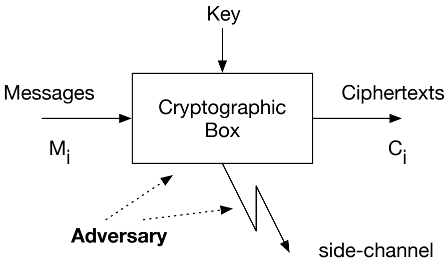
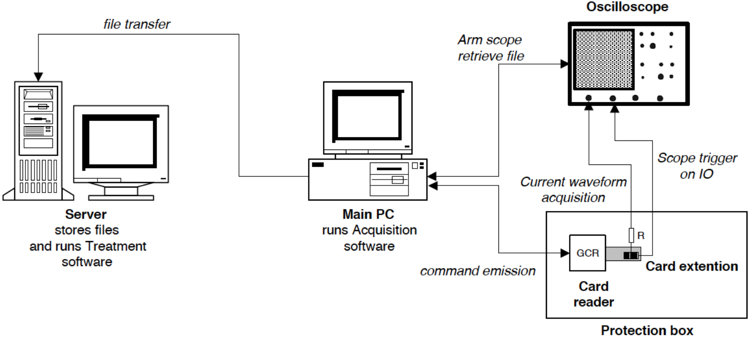
Power Attacks and Countermeasures



Information Leakage Hypothesis

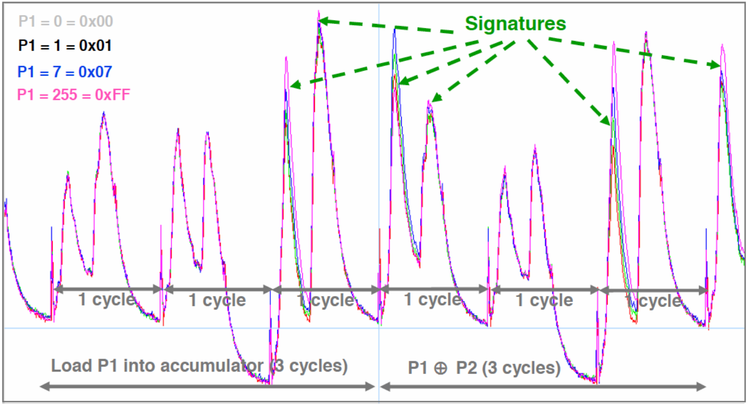
- The power consumption of a chip depends on the manipulated data and the executed instruction
- Information leakage model (assumption): The consumed power is related to the Hamming weight of data (or address, op code)
- $H(0) = 0$
- $H(1) = H(2) = H(4) = H(8) = \cdots = 1$
- $H(3) = H(5) = H(6) = H(9) = \cdots = 2$
- $\cdots$
- $H(P_i \oplus P_{i-1})$

Equipment Setup for Power Attacks



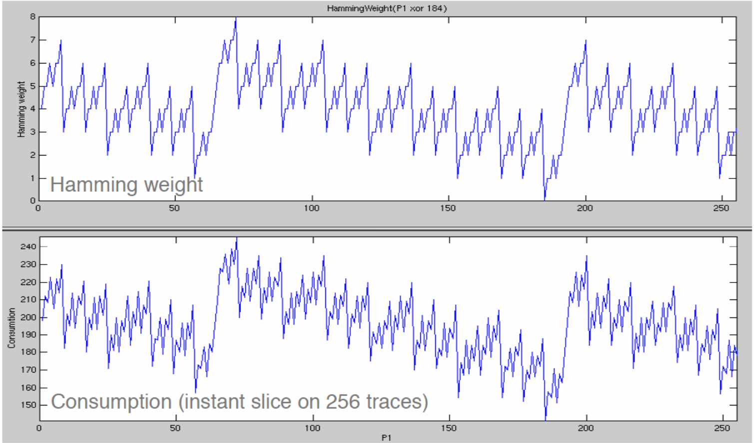
Information Leakage

- Load P_1 and XOR with $P_2 = 0$ such that $P_1 = 0, 1, 7, 255$



Information Leakage

- $H(P_1 \oplus 184)$ for $P_1 = 0, 1, 2, \ldots, 255$



Simple Power Analysis (SPA)

- The objective is to find the secret or private key
- Algorithm is known
- Implementation is unknown however some background is available
- Reverse engineering is required
- A single power curve may be sufficient
- A known plaintext, ciphertext pair may be required

SPA Attack on RSA Signature Operation

- The signature computation

$$s = m^d \pmod{n}$$

- n is large modulus, say 1024 bits or more
- m is the message
- m is the padded and hashed message
- s is the signature
- d is the private key such that $e \cdot d = 1 \bmod \phi(n)$
- The attacker aims to obtain d

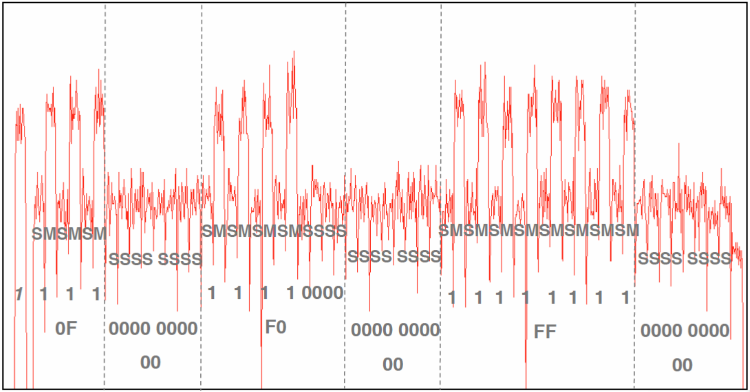
SPA Attack on RSA Signature Operation

- Implementation details:
  - n, m, s, and d are 128-byte buffers
  - the binary method of exponentiation
  - the exponent bits are scanned from MSB to LSB
  - k is the bit size of d

Input: m, $d = (d_{k-1}, \ldots, d_0)_2$, n
Output: $s = m^d \pmod{n}$
1. $s \leftarrow 1$
2. For $i = k - 1$ downto 0
  $s \leftarrow s \cdot s \pmod{n}$
  If $d_i = 1$ then $s \leftarrow s \cdot m \pmod{n}$
3. Return s

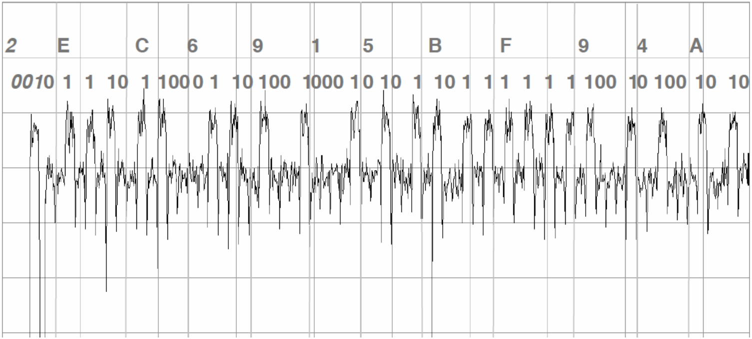
SPA Attack on RSA Signature Operation

- Test key value: 0F 00 F0 00 FF 00



SPA Attack on RSA Signature Operation

- Test key value: 2E C6 91 5B F9 4A



SPA Attack on RSA Signature Operation

- SPA uses implementation details
- SPA requires:
    - algorithm knowledge,
    - reverse engineering,
    - representation tuning, and
    - playing with implementation assumptions
- SPA depends on
    - Algorithm implementation
    - Application constraints
    - The technology (electrical properties) of the chip
    - Possible countermeasures

Countermeasures Against SPA Attack

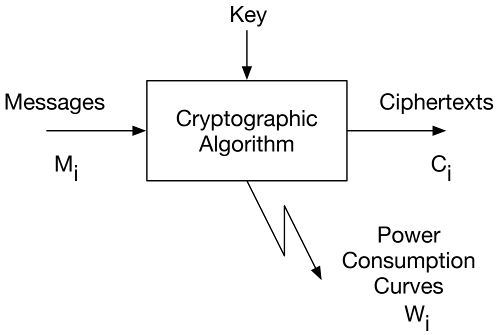
- What is a countermeasure?
- Anything that foils the attack
- Basic countermeasure: remove code branches that depend on secret or private key bits
- Advanced countermeasure:
  - Algorithm specification refinement
  - Data whitening (blinding)
  - Make changes in the instruction set
  - Electrical behavior changes (current scramblers, coprocessor usage)

Differential Power Analysis

- Also invented by Paul Kocher (1998)
- A powerful and generic power attack
- DPA uses statistics and signal processing
- DPA requires known random messages
- DPA targets a known algorithm
- Applicable to a smart card
- Big noise in crypto community
- Big fear in the smart card industry

Acquisition Procedure

- Apply the algorithm L times such that $10^3 < L < 10^5$



Selection and Prediction

- Assume the message is processed by a known deterministic function f (transfer, permutation)
- Knowing the message, one can recompute its image through f offline

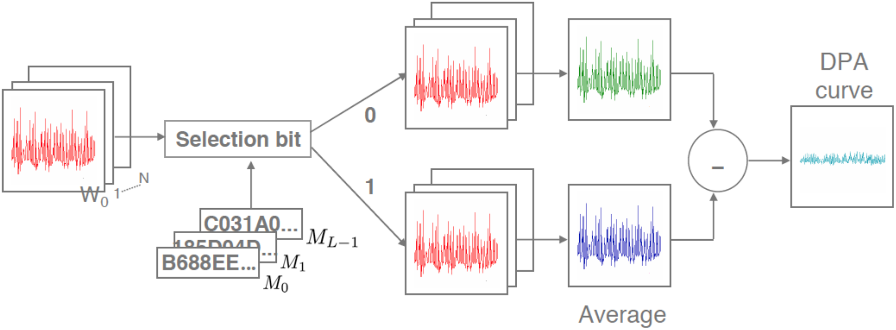
$$M_i \longrightarrow \boxed{\text{ f }} \longrightarrow M_i' = f(M_i)$$

- Now select a single bit from M' buffer
- One can predict the true story of its variations for $i = 0, 1, \ldots, L-1$

i	Message	bit
0	2A5A058FC295ED	0
1	17BD152B330F0A	1
2	BD9D5EE99FE1F8	0

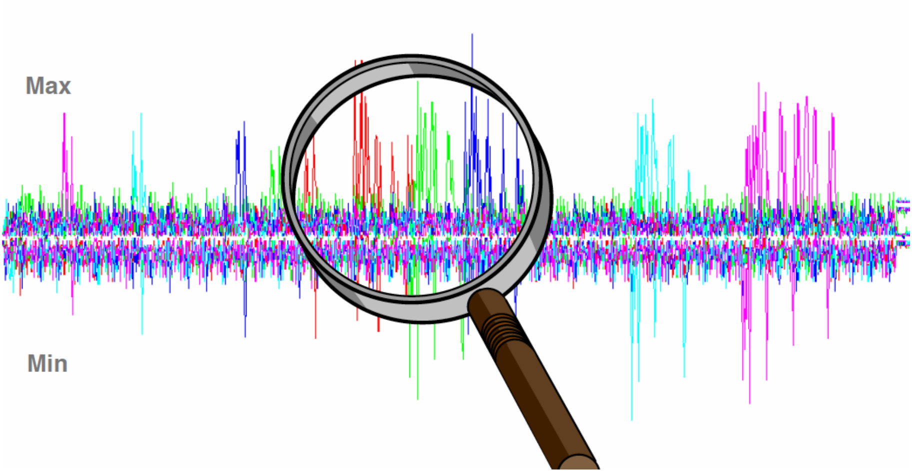
DPA Operator and Curve

- DPA curve construction



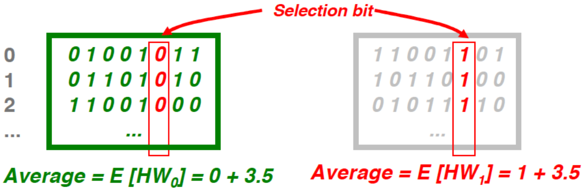
DPA Operator and Curve

- DPA curves for different selection bits



DPA Operator and Curve

- Spikes explanation: Hamming weight of the byte of the selection bits



$$\Delta = E(HW_1) - E(HW_0) = 1$$

- The peak height is proportional to $\sqrt{L}$
- If prediction was wrong, the selection bit would random

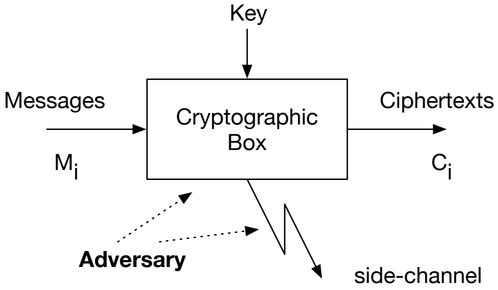
$$E(HW_1) = E(HW_0) = 4 \quad \Rightarrow \quad \Delta = 0$$

DPA on RSA

- The entire key (the private exponent d) is not handled together, rather bit by bit in progression
- The prediction can be done by time slices
- Prediction of the next bit requires the previous bit to be broken

RSA Countermeasures

- The binary method of exponentiation leaks information on private key



Square-and-Multiply Algorithm

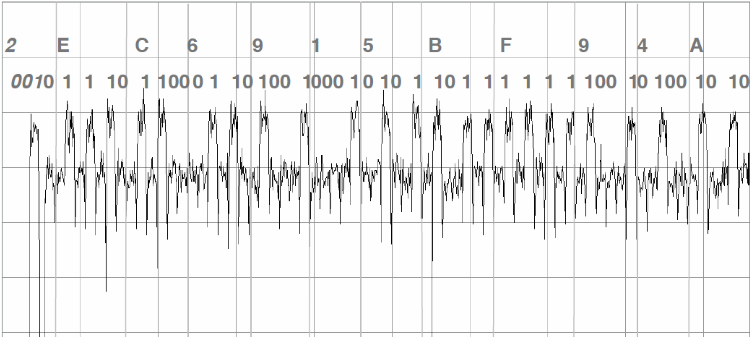
- The binary method is also known as Square-and-multiply algorithm

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$
2. For $i = k-1$ downto 0
  $R_0 \leftarrow R_0^2 \pmod{n}$
  If $d_i = 1$ then $R_0 \leftarrow R_0 \cdot m \pmod{n}$
3. Return R_0

- It performs exponentiation left to right
- 2 Temporary variables R_0 and m
- Susceptible to SPA-type attacks

Square-and-Multiply Algorithm

- The key: 2E C6 91 5B F9 4A



Square-and-Multiply-Always Algorithm

- One way to avoid leakage is to square and multiply at every step

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow 1$
2. For $i = k-1$ downto 0
$R_0 \leftarrow R_0^2 \pmod{n}$
$b \leftarrow 1 - d_i$; $R_b \leftarrow R_b \cdot m \pmod{n}$
3. Return R_0

- When $b = 1$ (i.e., $d_i = 0$), there is a dummy multiplication
- The power trace is a regular succession of squares and multiplies
- 3 Temporary variables: R_0, R_1 and m
- Not susceptible to SPA-type attacks
- Susceptible to Safe-Error attacks

Safe-Error Attacks

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

Montgomery Powering Ladder

- Montgomery exponentiation algorithm

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$
2. For $i = k - 1$ downto 0
$b \leftarrow 1 - d_i$; $R_b \leftarrow R_0 \cdot R_1 \pmod{n}$
$R_{d_i} \leftarrow R_{d_i}^2 \pmod{n}$
3. Return R_0

- This algorithm behaves regularly without dummy operations
- 2 Temporary variables: R_0 and R_1
- Not susceptible to SPA-type attacks
- Not susceptible to Safe-Error attacks

Square-and-Multiply Algorithm Example

- $e = 9 = (1001)_2$
- Square-and-Multiply Algorithm
- Start with $R_0 = 1$

i	d_i	Step 2a	Step 2b
3	1	$R_0 = R_0^2 = 1$	$R_0 = R_0 m = m$
2	0	$R_0 = R_0^2 = m^2$	
1	0	$R_0 = R_0^2 = m^4$	
0	1	$R_0 = R_0^2 = m^8$	$R_0 = R_0 m = m^9$

- Result: $R_0 = m^9$
- Total of 4 squarings and 2 multiplications

Square-and-Multiply-Always Algorithm Example

- $e = 9 = (1001)_2$
- Square-and-Multiply-Always Algorithm
- Start with $R_0 = 1$ and $R_1 = 1$

i	d_i	b	Step 2a	Step 2b
3	1	0	$R_0 = R_0^2 = 1$	$R_0 = R_0 m = m$
2	0	1	$R_0 = R_0^2 = m^2$	$R_1 = R_1 m = m$
1	0	1	$R_0 = R_0^2 = m^4$	$R_1 = R_1 m = m^2$
0	1	0	$R_0 = R_0^2 = m^8$	$R_0 = R_0 m = m^9$

- Result: $R_0 = m^9$
- Total of 4 squarings and 4 multiplications

Montgomery Powering Ladder Algorithm Example

- $e = 9 = (1001)_2$
- Montgomery Powering Ladder Algorithm
- Start with $R_0 = 1$ and $R_1 = m$

i	d_i	b	Step 2a	Step 2b
3	1	0	$R_0 = R_0 R_1 = m$	$R_1 = R_1^2 = m^2$
2	0	1	$R_1 = R_0 R_1 = m^3$	$R_0 = R_0^2 = m^2$
1	0	1	$R_1 = R_0 R_1 = m^5$	$R_0 = R_0^2 = m^4$
0	1	0	$R_0 = R_0 R_1 = m^9$	$R_1 = R_1^2 = m^{10}$

- Result: $R_0 = m^9$
- Total of 4 squarings and 4 multiplications

Comparing Exponentiation Algorithms

Algorithm	Temporary Variables	Number of Squ & Mul
Square-and-Multiply	2	$k + k/2$
Square-and-Multiply-Always	3	$k + k$
Montgomery Powering Ladder	2	$k + k$

- Are there better algorithms?
- Is it possible to compute $m^e \pmod{n}$ in a secure way, without introducing extra multiplications?
- The **Atomic Square-and-Multiply** algorithms by Marc Joye require $k + k/2$ squarings and multiplications as in the classical (unprotected) algorithm

Atomic Square-and-Multiply Algorithm

- Atomic Square-and-Multiply Algorithm by Marc Joye

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $i \leftarrow k-1$; $b \leftarrow 0$
2. While $i \geq 0$
$R_0 \leftarrow R_0 \cdot R_b \pmod{n}$
$b \leftarrow b \oplus d_i$; $i \leftarrow i - \bar{b}$
3. Return R_0

- This algorithm behaves regularly without dummy operations
- 2 Temporary variables: R_0 and R_1

Atomic Square-and-Multiply Algorithm Example

- $e = 9 = (1001)_2$
- Atomic Square-and-Multiply Algorithm by Marc Joye
- Start with $R_0 = 1$, $R_1 = m$, $i = k - 1 = 3$, and $b = 0$

i	d_i	b	Step 2a	Step 2b
3	1	0	$R_0 = R_0 R_0 = 1$	$b = b \oplus d_i = 1$; $i = i - \bar{b} = 3$
3	1	1	$R_0 = R_0 R_1 = m$	$b = b \oplus d_i = 0$; $i = i - \bar{b} = 2$
2	0	0	$R_0 = R_0 R_0 = m^2$	$b = b \oplus d_i = 0$; $i = i - \bar{b} = 1$
1	0	0	$R_0 = R_0 R_0 = m^4$	$b = b \oplus d_i = 0$; $i = i - \bar{b} = 0$
0	1	0	$R_0 = R_0 R_0 = m^8$	$b = b \oplus d_i = 1$; $i = i - \bar{b} = 0$
0	1	1	$R_0 = R_0 R_1 = m^9$	$b = b \oplus d_i = 0$; $i = i - \bar{b} = -1$

- Result: $R_0 = m^9$
- Total of 4 squarings and 2 multiplications

Right-to-Left Binary Algorithm

- The classical Right-to-Left Binary Algorithm

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $i \leftarrow 0$
2. While $i \leq k-1$
If $d_i = 1$ then $R_0 \leftarrow R_0 \cdot R_1 \pmod{n}$
$R_1 \leftarrow R_1^2 \pmod{n}$; $i \leftarrow i+1$
3. Return R_0

Right-to-Left Binary Algorithm Example

- $e = 9 = (1001)_2$
- The classical Right-to-Left Binary Algorithm
- Start with $R_0 = 1$, $R_1 = m$, and $i = 0$

i	d_i	Step 2a	Step 2b
0	1	$R_0 = R_0 R_1 = m$	$R_1 = R_1^2 = m^2$; $i = i + 1 = 1$
1	0		$R_1 = R_1^2 = m^4$; $i = i + 1 = 2$
2	0		$R_1 = R_1^2 = m^8$; $i = i + 1 = 3$
3	1	$R_0 = R_0 R_1 = m^9$	$R_1 = R_1^2 = m^{16}$; $i = i + 1 = 4$

- Result: $R_0 = m^9$
- Total of 4 squarings and 2 multiplications

Atomic Right-to-Left Binary Algorithm

- The atomic Right-to-Left Binary Algorithm by Marc Joye

Input: $m,\ d = (d_{k-1}, \ldots, d_0)_2,\ n$
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$; $i \leftarrow 0$; $b \leftarrow 1$
2. While $i \leq k-1$
$b \leftarrow b \oplus d_i$
$R_b \leftarrow R_b R_1 \pmod{n}$; $i \leftarrow i + b$
3. Return R_0

Atomic Right-to-Left Binary Algorithm Example

- $e = 9 = (1001)_2$
- Atomic Right-to-Left Binary Algorithm by Marc Joye
- Start with $R_0 = 1$, $R_1 = m$, $i = 0$, and $b = 1$

i	d_i	b	Step 2a	Step 2b
0	1	1	$b = b \oplus d_i = 0$	$R_0 = R_0 R_1 = m$; $i = i + b = 0$
0	1	0	$b = b \oplus d_i = 1$	$R_1 = R_1 R_1 = m^2$; $i = i + b = 1$
1	0	1	$b = b \oplus d_i = 1$	$R_1 = R_1 R_1 = m^4$; $i = i + b = 2$
2	0	1	$b = b \oplus d_i = 1$	$R_1 = R_1 R_1 = m^8$; $i = i + b = 3$
3	1	1	$b = b \oplus d_i = 0$	$R_0 = R_0 R_1 = m^9$; $i = i + b = 3$
3	1	0	$b = b \oplus d_i = 1$	$R_1 = R_1 R_1 = m^{16}$; $i = i + b = 4$

- Result: $R_0 = m^9$
- Total of 4 squarings and 2 multiplications

Preventing Side-Channel Attacks

- For SPA-type attacks: Use Montgomery ladder or Atomic algorithms of Marc Joye
- However, these algorithms are not sufficient to thwart DPA-like attacks
- To circumvent the DPA-type attacks, we use data whitening, or randomization, or blinding
- For RSA, randomization of m, d, or n is used in the computation of $s = m^d \pmod n$

DPA-Type Countermeasures — Randomizing m

- For a random r compute

$$\begin{aligned} m^* &= r^e \cdot m \pmod{n} \\ s^* &= (m^*)^d \pmod{n} \\ s &= s^* \cdot r^{-1} \pmod{n} \end{aligned}$$

- If e is unknown, compute

$$\begin{aligned} m^* &= r \cdot m \pmod{n} \\ s^* &= (m^*)^d \pmod{n} \\ s &= s^* \cdot r^{-d} \pmod{n} \end{aligned}$$

- For a short random $r < 2^u$, compute

$$\begin{aligned} m^* &= m + r \cdot n \\ n^* &= 2^u \cdot n \\ s^* &= (m^*)^d \pmod{n^*} \\ s &= s^* \pmod{n} \end{aligned}$$

DPA-Type Countermeasures — Randomizing d

- For a random r compute

$$\begin{aligned} d^* &= d + r \cdot \phi(n) \\ s &= m^{d^*} \pmod{n} \end{aligned}$$

- If $\phi(n)$ is unknown, compute

$$\begin{aligned} d^* &= d + r \cdot (e \cdot d - 1) \\ s &= m^{d^*} \pmod{n} \end{aligned}$$

- If e is unknown, for random $r < d$, compute

$$\begin{aligned} d^* &= d - r \\ s_1^* &= m^{d^*} \pmod{n} \\ s_2^* &= m^{r} \pmod{n} \\ s &= s_1^* \cdot s_2^* \pmod{n} \end{aligned}$$

DPA-Type Countermeasures — Randomizing n

- For short random numbers r_1 and $r_2 > r_1$, compute

$$
\begin{aligned}
m^* &= m + r_1 \cdot n \\
n^* &= r_2 \cdot n \\
s^* &= (m^*)^d \pmod{n^*} \\
s &= s^* \pmod{n}
\end{aligned}
$$

- For short random numbers r_1 and $r_2 > r_1$, compute

$$
\begin{aligned}
m^* &= m + r_1 \cdot n \\
n^* &= r_2 \cdot n \\
s^* &= (m^*)^d \pmod{n^*} \\
Y &= (m^*)^{d \bmod \phi(r_2)} \pmod{r_2} \\
c &= (S^* - Y + 1) \pmod{r_2} \\
s &= (s^*)^c \pmod{n}
\end{aligned}
$$

- Randomizing n also protects against fault attacks

Final Recommendations Against Side-Channel Attacks

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