

Masking Countermeasures in Cryptographic Hardware: Part I

Cryptographic Hardware for Embedded Systems

ECE 3170

Fall 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Reading

- This lecture is based on three sources:
- Chapter 9 of *Power Analysis Attacks: Revealing the Secrets of Smart Cards* by Mangard et al., 2007, ISBN-13: 978-0-387-30857-9, ISBN-10: 0-387-30857-1, e-ISBN-10: 0-387-38162-7.
- L. Goubin and J. Patarin, “DES and Differential Power Analysis The ‘Duplication’ Method,” Cryptographic Hardware for Embedded Systems (CHES) conference, 1999.
- Chapter 2 of *Handbook of Applied Cryptography* by Menezes et al., 1996, ISBN: 978-1-119-09672-6.

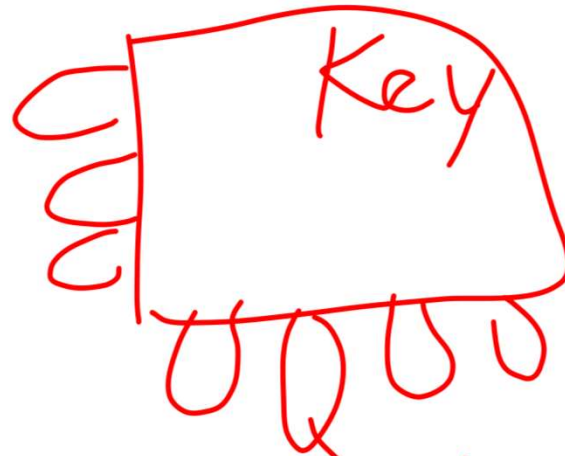
Mathematical Background

- Recall that $\mathbb{Z}$ is the set of integers (including negative numbers and zero)
 - The CHES paper by Goubin and Patarin use $\mathbf{Z}$ instead of $\mathbb{Z}$
- Let n be a positive integer. Then $\mathbb{Z}_n$ is $\{0,1,2,\dots,n-1\}$
- $\gcd(x,y)$ is the greatest common divisor of x and y
- The multiplicative group of $\mathbb{Z}_n$ is $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a,n) = 1\}$. In particular, if n is prime, then $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n-1\}$.
- Recall that $\oplus$ is linear, i.e., $f(V_1 \oplus V_2) = f(V_1) \oplus f(V_2)$
- S-boxes are nonlinear, i.e., $S(V_1 \oplus V_2) \neq S(V_1) \oplus S(V_2)$

Main Idea

- Replace each intermediate variable V with k variables $V_1, \dots, V_k$ such that $V_1, \dots, V_k$ can be used to recover (calculate) V
- Condition 1: from the knowledge of V and for any i (where $1 \leq i \leq k$), it is not feasible to deduce information about the set of possible values of V_i such that there exists values $V_1, \dots, V_{i-1}, V_{i+1}, \dots, V_k$ satisfying the equation $f(V_1, \dots, V_k) = V$
 - Obviously, take for example V_i has 8 bits, clearly V_i is equal to an 8-bit value between 0x00 and 0xFF. This fact is not information “deduced” about V_i from the value of V
- Condition 2: the function $f()$ is such that the transformations to be performed on $V_1, V_2, \dots$, or V_k during the computation (instead of transformations performed on V) can be implemented without explicit calculation of V

P



$S_{box}(P, k)$

Known power traces

not known

Example of Condition 1

- Choose $f(V_1, \dots, V_k) = V_1 \oplus V_2 \oplus \dots \oplus V_k$
- Clearly, for any particular i (where $1 \leq i \leq k$), V_i can take on any value
 - Therefore, even with knowledge of V , no limitation is placed on the value of V_i

Example of Condition 2

- Let $V \in$ multiplicative group $\mathbb{Z}_n^*$
 - The CHES paper by Goubin and Patarin use $\mathbf{Z}/n\mathbf{Z}$ instead of $\mathbb{Z}_n^*$ to indicate a multiplicative group
- $f(V_1, \dots, V_k) = V_1 * V_2 * \dots * V_k \bmod n$
 - where, for each i , $1 \leq i \leq k$, $V_i \in$ multiplicative group $\mathbb{Z}_n^*$
- Clearly, for $f(V_1, \dots, V_k)$ as defined, individual transformations can be performed on $V_1, V_2, \dots, V_k$ without calculating V
- Condition 1 is also satisfied as well

Mathematical Background (cont'd)

- Handbook of Applied Cryptography, Chapter 2.4, pp. 63-75
- Definition
 - The *multiplicative group* of $\mathbb{Z}_n$ is $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$
 - In particular, if n is prime, then $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n - 1\}$
- Definition
 - The *order* of $\mathbb{Z}_n^*$ is the number of elements in $\mathbb{Z}_n^*$, i.e., $|\mathbb{Z}_n^*|$
 - Note that if $a \in \mathbb{Z}_n^*$ and $b \in \mathbb{Z}_n^*$ then $a \cdot b \in \mathbb{Z}_n^*$, i.e., $\mathbb{Z}_n^*$ is closed under multiplication (recall that all multiplication in $\mathbb{Z}_n$ is mod n)
- Example 1: $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$
- Example 2: $\mathbb{Z}_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Example of Example of Condition 2

- First note the multiplicative groups are important for asymmetric encryption schemes such as RSA
- Consider $\mathbb{Z}_{13}^*$
- $12 = 3 * 4$
- So if $V = 12$, $V_1 = 3$ and $V_2 = 4$, $f(V_1, \dots, V_k) = V_1 * V_2 \bmod 13$
- The mod function provides the result that Condition 2 holds

A DES Round

- Key bits shifted, then 48 bits selected
- 1) R_{i-1} expanded to 48 bits
- 2) Key bits permuted and XORed with R_{i-1}
- 3) Eight S-boxes produce 32 bits
- 4) 32 bits are permuted
- Function f is comprised of the above four steps
- Output of f XORed w/ L_{i-1}
 - Result: R_i
- $L_i = R_{i-1}$

Recall Slide 7 of Lecture 4 DES !

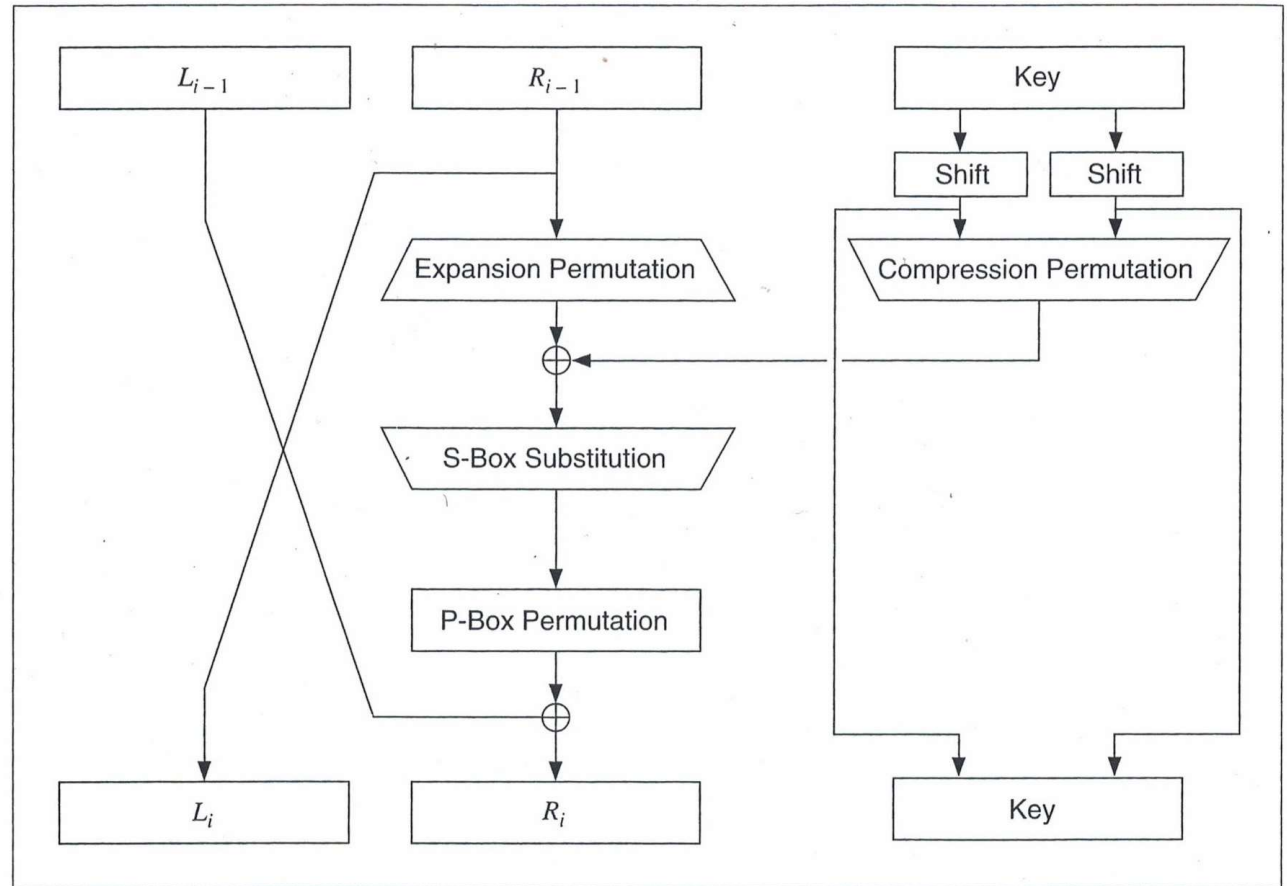


Figure 12.2 One round of DES.

Example: DES

- Consider intermediate variable V
- Separate V into two components: V_1 and V_2
- E.g., choose a function $f(V_1, V_2) = V = V_1 \oplus V_2$
- Condition 1 is satisfied
- All DES transformations fall into one of the following 5 categories:
 - Permutation of the bits of V
 - Expansion of the bits of V
 - $\oplus$ between V and another variable V' of the same type
 - $\oplus$ between V and another variable C depending only on the key
 - Transformations of V using a substitution box

Example: DES (cont'd)

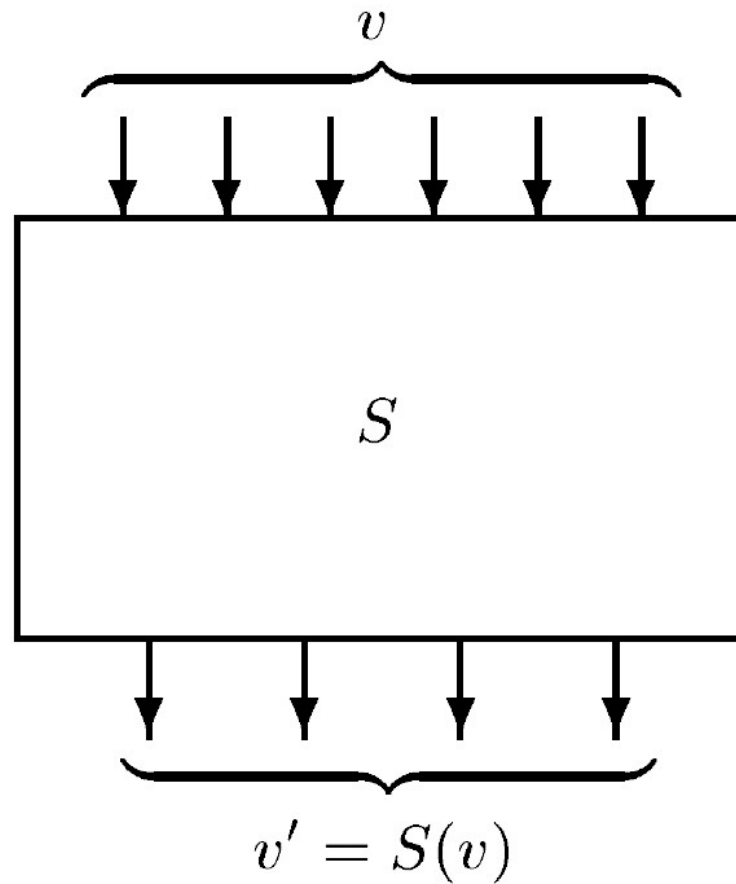
- First two consist of linear transformations
 - To satisfy Condition 2, just perform the permutation and expansion first on V_1 then V_2
 - From linearity, $f(V_1, V_2) = V$ holds after these transformations as well
 - For the third category, just replace $V'' = V \oplus V'$ by (1) $V_1'' = V_1 \oplus V_1'$ and (2) $V_2'' = V_2 \oplus V_2'$
 - Also from linearity, $f(V_1, V_2) = V$ and $f(V_1', V_2') = V'$ result in $f(V_1'', V_2'') = V''$
 - Thus, condition 2 also holds for this category
 - The fourth category similarly maintains Condition 2, just replace $V \oplus C$ with $V_1 \oplus C$ (or with $V_2 \oplus C$)
- Permutation of the bits of V
 - Expansion of the bits of V
 - $\oplus$ between V and another variable V' of the same type
 - $\oplus$ between V and another variable depending only on the key
 - Transformations of V using a substitution box

Main Idea (REPEATED from Slide 4!)

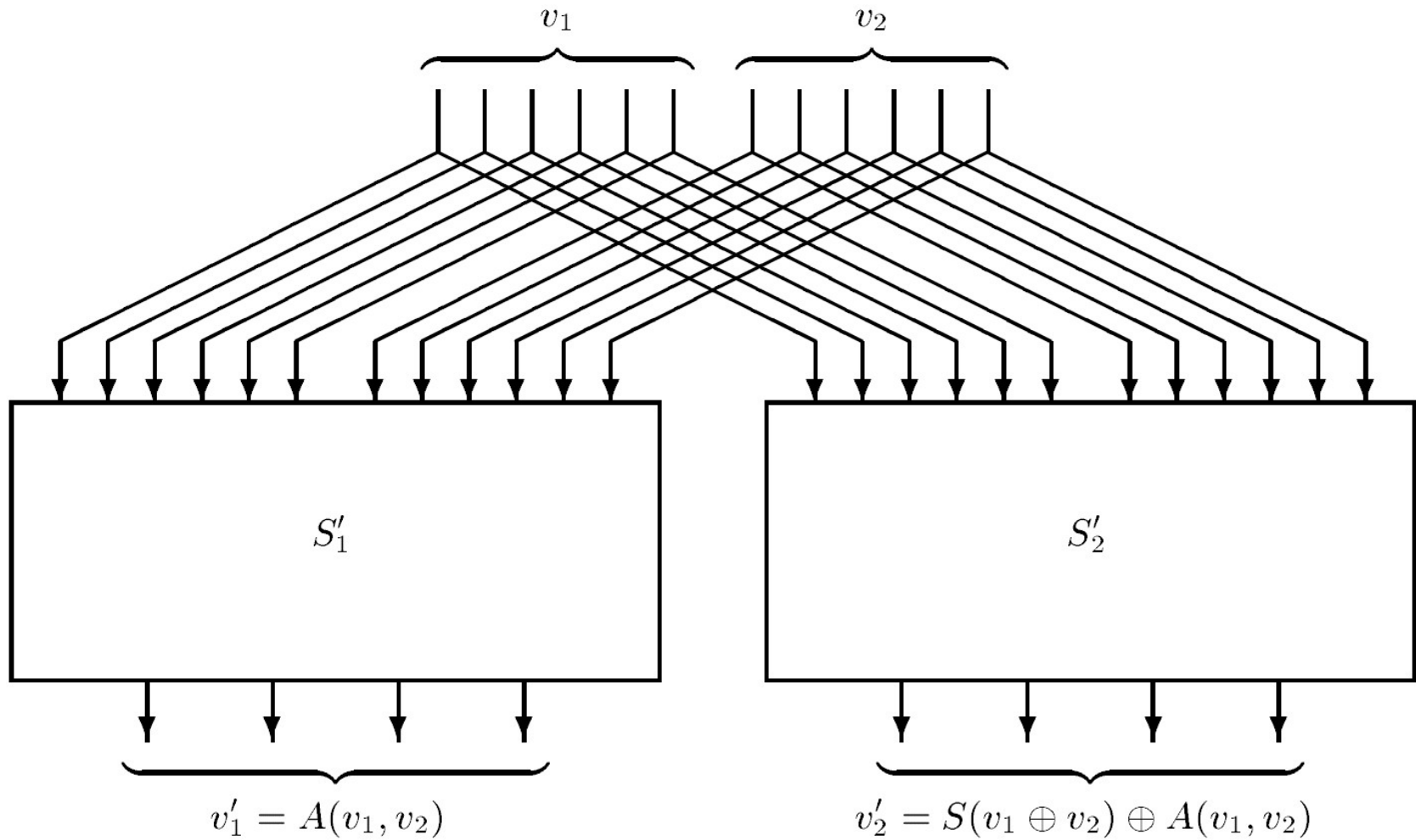
- Replace each intermediate variable V with k variables $V_1, \dots, V_k$ such that $V_1, \dots, V_k$ can be used to recover (calculate) V
- Condition 1: from the knowledge of V and for any i (where $1 \leq i \leq k$), it is not feasible to deduce information about the set of possible values of V_i such that there exists values $V_1, \dots, V_{i-1}, V_{i+1}, \dots, V_k$ satisfying the equation $f(V_1, \dots, V_k) = V$
 - Obviously, take for example V_i has 8 bits, clearly V_i is equal to an 8-bit value between 0x00 and 0xFF. This fact is not information “deduced” about V_i from the value of V
- Condition 2: the function $f()$ is such that the transformations to be performed on $V_1, V_2, \dots$, or V_k during the computation (instead of transformations performed on V) can be implemented without explicit calculation of V

Example: DES (cont'd 2)

- The fifth category is nonlinear
- Idea: design another substitution function $A()$ from 12 bits to 4 such that $(V_1', V_2') = (A(V_1, V_2), S(V_1 \oplus V_2) \oplus A(V_1, V_2))$



**Initial implementation: the predictable values
 v and v' appear in RAM at some time**



Modified implementation: the values $v = v_1 \oplus v_2$ and $v' = v'_1 \oplus v'_2$ never explicitly appear in RAM

Example: DES (cont'd 3)

- The result is two larger substitution boxes S_1' and S_2'
- S_1' implements function A from 12 bits to 4 such that $V_1' = A(V_1, V_2)$
- S_2' implements function $S(V_1, V_2) \oplus A(V_1, V_2)$ from 12 bits (V_1, V_2) to 4 (V_2') such that $V_2' = S(V_1 \oplus V_2) \oplus A(V_1, V_2)$
- Substitution function A satisfies Condition 1
- Table look-up never explicitly calculates $V_1 \oplus V_2$
 - Thus, Condition 2 is satisfied

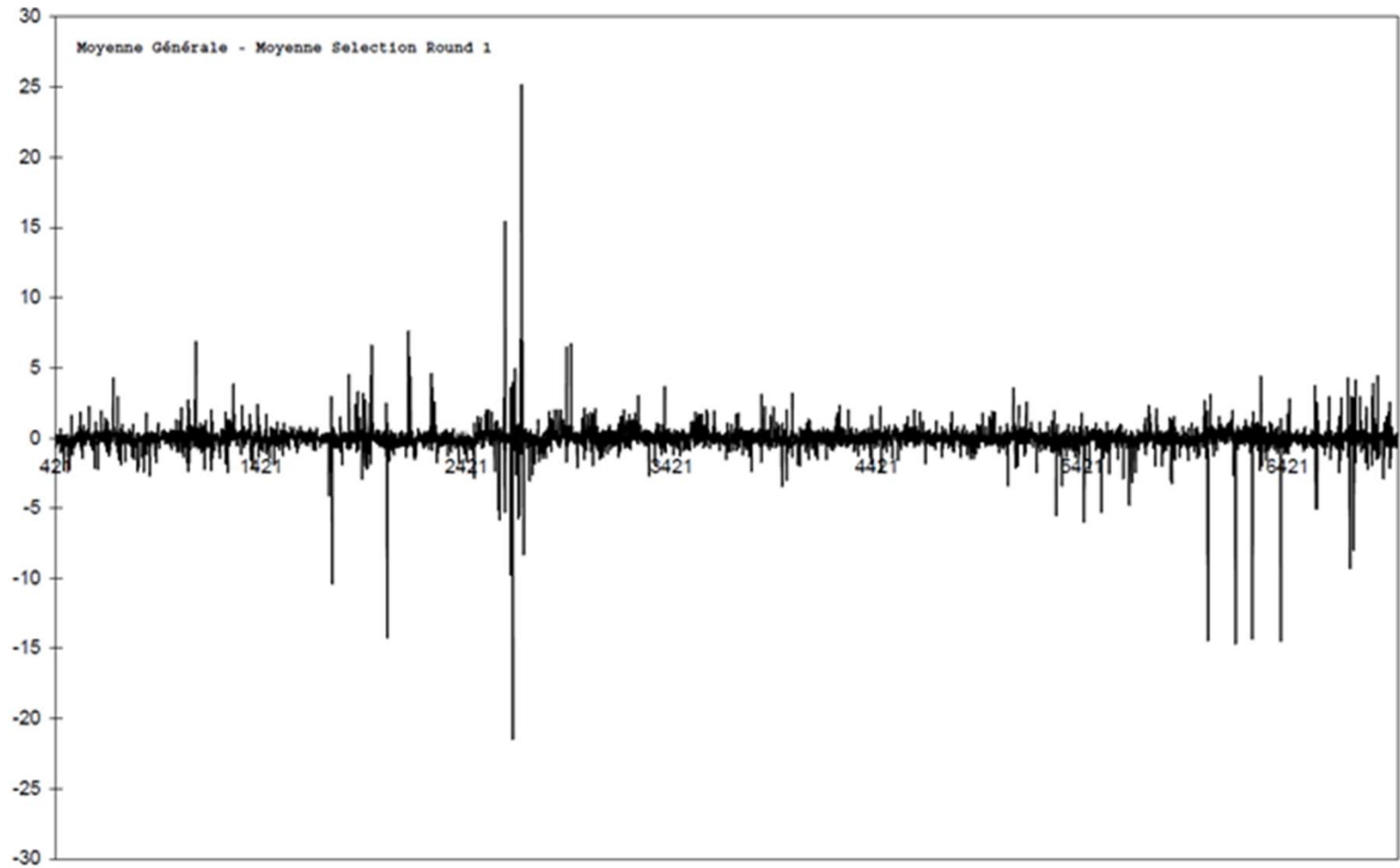


Fig. 6. An example of difference of the curves MC and MC' when the 6 bits are false

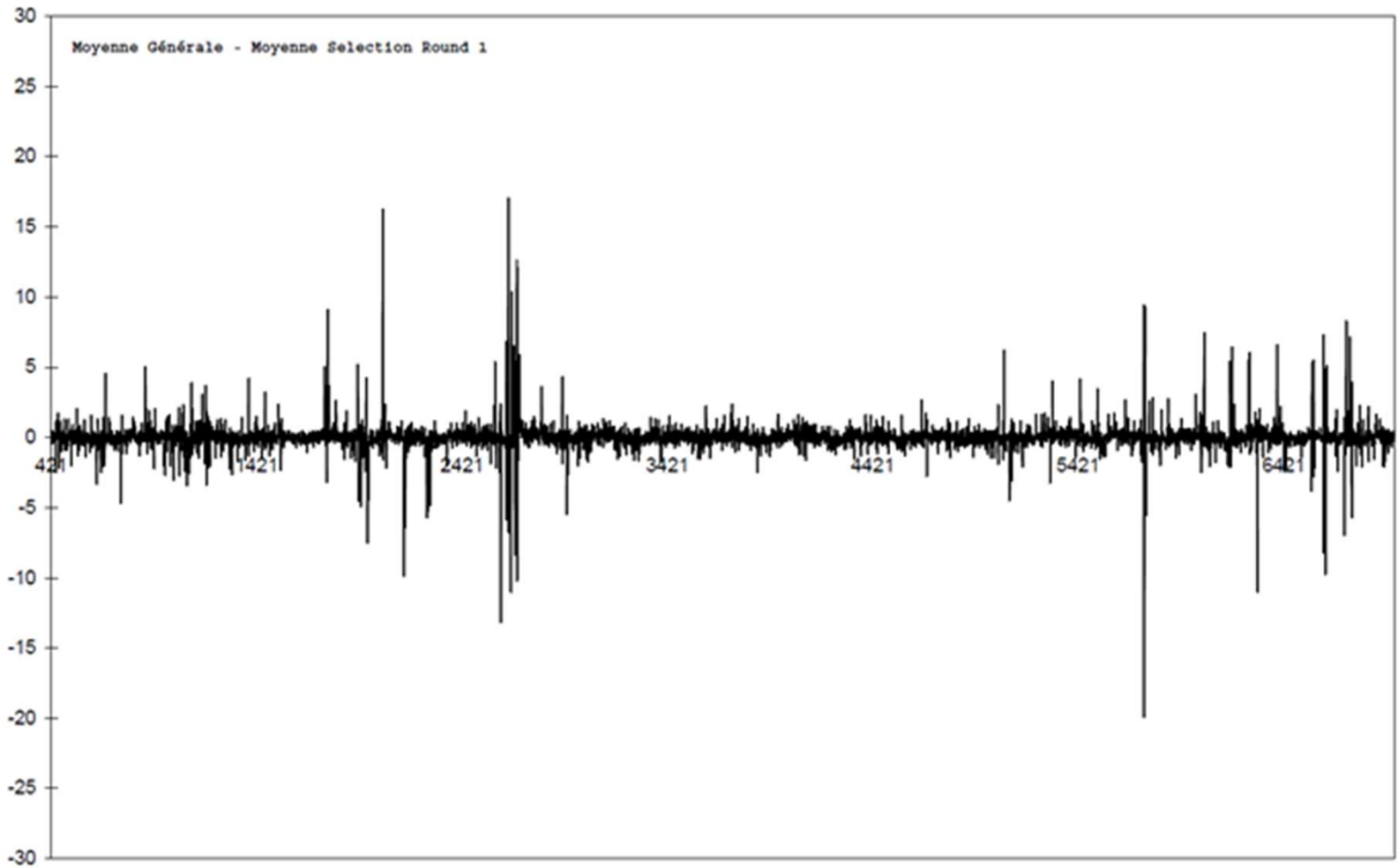


Fig. 7. Difference of the curves MC and MC' when the 6 bits are correct

©Georgia Institute of Technology, 2018-2025

Summary of Masking Types

- Boolean
 - Intermediate value v is concealed by XOR with mask m
 - $v_m = v \oplus m$
- Additive
 - Intermediate value v is concealed by modular addition with mask m
 - $v_m = v + m \pmod{n}$
- Multiplicative
 - Intermediate value v is concealed by modular multiplication with mask m
 - $v_m = v * m \pmod{n}$

Arithmetic Masking Example: RSA

- Consider the “square and multiply” implementation
 - Perform computation of $x^d \bmod n$
 - Exponent d is part of the key
 - d has m bits $d_{m-1}d_{m-2} \dots d_1d_0$
 - 1. $z \leftarrow 1$;
 - For i going backwards from $m - 1$ to 0 do:
 - 2. $z \leftarrow z^2 \bmod n$;
 - 3. if $d_i = 1$ then $z \leftarrow z * x \bmod n$;

Arithmetic Masking Example: new RSA

- Consider again the “square and multiply” implementation
 - Recall we replace intermediate variable V with variables V_1 and V_2
 - In this case we replace x by x_1 and x_2
 - Need to compute $x_1^d \bmod n$ and $x_2^d \bmod n$
 - Final computation will be $x^d \bmod n = (x_1^d \bmod n) * (x_2^d \bmod n) \bmod n$
 - As before d has m bits $d_{m-1}d_{m-2} \dots d_1d_0$

1. $z_1 \leftarrow 1$;	1. $z_2 \leftarrow 1$;
For i going backwards from $m - 1$ to 0 do:	For i going backwards from $m - 1$ to 0 do:
2. $z_1 \leftarrow z_1^2 \bmod n$;	2. $z_2 \leftarrow z_2^2 \bmod n$;
3. if $d_i = 1$ then $z_1 \leftarrow z_1 * x_1 \bmod n$;	3. if $d_i = 1$ then $z_2 \leftarrow z_2 * x_2 \bmod n$;
 - The above steps can be done in a random fashion including variations on overlapping versus sequential execution