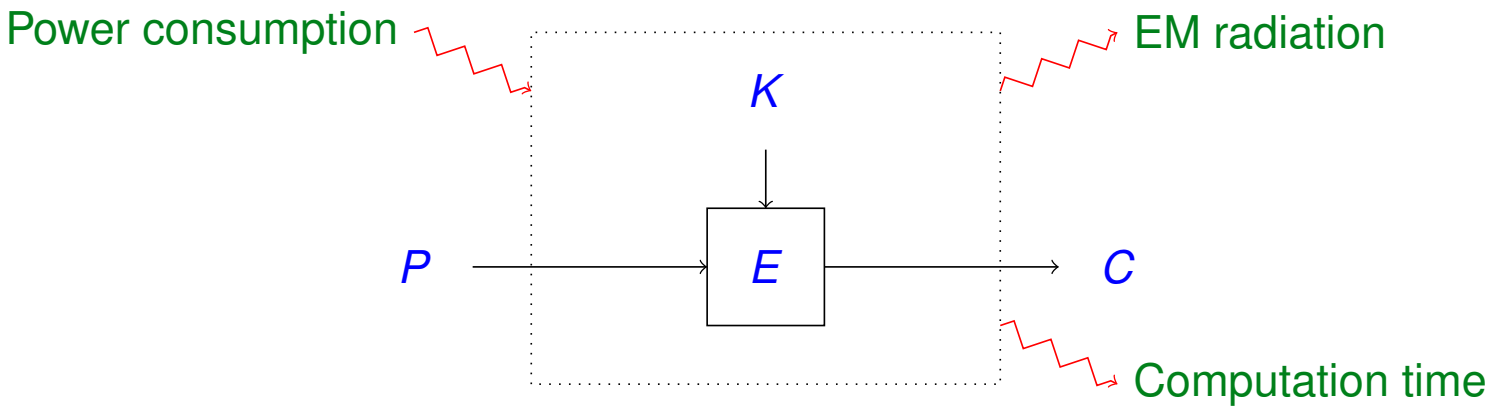
Countermeasures Against Side-Channel Attacks

Ulrich Kühne
ulrich.kuhne@telecom-paris.fr
2019–2020

Side-Channel Attacks



- Side-channels depend on the implementation of an algorithm (hardware or software)
- Side-channels cannot be observed on the algorithmic (mathematical, cryptanalytic) level.
- The implementation may leak sensitive information (secrets) via side-channels, even if those secrets never appear on the input/output interface.

Making Attacks Difficult

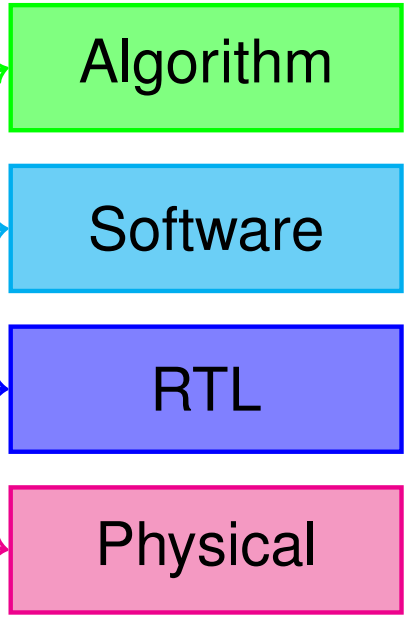
Pseudo countermeasures

- These countermeasures make attacks difficult (but not impossible)
- Examples
  - Noise generator
  - Dis-aligning the traces
    - Variable clock
    - Insertion of dummy operations
- But: There are techniques to remove noise and jitter from the traces

Countermeasures

Different levels of abstraction

- Algorithm / protocol level
- Software / binary level
- Register Transfer Level (RTL)
- Netlist / physical level

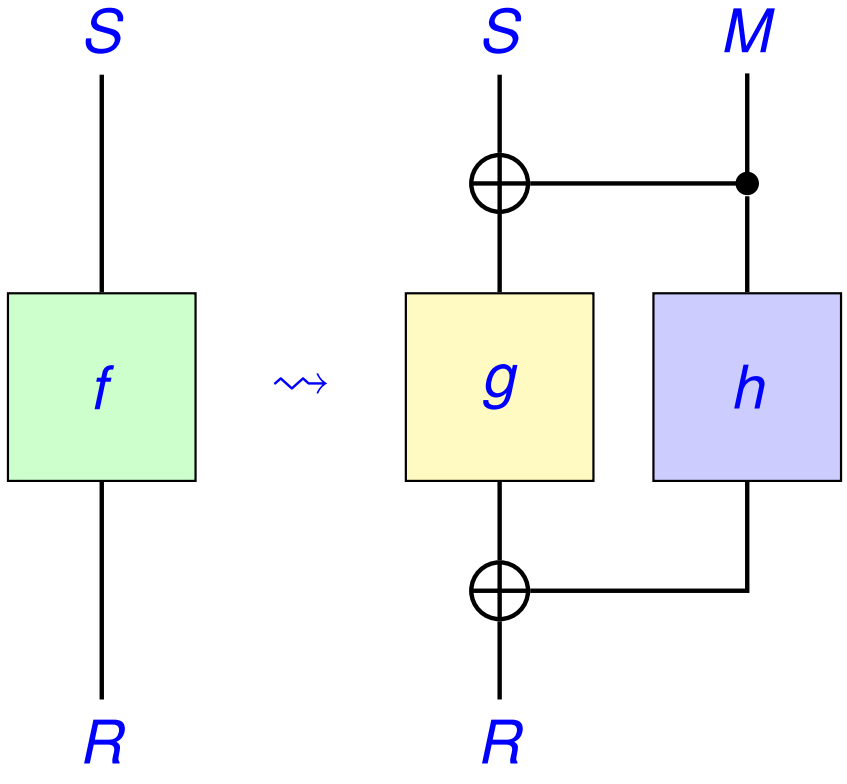


Protocol

Key rotation

- If 200 operations are needed to recover the key, it is sufficient to replace the key after 100 operations
- This demands precise knowledge on existing attacks
- *In extremis*, we can change the key after each operation

Masking



Masking

- Masking the secret with the help of a random variable
- Original function: $R = f(S)$, where S is the secret
- Masked function: $S' = S \oplus M$, $R' = g(S')$, $M' = h(M)$ such that $R = R' \oplus M'$ (Boolean masking)
- M (the mask) is a fresh random variable for each operation
- Need to take care never to manipulate S directly
- If the attacker finds S', she does not learn anything about S
- To find S, she needs to find both S' and M' simultaneously (second order attack)

Masking

Example

Galois Field Arithmetic

- Given an n bit variable X
- X can be interpreted over the finite field $GF(2^n)$
- Addition over $GF(2^n)$ corresponds to bit-wise XOR
- Subtraction is identical to addition: $X + X = 0$
- Multiplication is defined modulo an irreducible polynomial g

Masking

Example (continued)

Consider a simple affine function over $GF(2^n)$

$$f(X) = a \cdot X + b$$

Using a Boolean mask M, this becomes

$$f(X + M) = a \cdot X + \underbrace{a \cdot M}_{\text{correction term}} + b$$

Masking

Example (continued)

Consider a simple affine function over $GF(2^n)$

$$f(X) = a \cdot X + b$$

Using a Boolean mask M, this becomes

$$f(X + M) = a \cdot X + \underbrace{a \cdot M}_{\text{correction term}} + b$$

Since each element is its additive inverse, we have

$$g(M) = a \cdot M$$

Masking

Higher-order masking

- To protect a system against higher-order attacks, multiple shares can be used
- Example: *Threshold Implementation* (TI) [4]

Threshold Implementation

Secret sharing

A variable x is said to be split into n shares x_i if

$$x = \bigoplus_{i=1}^{n} x_i$$

In a perfect (n, n) secret sharing scheme, to recover x, an attacker needs to know all n shares, i.e. $n - 1$ shares do not reveal any information on x.

Threshold Implementation

Example: Three shares

Given secret variable x and two uniform random variables r_1 and r_2, we can define a $(3, 3)$ secret sharing as follows:

$$x_1 = x \oplus r_1$$
$$x_2 = x \oplus r_2$$
$$x_3 = x \oplus r_1 \oplus r_2,$$

x	r_1	r_2	x_1	x_2	x_3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	0	1

and we have $x = x_1 \oplus x_2 \oplus x_3$.

Threshold Implementation

Shared functions

Given a function $z = f(x, y, \dots)$, it can be split into n shares

$$
\begin{aligned}
z_1 &= f_1(\overline{x_1}, \overline{y_1}, \dots) \\
z_2 &= f_2(\overline{x_2}, \overline{y_2}, \dots) \\
&\dots \\
z_n &= f_n(\overline{x_n}, \overline{y_n}, \dots),
\end{aligned}
$$

where each sub-function takes as inputs subsets $\overline{x_i} \subseteq \{x_1, \dots, x_n\}$ of the input shares.

Threshold Implementation

Conditions for a TI implementation

1. Correctness: The sum of all shared functions is equal to the original function $f = \bigoplus_{i=1}^{n} f_i$.
2. Non-completeness: Each shared function is independent of at least one share of each input variable.
3. Uniformity: For any input value x, the corresponding output shares $z_1, \ldots, z_n$ are uniformly distributed for $z = f(x)$

Threshold Implementation

Example: Boolean AND

Consider multiplication in $GF(2)$, i.e. Boolean AND:

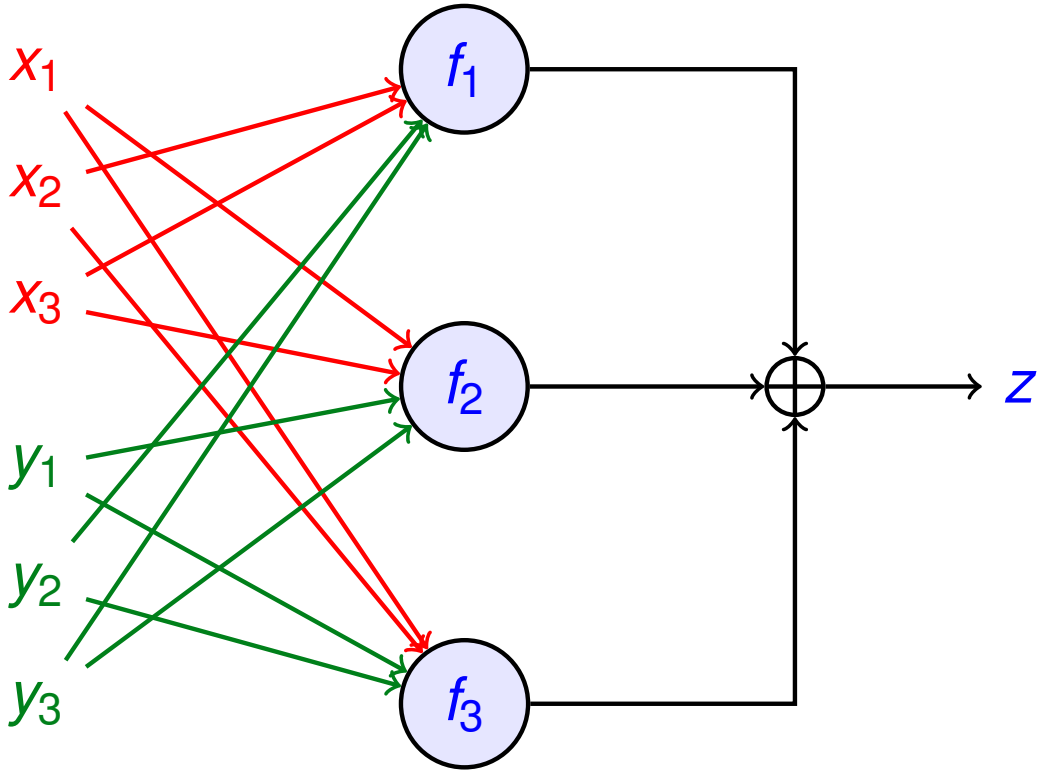
$$f(x, y) = x \cdot y$$

For $n = 3$, a three share implementation of f is given by

$$z_1 = f_1(x_2, x_3, y_2, y_3) = x_2 \cdot y_2 \oplus x_2 \cdot y_3 \oplus x_3 \cdot y_2$$
$$z_2 = f_2(x_1, x_3, y_1, y_3) = x_3 \cdot y_3 \oplus x_1 \cdot y_3 \oplus x_3 \cdot y_1$$
$$z_3 = f_3(x_1, x_2, y_1, y_2) = x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1$$

Threshold Implementation

Example: Non-completeness



$\Rightarrow$ The implementation is non-complete

Threshold Implementation

Example: Uniformity

- For each (un-shared) input value, find the corresponding valid input shares
  - Valid shares for $x = 0$:
    $(x_1, x_2, x_3) \in \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}$
  - Valid shares for $x = 1$:
    $(x_1, x_2, x_3) \in \{(0,0,1), (0,1,0), (1,0,0), (1,1,1)\}$
- For each combination of (un-shared) inputs, compute the distribution of the corresponding output shares

Threshold Implementation

Example: Uniformity

Compute the uniformity table for our three share AND gate:

Threshold Implementation

Example: Uniformity

Compute the uniformity table for our three share AND gate:

	number of occurrences of output share (z_1, z_2, z_3)							
(x, y)	(000)	(011)	(101)	(110)	(001)	(010)	(100)	(111)
(0, 0)	7	3	3	3	0	0	0	0
(0, 1)	7	3	3	3	0	0	0	0
(1, 0)	7	3	3	3	0	0	0	0
(1, 1)	0	0	0	0	5	5	5	1

Threshold Implementation

Example: Uniformity

Compute the uniformity table for our three share AND gate:

	number of occurrences of output share (z_1, z_2, z_3)							
(x, y)	(000)	(011)	(101)	(110)	(001)	(010)	(100)	(111)
(0, 0)	7	3	3	3	0	0	0	0
(0, 1)	7	3	3	3	0	0	0	0
(1, 0)	7	3	3	3	0	0	0	0
(1, 1)	0	0	0	0	5	5	5	1

- Our implementation is not uniform ☹
- It turns out that there is no three share Threshold Implementation for any non-linear two-input function [5]

Threshold Implementation

Example: Boolean Equality (XNOR)

Consider equality in $GF(2)$, i.e. Boolean XNOR:

$$f(x, y) = x \equiv y$$

For $n = 3$, a three share implementation of f is given by

$$z_1 = f_1(x_2, y_2) = x_2 \oplus y_2 \oplus 1$$
$$z_2 = f_2(x_3, y_3) = x_3 \oplus y_3 \oplus 1$$
$$z_3 = f_3(x_1, y_1) = x_1 \oplus y_1 \oplus 1$$

Threshold Implementation

Example: Boolean Equality (XNOR)

Consider equality in $GF(2)$, i.e. Boolean XNOR:

$$f(x, y) = x \equiv y$$

For $n = 3$, a three share implementation of f is given by

$$\begin{aligned} z_1 &= f_1(x_2, y_2) = x_2 \oplus y_2 \oplus 1 \\ z_2 &= f_2(x_3, y_3) = x_3 \oplus y_3 \oplus 1 \\ z_3 &= f_3(x_1, y_1) = x_1 \oplus y_1 \oplus 1 \end{aligned}$$

- This TI is correct, non-complete, and uniform! ☺

Threshold Implementation

- Three share TI exist for various non-linear functions
  - For example multiplication in $GF(2^2)$
- Finding uniform sharings is non-trivial
  - Add correction terms to an even number of shares
  - Check for uniformity
  - Try again...
- Decomposition for higher degree non-linear functions

Threshold Implementation

(De-)composition

- Rewrite function f as composition $g \circ h$
- Build TI for (simpler) component functions g and h
- Add registers

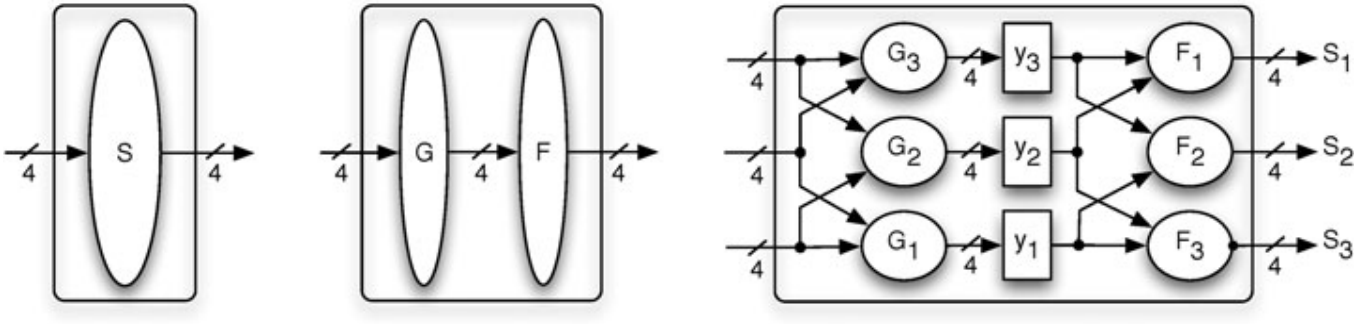
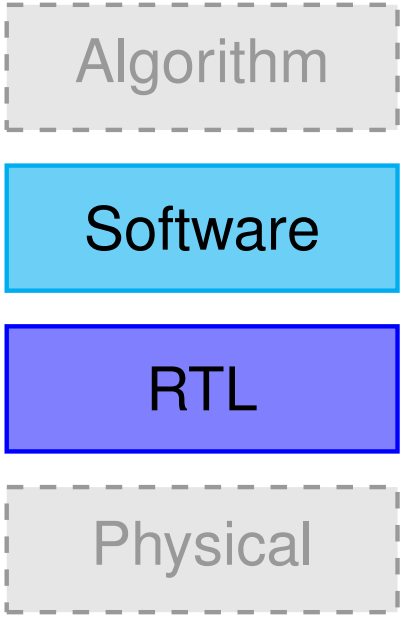


Figure: Three share TI for Present S-box [6]

Masking

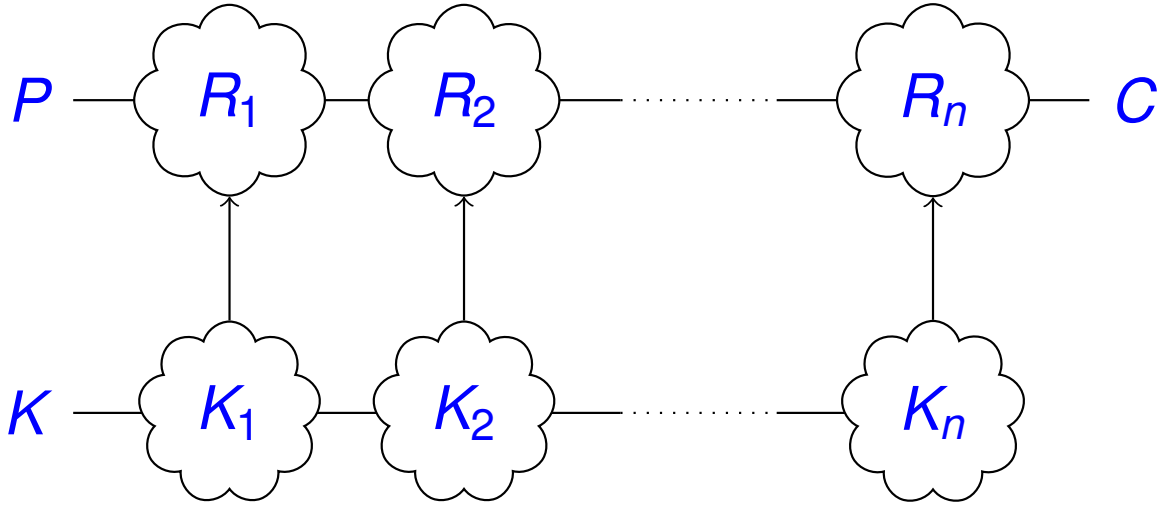
Summary

- (Provable) effective countermeasure
- Can be generalized against higher order attacks
- Applicable on different levels of abstraction
- Needs a reliable source of randomness

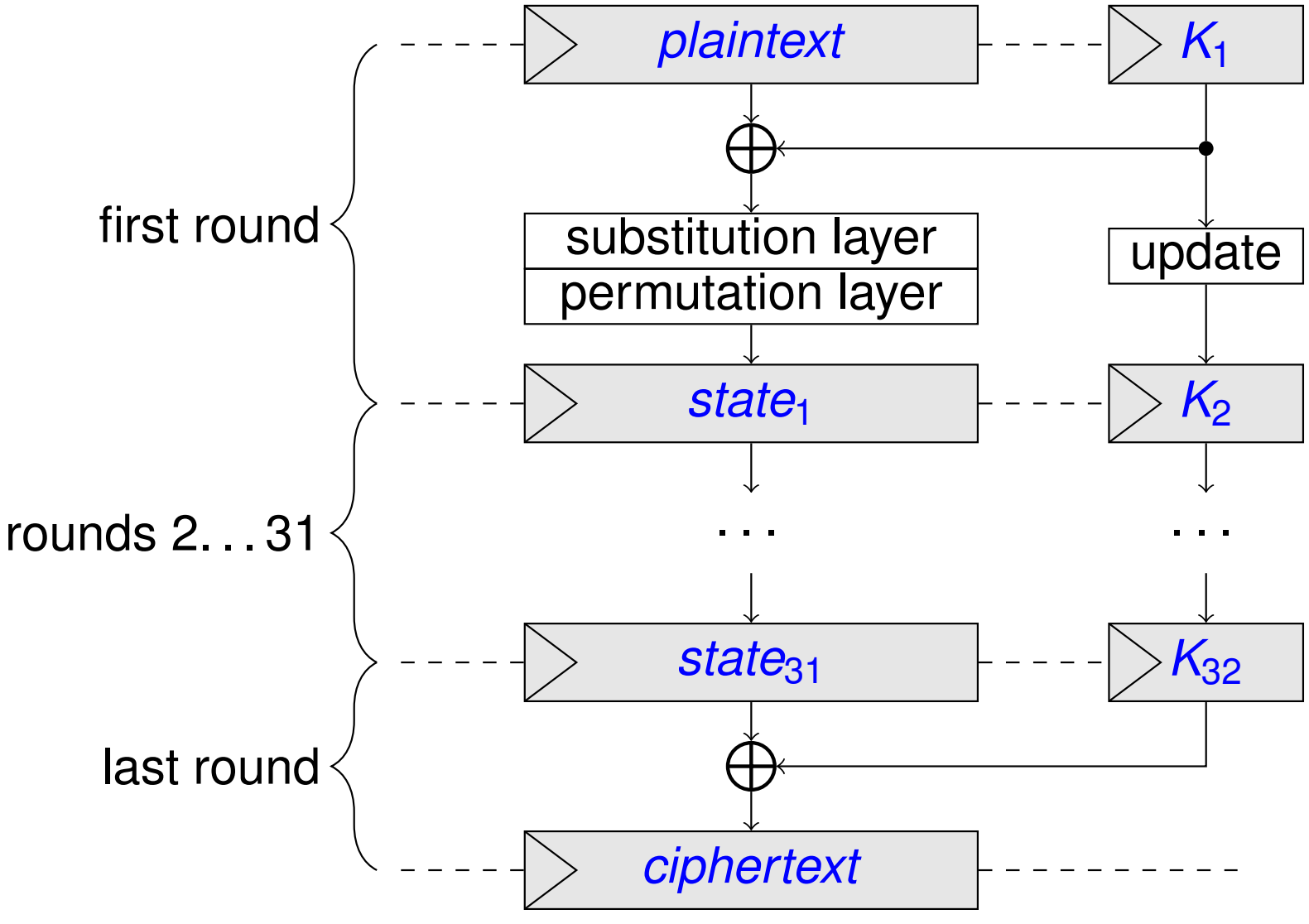


Unrolled Implementation [1]

- In CMOS circuits, the easiest to exploit leakage is due to register updates
- Remove registers to reduce information leakage
- This corresponds to unrolling partially or completely the data path of an implementation

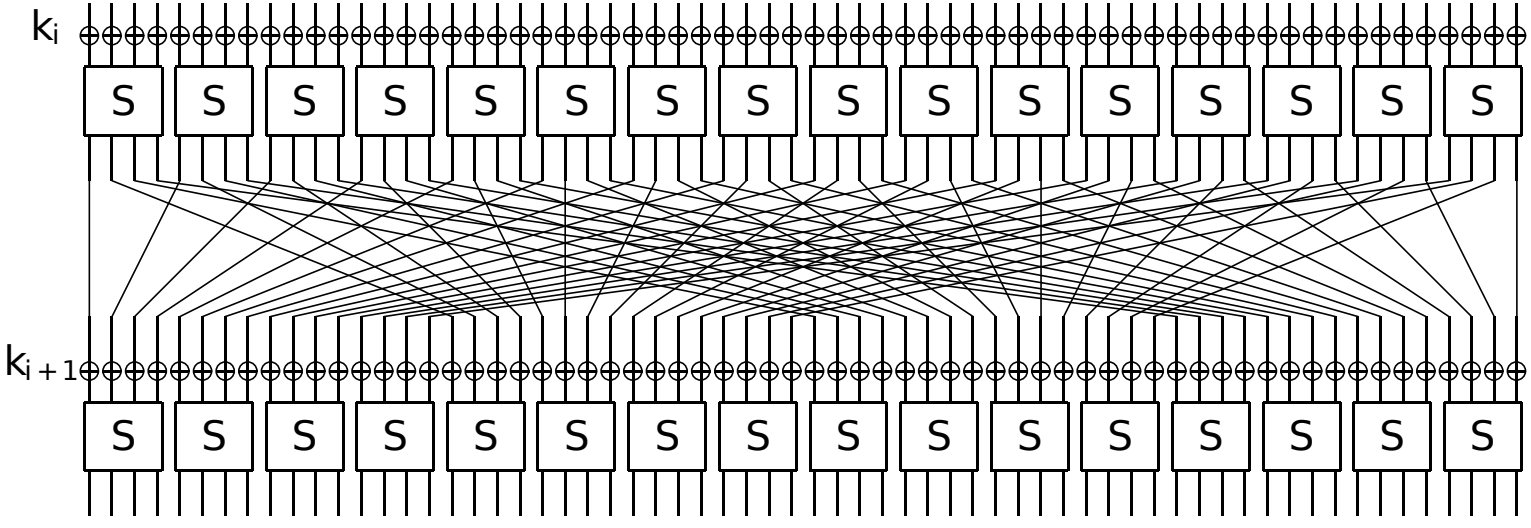


Example: PRESENT [2]



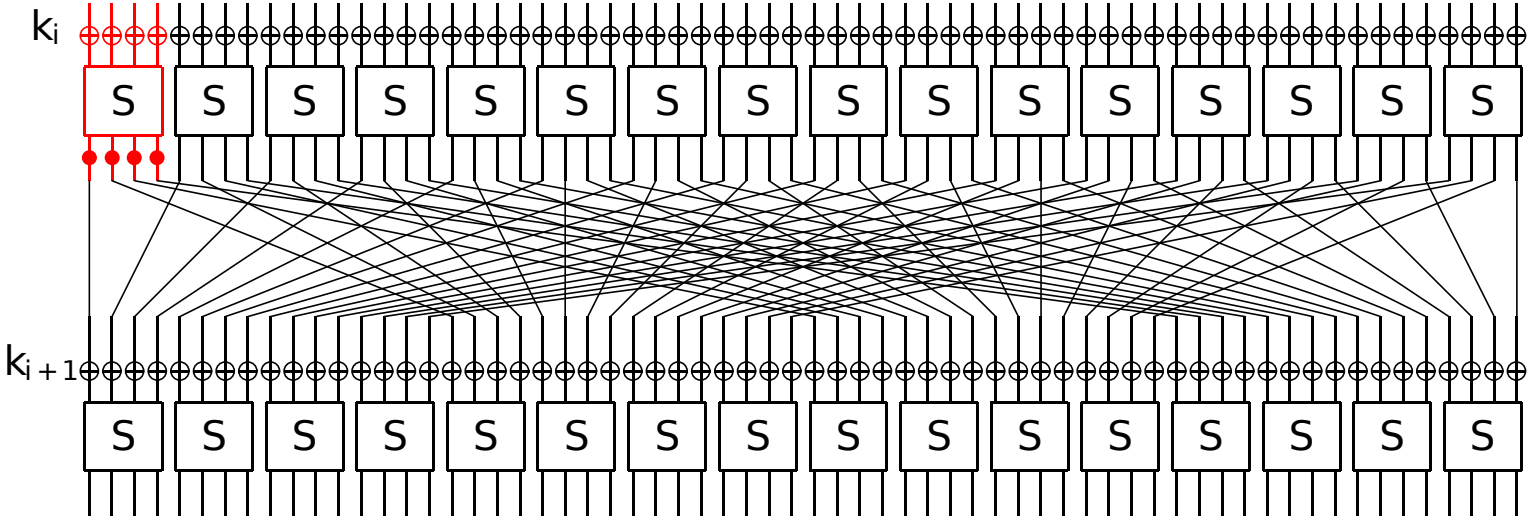
Example: PRESENT [2]

Substitution and permutation layer



Example: PRESENT [2]

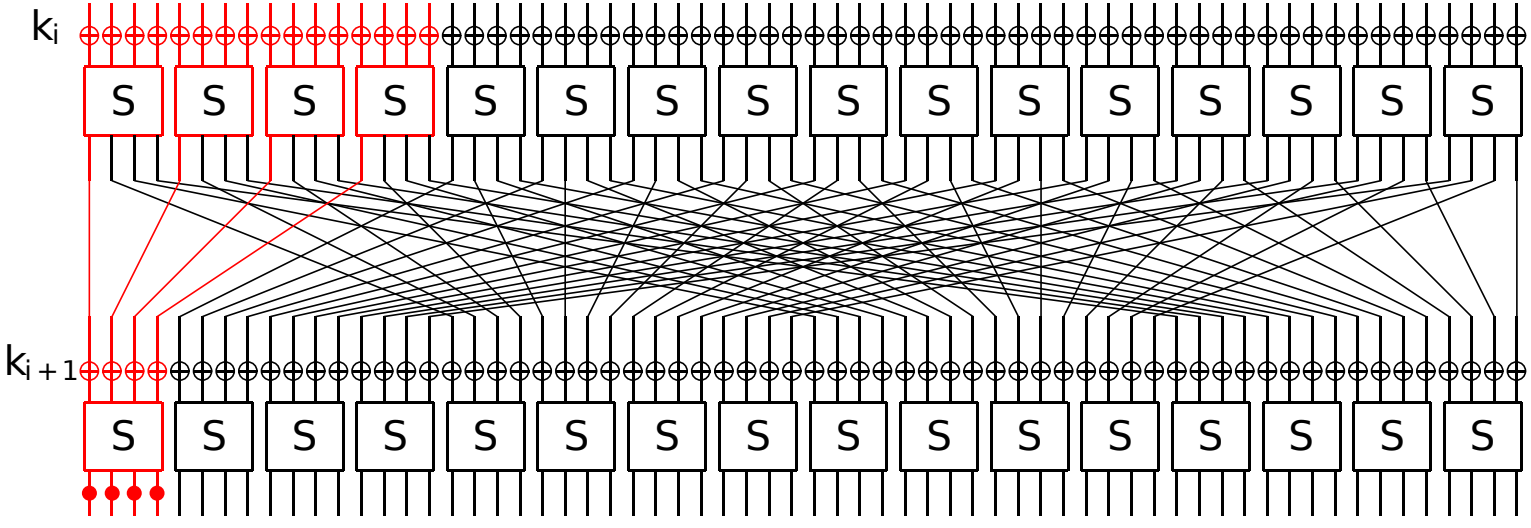
Substitution and permutation layer



- First round attack: 4 bits of K_1

Example: PRESENT [2]

Substitution and permutation layer



- First round attack: 4 bits of K_1
- Second round attack: 4 bits of K_2 + 16 bits of K_1 = 2^{20} possible key hypotheses

Balancing

- Try to hide sensitive information
- Make the behavior of the system constant with respect to the considered side-channel
  - Constant computation time
  - Identical power consumption
- Can be very tricky to achieve (cf localized EM radiation)

Balancing

Example: RSA

Inputs : M , K
$R = 1$; $S = M$;
for $i = |K| - 1; i \geq 0$; $i - -$ **do**
    `/* Balanced branching */`
    if $K_i == 1$ **then**
        $R = R \times S$; $S = S^2$;
    else
        $S = S \times R$; $R = R^2$;
    end if
end for
Return $R = M^K$;

- Modular exponent calculation using Montgomery ladder exponentiation algorithm

Balancing

Dual-Rail Logic with Precharge (DPL) [7, 3]

- Each Boolean variable a is represented by two signals a_T and a_F

a_T	a_F	state	a
0	0	NULL0	-
0	1	VALID0	0
1	0	VALID1	1
1	1	NULL1	-

- A DPL function $(s_T, s_F) = f((a_T, a_F), (b_T, b_F))$ must satisfy the following conditions:
  - If a and b are NULL0, s is NULL0
  - If a and b are VALID, s is VALID

Balancing

Dual-Rail Logic with Precharge (DPL)

- Example of Boolean AND function:
  - $s_T = a_T \cdot b_T$
  - $s_F = a_F + b_F$
- Precharge: The computation alternates between NULL0 and valid phases
- This ensures that we can only observe the following transitions:
  - $(0,0) \rightarrow (0,1) \rightarrow (0,0)$
  - $(0,0) \rightarrow (1,0) \rightarrow (0,0)$
- Thus, at each transition, exactly one signal changes its value, leading to identical power consumption

Balancing

Problems

- Early evaluation: If $f(VALID, NULL) = VALID$, information can leak if the input signals arrive at different times
- The *true* and *false* networks must be close together to avoid timing and power consumption variance

Conclusion

- Protections at different abstraction layers (protocol to physical)
- Security is always a trade-off
- Arms race between protections and novel attacks
- Defender needs to know state-of-the-art attacks

Bibliography I

[1] Shivam Bhasin, Sylvain Guilley, Laurent Sauvage, and Jean-Luc Danger.
Unrolling Cryptographic Circuits: A Simple Countermeasure Against Side-Channel Attacks.
In *RSA Cryptographers' Track, CT-RSA*, volume 5985 of *LNCS*, pages 195–207. Springer, March 1-5 2010.
San Francisco, CA, USA. DOI: 10.1007/978-3-642-11925-5_14.

[2] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe.
PRESENT: An Ultra-Lightweight Block Cipher.
In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727, pages 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3] Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley.
BCDL: A high performance balanced DPL with global precharge and without early-evaluation.
In *DATE'10*, pages 849–854. IEEE Computer Society, March 8-12 2010.
Dresden, Germany.

[4] Svetla Nikova, Christian Rechberger, and Vincent Rijmen.
Threshold Implementations Against Side-Channel Attacks and Glitches.
In *Information and Communications Security: 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006. Proceedings*, pages 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] Svetla Nikova, Vincent Rijmen, and Martin Schläffer.
Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches.
In *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.
Seoul, Korea.

[6] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling.
Side-Channel Resistant Crypto for Less than 2,300 GE.
Journal of Cryptology, 24(2):322–345, April 2011.

Bibliography II

[7] Kris Tiri and Ingrid Verbauwhede.
A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation.
In *DATE'04*, pages 246–251. IEEE Computer Society, February 2004.
Paris, France. DOI: 10.1109/DATE.2004.1268856.