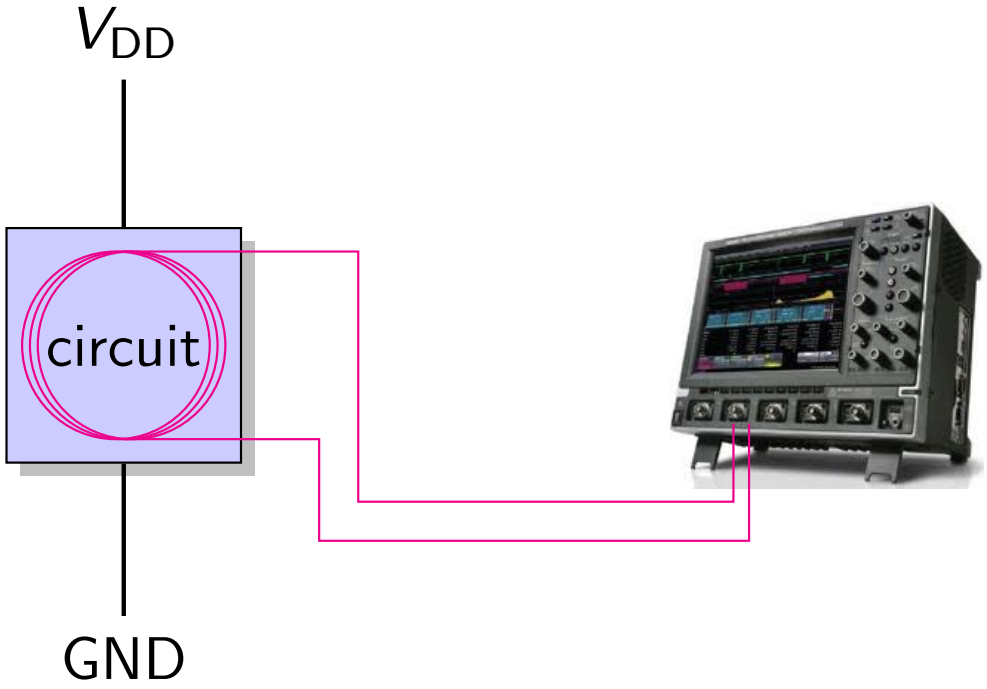
General principle: use a probe to measure the EMR



EMR measurement:

Electromagnetic Radiation Analysis

General principle: use a probe to measure the EMR

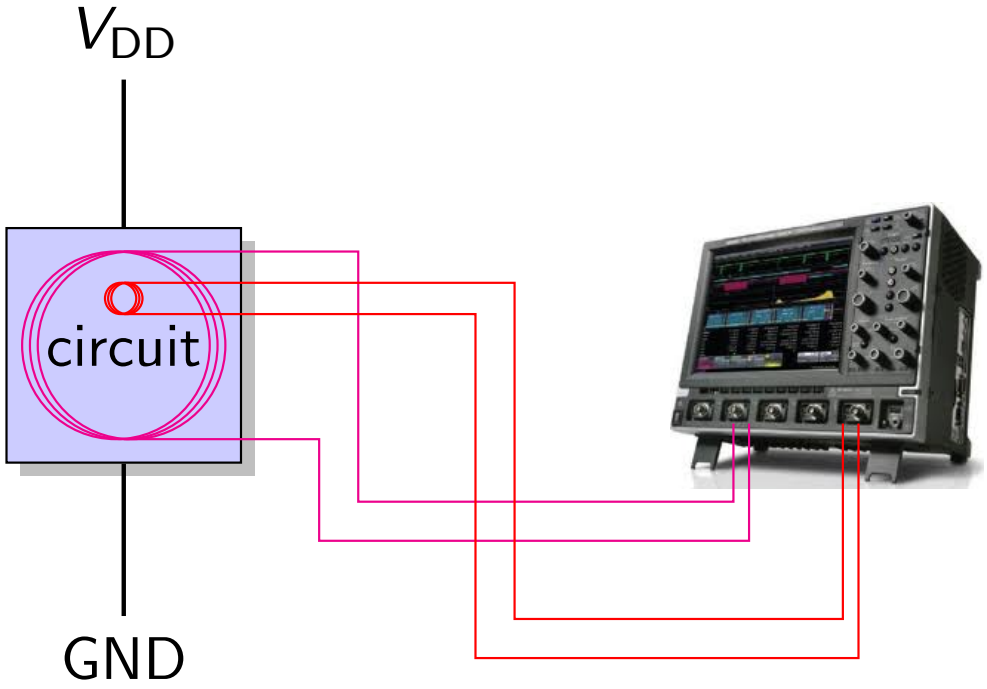


EMR measurement:

- global EMR with a large probe

Electromagnetic Radiation Analysis

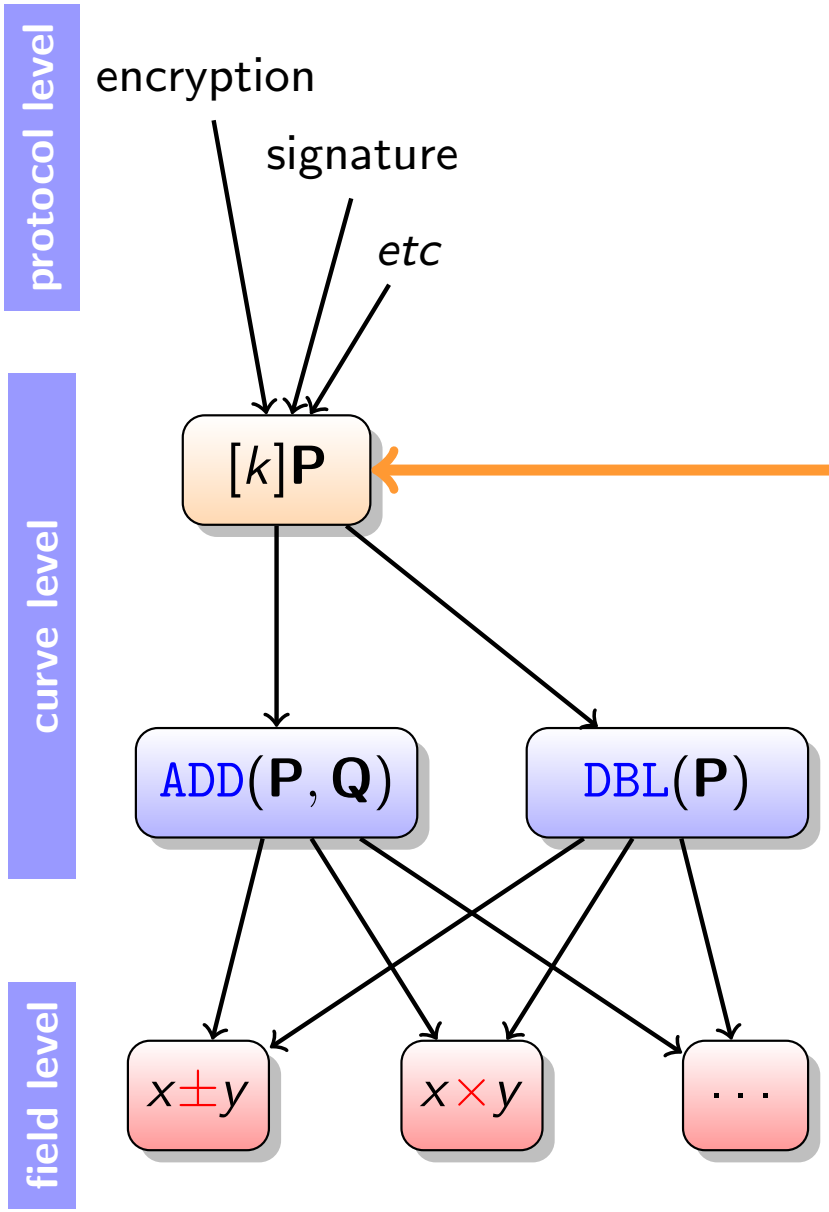
General principle: use a probe to measure the EMR



EMR measurement:

- global EMR with a large probe
- local EMR with a micro-probe

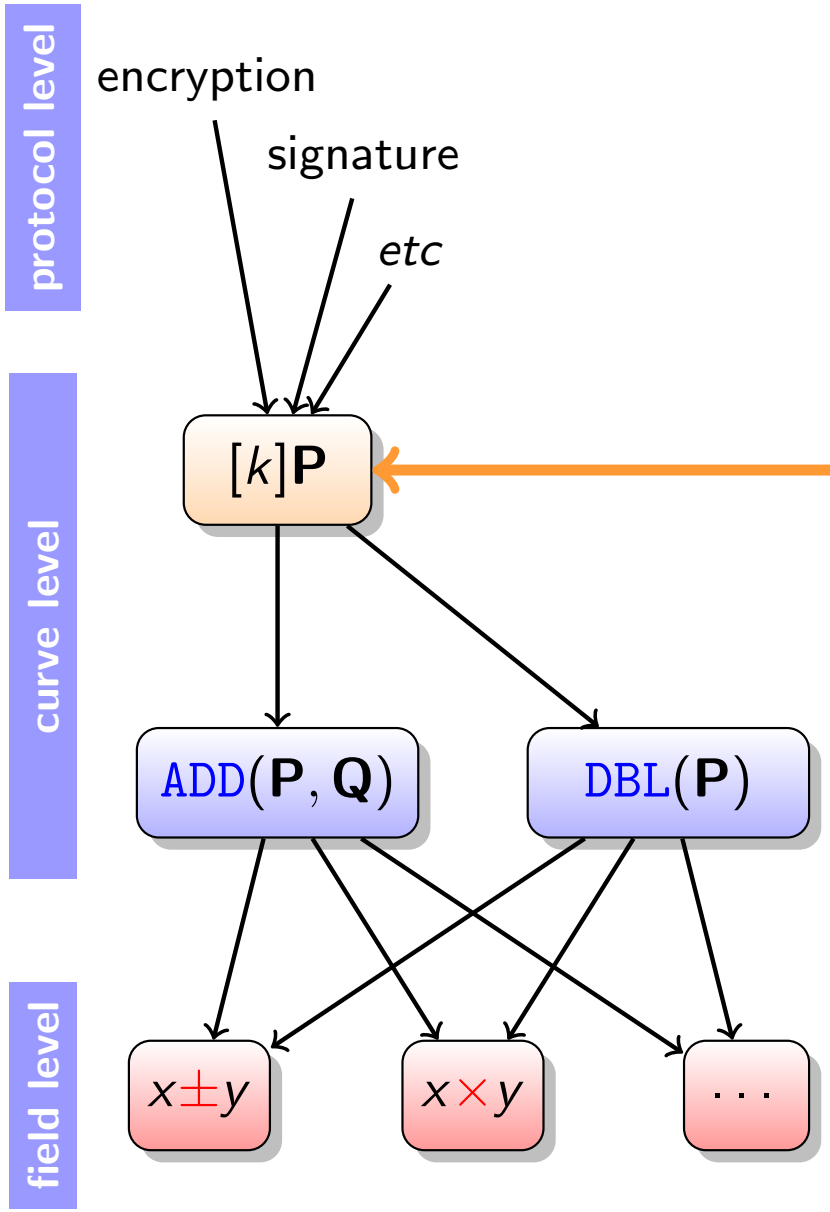
Side Channel Attack on ECC

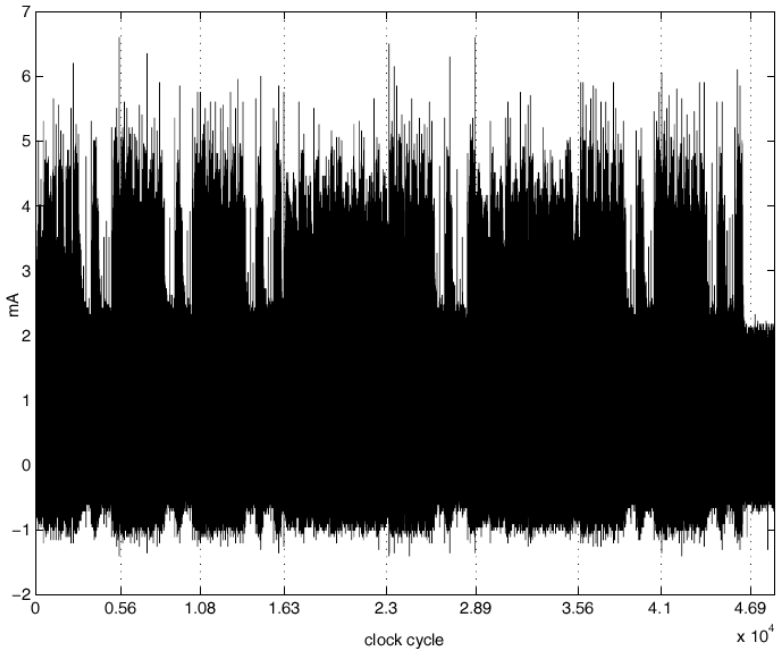


Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Side Channel Attack on ECC

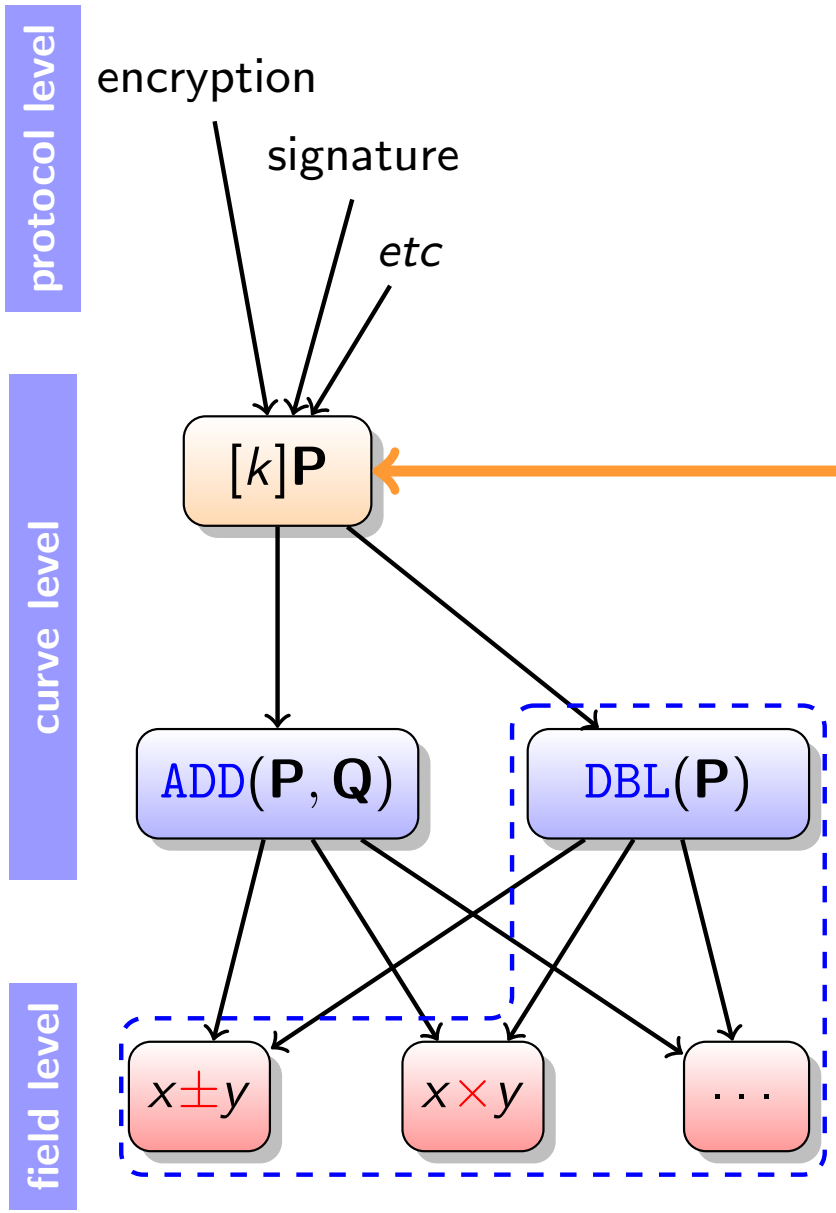


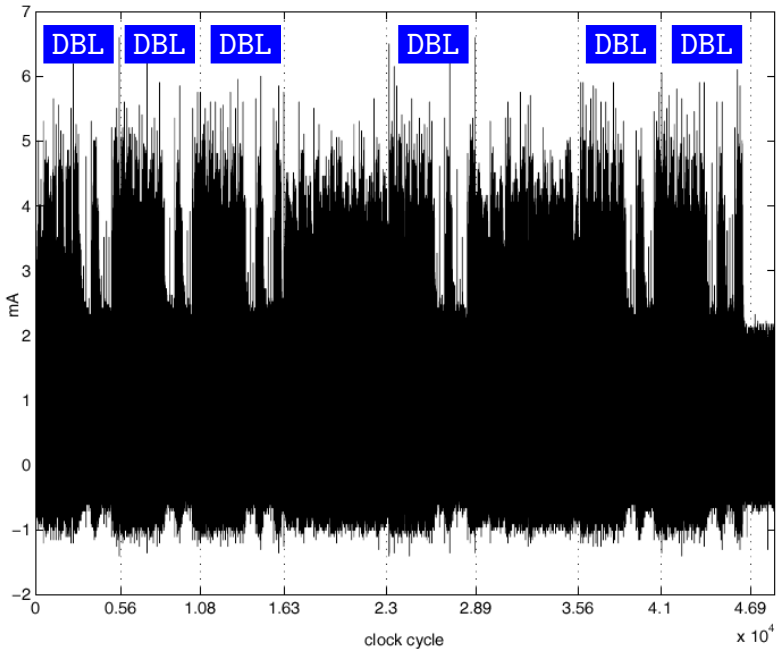


Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Side Channel Attack on ECC

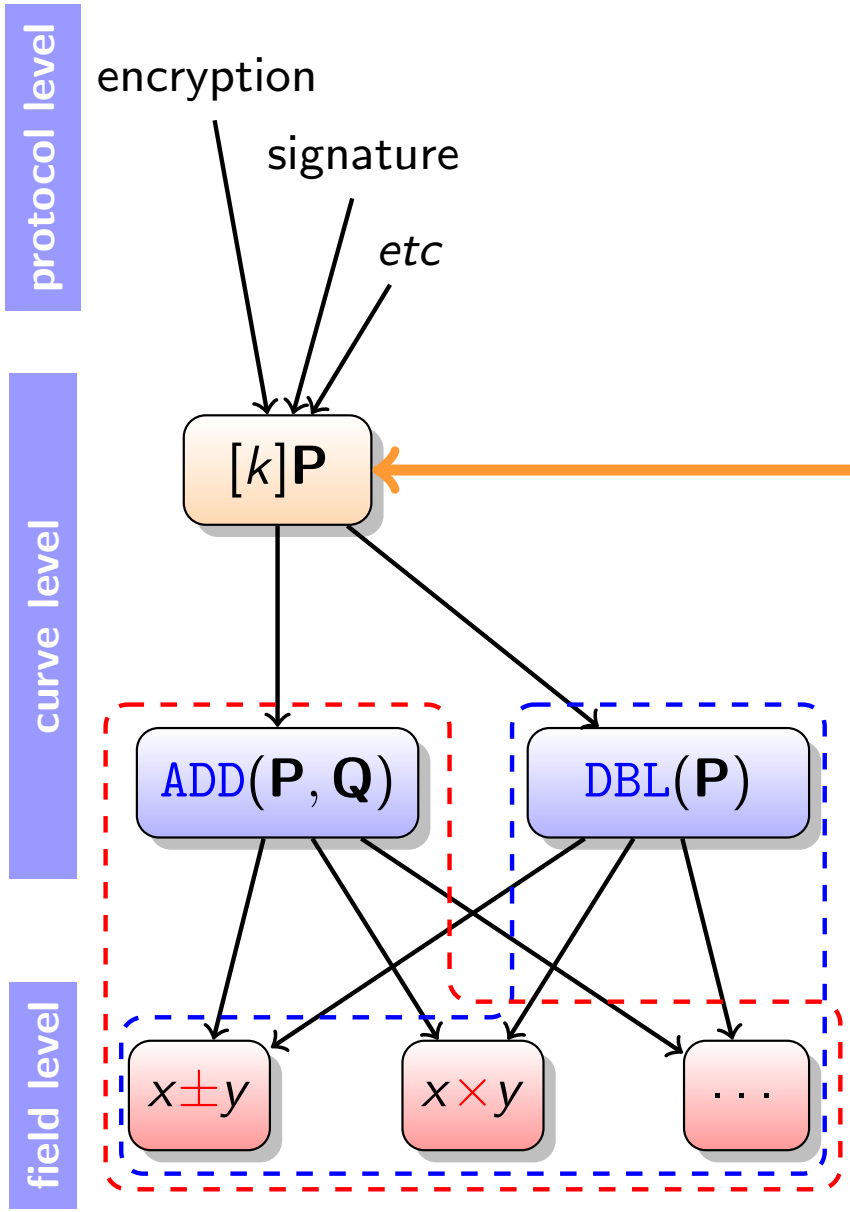


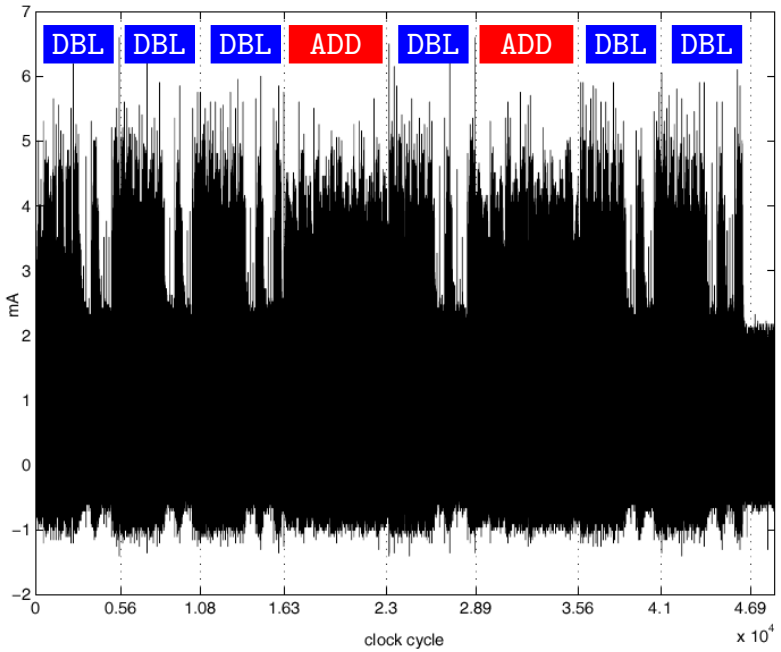


Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Side Channel Attack on ECC

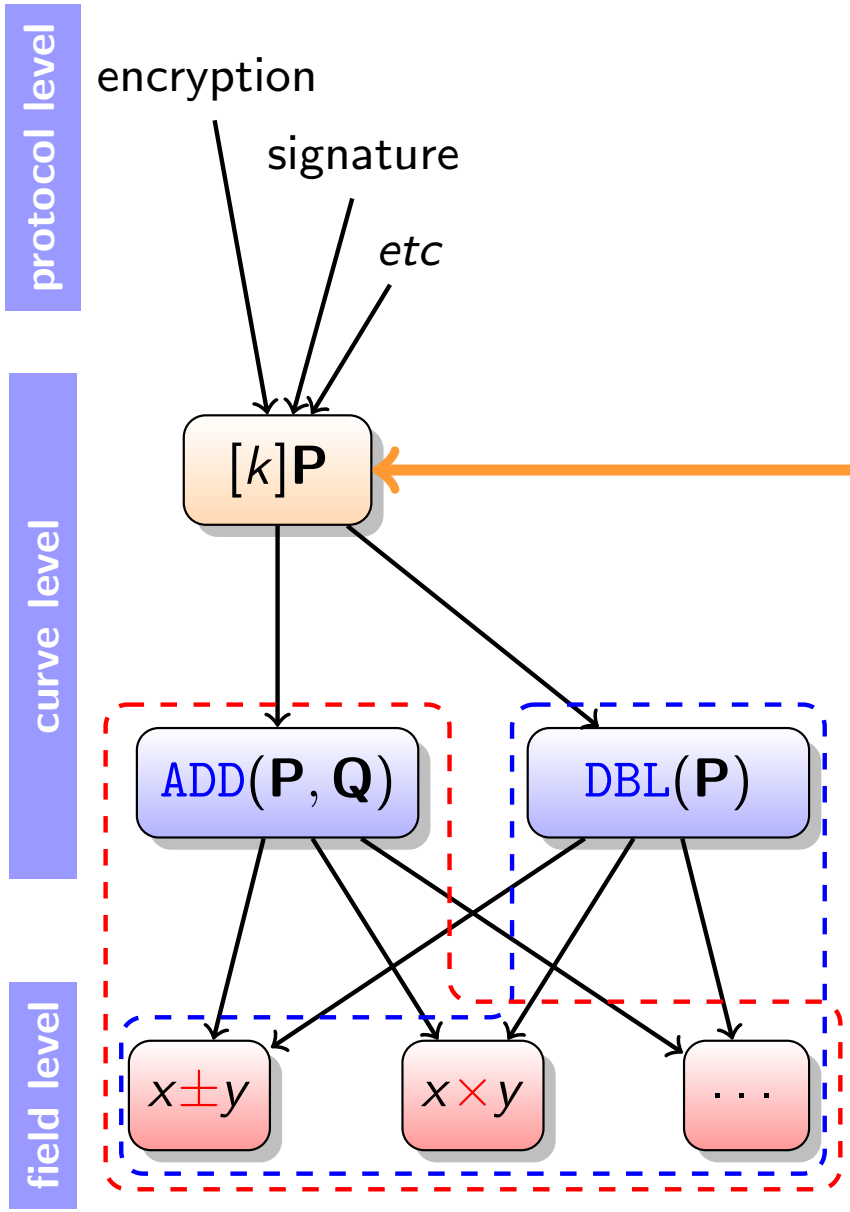


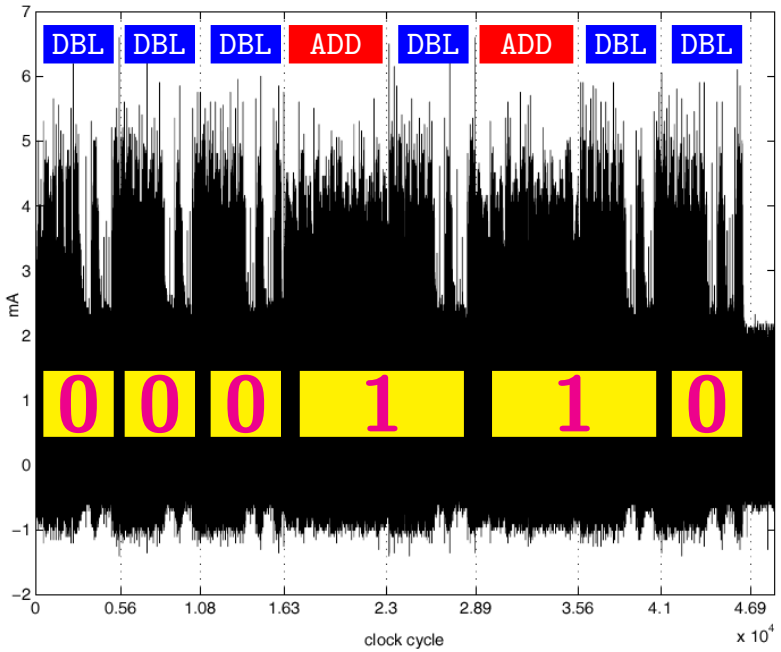


Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

Side Channel Attack on ECC



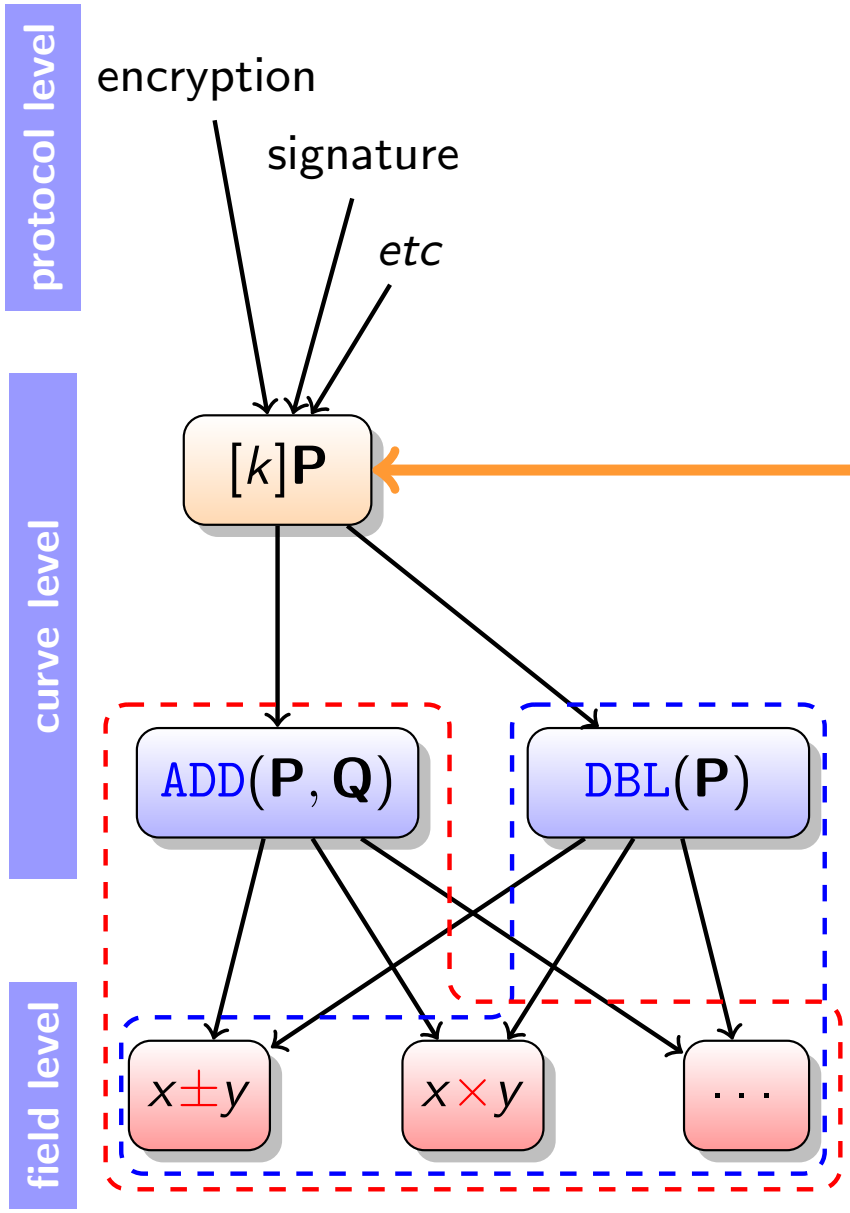


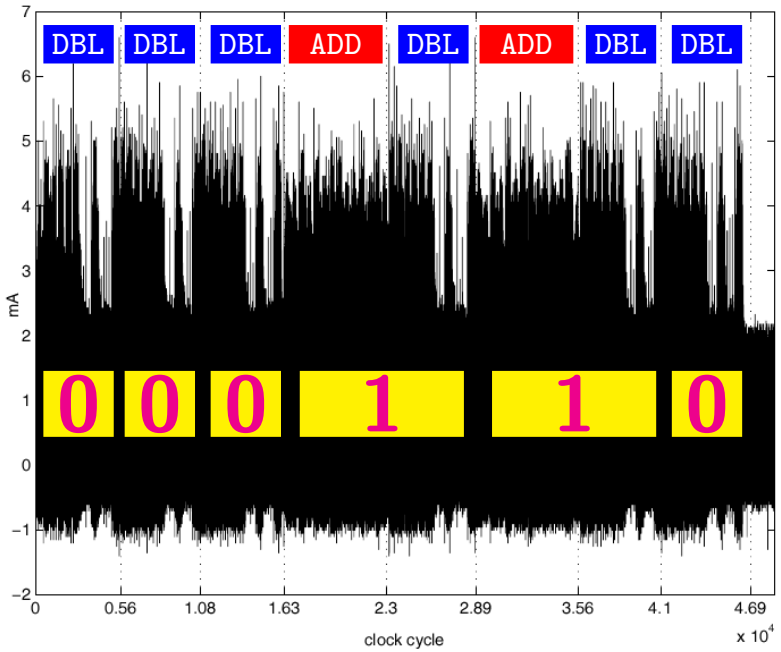
Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

- simple power analysis (& variants)

Side Channel Attack on ECC





Scalar multiplication operation

```
for i from 0 to t − 1 do
    if k_i = 1 then Q = ADD(P, Q)
    P = DBL(P)
```

- simple power analysis (& variants)
- differential power analysis (& variants)
- horizontal/vertical/templates/...attacks

Fault Injection Attacks

Objective: alter the correct functioning of a system "from outside"

Fault effects examples:

- modify a value in a register
- modify a value in the memory hierarchy
- modify an address (data location or code location)
- modify a control signal (e.g. status flag, branch direction)
- skip/modify the instruction decoding
- delay/advance propagation of internal control signals
- etc.

Also called perturbation attacks

Fault Injection Techniques

Typical techniques:

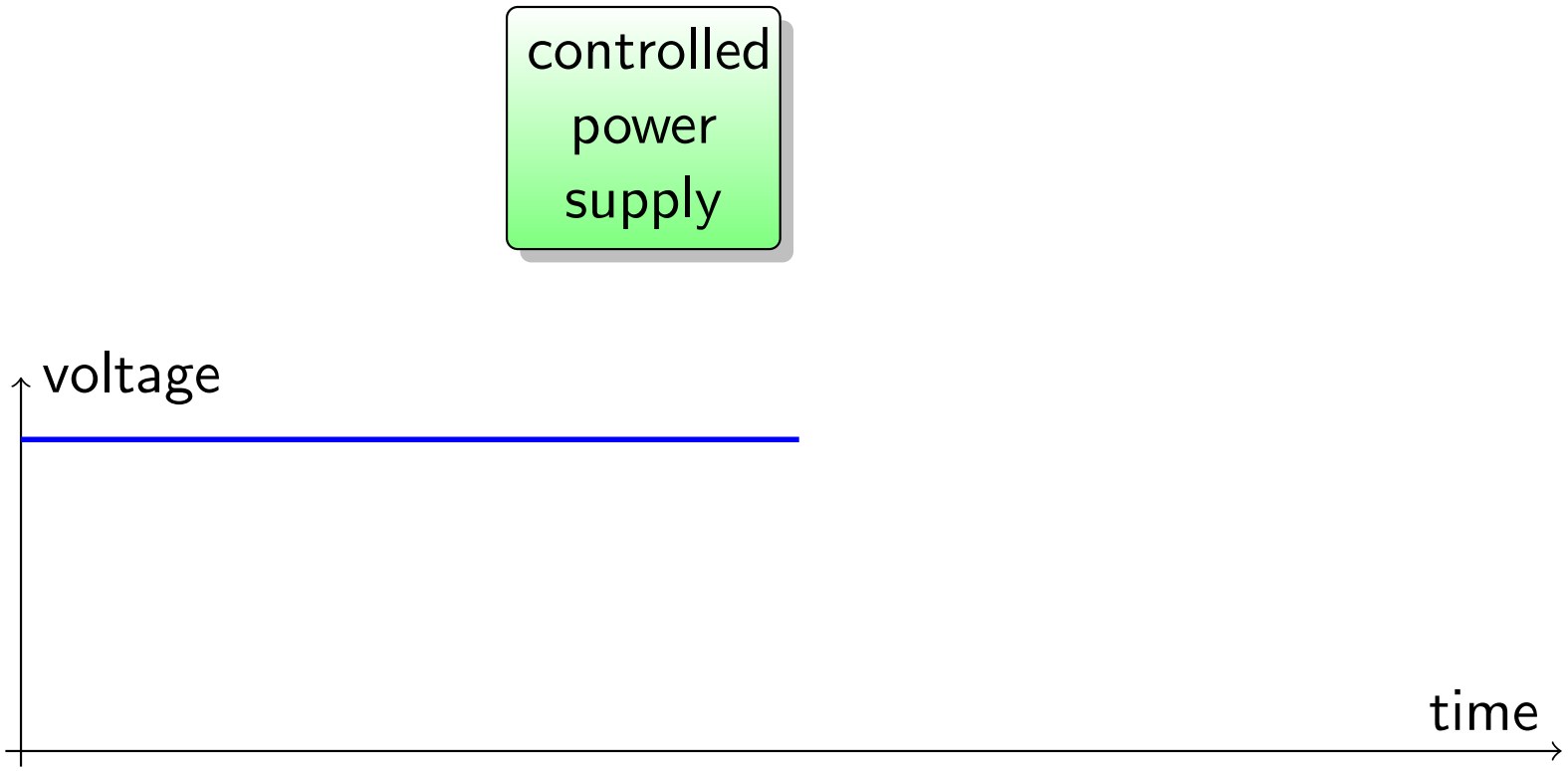
- perturbation in the power supply voltage
- perturbation of the clock signal
- temperature (over/under-heating the chip)
- radiation or electromagnetic (EM) disturbances
- exposing the chip to intense lights or beams
- etc

Accuracy:

- time: part of clock cycle, clock cycle, code block (instruction sequence)
- space: gate, block, unit, core, chip, package
- value: set to a specific value, bit flip, stuck-at 0 or 1, random modification

Perturbation on the Power Supply

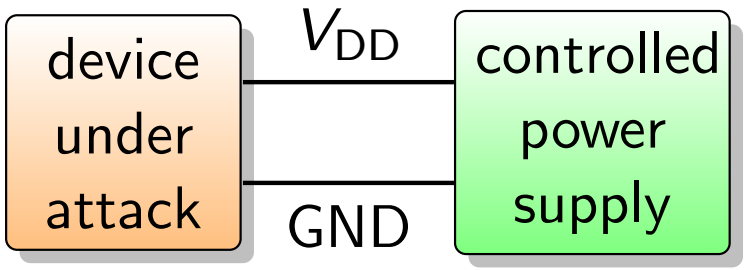
Principle:

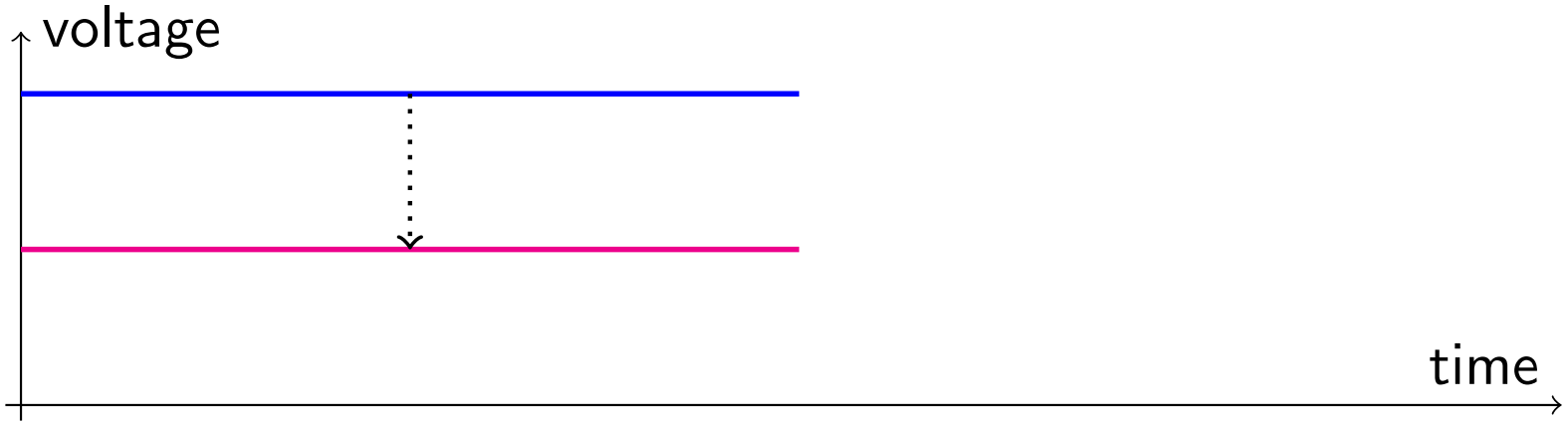


- Nominal power supply (e.g. $\approx [0.7, 1.2]$ V for current technologies)

Perturbation on the Power Supply

Principle:

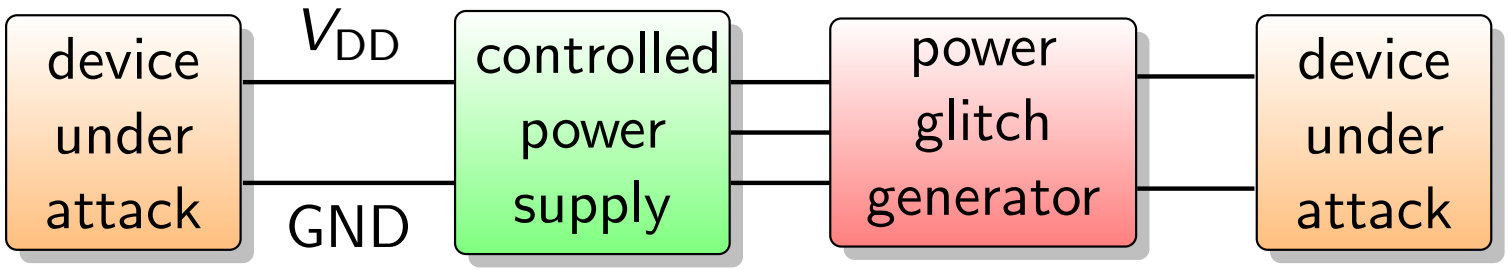


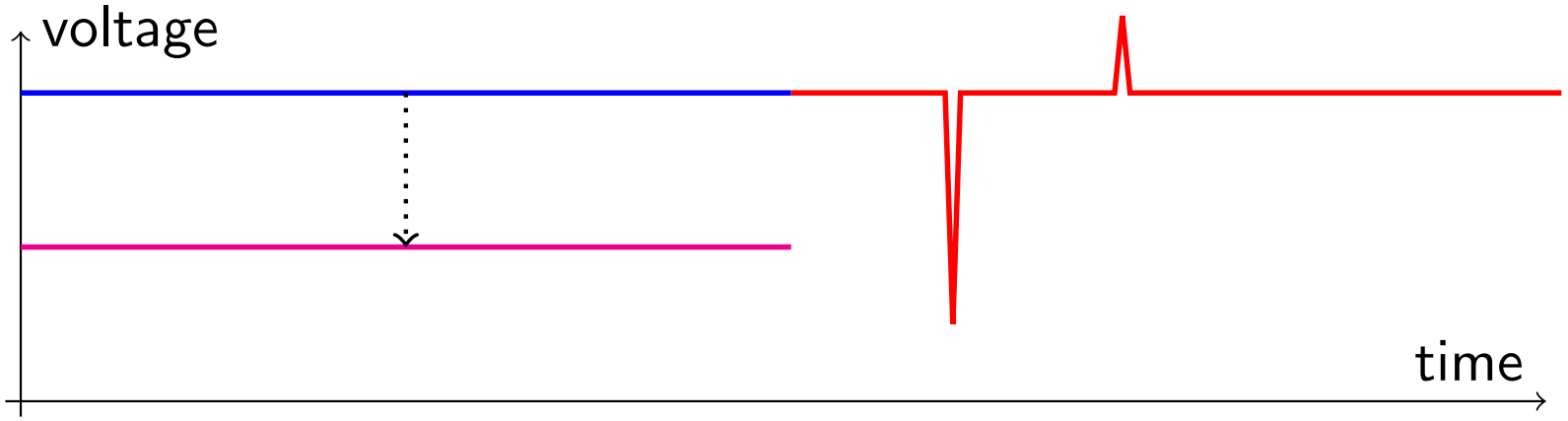


- Nominal power supply (e.g. $\approx [0.7, 1.2]$ V for current technologies)
- Non-nominal constant power supply (e.g. 0.7 V instead of 1.2 V)

Perturbation on the Power Supply

Principle:





- Nominal power supply (e.g. $\approx [0.7, 1.2]$ V for current technologies)
- Non-nominal constant power supply (e.g. 0.7 V instead of 1.2 V)
- Glitches (dips, spikes) in the power supply at some selected moments

Under Powering Example

Source: paper [19] presented at EDCC 2008 conference

Setup: 130 nm smart card (1.2 V nominal V_{DD}) with AES crypto-processor

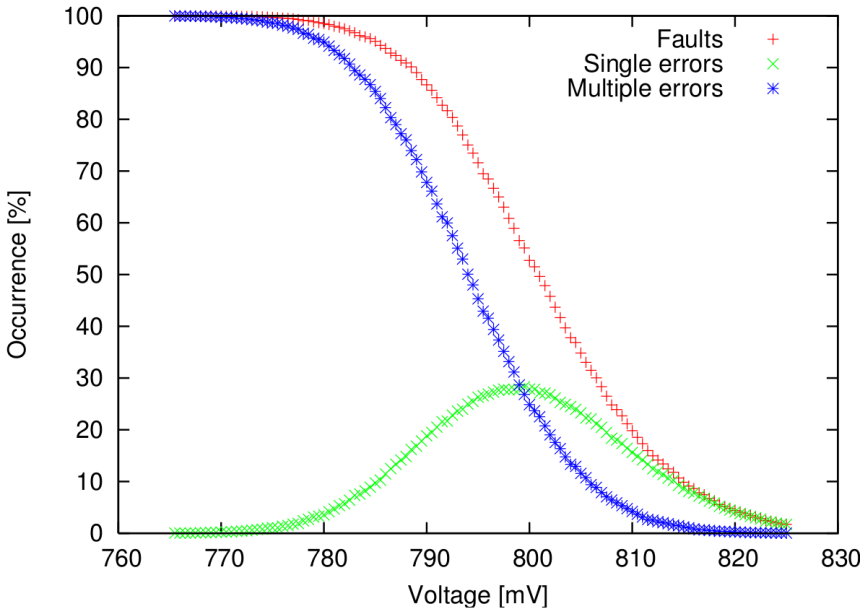
Measurement campaign: triples (msg, key, cypher) recorded for 100 V_{DD} in [775, 825] mV over 20,000 encryptions with comparison to a (RTL) simulation for one byte corruption in the state matrix at various rounds

Under Powering Example

Source: paper [19] presented at EDCC 2008 conference

Setup: 130 nm smart card (1.2 V nominal V_{DD}) with AES crypto-processor

Measurement campaign: triples (msg, key, cypher) recorded for 100 V_{DD} in [775, 825] mV over 20,000 encryptions with comparison to a (RTL) simulation for one byte corruption in the state matrix at various rounds



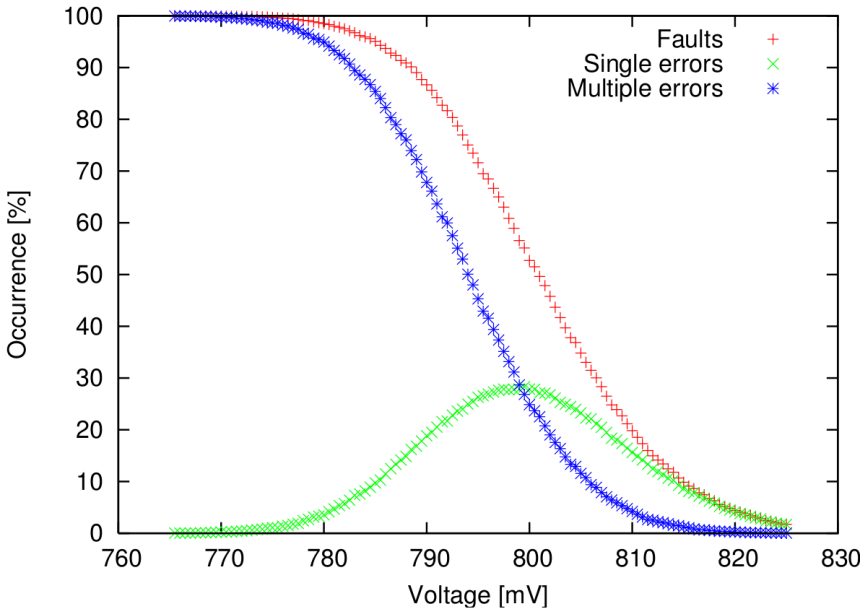
Observed behavior is compatible with setup violation model on a critical path (bell shape due to only one or multiple paths)

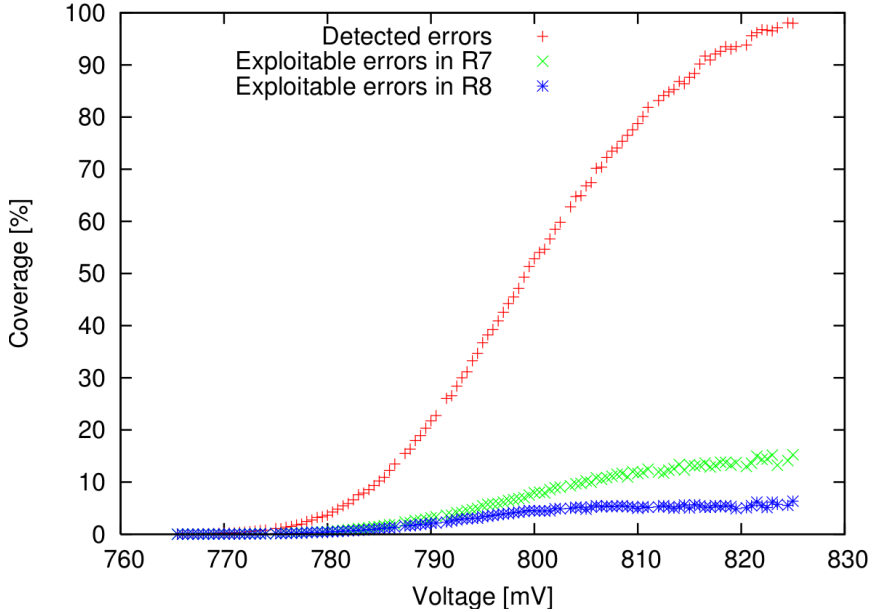
Under Powering Example

Source: paper [19] presented at EDCC 2008 conference

Setup: 130 nm smart card (1.2 V nominal V_{DD}) with AES crypto-processor

Measurement campaign: triples (msg, key, cypher) recorded for 100 V_{DD} in [775, 825] mV over 20,000 encryptions with comparison to a (RTL) simulation for one byte corruption in the state matrix at various rounds



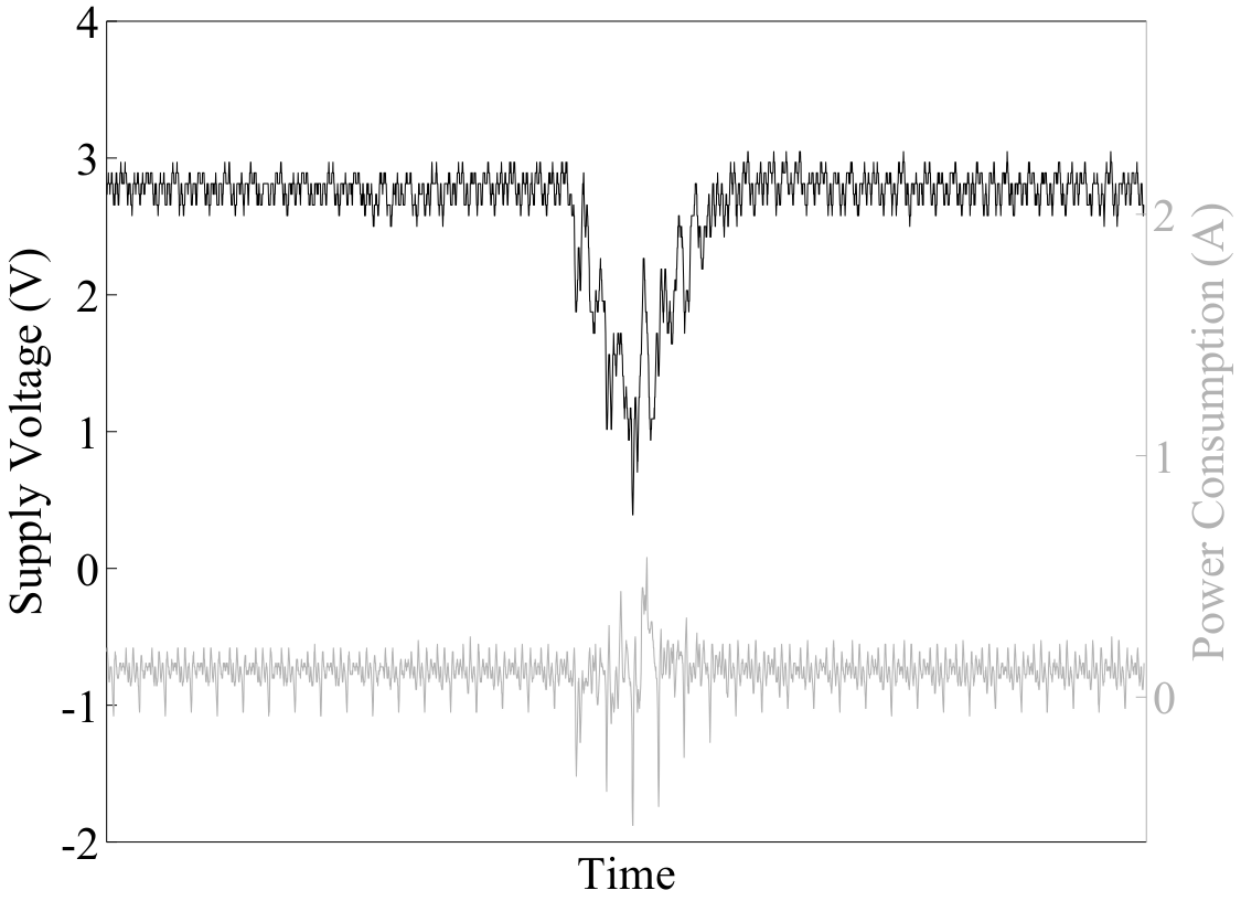


Observed behavior is compatible with setup violation model on a critical path (bell shape due to only one or multiple paths)

Power Glitching Example

Source: FDTC 2008 conference paper [18]

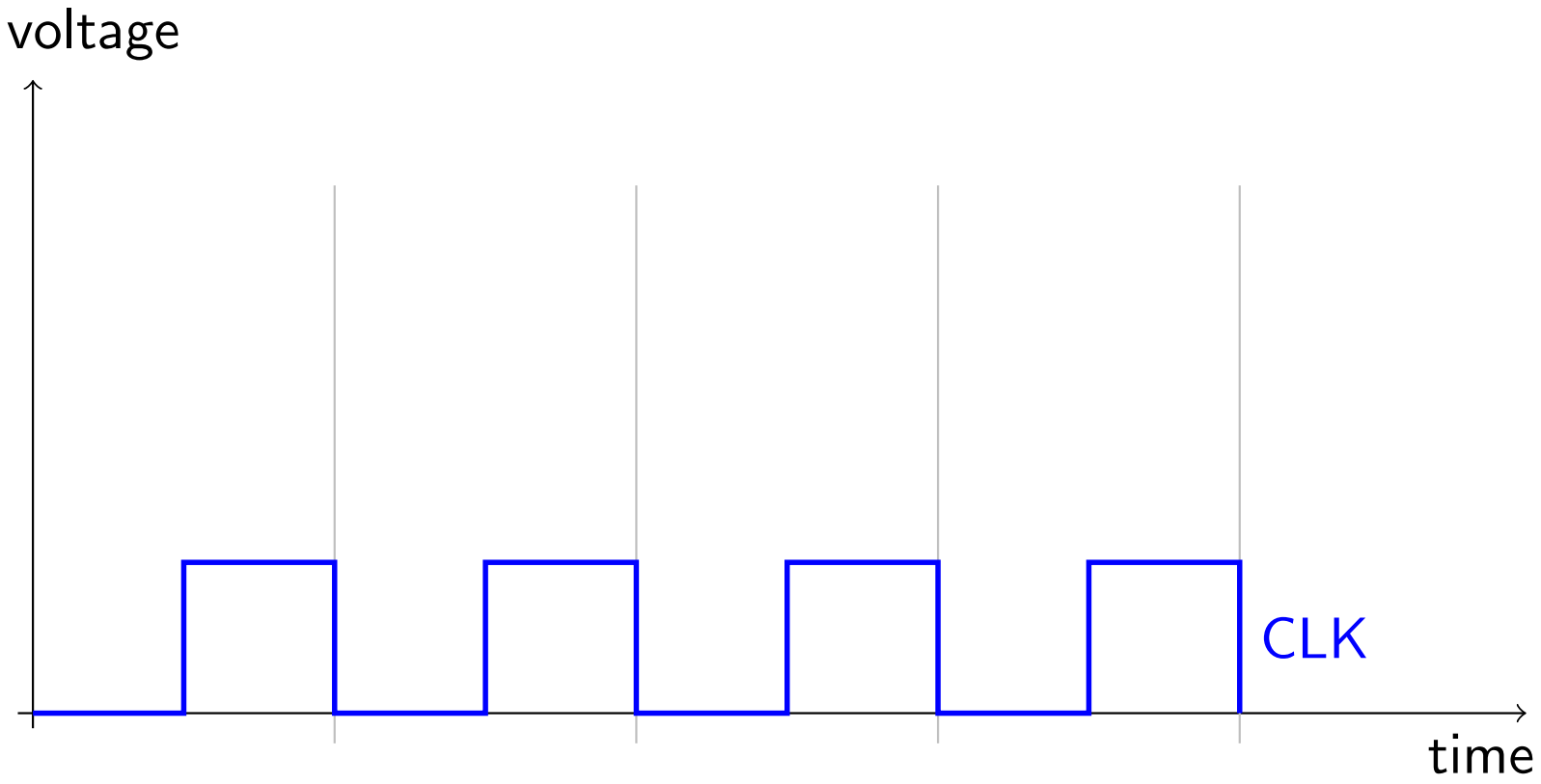
Setup: AVR microcontroller with RSA implementation



Attack result: a power glitch causes to skip some instruction

Perturbation on the External Clock

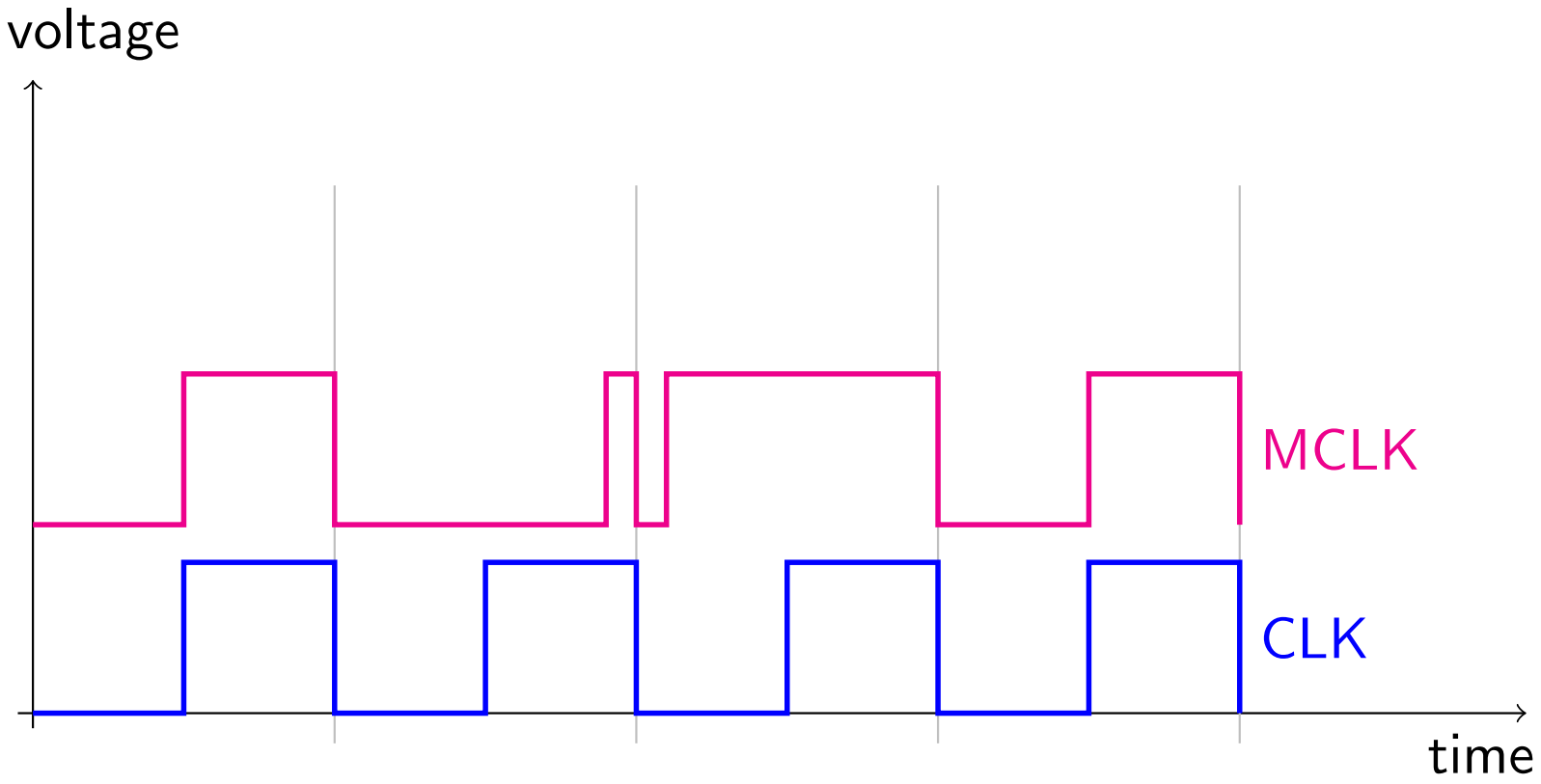
Principle:



- Normal clock (at a given frequency, duty cycle $\approx 50\%$)

Perturbation on the External Clock

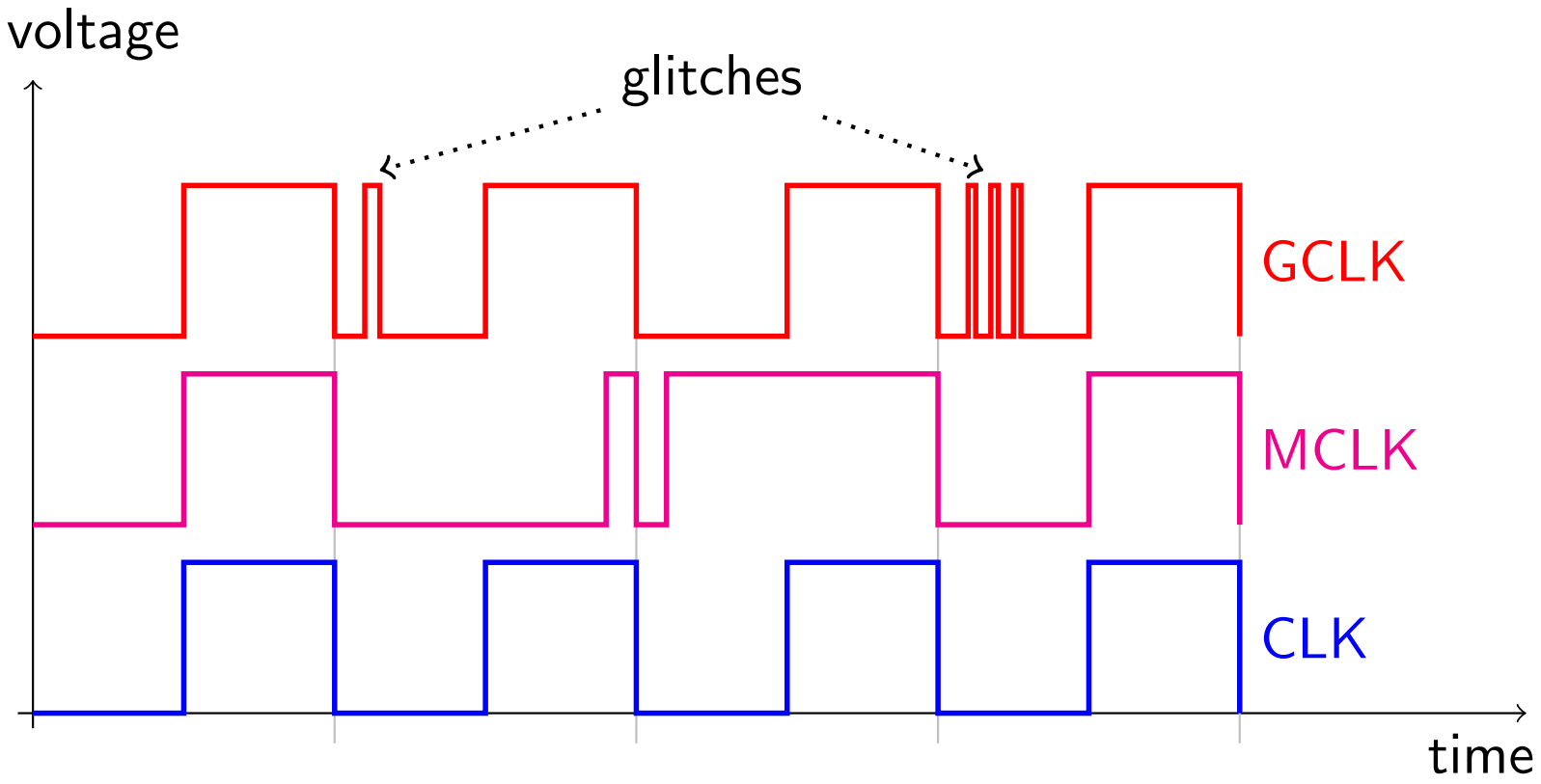
Principle:



- Normal clock (at a given frequency, duty cycle $\approx 50\%$)
- Clock with a modified duty cycle

Perturbation on the External Clock

Principle:



- Normal clock (at a given frequency, duty cycle $\approx 50\%$)
- Clock with a modified duty cycle
- Glitched clock
- Etc.

Glitchy Clock Generation Example

Source: paper [10] published in J. Crypto. Eng. 2011

Setup: Virtex-II Pro FPGA (on SASEBO card) used to generate a "glitchy" clock for several programmable time parameters

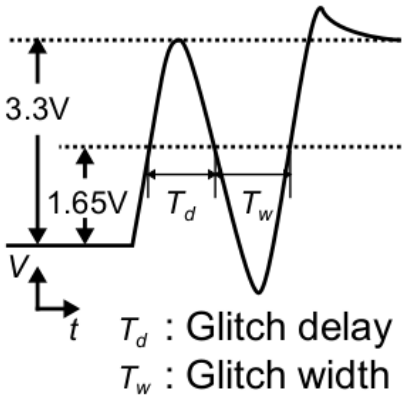


Fig. 3. Image of glitchy-clock cycle.

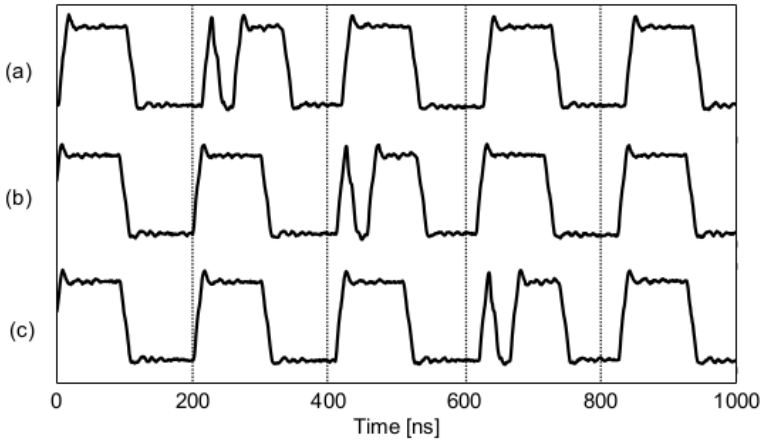
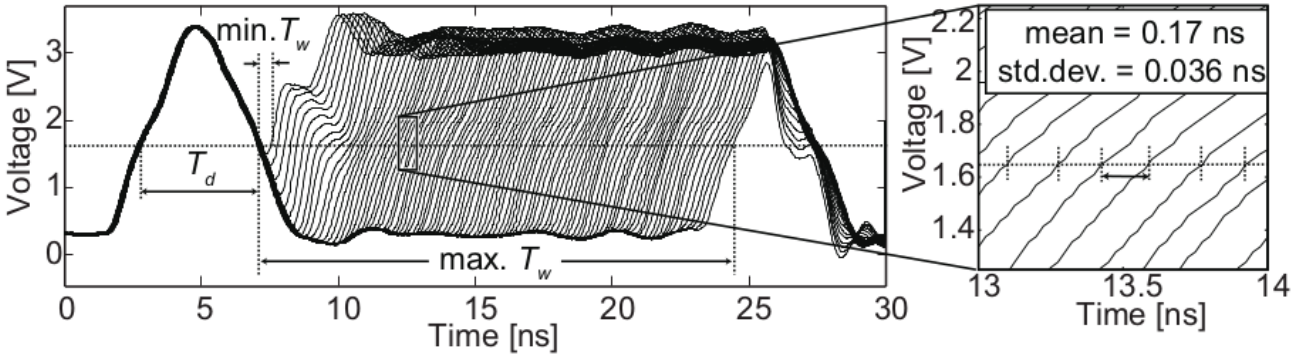


Fig. 4. Examples of glitchy-clock signals.



Clock Glitch Attack Example

Source: paper [1] presented at FDTC 2011 conference

Setup: AVR ATMega 163 microcontroller @ 1MHz

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`EOR R15,R5`	0010 0100 1111 0101

Clock Glitch Attack Example

Source: paper [1] presented at FDTC 2011 conference

Setup: AVR ATMega 163 microcontroller @ 1MHz

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`EOR R15,R5`	0010 0100 1111 0101
glitch	59 ns	$i+1$	`NOP`	0000 0000 0000 0000

Clock Glitch Attack Example

Source: paper [1] presented at FDTC 2011 conference

Setup: AVR ATMega 163 microcontroller @ 1MHz

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`EOR R15,R5`	0010 0100 1111 0101
glitch	59 ns	$i+1$	`NOP`	0000 0000 0000 0000

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`SER R18`	1110 1111 0010 1111

Clock Glitch Attack Example

Source: paper [1] presented at FDTC 2011 conference

Setup: AVR ATMega 163 microcontroller @ 1MHz

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`EOR R15,R5`	0010 0100 1111 0101
glitch	59 ns	$i+1$	`NOP`	0000 0000 0000 0000

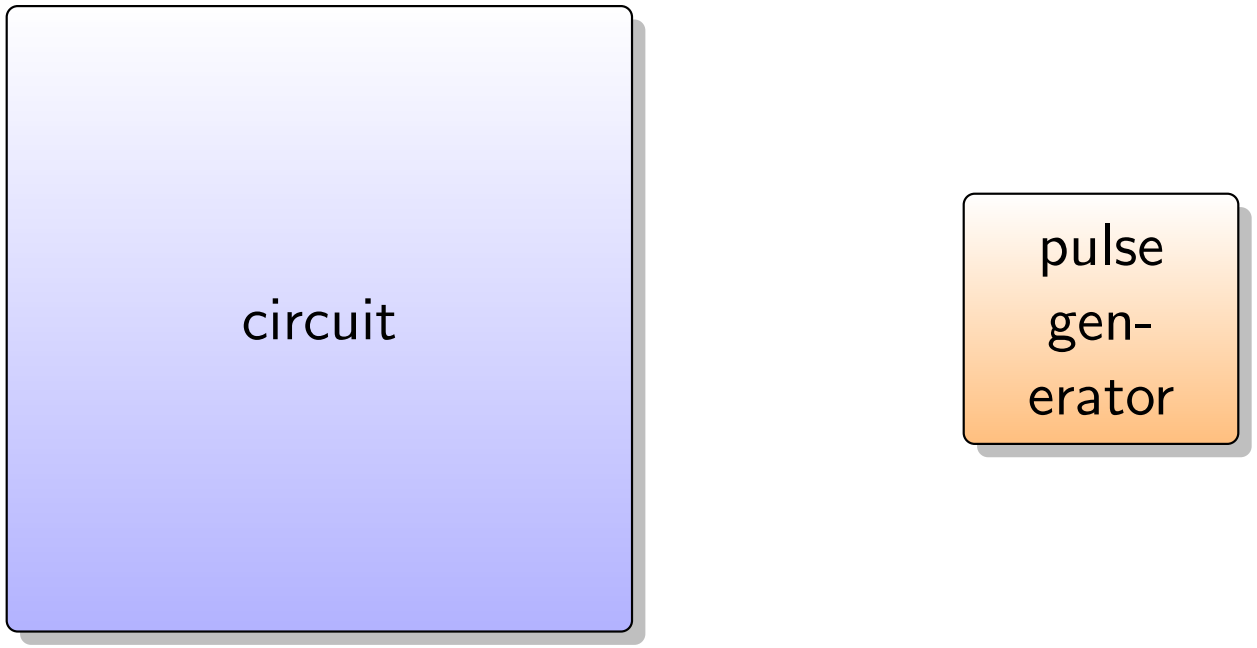
mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`NOP`	0000 0000 0000 0000
normal	-	$i+1$	`SER R18`	1110 1111 0010 1111
glitch	61 ns	$i+1$	`LDI R18,0xEF`	1110 1110 0010 1111
glitch	60 ns	$i+1$	`SBC R12,R15`	0000 1000 0010 1111
glitch	59 ns	$i+1$	`NOP`	0000 0000 0000 0000

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`TST R12`	0010 0000 1100 1100
normal	-	$i+1$	`BREQ PC+0x02`	1111 0000 0000 1001
normal	-	$i+2$	`SER R26`	1110 1111 1010 1111

mode	glitch period	cycle	instruction	opcode (bin)
normal	-	i	`TST R12`	0010 0000 1100 1100
normal	-	$i+1$	`BREQ PC+0x02`	1111 0000 0000 1001
normal	-	$i+2$	`SER R26`	1110 1111 1010 1111
glitch	57 ns	$i+2$	`LDI R26,0xEF`	1110 1110 1010 1111
glitch	56 ns	$i+2$	`LDI R26,0xCF`	1110 1100 1010 1111
glitch	52 ns	$i+2$	`LDI R26,0x0F`	1110 0000 1010 1111
glitch	45 ns	$i+2$	`LDI R16,0x09`	1110 0000 0000 1001
glitch	32 ns	$i+2$	`LD R0,Y+0x01`	1000 0000 0000 1001
glitch	28 ns	$i+2$	`LD R9,Y`	1000 0000 0000 1000
glitch	27 ns	$i+2$	`LDI R16,0x09`	1110 0000 0000 1001
glitch	15 ns	$i+2$	`BREQ PC+0x02`	1111 0000 0000 1001

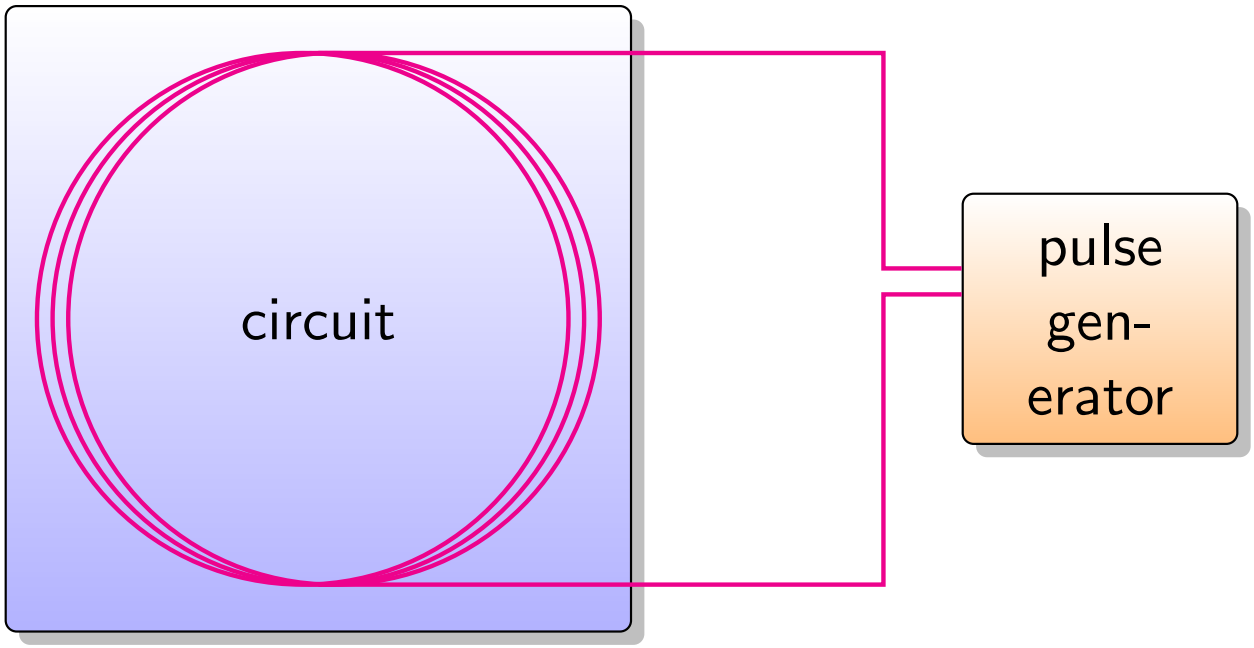
Electromagnetic Perturbations

Principle:



Electromagnetic Perturbations

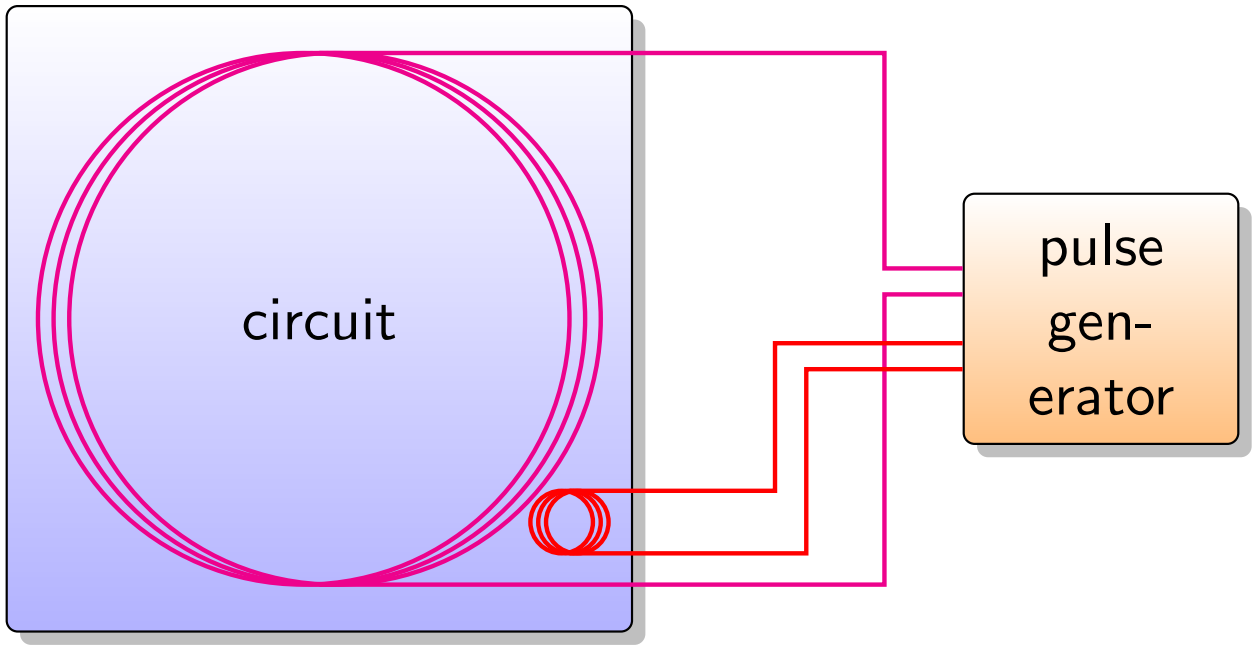
Principle:



- large antenna

Electromagnetic Perturbations

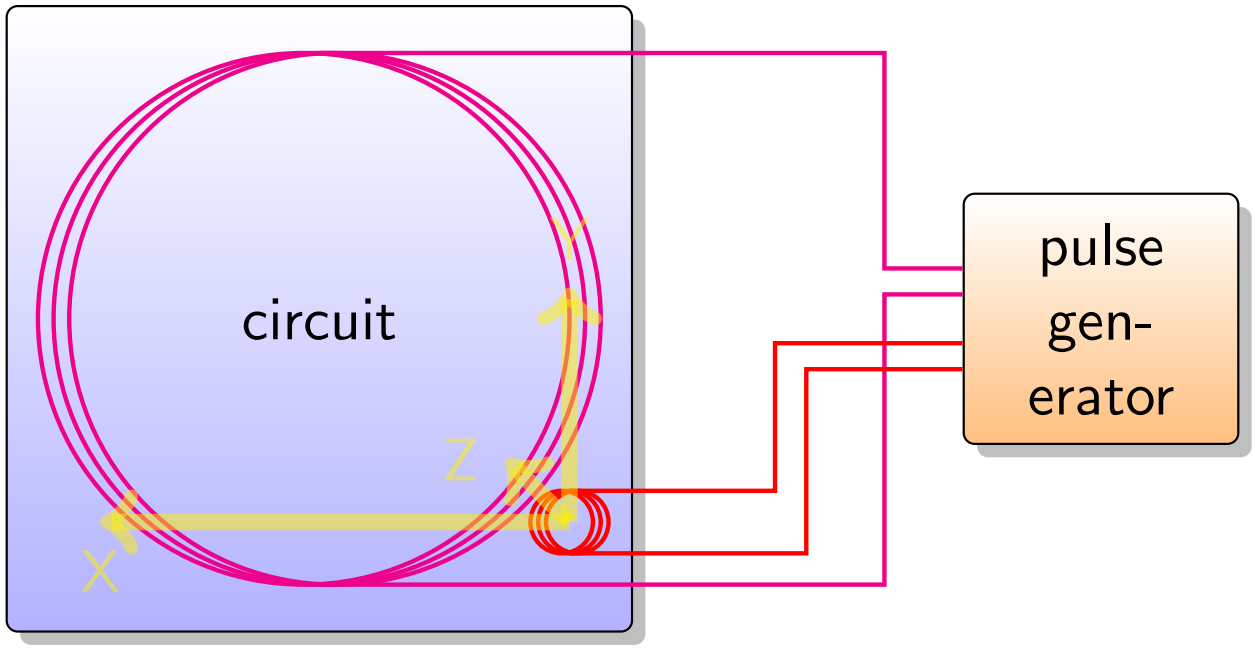
Principle:



- large antenna
- micro-antenna

Electromagnetic Perturbations

Principle:



- large antenna
- micro-antenna with motorized (X,Y,Z) stage/table

Electromagnetic Attack Example

Source: article [12] presented at FDTC 2013 conference

Setup: 32-b Cortex-M3 ARM microprocessor (CMOS 130 nm SoC at 56 MHz), magnetic antenna with pulses in [-200, 200] V and [10, 200] ns

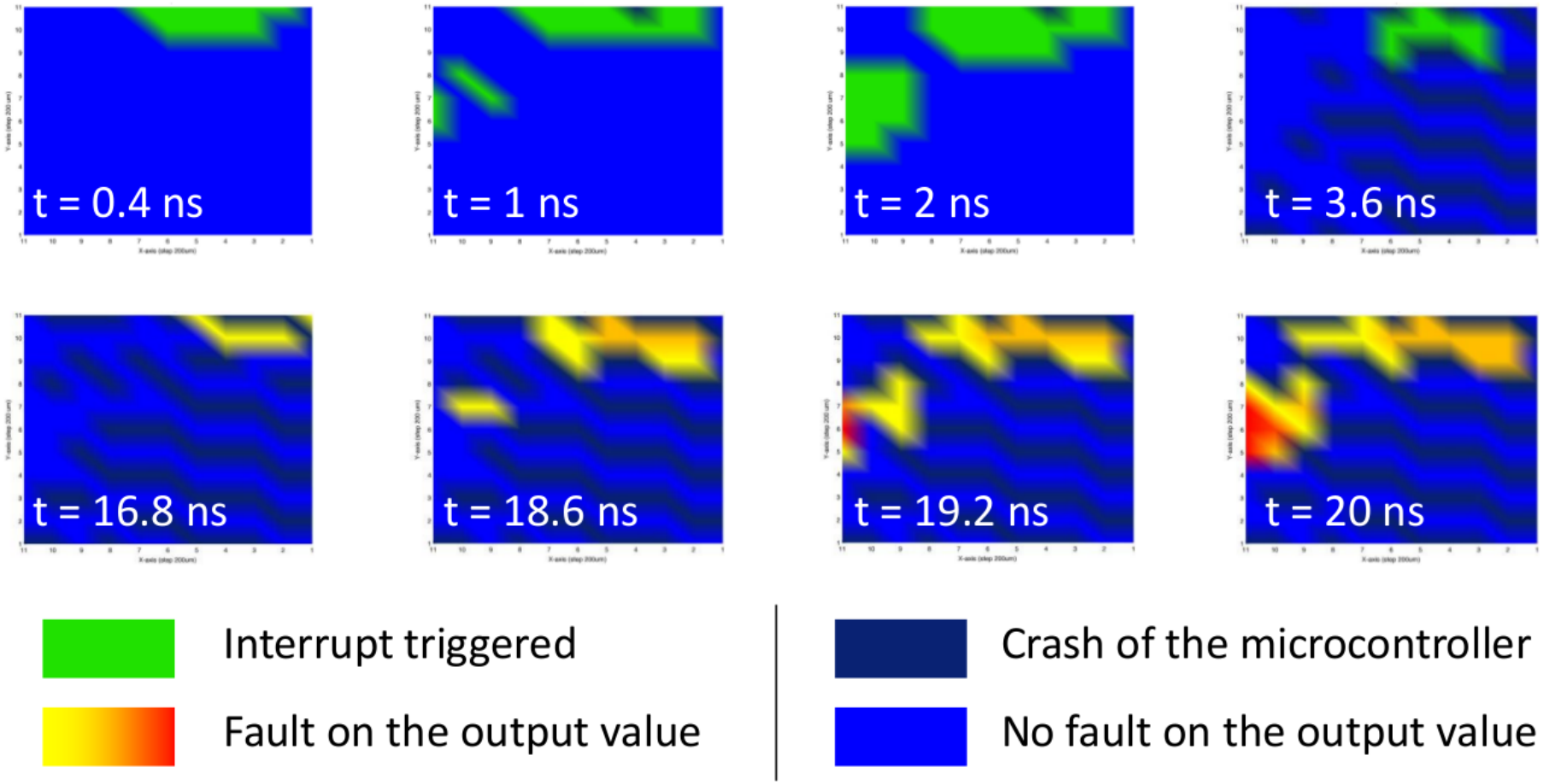


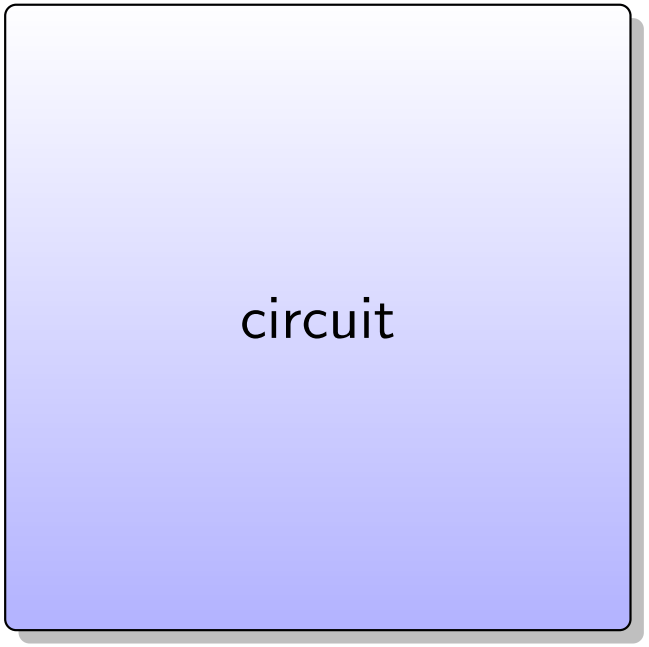
Figure 3: Impact of the probe's position

Loaded value: 12345678

Pulse voltage [V]	Loaded value	Occurrence rate [%]
170	1234 5678	100
172	1234 5678	100
174	9234 5678	73
176	FE34 5678	30
178	FFF4 5678	53
180	FFFD 5678	50
182	FFFF 7F78	46
184	FFFF FFFB	40
186	FFFF FFFF	100
188	FFFF FFFF	100
190	FFFF FFFF	100

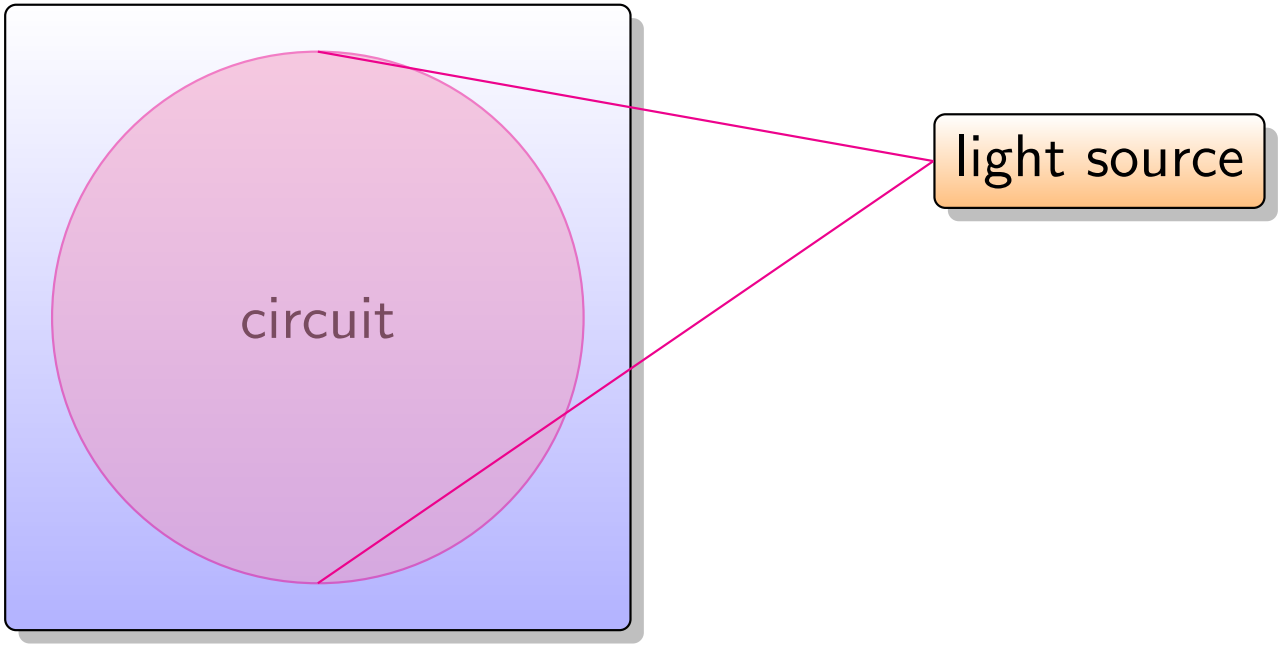
Lights / Lasers

Principle:



Lights / Lasers

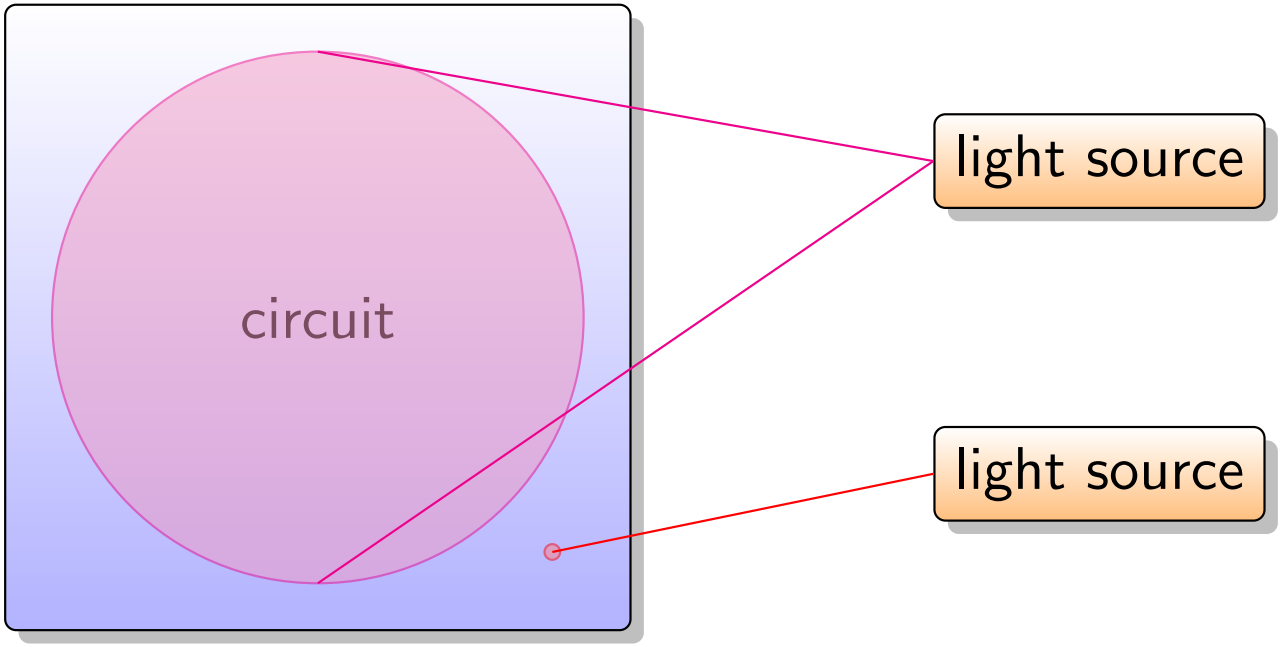
Principle:



- large illuminated area (flash light with microscope)

Lights / Lasers

Principle:



- large illuminated area (flash light with microscope)
- small "spot" (laser with variable locations)

Differential Fault Analysis

Most of time, exploiting only one fault does not provide enough information

- Accurately injecting fault is difficult
- The fault causes a few perturbations

Differential Fault Analysis

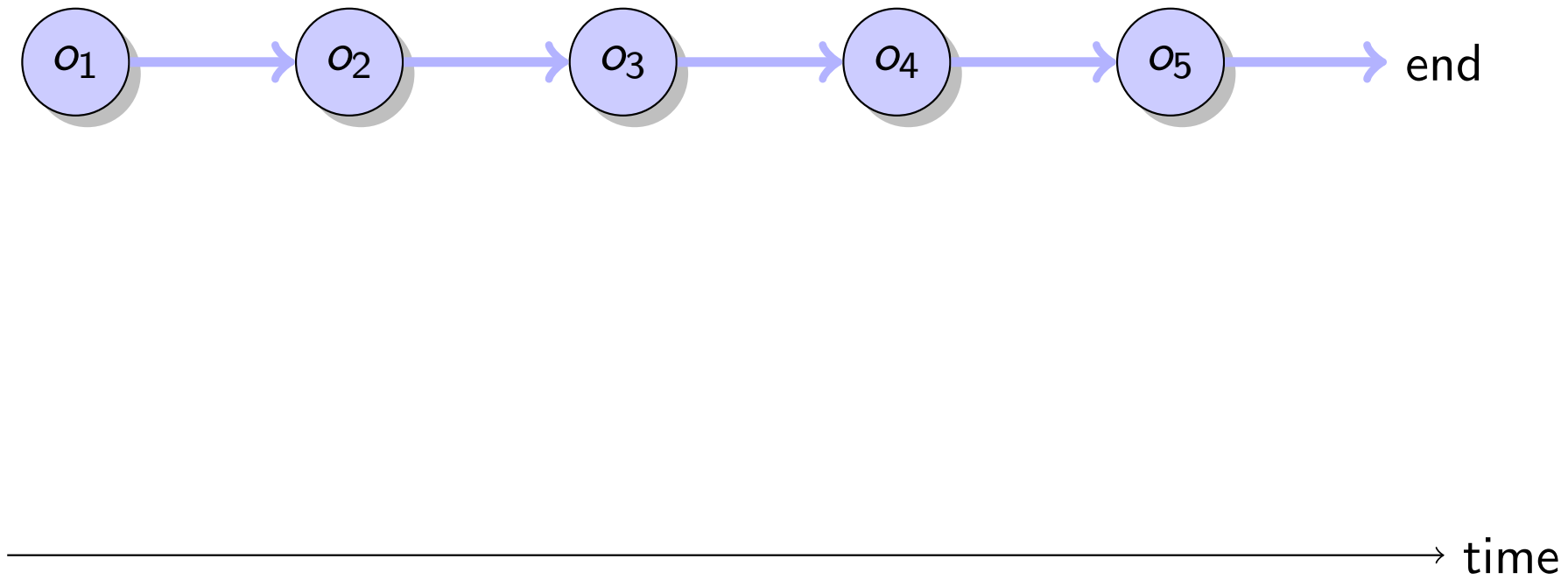
Most of time, exploiting only one fault does not provide enough information

- Accurately injecting fault is difficult
- The fault causes a few perturbations

Then, use statistical correlation(s)

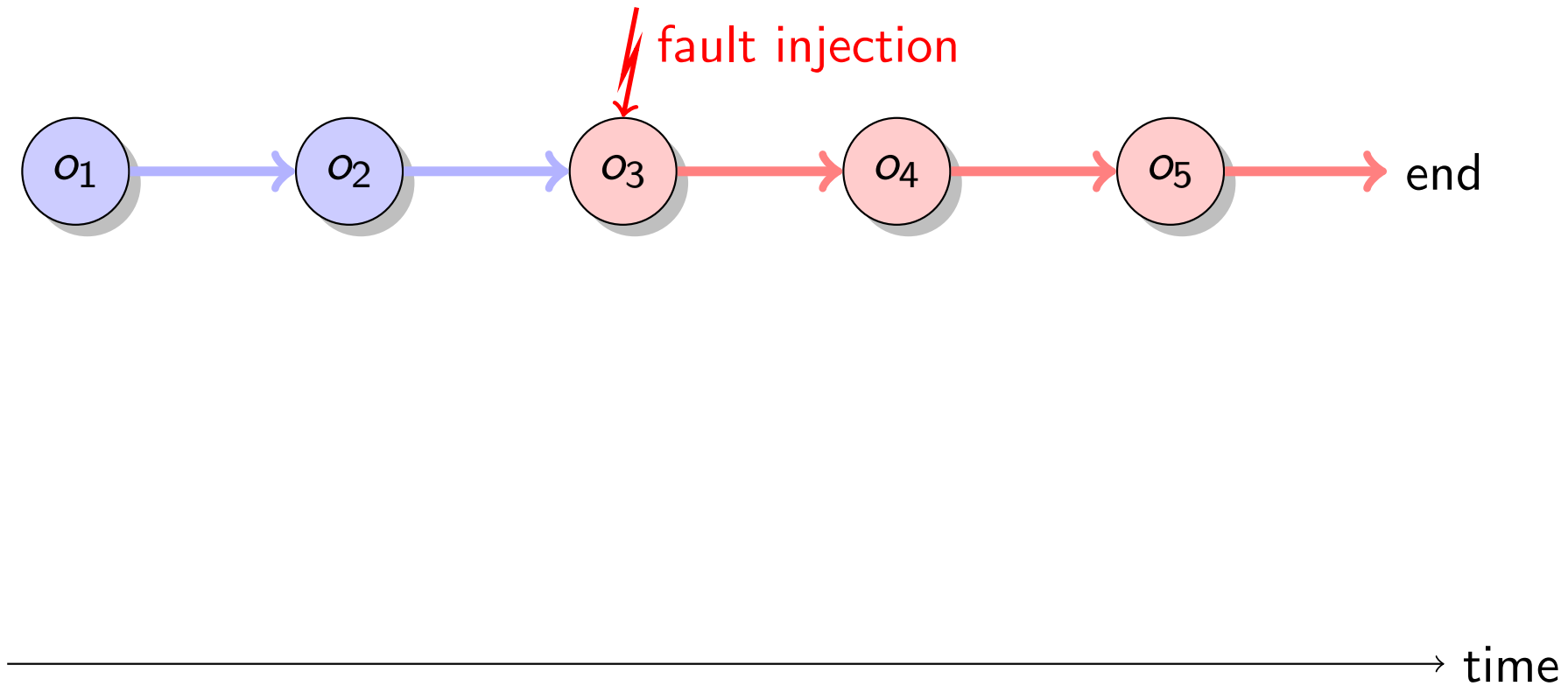
Safe Error Attack

Principle: exploit the link (or the lack of link) between injected fault(s) during "useful" (or "useless") operations and the final result



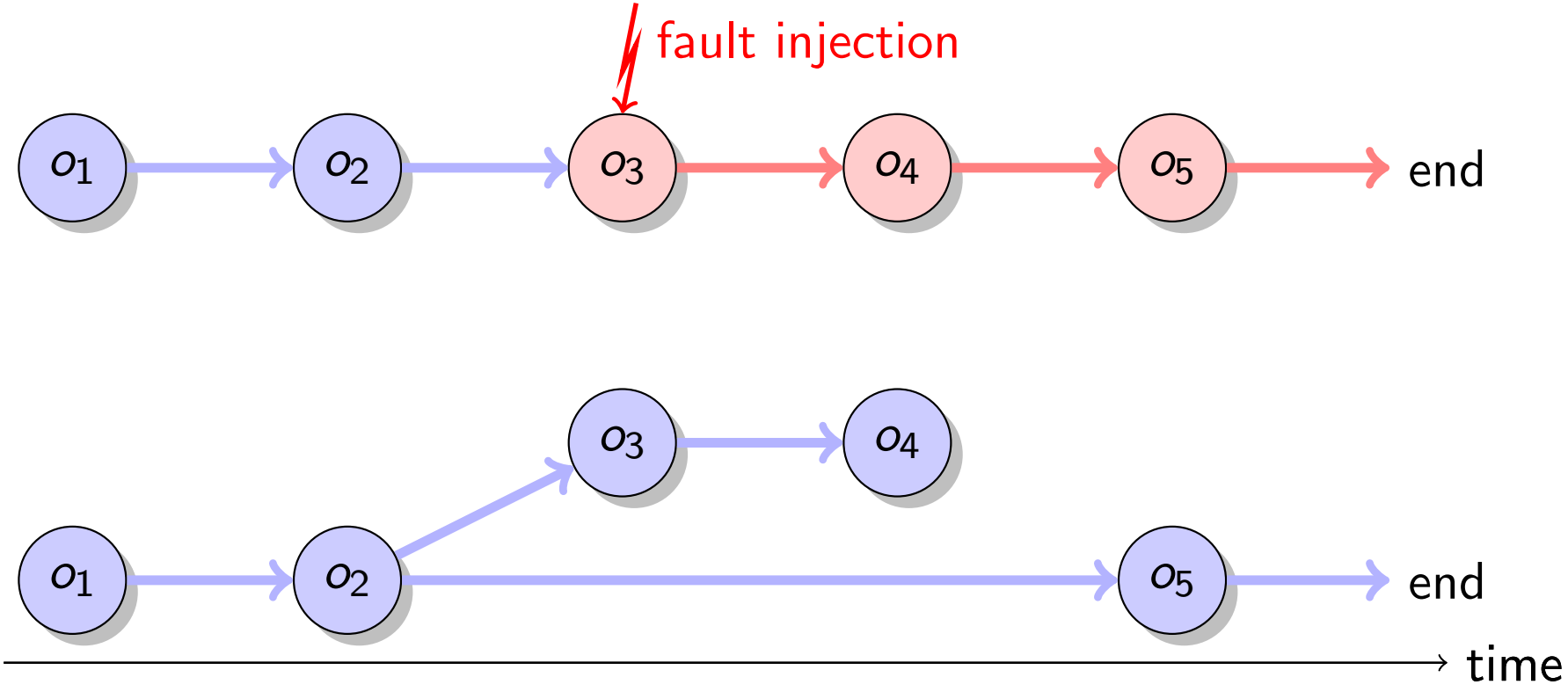
Safe Error Attack

Principle: exploit the link (or the lack of link) between injected fault(s) during "useful" (or "useless") operations and the final result



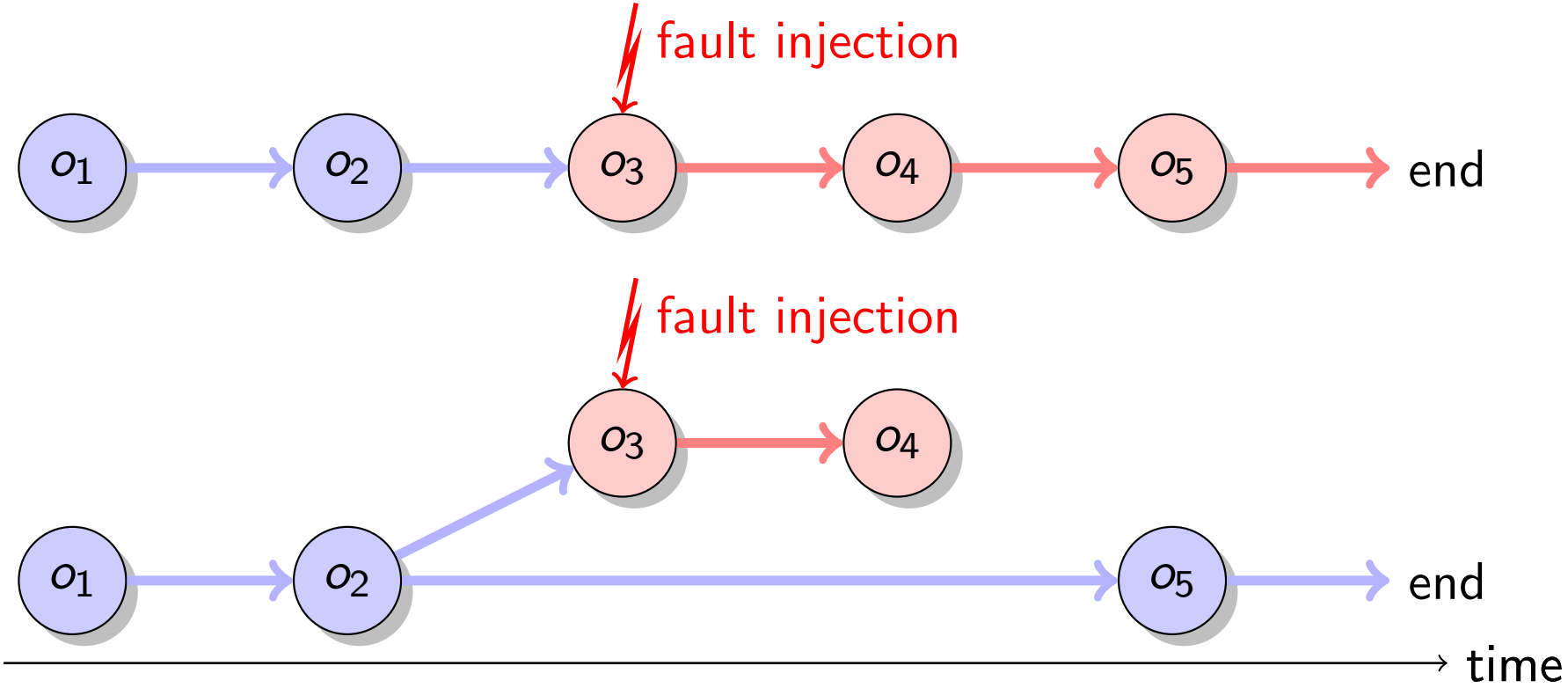
Safe Error Attack

Principle: exploit the link (or the lack of link) between injected fault(s) during "useful" (or "useless") operations and the final result

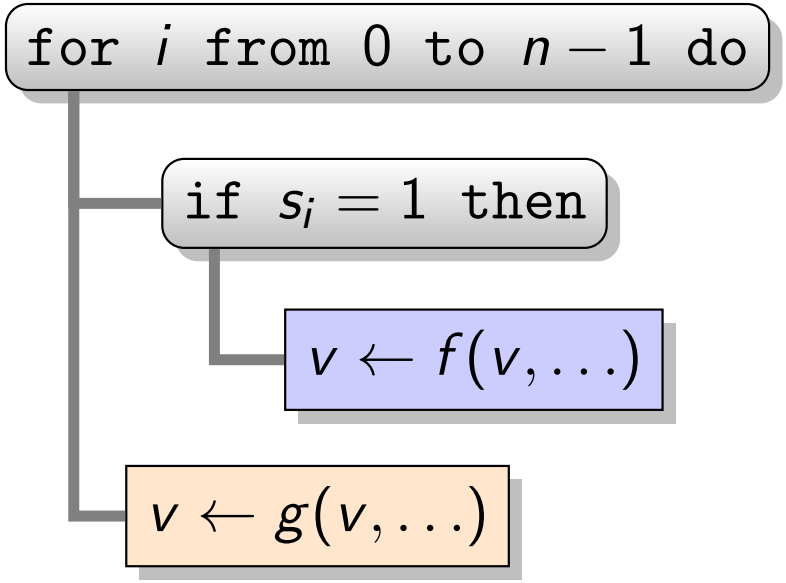


Safe Error Attack

Principle: exploit the link (or the lack of link) between injected fault(s) during "useful" (or "useless") operations and the final result

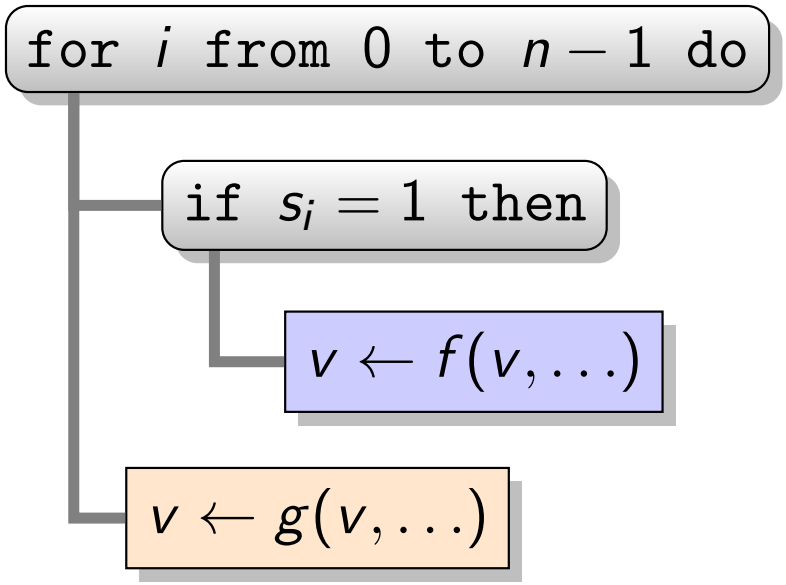


Safe Error Attack Example in Asymmetric Crypto



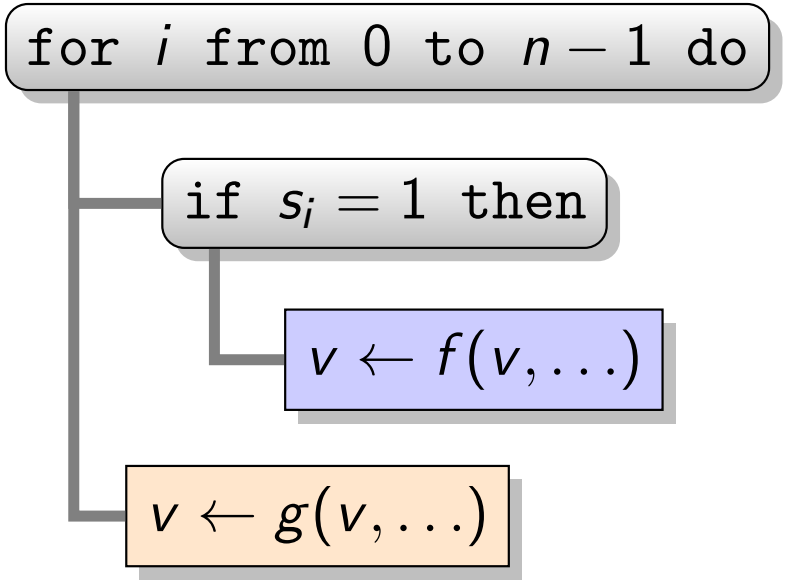
Safe Error Attack Example in Asymmetric Crypto

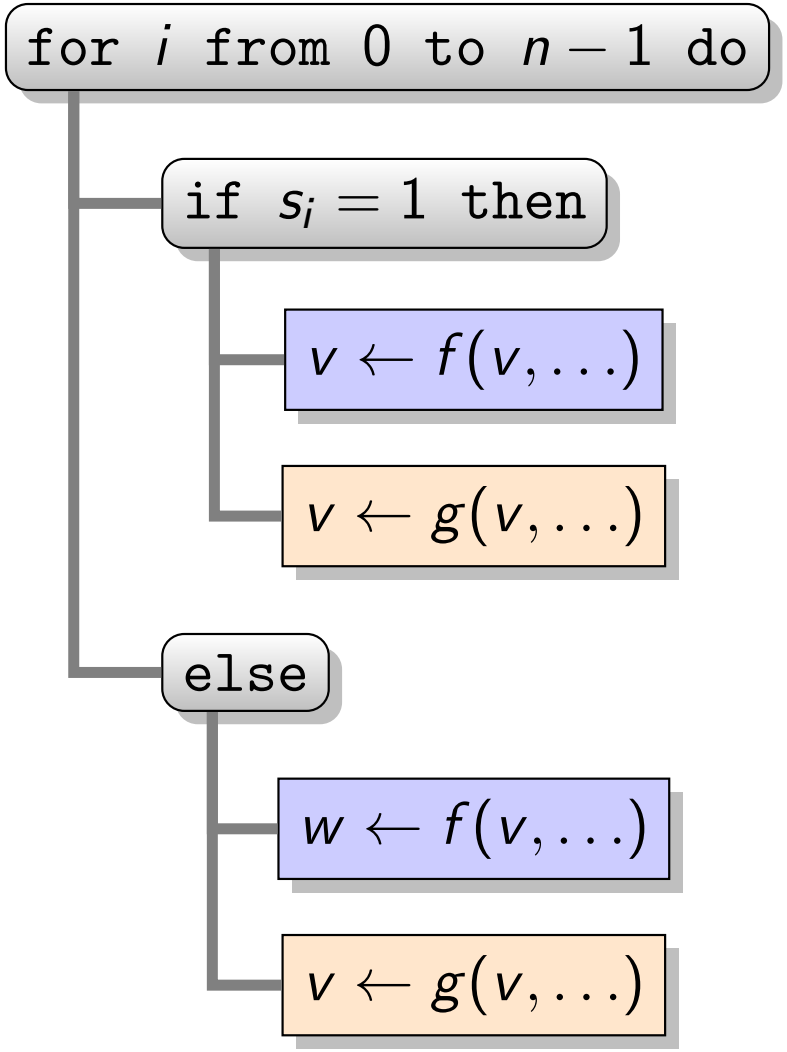
WEAK against SPA



Safe Error Attack Example in Asymmetric Crypto

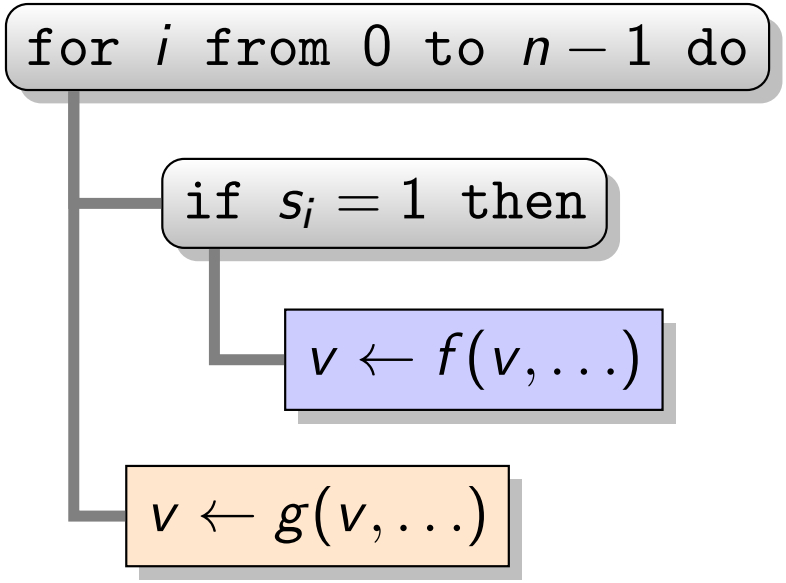
WEAK against SPA



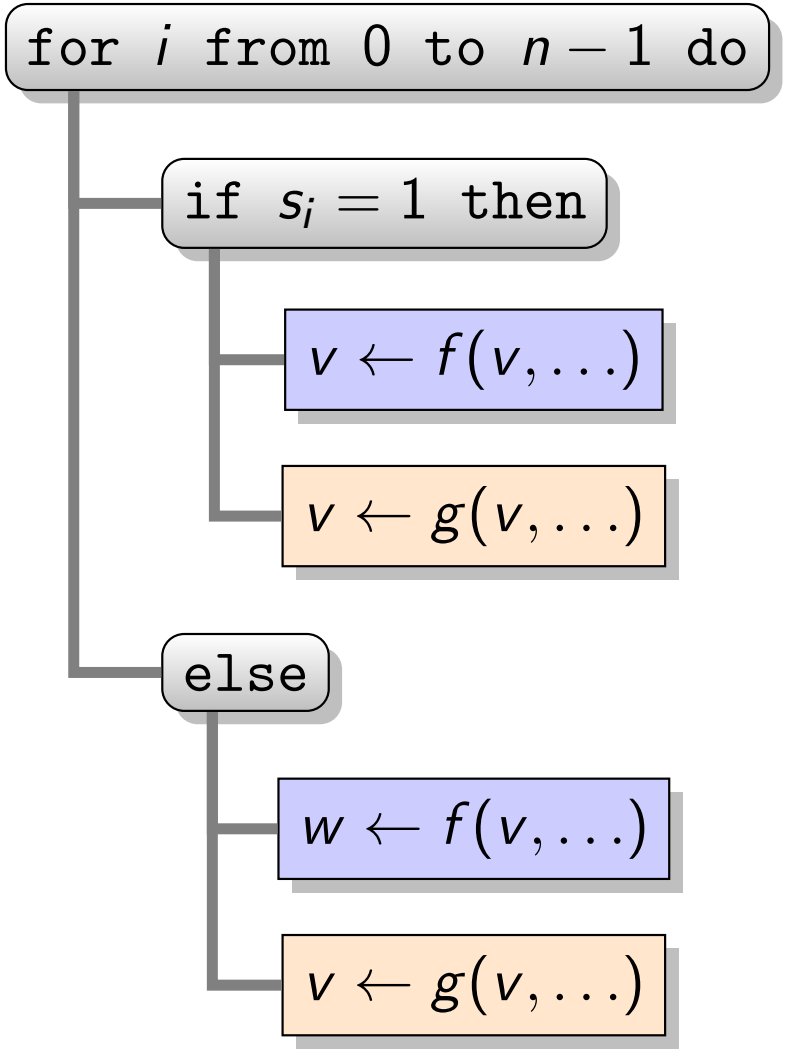


Safe Error Attack Example in Asymmetric Crypto

WEAK against SPA

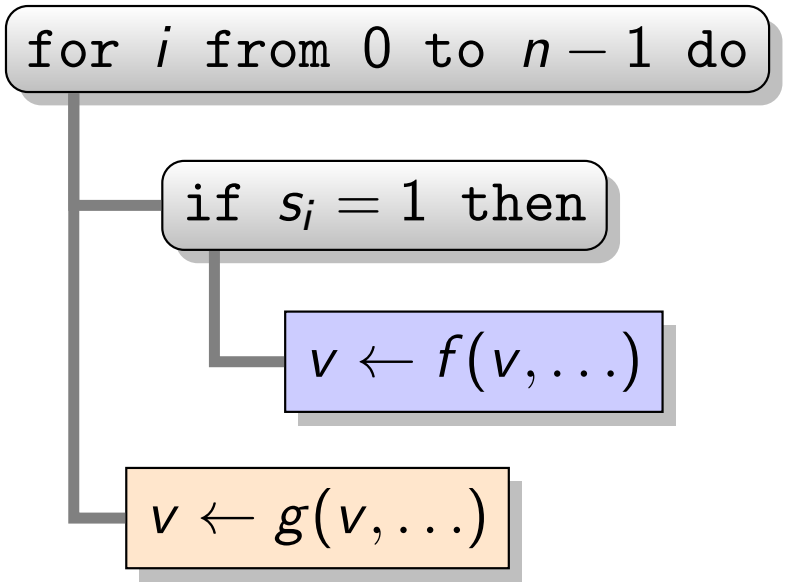


WEAK against SEA

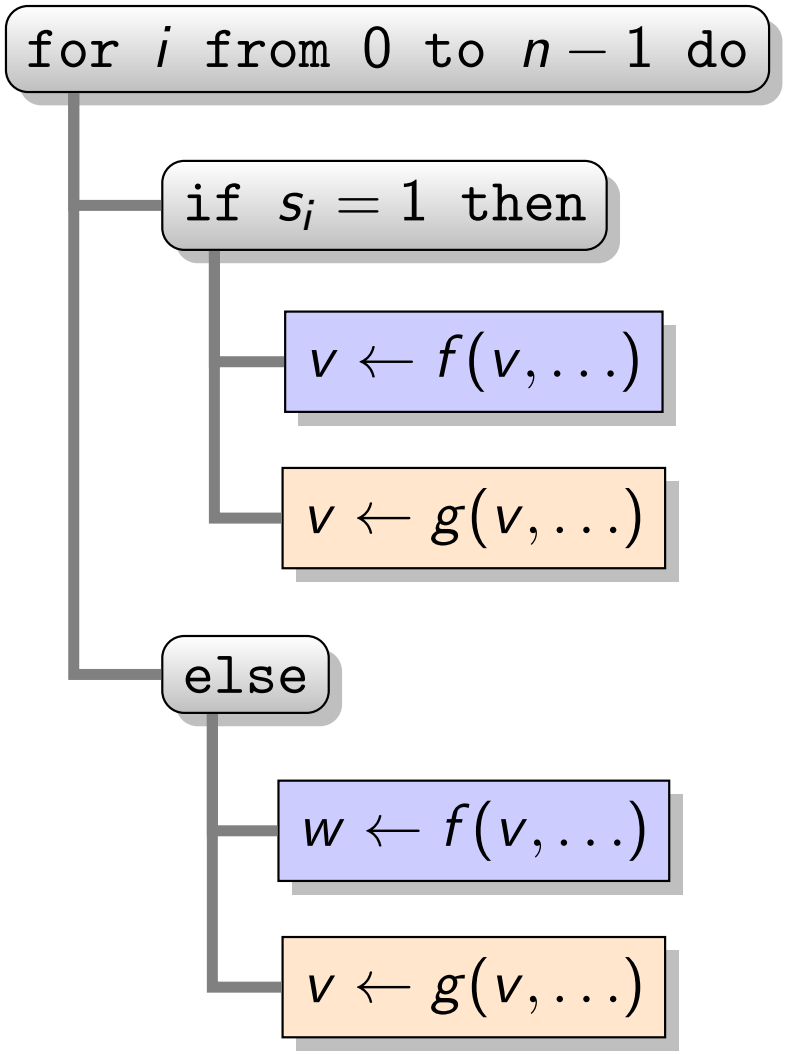


Safe Error Attack Example in Asymmetric Crypto

WEAK against SPA



WEAK against SEA



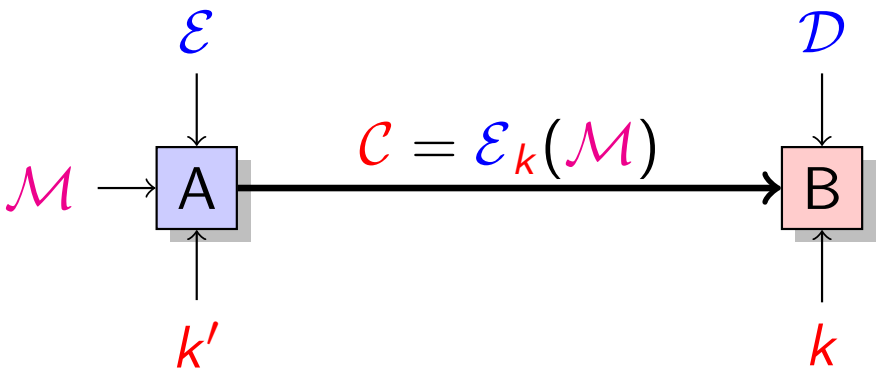
Useless or dummy operations are a bad idea

Fault Attack Example: Bit Flip on RSA Decryption

Fault Attack Example: Bit Flip on RSA Decryption

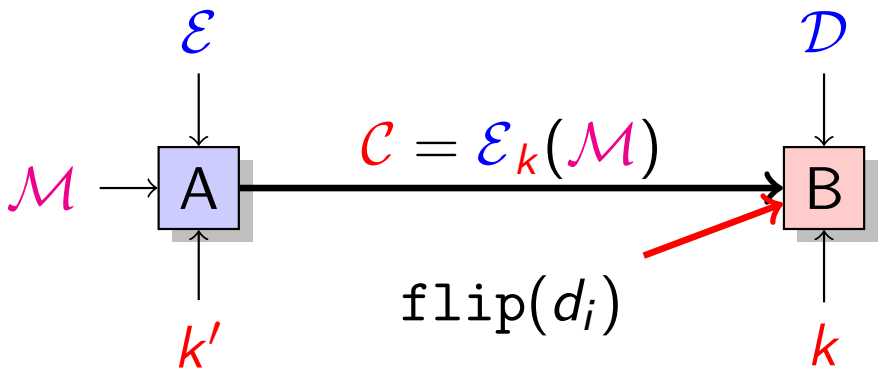
- choose a plaintext message $\mathcal{M}$

Fault Attack Example: Bit Flip on RSA Decryption



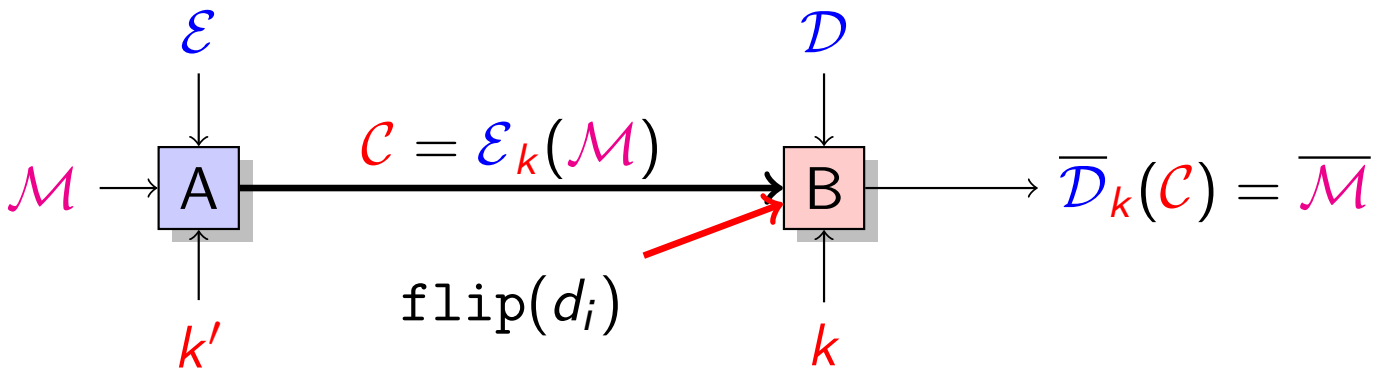
- choose a plaintext message $\mathcal{M}$
- encrypt $\mathcal{M}$ into $\mathcal{C} = \mathcal{E}_k(\mathcal{M})$

Fault Attack Example: Bit Flip on RSA Decryption



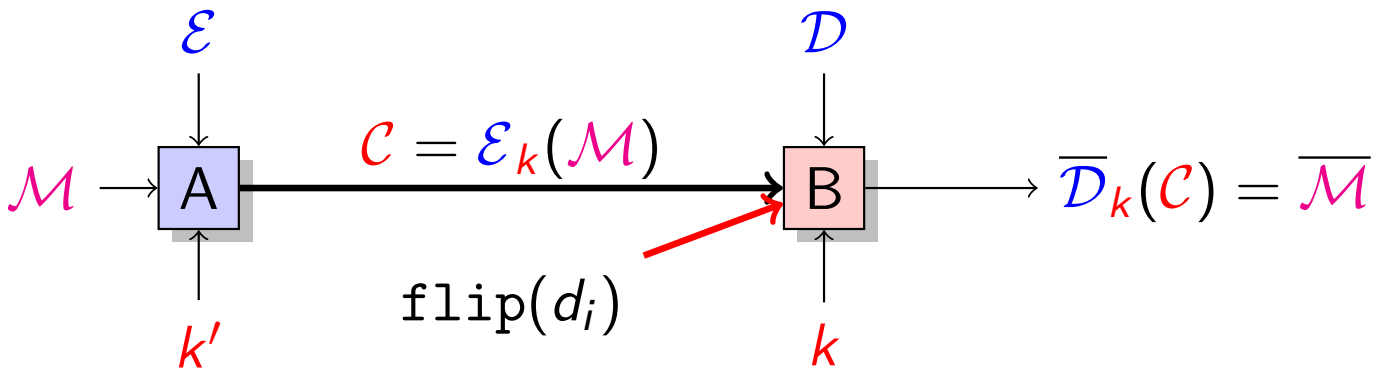
- choose a plaintext message $\mathcal{M}$
- encrypt $\mathcal{M}$ into $\mathcal{C} = \mathcal{E}_k(\mathcal{M})$
- inject a fault by fliping d_i for a random i (d is the secret key)

Fault Attack Example: Bit Flip on RSA Decryption



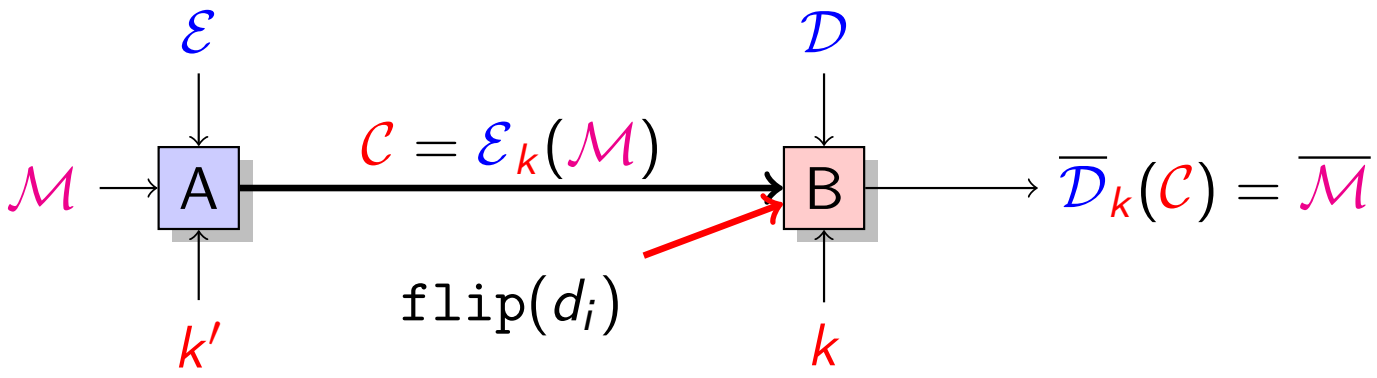
- choose a plaintext message $\mathcal{M}$
- encrypt $\mathcal{M}$ into $\mathcal{C} = \mathcal{E}_k(\mathcal{M})$
- inject a fault by fliping d_i for a random i (d is the secret key)
- compute $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = \frac{c^{2^i \overline{d_i}}}{c^{2^i d_i}}$

Fault Attack Example: Bit Flip on RSA Decryption



- choose a plaintext message $\mathcal{M}$
- encrypt $\mathcal{M}$ into $\mathcal{C} = \mathcal{E}_k(\mathcal{M})$
- inject a fault by fliping d_i for a random i (d is the secret key)
- compute $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = \frac{c^{2^i \overline{d_i}}}{c^{2^i d_i}}$
- test:
  - $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = \frac{1}{c^{2^i}} \bmod N \Longrightarrow d_i = 1$
  - $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = c^{2^i} \bmod N \Longrightarrow d_i = 0$

Fault Attack Example: Bit Flip on RSA Decryption



- choose a plaintext message $\mathcal{M}$
- encrypt $\mathcal{M}$ into $\mathcal{C} = \mathcal{E}_k(\mathcal{M})$
- inject a fault by fliping d_i for a random i (d is the secret key)
- compute $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = \frac{c^{2^i \overline{d_i}}}{c^{2^i d_i}}$
- test:
  - $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = \frac{1}{c^{2^i}} \bmod N \Longrightarrow d_i = 1$
  - $\frac{\overline{\mathcal{M}}}{\mathcal{M}} = c^{2^i} \bmod N \Longrightarrow d_i = 0$
- retry for several i ($\Longrightarrow$ get small parts of d, then mathematical attacks)

Many other fault attacks. . .

Countermeasures

Principles for preventing attacks:

- embed additional protection blocks
- modify the original circuit into a secured version
- application levels: circuit, architecture, algorithm, protocol...

Countermeasures

Principles for preventing attacks:

- embed additional protection blocks
- modify the original circuit into a secured version
- application levels: circuit, architecture, algorithm, protocol...

Countermeasures:

- electrical shielding
- detectors, estimators, decoupling
- use uniform computation durations and power consumption
- use detection/correction codes (for fault injection attacks)
- provide a random behavior (algorithms, representation, operations...)
- add noise (e.g. masking, useless instructions/computations)
- circuit reconfiguration (algorithms, block location, representation of values...)

Low-Level Coding and Circuit Activity

Assumptions:

- b is a bit (i.e. $b \in \{0, 1\}$, logical or mathematical value)
- electrical states for a wire ▬ : V_{DD} (logical 1) or GND (logical 0)

Low-Level Coding and Circuit Activity

Assumptions:

- b is a bit (i.e. $b \in \{0, 1\}$, logical or mathematical value)
- electrical states for a wire ▬ : V_{DD} (logical 1) or GND (logical 0)

Low-level codings of a bit:

	$b = 0$	$b = 1$
standard	▬ GND	▬ V_{DD}

Low-Level Coding and Circuit Activity

Assumptions:

- b is a bit (i.e. $b \in \{0, 1\}$, logical or mathematical value)
- electrical states for a wire : V_{DD} (logical 1) or GND (logical 0)

Low-level codings of a bit:

	$b = 0$	$b = 1$
standard	GND	V_{DD}
dual rail	$r_0 = V_{\mathrm{DD}}$, $r_1 = \mathrm{GND}$ $(1, 0)_{\mathrm{DR}}$	$r_0 = \mathrm{GND}$, $r_1 = V_{\mathrm{DD}}$ $(0, 1)_{\mathrm{DR}}$

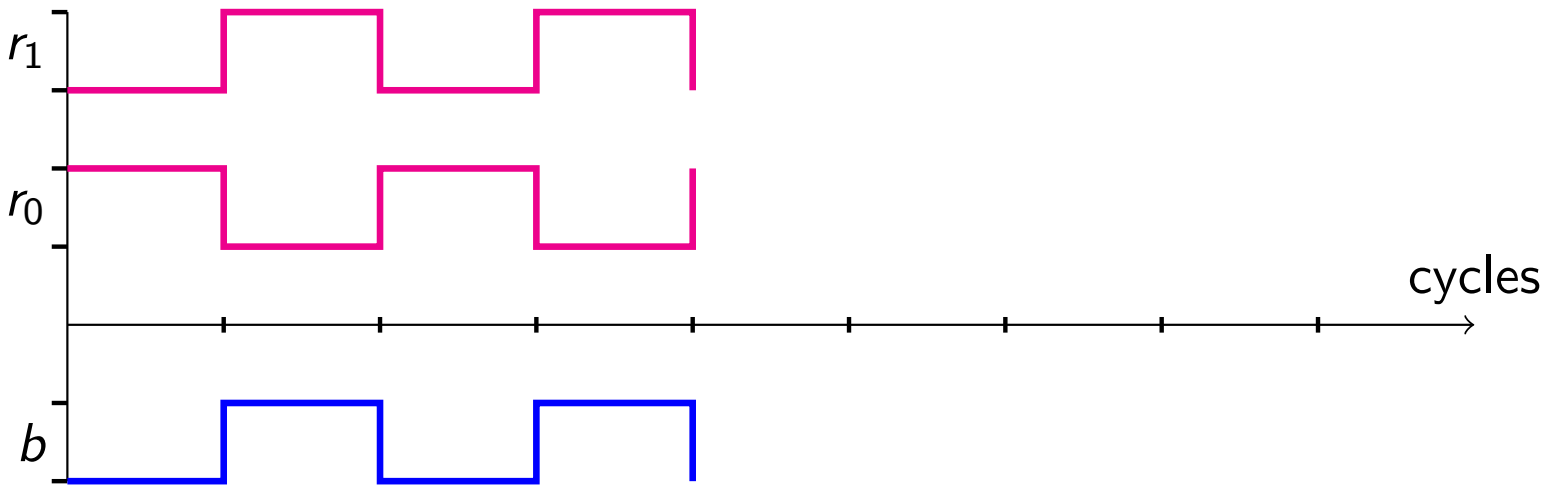
Low-Level Coding and Circuit Activity

Assumptions:

- b is a bit (i.e. $b \in \{0, 1\}$, logical or mathematical value)
- electrical states for a wire : V_{DD} (logical 1) or GND (logical 0)

Low-level codings of a bit:

	$b = 0$	$b = 1$
standard	GND	V_{DD}
dual rail	$r_0 = V_{\mathrm{DD}}$, $r_1 = \mathrm{GND}$ $(1, 0)_{\mathrm{DR}}$	$r_0 = \mathrm{GND}$, $r_1 = V_{\mathrm{DD}}$ $(0, 1)_{\mathrm{DR}}$



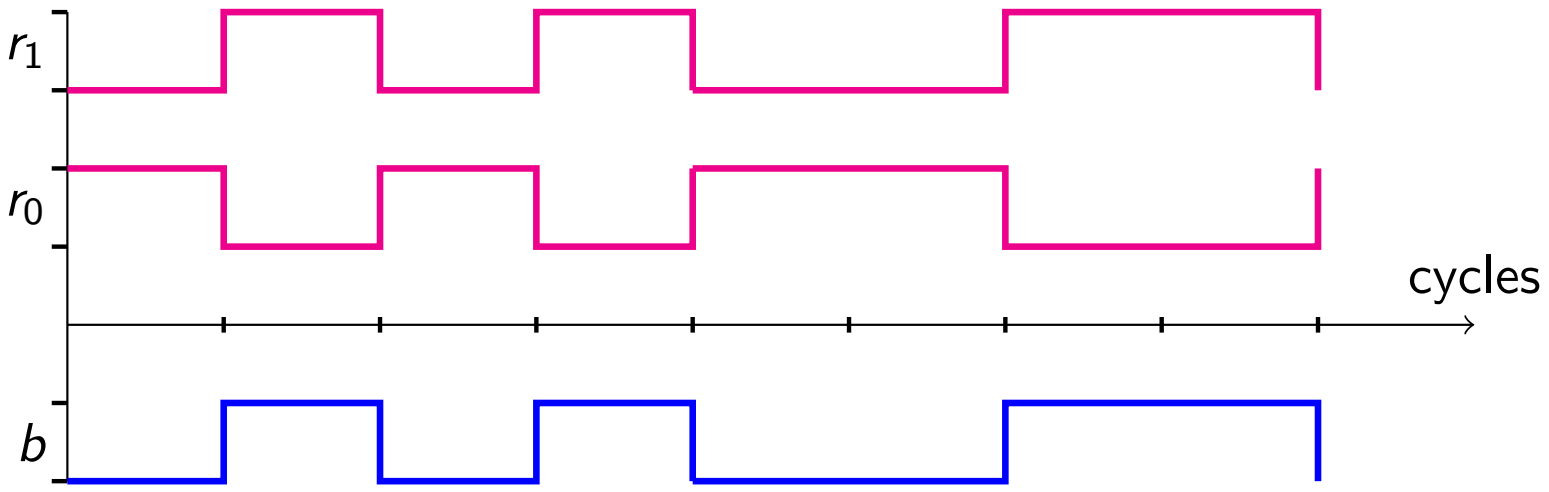
Low-Level Coding and Circuit Activity

Assumptions:

- b is a bit (i.e. $b \in \{0, 1\}$, logical or mathematical value)
- electrical states for a wire : V_{DD} (logical 1) or GND (logical 0)

Low-level codings of a bit:

	$b = 0$	$b = 1$
standard	GND	V_{DD}
dual rail	$r_0 = V_{\mathrm{DD}}$, $r_1 = \mathrm{GND}$ $(1, 0)_{\mathrm{DR}}$	$r_0 = \mathrm{GND}$, $r_1 = V_{\mathrm{DD}}$ $(0, 1)_{\mathrm{DR}}$



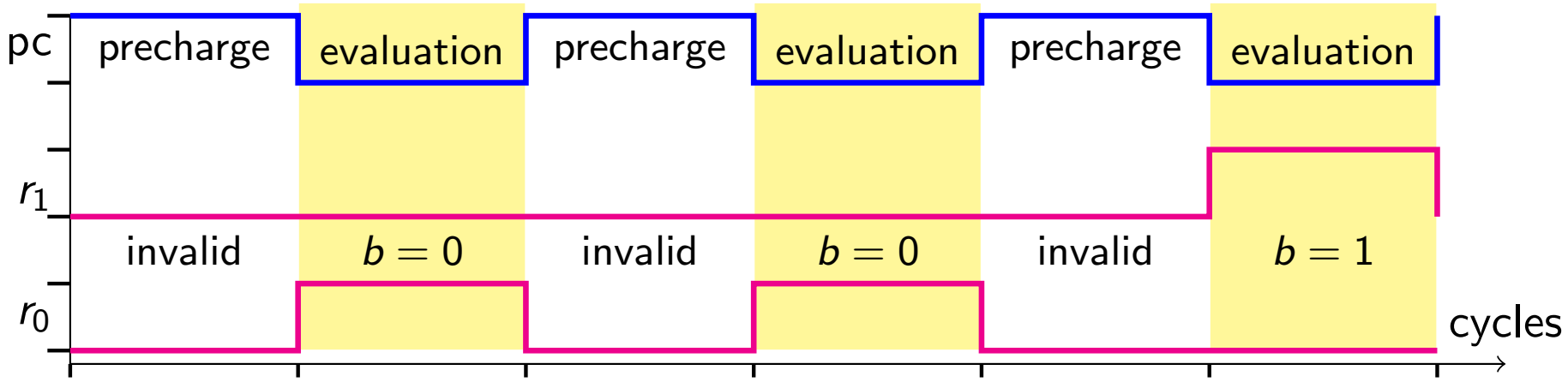
Circuit Logic Styles

Countermeasure principles: uniformize circuit activity and exclusive coding

Circuit Logic Styles

Countermeasure principles: uniformize circuit activity and exclusive coding

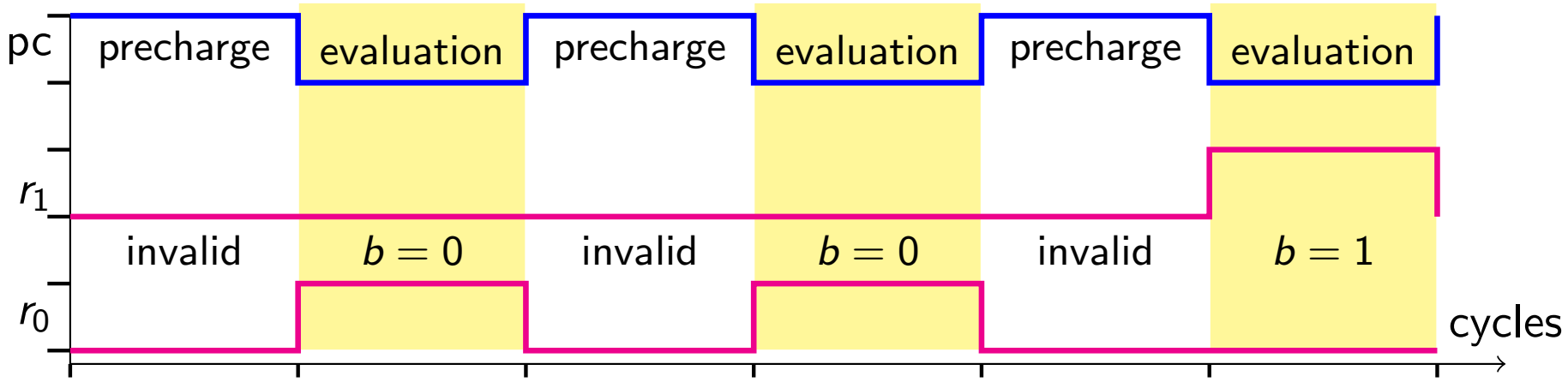
Solution based on precharge logic and dual-rail coding:



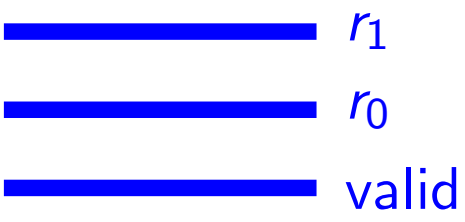
Circuit Logic Styles

Countermeasure principles: uniformize circuit activity and exclusive coding

Solution based on precharge logic and dual-rail coding:

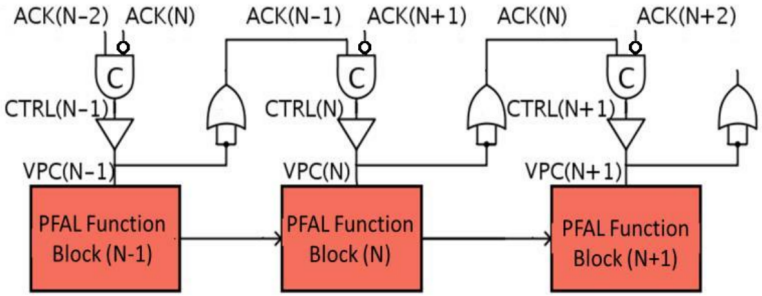


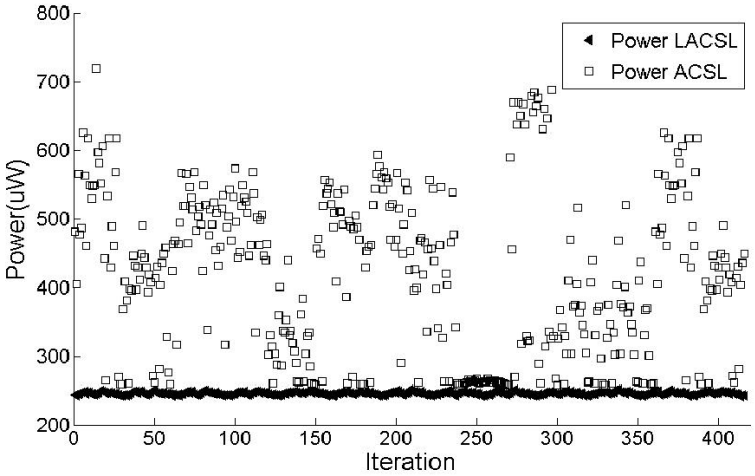
Solution based on validity line and dual-rail coding:

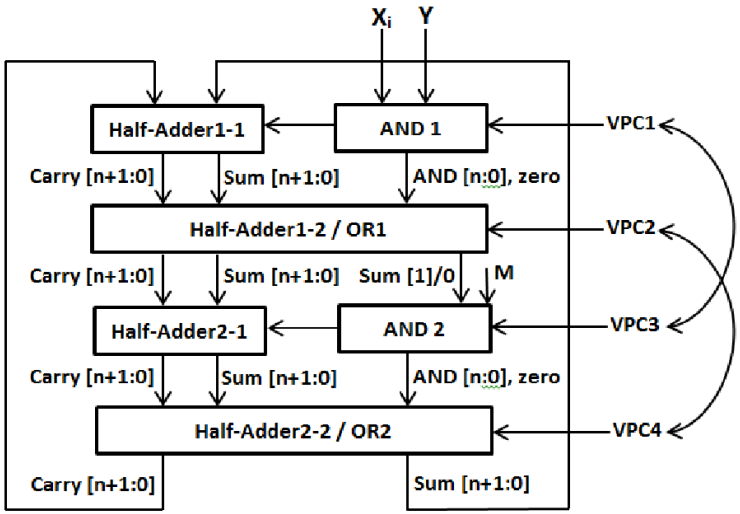


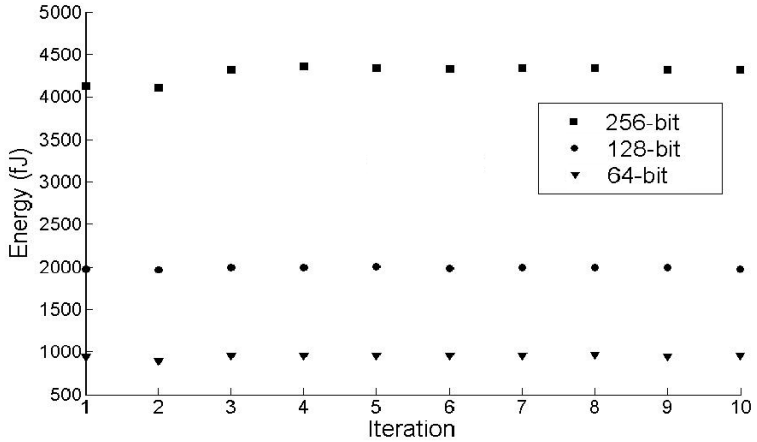
Important overhead: silicon area and local storage (registers)

Circuit-Level Protections for Arithmetic Operators







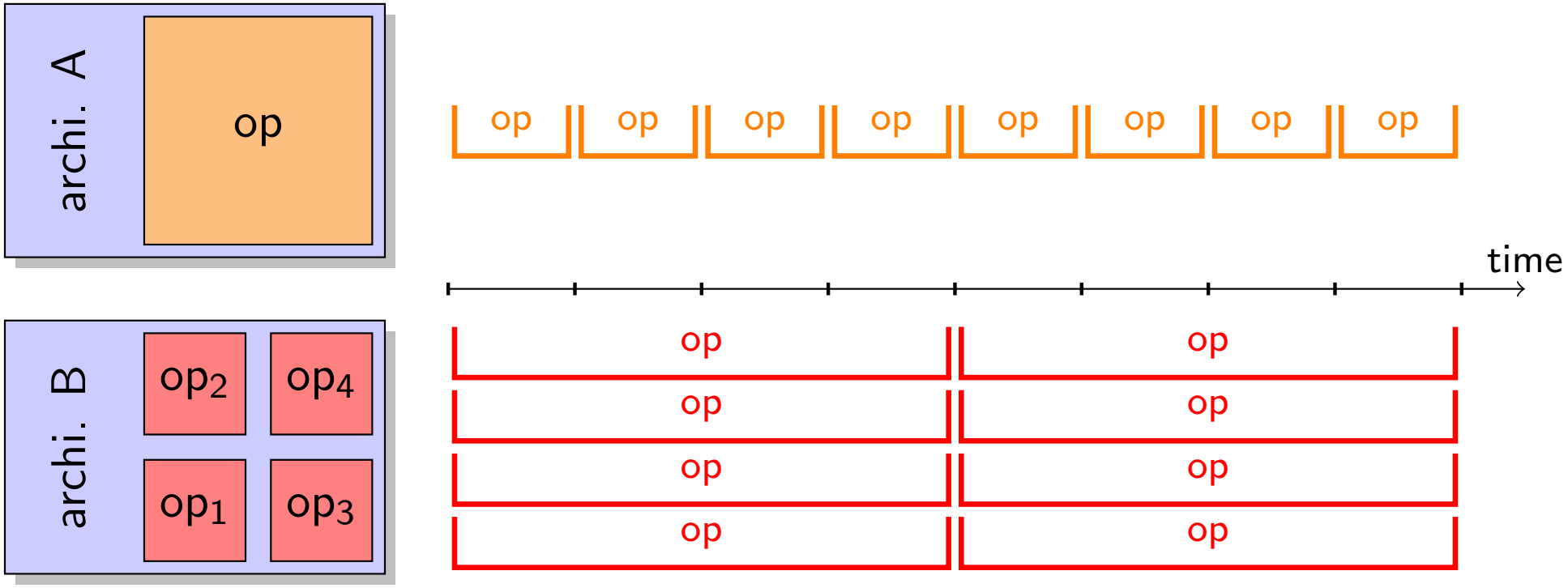


References: [8] and [9]

Countermeasure: Architecture

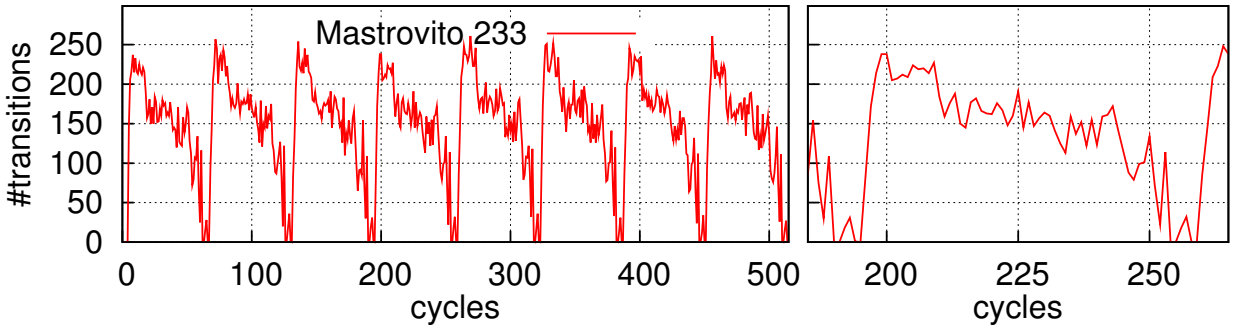
Increase internal parallelism:

- replace one fast but big operator
- by several instances of a small but slow one



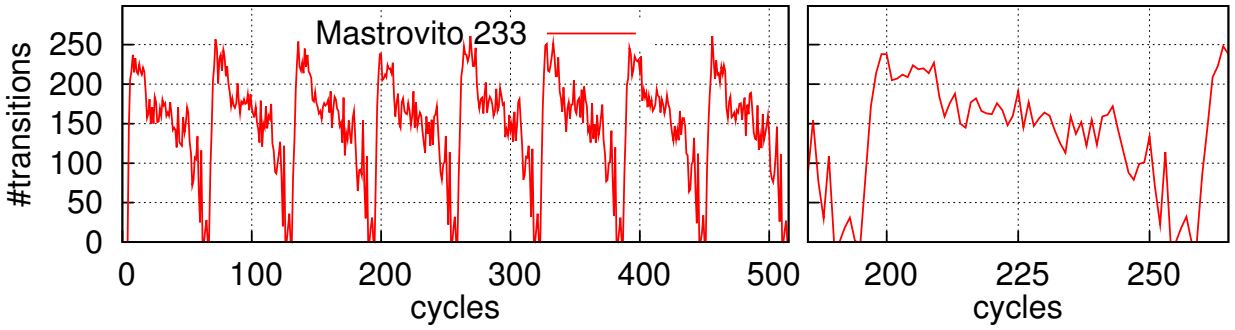
Protected Multipliers

Unprotected



Protected Multipliers

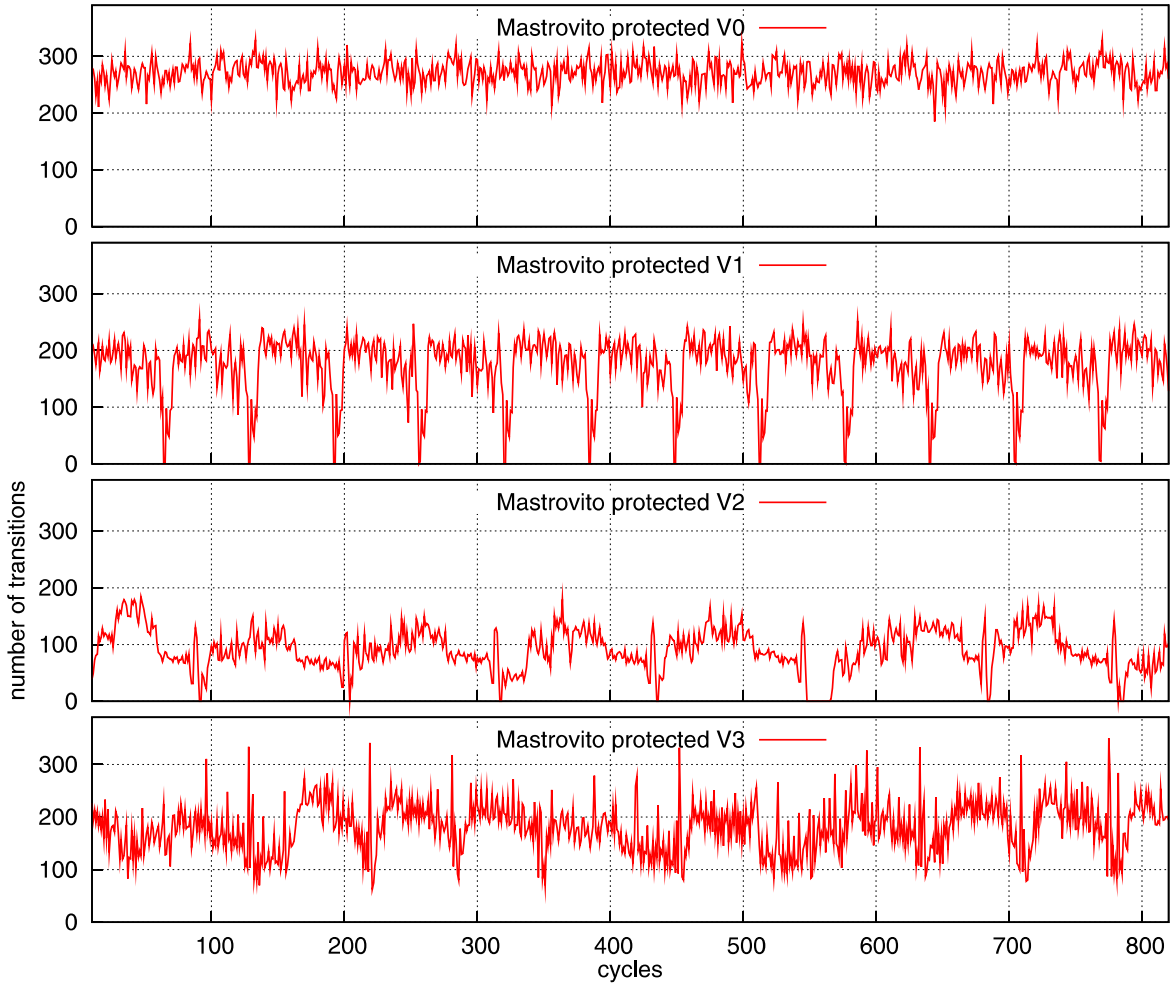
Unprotected



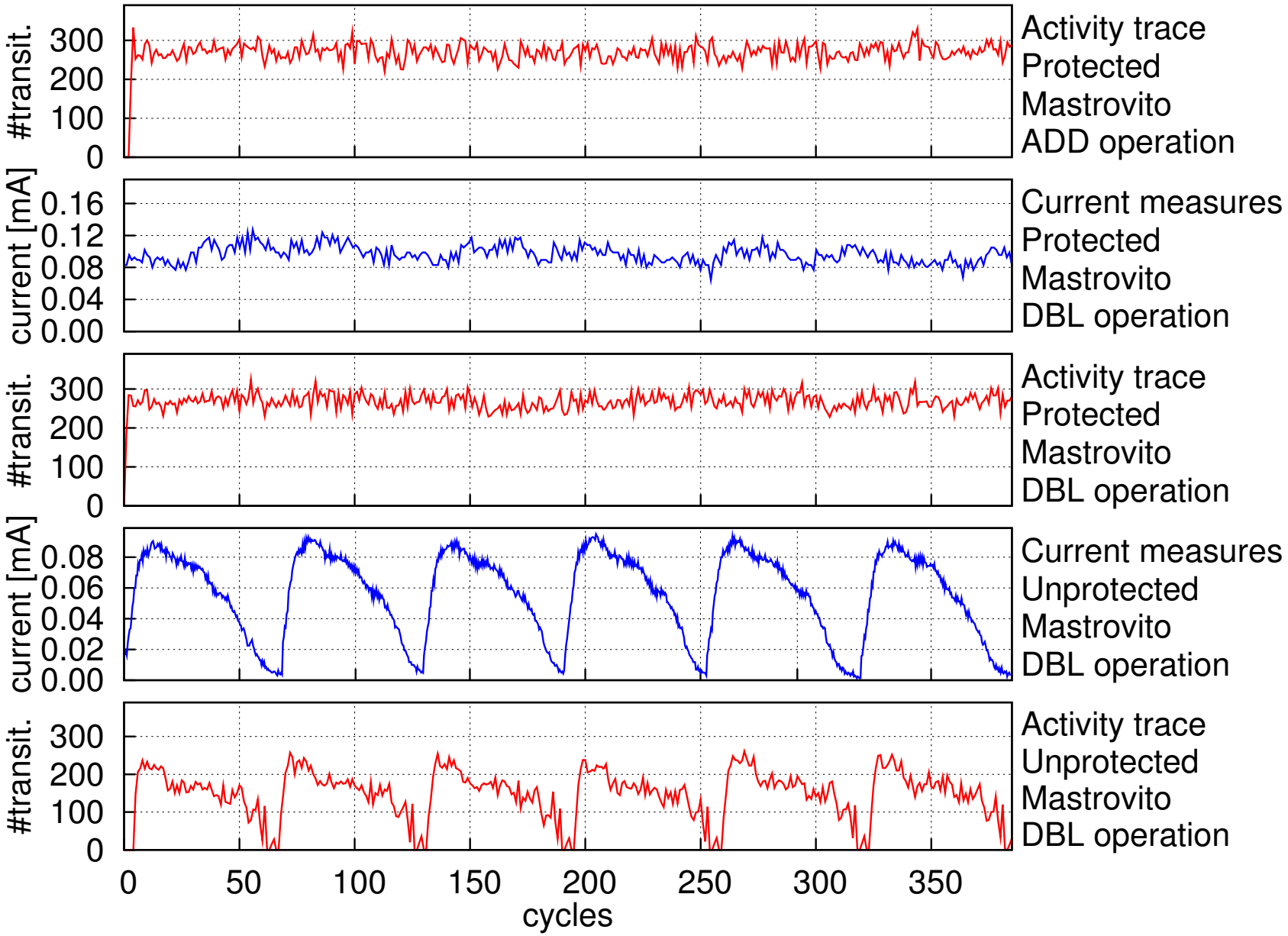
Protected

Overhead:
Area/time $< 10\,\%$

References:
PhD D. Pamula [13]
Articles: [16], [15], [14]



Protected ECC Accelerator

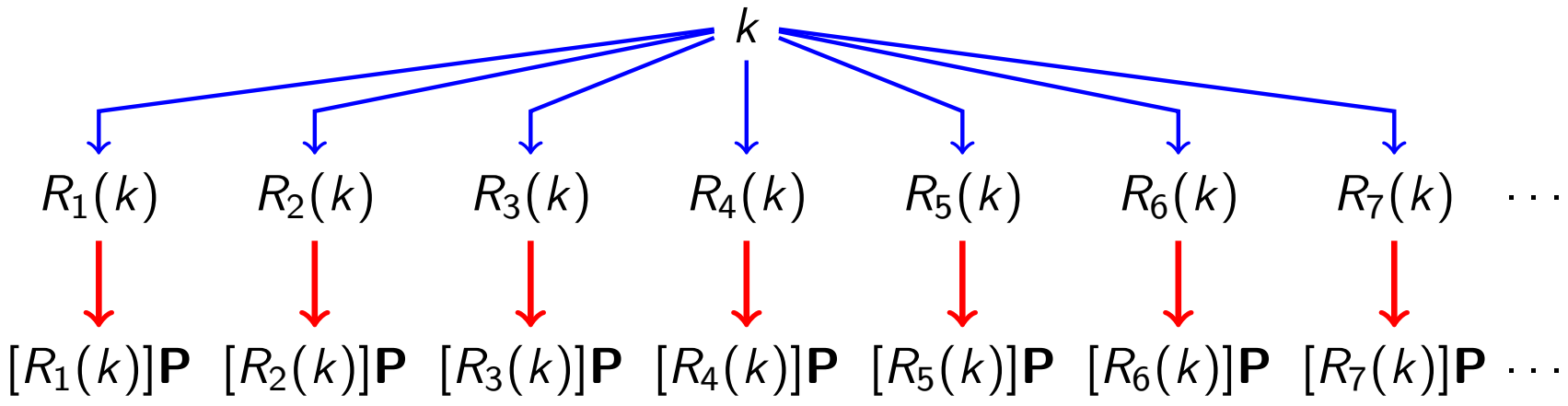


Warning: old dedicated accelerator (similar behavior is expected for our new one)

Arithmetic Level Countermeasures

Redundant number system =

- a way to improve the performance of some operations
- a way to represent a value with different representations



Important property: $\forall i \quad [R_i(k)]\mathbf{P} = [k]\mathbf{P}$

Proposed solution: use random redundant representations of k

Double-Base Number System

Standard radix-2 representation:

$$k = \sum_{i=0}^{t-1} k_i 2^i = \boxed{k_{t-1}}\boxed{k_{t-2}}\boxed{\cdots}\boxed{k_2}\boxed{k_1}\boxed{k_0} \quad t \text{ explicit digits}$$

Double-Base Number System

Standard radix-2 representation:

$$k = \sum_{i=0}^{t-1} k_i 2^i =$$

2^{t-1}	2^{t-2}	$\cdots$	2^2	2^1	2^0	implicit weights
k_{t-1}	k_{t-2}	$\cdots$	k_2	k_1	k_0	t explicit digits

Digits: $k_i \in \{0, 1\}$, typical size: $t \in \{160, \ldots, 600\}$

Double-Base Number System

Standard radix-2 representation:

$$k = \sum_{i=0}^{t-1} k_i 2^i =$$

2^{t-1}	2^{t-2}	$\cdots$	2^2	2^1	2^0	implicit weights
k_{t-1}	k_{t-2}	$\cdots$	k_2	k_1	k_0	t explicit digits

Digits: $k_i \in \{0, 1\}$, typical size: $t \in \{160, \ldots, 600\}$

Double-Base Number System (DBNS):

$$k = \sum_{j=0}^{n-1} k_j 2^{a_j} 3^{b_j} =$$

Double-Base Number System

Standard radix-2 representation:

$$k = \sum_{i=0}^{t-1} k_i 2^i =$$

2^{t-1}	2^{t-2}	$\cdots$	2^2	2^1	2^0	implicit weights
k_{t-1}	k_{t-2}	$\cdots$	k_2	k_1	k_0	t explicit digits

Digits: $k_i \in \{0, 1\}$, typical size: $t \in \{160, \ldots, 600\}$

Double-Base Number System (DBNS):

$$k = \sum_{j=0}^{n-1} k_j 2^{a_j} 3^{b_j} =$$

k_{n-1}	$\cdots$	k_1	k_0	n $(2,3)-$terms
a_{n-1}	$\cdots$	a_1	a_0	explicit "digits"
b_{n-1}	$\cdots$	b_1	b_0	explicit ranks

$a_j, b_j \in \mathbb{N}$, $k_j \in \{1\}$ or $k_j \in \{-1, 1\}$, size $n \approx \log t$

Double-Base Number System

Standard radix-2 representation:

$$k = \sum_{i=0}^{t-1} k_i 2^i =$$

2^{t-1}	2^{t-2}	$\cdots$	2^2	2^1	2^0	implicit weights
k_{t-1}	k_{t-2}	$\cdots$	k_2	k_1	k_0	t explicit digits

Digits: $k_i \in \{0, 1\}$, typical size: $t \in \{160, \ldots, 600\}$

Double-Base Number System (DBNS):

$$k = \sum_{j=0}^{n-1} k_j 2^{a_j} 3^{b_j} =$$

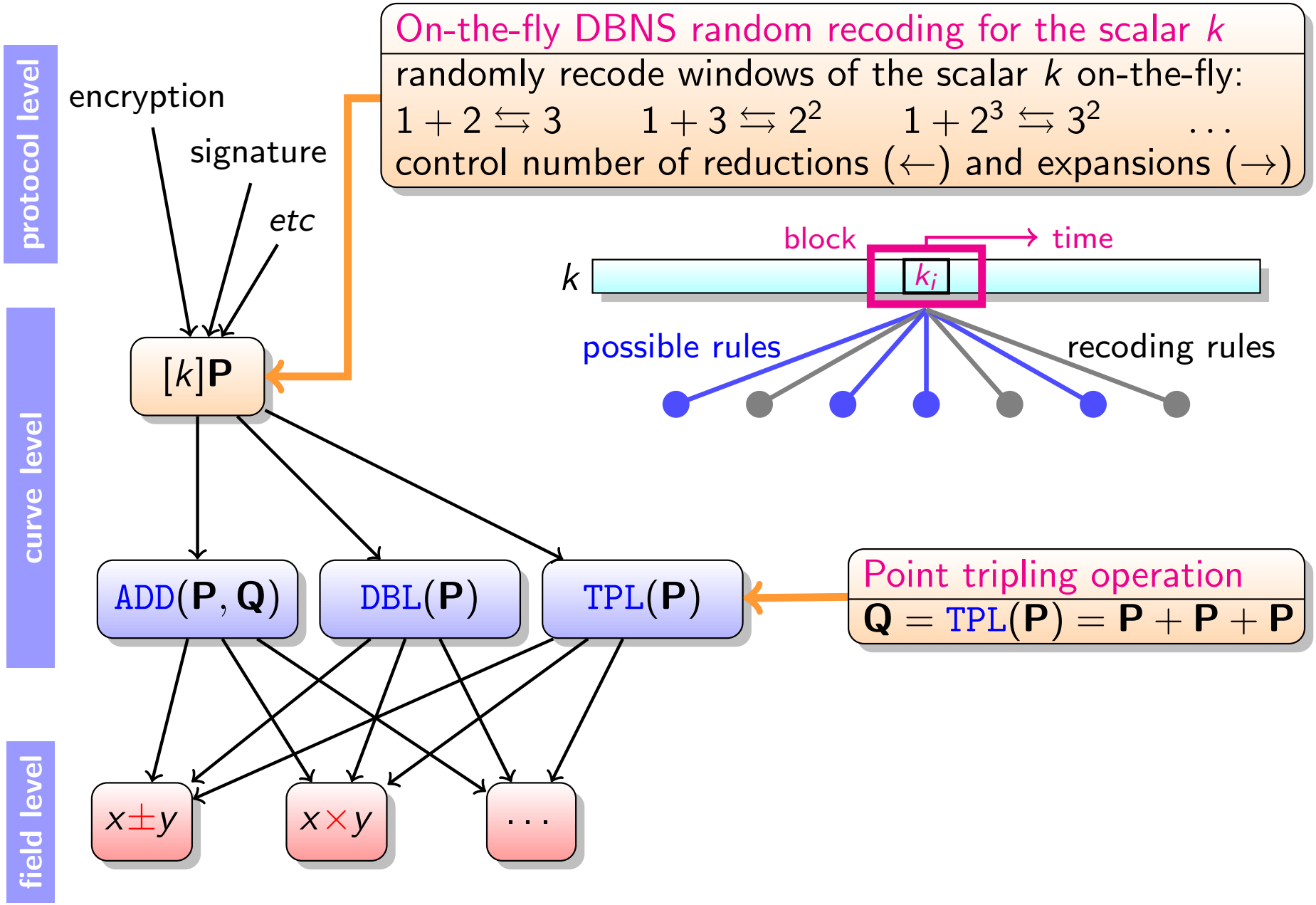
k_{n-1}	$\cdots$	k_1	k_0	n $(2,3)-$terms
a_{n-1}	$\cdots$	a_1	a_0	explicit "digits"
b_{n-1}	$\cdots$	b_1	b_0	explicit ranks

$a_j, b_j \in \mathbb{N}$, $k_j \in \{1\}$ or $k_j \in \{-1, 1\}$, size $n \approx \log t$

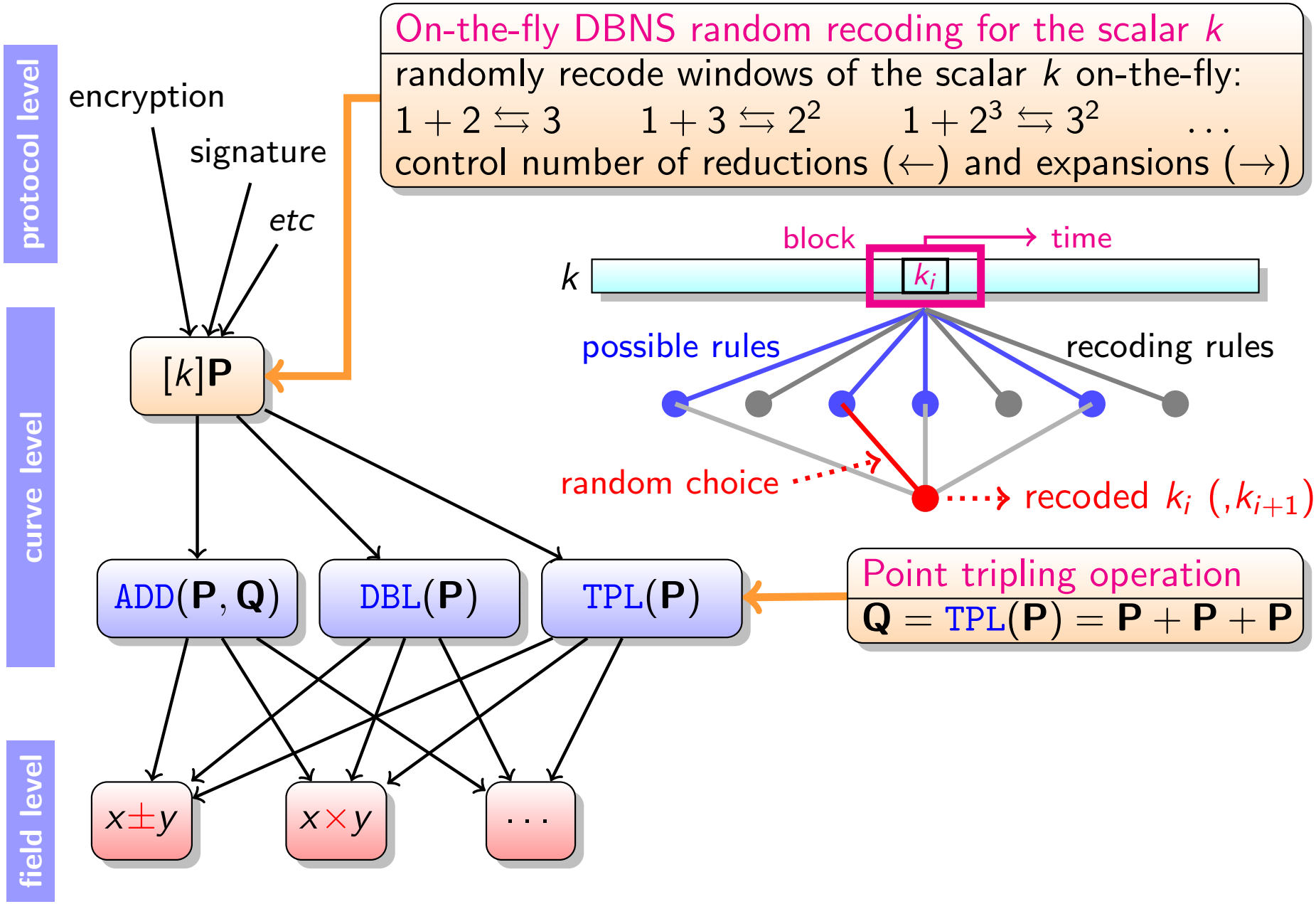
DBNS is a very redundant and sparse representation: $1701 = (11010100101)_2$

$$\begin{aligned}
1701 &= 243 + 1458 &= 2^0 3^5 + 2^1 3^6 &= (1, 0, 5), (1, 1, 6) \\
&= 1728 - 27 &= 2^6 3^3 - 2^0 3^3 &= (1, 6, 3), (-1, 0, 3) \\
&= 729 + 972 &= 2^0 3^6 + 2^2 3^5 &= (1, 0, 6), (1, 2, 5) \\
&\ldots
\end{aligned}$$

Randomized DBNS Recoding of the Scalar k



Randomized DBNS Recoding of the Scalar k



Randomized DBNS Recoding of the Scalar k

protocol level

encryption

signature

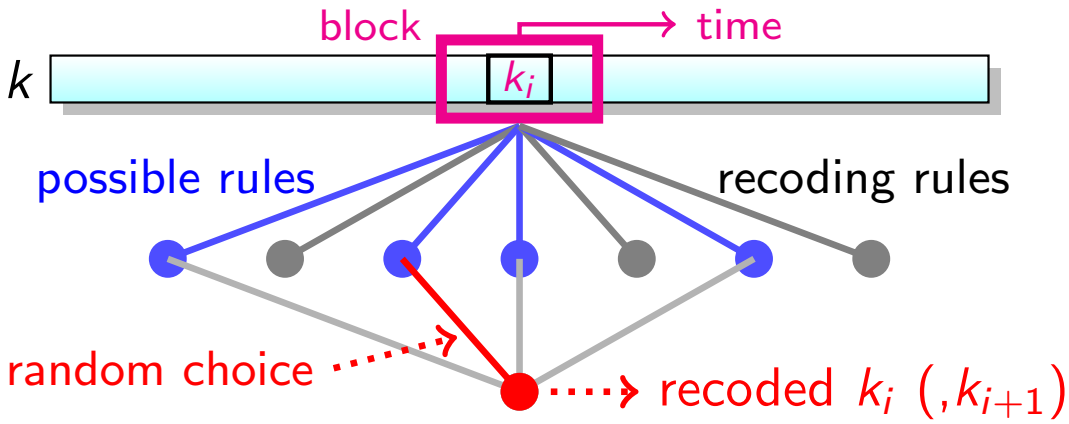
etc

On-the-fly DBNS random recoding for the scalar k

randomly recode windows of the scalar k on-the-fly:

$1+2 \leftrightarrows 3 \qquad 1+3 \leftrightarrows 2^2 \qquad 1+2^3 \leftrightarrows 3^2 \qquad \ldots$

control number of reductions ($\leftarrow$) and expansions ($\rightarrow$)



curve level

$[k]\mathbf{P}$

$\mathtt{ADD}(\mathbf{P},\mathbf{Q})$ $\mathtt{DBL}(\mathbf{P})$ $\mathtt{TPL}(\mathbf{P})$

Point tripling operation

$\mathbf{Q} = \mathtt{TPL}(\mathbf{P}) = \mathbf{P}+\mathbf{P}+\mathbf{P}$

field level

$x \pm y$ $x \times y$ $\ldots$

DBNS is redundant $\Rightarrow$ security $\nearrow$

DBNS is sparse $\Rightarrow$ 20–30 % speed $\nearrow$

Ref: [7] Chabrier, Pamula & Tisserand. Asilomar 2009

Hardware Implementation of RNS for ECC (1/2)

RNS: Residue Number System

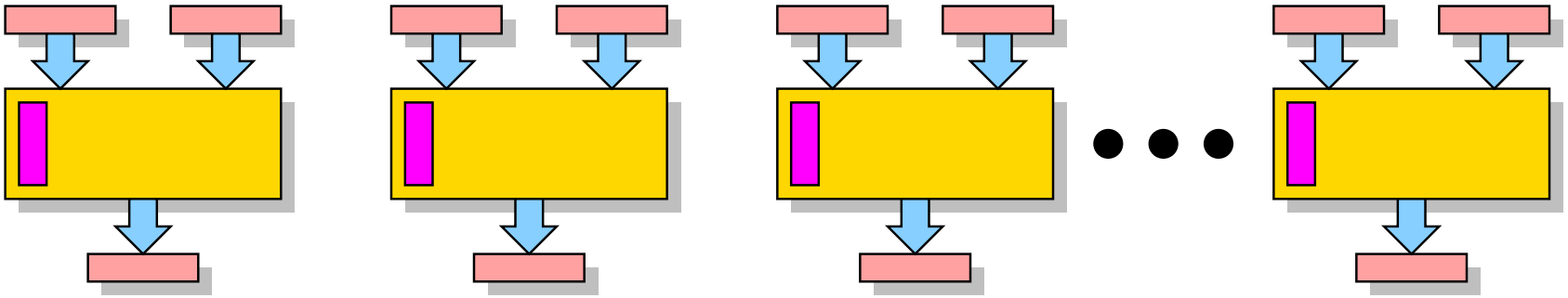
- Base $\mathcal{B} = (m_1, m_2, \ldots, m_k)$ of k relatively prime moduli
- Size of the base: k

$$A = \{a_1, a_2, \ldots, a_k\}, \quad \forall i \ a_i = A \bmod m_i$$

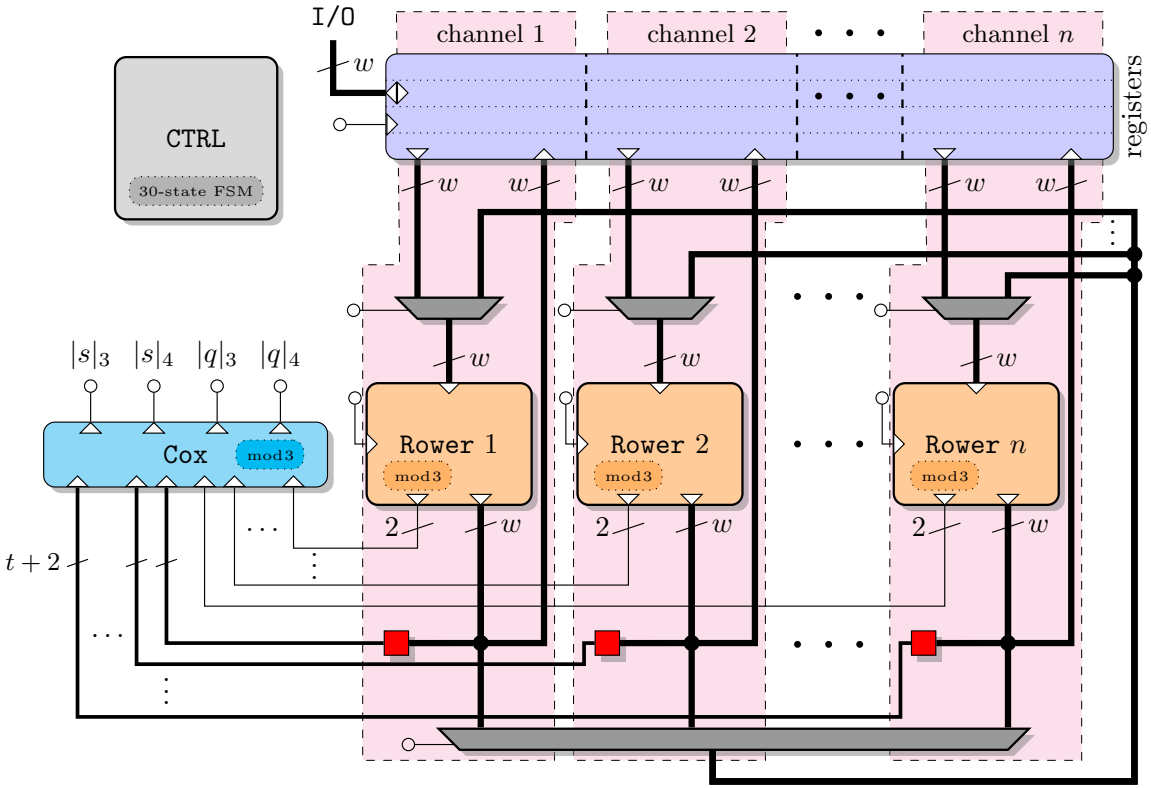
Operations:

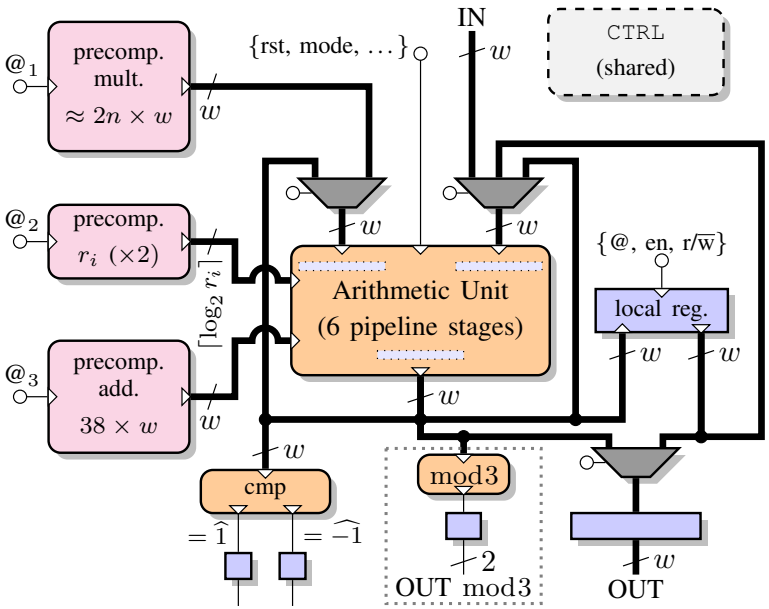
$$A \pm B = (|a_1 \pm b_1|_{m_1}, \ldots, |a_k \pm b_k|_{m_k})$$

$$A \times B = (|a_1 \times b_1|_{m_1}, \ldots, |a_k \times b_k|_{m_k})$$



Hardware Implementation of RNS for ECC (2/2)

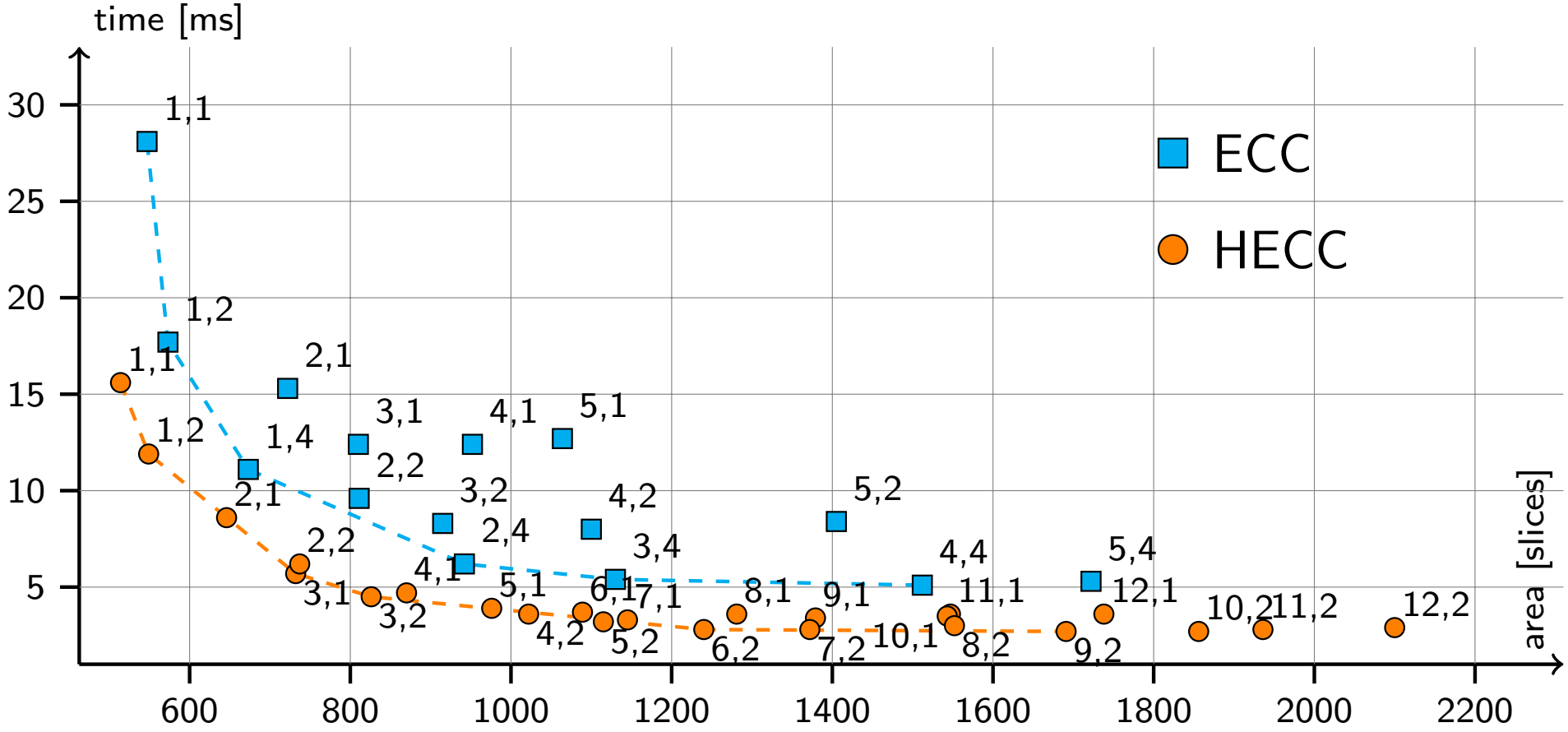




Optimized algorithms and implementations for $\mathrm{GF}(p)$ operations:

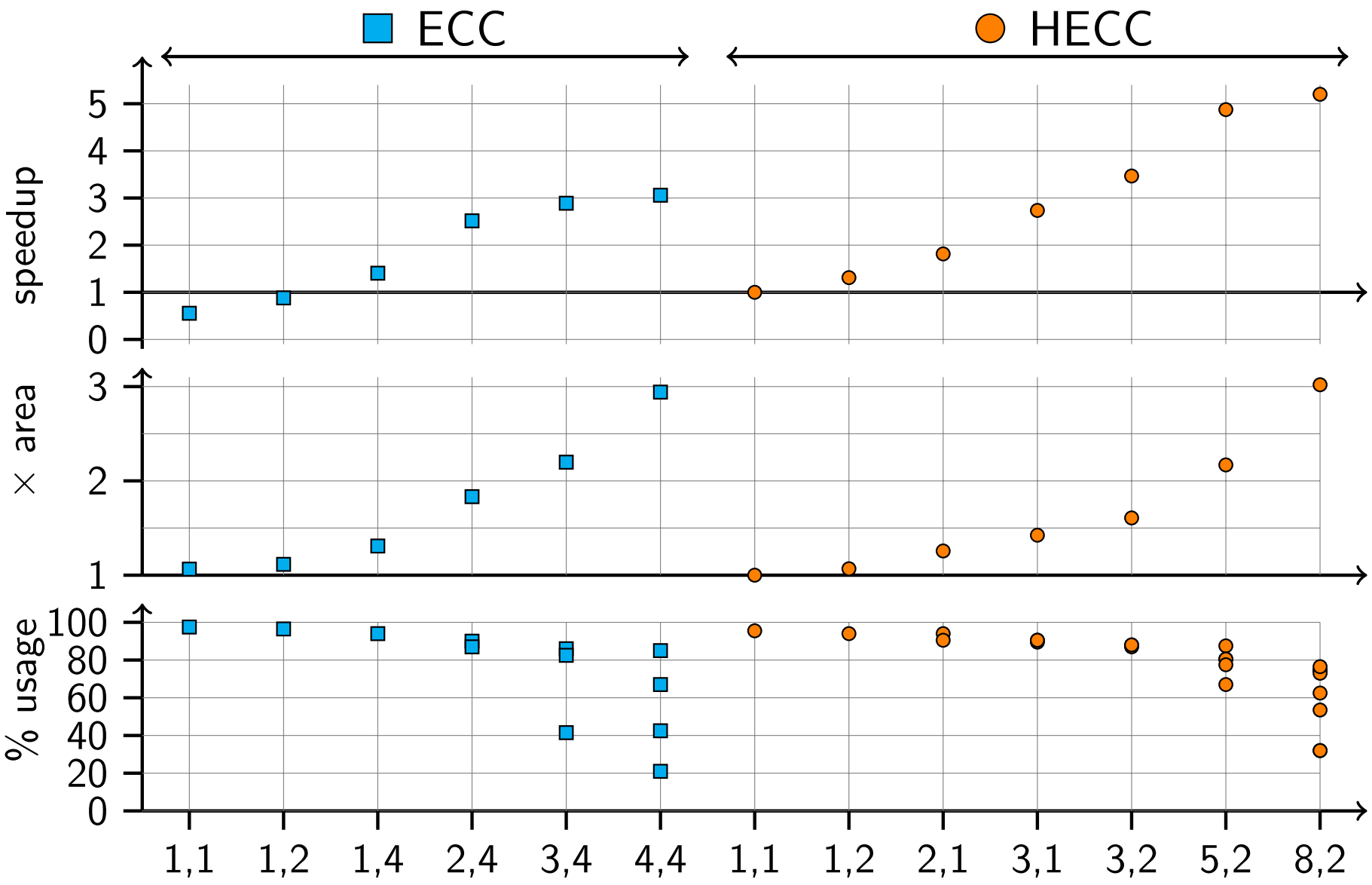
- fast operations: inversion [3], modular multiplication [5], patterns [4]
- PhD Thesis Karim Bigou [2]
- hybrid positio-residues (HPR) representation [6]

Comparison ECC 256 vs HECC 128 (1/2)



On average HECC is 40 % faster than ECC for a similar silicon cost

Comparison ECC 256 vs HECC 128 (2/2)



Conclusion

- Side channel and fault attacks are serious threats
- Attacks are more and more efficient (many variants)
- Security analysis is mandatory at all levels (specification, algorithm, operation, implementation)
- Security = trade-off between performances, robustness and cost
- Security = *func*(secret value, attacker capabilities)
- security = computer science + microelectronics + mathematics

Conclusion

- Side channel and fault attacks are serious threats
- Attacks are more and more efficient (many variants)
- Security analysis is mandatory at all levels (specification, algorithm, operation, implementation)
- Security = trade-off between performances, robustness and cost
- Security = *func*(secret value, attacker capabilities)
- security = computer science + microelectronics + mathematics

Current works examples:

- Methods/tools for automating security analysis
- Circuit reconfiguration (representations, algorithms)
- Circuits with reduced activity variations
- Representation of numbers with error detection/correction "codes"
- Design space exploration
- CAD tools with security improvement capabilities

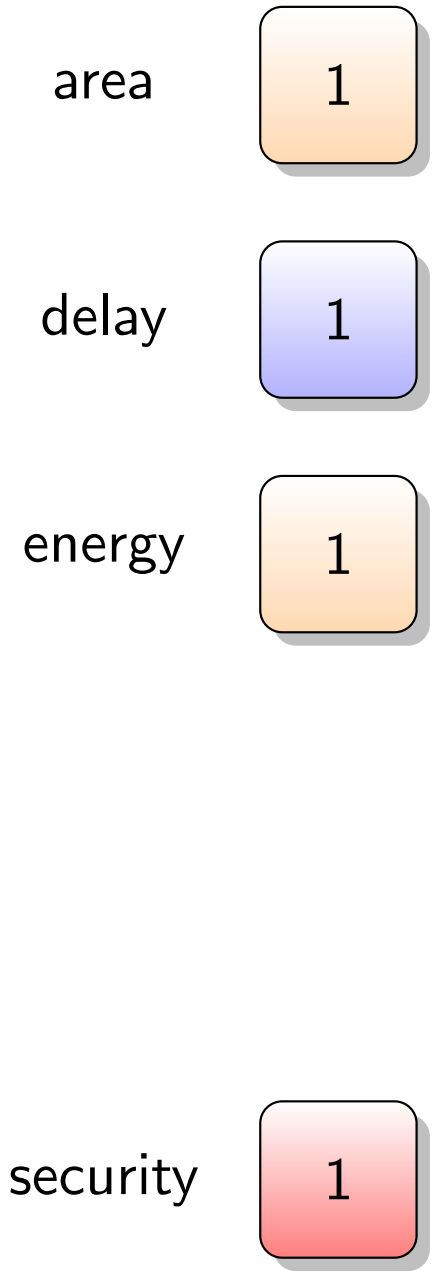
Our Long Term Objectives

Study the links between:

- cryptosystems
- arithmetic algorithms
- $\mathbb{F}_q$, pts representations
- architectures & units
- circuit optimisations

to ensure

- high security against
  - theoretical attacks
  - physical attacks
- low design cost
- low silicon cost
- low energy(/power)
- high performances
- high flexibility



Our Long Term Objectives

Study the links between:

- cryptosystems
- arithmetic algorithms
- $\mathbb{F}_q$, pts representations
- architectures & units
- circuit optimisations

to ensure

- high security against
  - theoretical attacks
  - physical attacks
- low design cost
- low silicon cost
- low energy(/power)
- high performances
- high flexibility

area $1 \longrightarrow 1 + a$

delay $1 \longrightarrow 1 + t$

energy $1 \longrightarrow 1 + e$

$a, t, e \in 0\%, 5\%, 10\%, \ldots, 100\%$

security 1

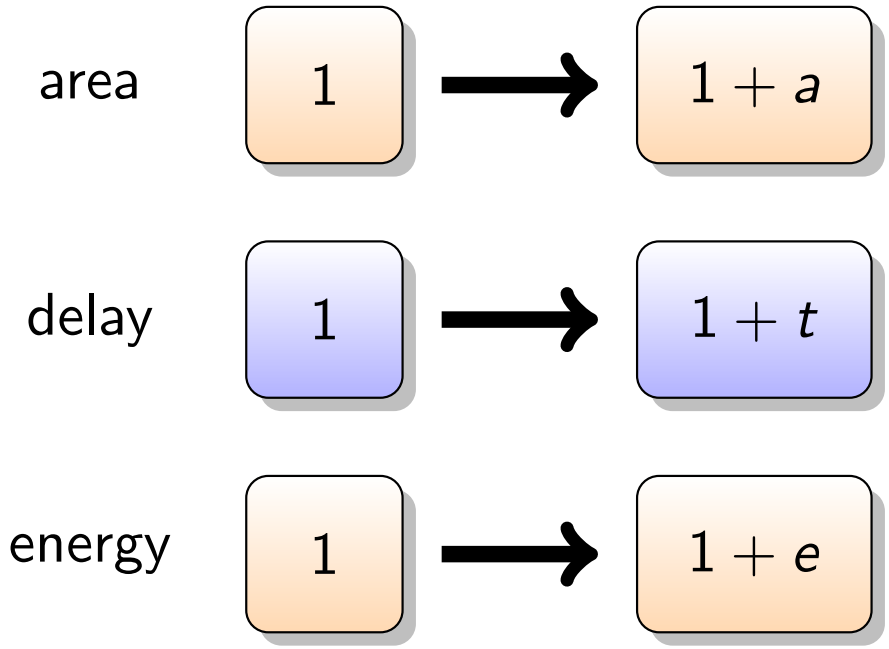
Our Long Term Objectives

Study the links between:

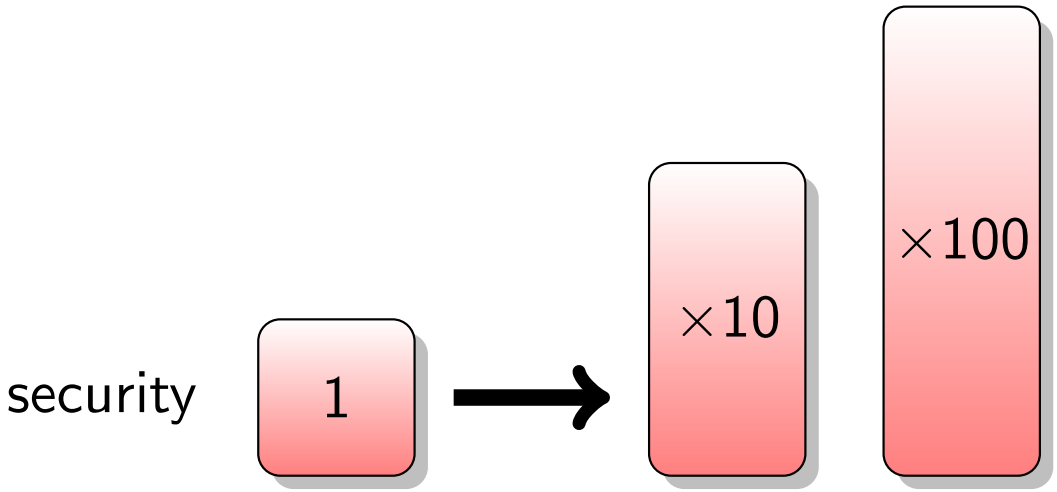
- cryptosystems
- arithmetic algorithms
- $\mathbb{F}_q$, pts representations
- architectures & units
- circuit optimisations

to ensure

- high security against
  - theoretical attacks
  - physical attacks
- low design cost
- low silicon cost
- low energy(/power)
- high performances
- high flexibility



$a, t, e \in 0\%, 5\%, 10\%, \ldots, 100\%$



References I

[1] J. Balasch, B. Gierlichs, and I. Verbauwhede.
An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs.
In *Proc. 8th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 105–114, Nara, Japan, September 2011. IEEE.

[2] K. Bigou.
Étude théorique et implantation matérielle d'unités de calcul en représentation modulaire des nombres pour la cryptographie sur courbes elliptiques.
Phd thesis, University Rennes 1, Lannion, France, November 2014.

[3] K. Bigou and A. Tisserand.
Improving modular inversion in RNS using the plus-minus method.
In G. Bertoni and J.-S. Coron, editors, *Proc. 15th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 8086 of *LNCS*, pages 233–249, Santa Barbara, CA, USA, August 2013. Springer.

[4] K. Bigou and A. Tisserand.
RNS modular multiplication through reduced base extensions.
In H. Fu and D. Thomas, editors, *Proc. 25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 57–62, Zurich, Switzerland, June 2014. IEEE.

[5] K. Bigou and A. Tisserand.
Single base modular multiplication for efficient hardware RNS implementations of ECC.
In T. Guneysu and H. Handschuh, editors, *Proc. 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 9293 of *LNCS*, pages 123–140, Saint-Malo, France, September 2015. Springer.

[6] K. Bigou and A. Tisserand.
Hybrid position-residues number system.
In J. Hormigo, S. Oberman, and N. Revol, editors, *Proc. 23rd Symposium on Computer Arithmetic (ARITH)*, pages 126–133, Santa Clara, CA, U.S.A, July 2016. IEEE Computer Society.

[7] T. Chabrier, D. Pamula, and A. Tisserand.
Hardware implementation of DBNS recoding for ECC processor.
In *Proc. 44rd Asilomar Conference on Signals, Systems and Computers*, pages 1129–1133, Pacific Grove, California, U.S.A., November 2010. IEEE.

References II

[8] J. Chen, A. Tisserand, E. M. Popovici, and S. Cotofana.
Robust sub-powered asynchronous logic.
In J. Becker and M. R. Adrover, editors, *Proc. 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–7, Palma de Mallorca, Spain, September 2014. IEEE.

[9] J. Chen, A. Tisserand, E. M. Popovici, and S. Cotofana.
Asynchronous charge sharing power consistent Montgomery multiplier.
In J. Sparso and E Yahya, editors, *Proc. 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 132–138, Mountain View, California, USA, May 2015.

[10] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh.
An on-chip glitchy-clock generator for testing fault injection attacks.
Journal of Cryptographic Engineering, 1(4):265–270, December 2011.

[11] P. C. Kocher, J. Jaffe, and B. Jun.
Differential power analysis.
In *Proc. Advances in Cryptology (CRYPTO)*, volume 1666 of *LNCS*, pages 388–397. Springer, August 1999.

[12] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz.
Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller.
In *Proc. 10th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 77–88, Santa Barbara, CA, USA, August 2013. IEEE.

[13] D. Pamula.
Arithmetic Operators on GF(2^m) for Cryptographic Applications: Performance - Power Consumption - Security Tradeoffs.
Phd thesis, University of Rennes 1 and Silesian University of Technology, December 2012.

[14] D. Pamula, E. Hrynkiewicz, and A. Tisserand.
Analysis of GF(2^{233}) multipliers regarding elliptic curve cryptosystem applications.
In *11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDeS)*, pages 271–276, Brno, Czech Republic, May 2012.

References III

[15] D. Pamula and A. Tisserand.
GF(2^m) finite-field multipliers with reduced activity variations.
In *4th International Workshop on the Arithmetic of Finite Fields*, volume 7369 of *LNCS*, pages 152–167, Bochum, Germany, July 2012. Springer.

[16] D. Pamula and A. Tisserand.
Fast and secure finite field multipliers.
In *Proc. 18th Euromicro Conference on Digital System Design (DSD)*, pages 653–660, Madeira, Portugal, August 2015.

[17] R. L. Rivest, A. Shamir, and L. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Communications of the ACM, 21(2):120–126, February 1978.

[18] J. Schmidt and C. Herbst.
A practical fault attack on square and multiply.
In *Proc. 5th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 53–58, Washington, DC, USA, August 2008. IEEE.

[19] N. Selmane, S. Guilley, and J.-L. Danger.
Practical setup time violation attacks on AES.
In *Proc. 7th European Dependable Computing Conference (EDCC)*, Kaunas, Lithuania, 2008.

The end, questions ?

Contact:

- mailto:arnaud.tisserand@univ-ubs.fr
- http://www-labsticc.univ-ubs.fr/~tisseran
- CNRS, Lab-STICC Laboratory
  University South Brittany (UBS),
  Centre de recherche C. Huygens, rue St Maudé, BP 92116,
  56321 Lorient cedex, France

Thank you