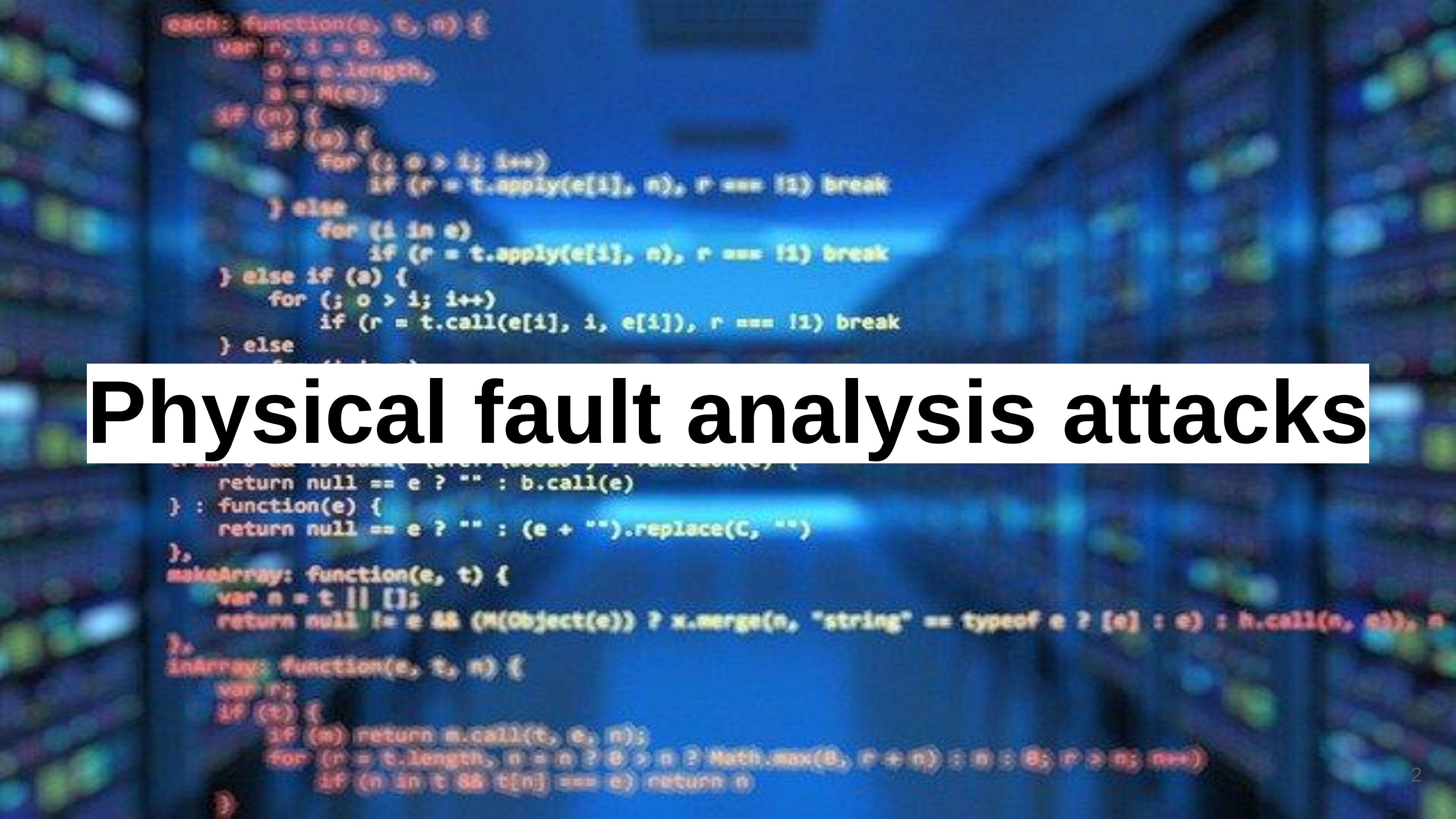
Cryptographic Engineering

L04: Physical fault analysis attacks + countermeasures

Prof. Nele Mentens

February 26, 2024

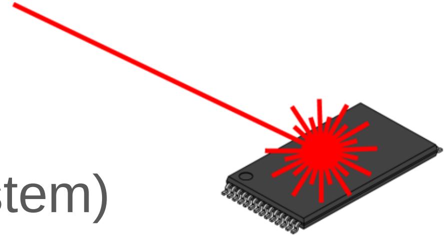


Physical fault analysis attacks

Fault analysis vs. side-channel analysis

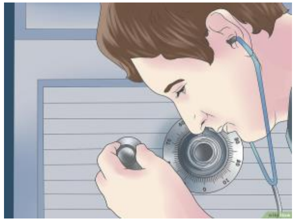
- Goal: introduce a computational fault to expose secrets or to enforce access rights

- Fault analysis attack
  = active (**introduces changes** to the running system)



(source: Ruhr university Bochum)

- Side-channel analysis attack
  = passive (**observes** the running system)



(source: wikiHow)

Simple example

Example: introduce a computational fault that forces the approval of the wrong PIN code

```
for (i = 0; i < 4; i++)
{
        if (pin[i] == input[i])
                ok_digits++;
}
if (ok_digits == 4)
        respond_code(NO_ERROR);
else
        respond_code(ERROR);
```

always execute this branch

Fault analysis

Goal: introduce a computational fault to expose secrets or to enforce access rights

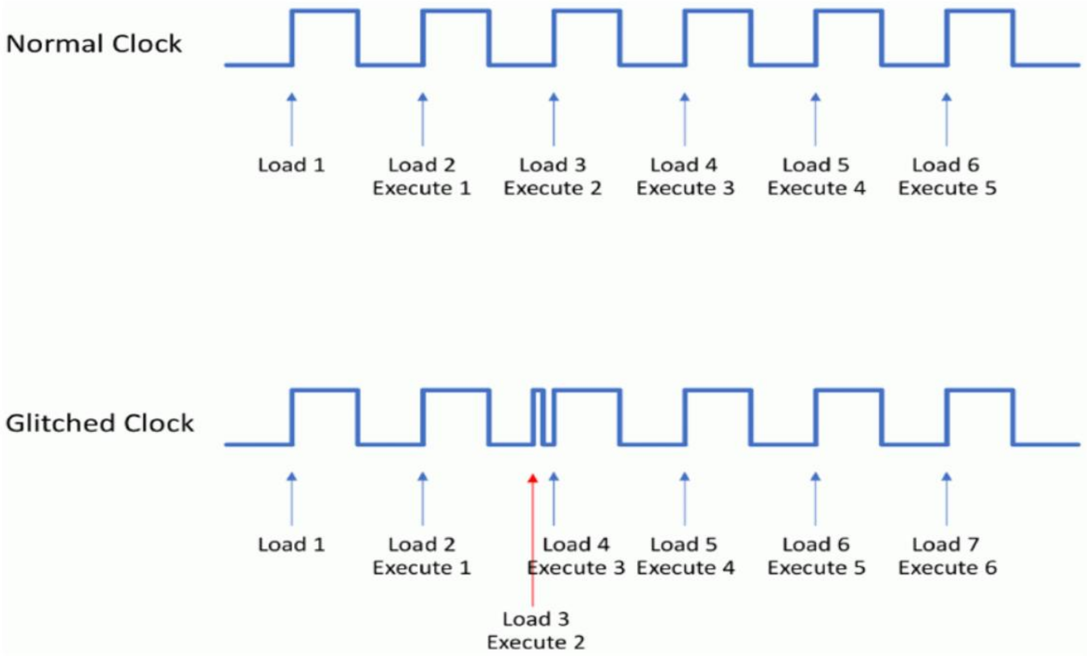
Fault injection

Fault exploitation

Fault injection - How to introduce a fault?

Overclocking: increase the clock frequency of the device such that one or more operations will not be successfully executed

- Each instruction needs a certain time to complete, regulated by the clock
- When the clock period decreases, the instruction does not finish in time

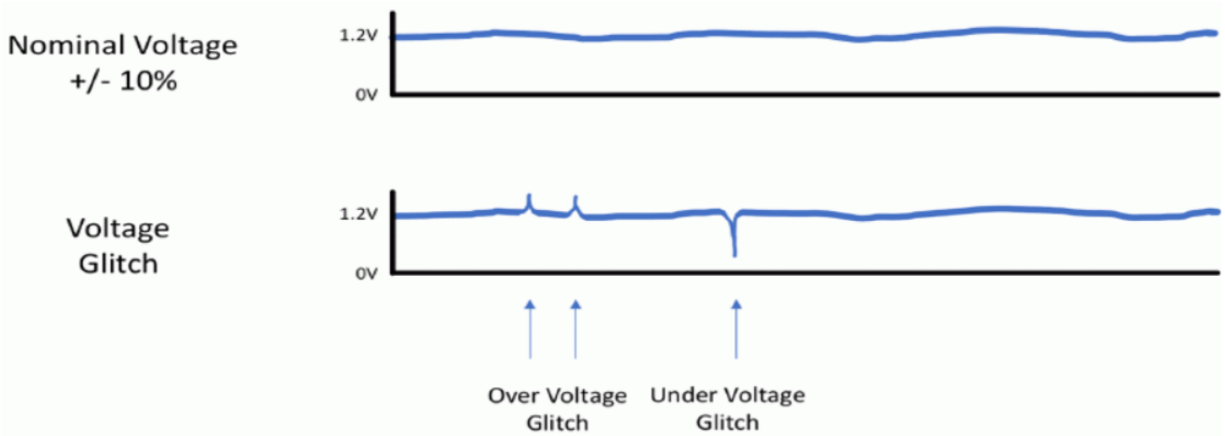


(source: eeweb)

Fault injection - How to introduce a fault?

Underpowering: reduce the power supply voltage of the device such that one or more operations will not be successfully executed

- Lowering the supply voltage will make the system slower
- Instructions running on a lower supply voltage do not complete within one clock period



(source: eeweb)

Fault injection - How to introduce a fault?

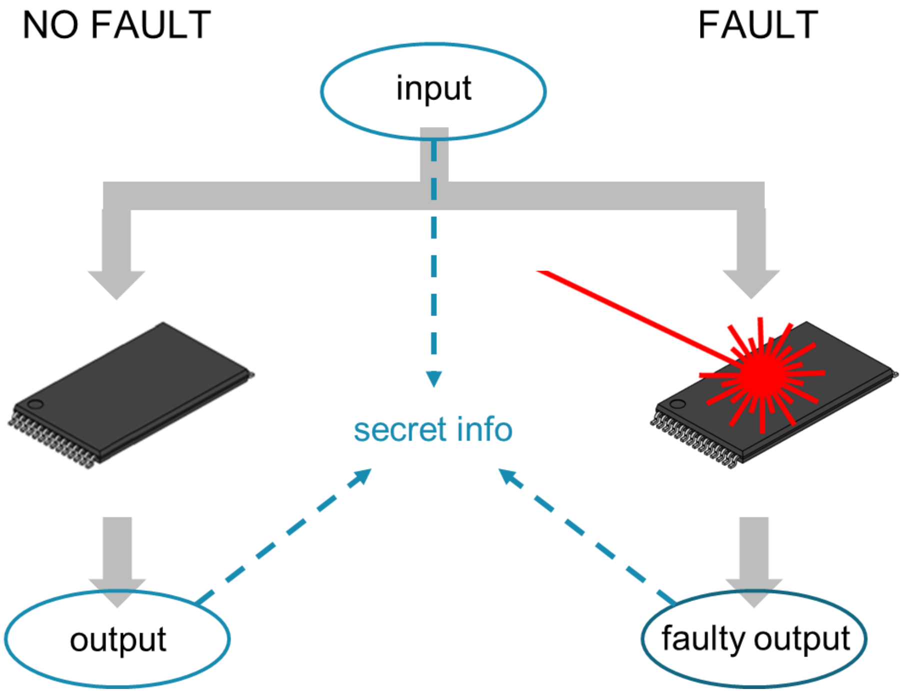
Optical fault injection: use a laser to make transistors switch

- The photons in the laser ionize the charge carriers in the semiconductor

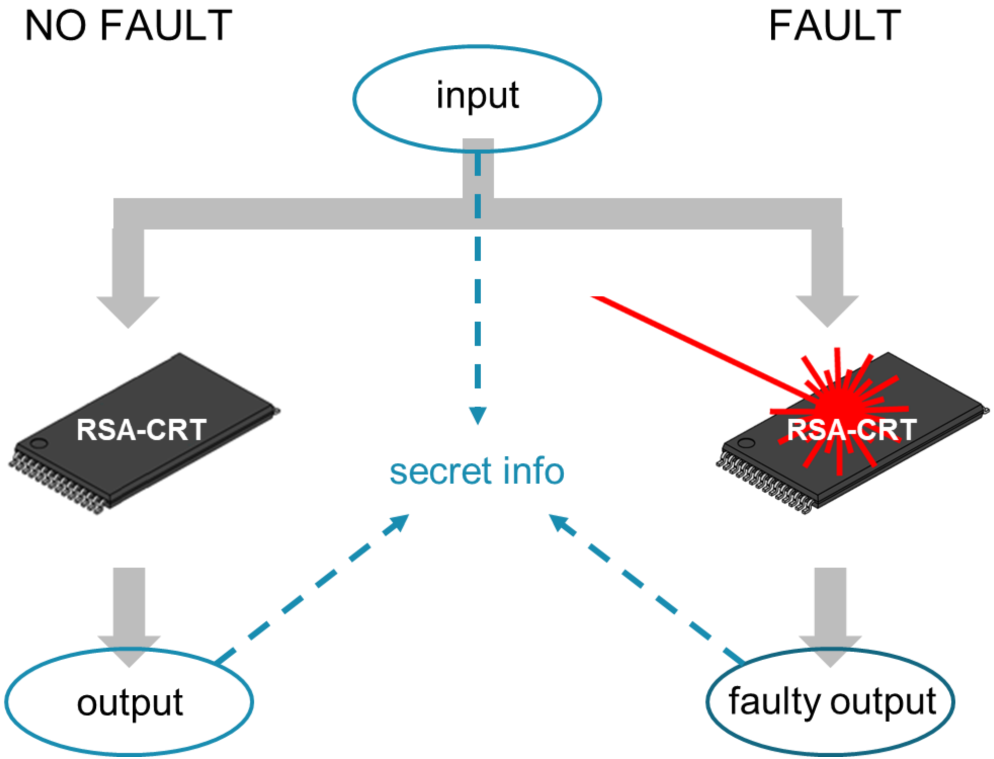
Electromagnetic fault injection: use electromagnetic pulses to make transistors switch

- An electromagnetic pulse generates an eddy current that makes a transistor switch

Fault exploitation – How to extract secret info from cryptographic algorithms?



Example: Bellcore attack on RSA with CRT



RSA with CRT (Chinese Remainder Theorem)

RSA signatures:

- Choose two different large primes (p and q) and compute n = p*q
- Compute $\lambda(n)$
  $= \text{lcm}(p-1,q-1)$
  $= (p-1)*(q-1)/\text{gcd}(p-1,q-1)$
  (lcm = least common multiple)
  (gcd = greatest common divider)
- Choose $e < \lambda(n)$ and relatively prime to $\lambda(n)$
- Calculate $d = e^{-1} \bmod \lambda(n)$

Public key = (e,n)
Private key = (p,q,d)

RSA with CRT (Chinese Remainder Theorem)

RSA signatures:

- Choose two different large primes (p and q) and compute $n = p*q$

- Compute $\lambda(n)$
  $= \mathrm{lcm}(p-1,q-1)$
  $= (p-1)*(q-1)/\gcd(p-1,q-1)$
  (lcm = least common multiple)
  (gcd = greatest common divider)

- Choose $e < \lambda(n)$ and relatively prime to $\lambda(n)$

- Calculate $d = e^{-1} \bmod \lambda(n)$

Public key = (e,n)
Private key = (p,q,d)

Signature generation without CRT: $s = m^d \bmod n$

RSA with CRT (Chinese Remainder Theorem)

RSA signatures:

- Choose two different large primes (p and q) and compute $n = p*q$
- Compute $\lambda(n)$
  $= lcm(p-1,q-1)$
  $= (p-1)*(q-1)/gcd(p-1,q-1)$
  (lcm = least common multiple)
  (gcd = greatest common divider)
- Choose $e < \lambda(n)$ and relatively prime to $\lambda(n)$
- Calculate $d = e^{-1} \bmod \lambda(n)$

Public key = (e,n)
Private key = (p,q,d)

Signature generation without CRT: $s = m^d \bmod n$

Signing with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Bellcore attack

- Insert fault in **one** of the partial signatures
- Compute the faulty signature ŝ
- Reveal p or q by computing gcd(s-ŝ,n)

Signing with CRT:

- Pre-compute dp = d mod (p-1)
- Pre-compute dq = d mod (q-1)
- Pre-compute pinv = p^{-1} mod q (with p < q)

Compute first partial signature sp = m^{dp} mod p
Compute second partial signature sq = m^{dq} mod q

- Compute signature
  s = (((sq-sp)*pinv) mod q)*p + sp mod n

Example Bellcore attack – correct signature

RSA signatures:

- Choose two different large primes (p and q) and compute n = p*q

$p = 19$
$q = 23$

- Compute $\lambda(n)$
  $= \text{lcm}(p-1,q-1)$
  $= (p-1)*(q-1)/\gcd(p-1,q-1)$
  (lcm = least common multiple)
  (gcd = greatest common divider)

$\lambda(n) = 18*22/\gcd(18,22) = 18*22/2 = 198$

- Choose $e < \lambda(n)$ and relatively prime to $\lambda(n)$

$e = 13$

- Calculate $d = e^{-1} \bmod \lambda(n)$

$d = 13^{-1} \bmod 198$
$= 61$

(because
$61*13 \bmod 198 = 1$)

Extended Euclidean algorithm

$198 = 15*13 + 3$
$13 = 4*3 + 1$

$1 = 13 - 4*3$
$1 = 13 - 4*(198-15*13)$
$1 = 61*13 - 4*198$

Public key = (e,n)
Private key = (p,q,d)

Example Bellcore attack – correct signature

From previous slide:
p = 19, q = 23, e = 13, d = 61

dp = 61 mod 18 = 7
dq = 61 mod 22 = 17
pinv = 19^{-1} mod 23 = -6 mod 23 = 17

23 = 1*19 + 4
19 = 4*4 + 3
4 = 1*3 + 1

1 = 4 - 1*3
1 = 4 - 1*(19-4*4)
1 = 5*4 - 1*19
1 = 5*(23-1*19) - 1*19
1 = 5*23-6*19

(because
-6*19 mod 23 = 1)

Signature generation without CRT: s = m^d mod n

Signature generation with CRT:

- Pre-compute dp = d mod (p-1)
- Pre-compute dq = d mod (q-1)
- Pre-compute pinv = p^{-1} mod q (with p < q)

- Compute first partial signature sp = m^{dp} mod p
- Compute second partial signature sq = m^{dq} mod q

- Compute signature
  s = (((sq-sp)*pinv) mod q)*p + sp mod n

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Example Bellcore attack – correct signature

From previous slide:
$p = 19$, $q = 23$, $e = 13$, $d = 61$

$dp = 61 \bmod 18 = 7$
$dq = 61 \bmod 22 = 17$
$pinv = 19^{-1} \bmod 23 = -6 \bmod 23 = 17$

E.g. $m = 123 \rightarrow sp = 123^7 \bmod 19 = 4$
$sq = 123^{17} \bmod 23 = 13$

$$
\begin{aligned}
123^7 \bmod 19 &= (123 \bmod 19)^7 \bmod 19 \\
&= 9^7 \bmod 19 \\
&= (9^2 \bmod 19)^3 * 9 \bmod 19 \\
&= 5^3 * 9 \bmod 19 = \mathbf{4} \\
123^{17} \bmod 23 &= (123 \bmod 23)^{17} \bmod 23 \\
&= 8^{17} \bmod 23 \\
&= ((8^2 \bmod 23)^2)^2)^2 * 8 \bmod 23 \\
&= (18^2 \bmod 23)^2)^2 * 8 \bmod 23 = \mathbf{13}
\end{aligned}
$$

Signature generation without CRT: $s = m^d \bmod n$

Signature generation with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Example Bellcore attack – correct signature

From previous slide:
$p = 19$, $q = 23$, $e = 13$, $d = 61$

$dp = 61 \bmod 18 = 7$
$dq = 61 \bmod 22 = 17$
$pinv = 19^{-1} \bmod 23 = -6 \bmod 23 = 17$

E.g. $m = 123 \rightarrow sp = 123^{7} \bmod 19 = 4$
$sq = 123^{17} \bmod 23 = 13$

$s = (((13-4)*17) \bmod 23)*19 + 4 \bmod 437$
= 289

Signature generation without CRT: $s = m^{d} \bmod n$

Signature generation with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$

Modular exponentiation with smaller modulus $\rightarrow$ cheaper computation in terms of energy, delay

Example Bellcore attack – faulty signature

Force one of the partial signatures to be wrong

$dp = 61 \bmod 18 = 7$
$dq = 61 \bmod 22 = 17$
$pinv = 19^{-1} \bmod 23 = -6 \bmod 23 = 17$

E.g. $m = 123 \rightarrow sp = 123^{7} \bmod 19 =$ **3**
$sq = 123^{17} \bmod 23 = 13$

Signature generation without CRT: $s = m^{d} \bmod n$

Signature generation with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $$s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$$

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Example Bellcore attack – faulty signature

Force one of the partial signatures to be wrong

$dp = 61 \bmod 18 = 7$
$dq = 61 \bmod 22 = 17$
$pinv = 19^{-1} \bmod 23 = -6 \bmod 23 = 17$

E.g. $m = 123 \rightarrow sp = 123^{7} \bmod 19 = \mathbf{3}$
$sq = 123^{17} \bmod 23 = 13$

$\hat{s} = (((13-\mathbf{3})*17) \bmod 23)*19 + \mathbf{3} \bmod 437$
$= \mathbf{174}$

Signature generation without CRT: $s = m^{d} \bmod n$

Signature generation with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Example Bellcore attack – faulty signature

Force one of the partial signatures to be wrong

$dp = 61 \bmod 18 = 7$
$dq = 61 \bmod 22 = 17$
$pinv = 19^{-1} \bmod 23 = -6 \bmod 23 = 17$

E.g. $m = 123 \rightarrow sp = 123^7 \bmod 19 = \mathbf{3}$
$sq = 123^{17} \bmod 23 = 13$

$\hat{s} = (((13-\mathbf{3})*17) \bmod 23)*19 + \mathbf{3} \bmod 437$
$= \mathbf{174}$

gcd(s-ŝ,n) = gcd(289-174,437)
= gcd(115,437)
= 23 → q revealed!!!

Signature generation without CRT: $s = m^d \bmod n$

Signature generation with CRT:

- Pre-compute $dp = d \bmod (p-1)$
- Pre-compute $dq = d \bmod (q-1)$
- Pre-compute $pinv = p^{-1} \bmod q$ (with $p < q$)

- Compute first partial signature $sp = m^{dp} \bmod p$
- Compute second partial signature $sq = m^{dq} \bmod q$

- Compute signature
  $s = (((sq-sp)*pinv) \bmod q)*p + sp \bmod n$

Modular exponentiation with smaller modulus → cheaper computation in terms of energy, delay

Exercise RSA-CRT and the Bellcore attack

Assume p = 5, q = 11 and e = 17.

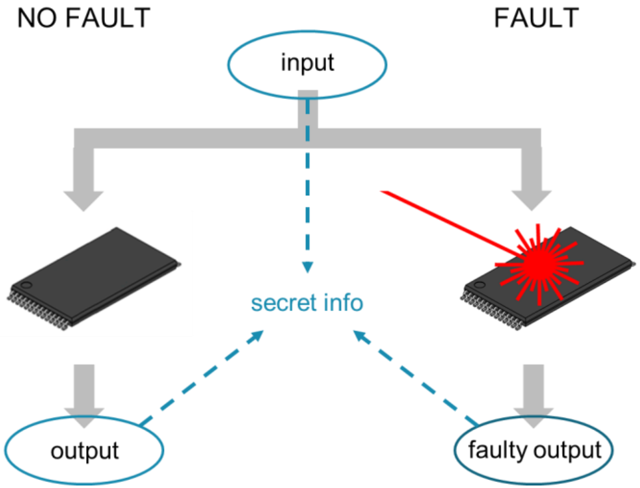
- Compute n and $\lambda(n)$.
- Compute d, the private key.
- Generate the digital signature of the message m = 6 using RSA-CRT.
- Introduce a random error in one of the partial signatures and show how the fault can reveal p or q.

t

Types of fault analysis attacks

Differential Fault Analysis (DFA):

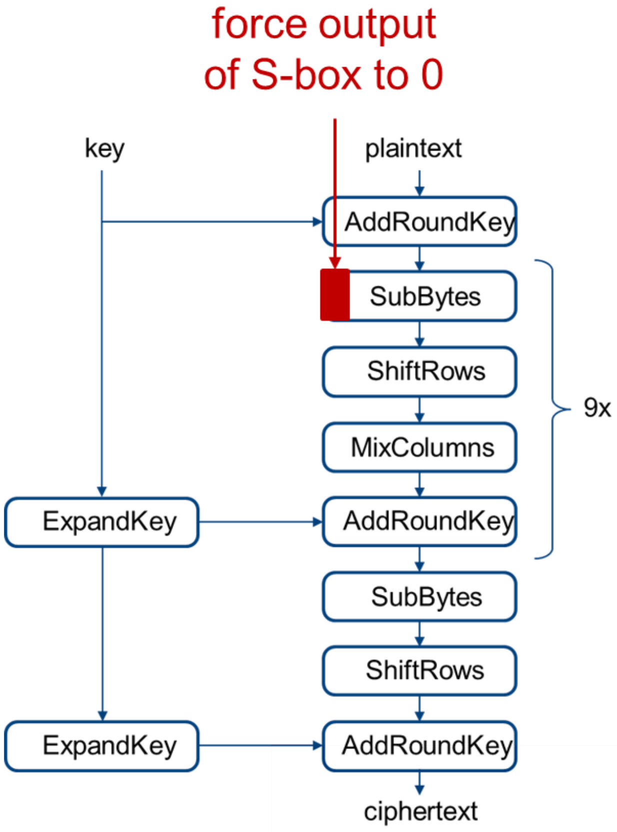
- Execute the cryptographic algorithm twice, once without a fault and once with a fault
- Use the outputs of both computations and compute the difference
- Extract secret information from this difference
- Only one fault needed
- Example:
    - the Bellcore attack: see previous slides
    - DFA on AES: see https://www.youtube.com/watch?v=uA6YjCrPqxE from 14’11” to 34’27”

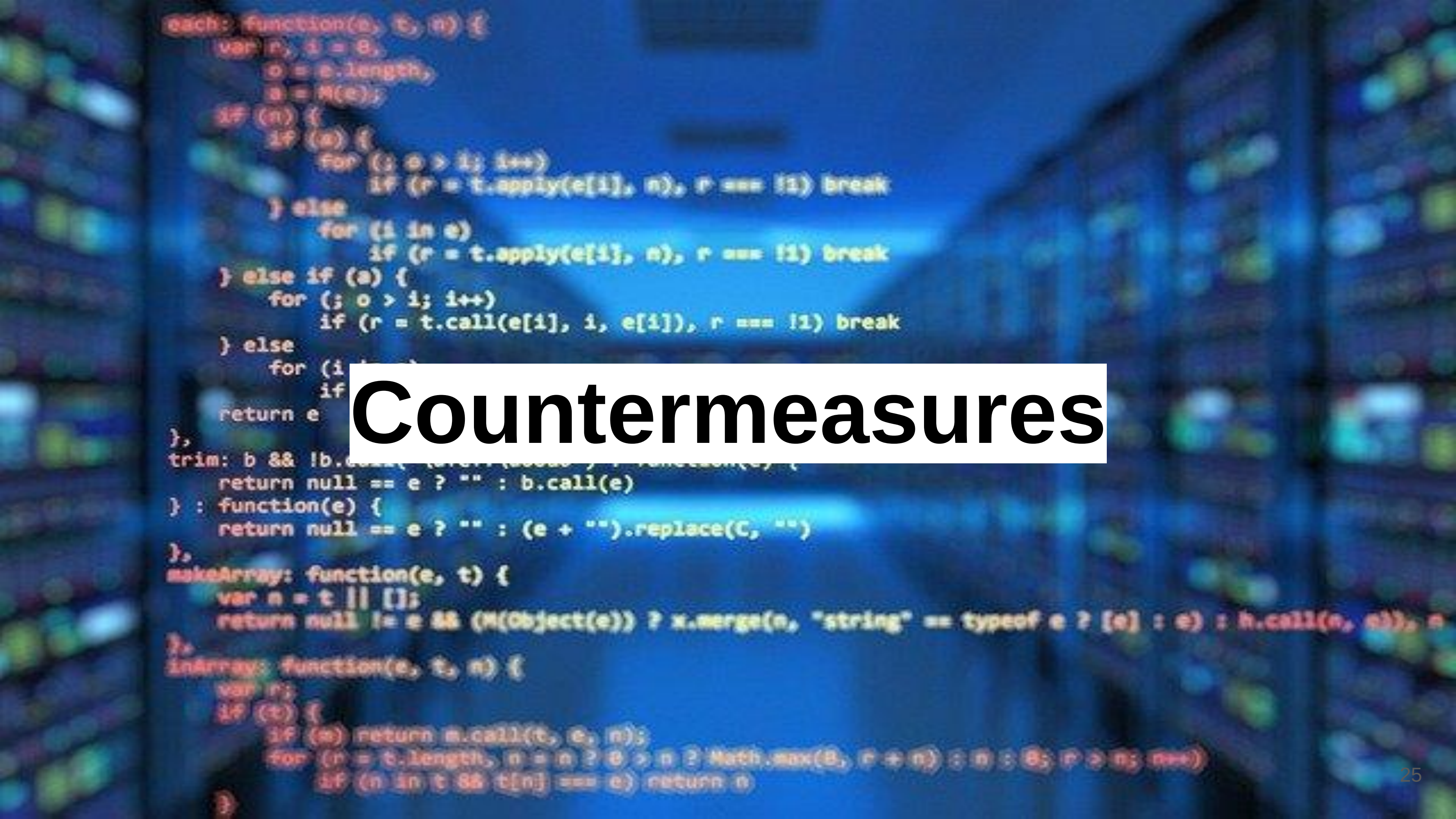


Types of fault analysis attacks

Collision Fault Analysis (CFA):

- Execute cryptographic algorithm once without fault, result is C
- Execute cryptographic algorithm many times with fault until a collision is found at the output, i.e. C = Ĉ
- Extract the partial key by calculating for which key value the equality holds
- Example:
    - Force the output of one 8-bit AES S-box to 0
    - When a plaintext is found for which C = Ĉ, try all possible partial keys (2^8 = 256 options) until the correct 8-bit partial key is found





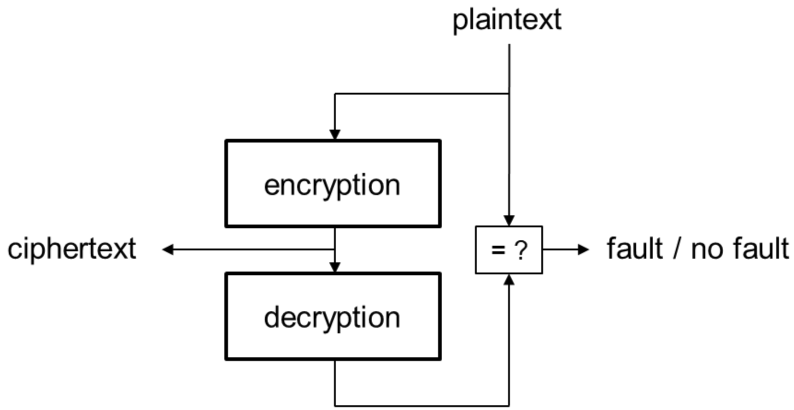
Countermeasures

Protection mechanisms against fault analysis

- Hide the sensitive operations to prevent targeted attacks
    - Temporal: hide the moment in time when an instruction is executed
    - Spatial: hide the location on the chip where certain functionality is implemented

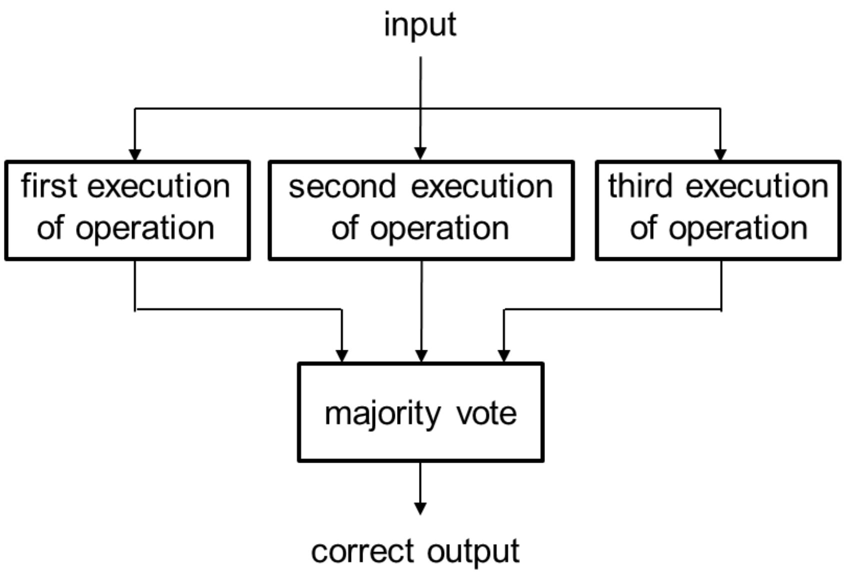
Protection mechanisms against fault analysis

- Detect the injection of a fault
  - Clock monitors
  - Supply voltage monitors
  - Execute each sensitive operation twice and check if the results are the same
  - Execute the inverse operation to check if there was a fault



Protection mechanisms against fault analysis

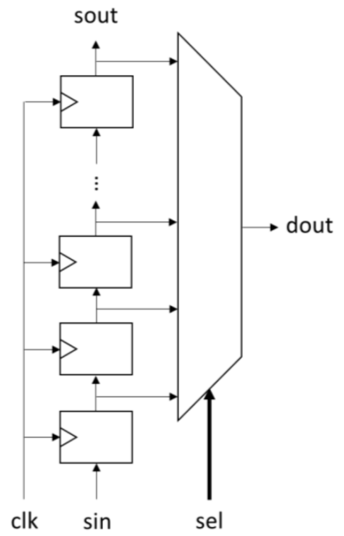
- Correct the fault
  - Triple Modular Redundancy (TMR) (in time or space)
  - Majority voting system to determine the correct output
    - Assuming it is very difficult to inject the same fault in all three executions of the same operation

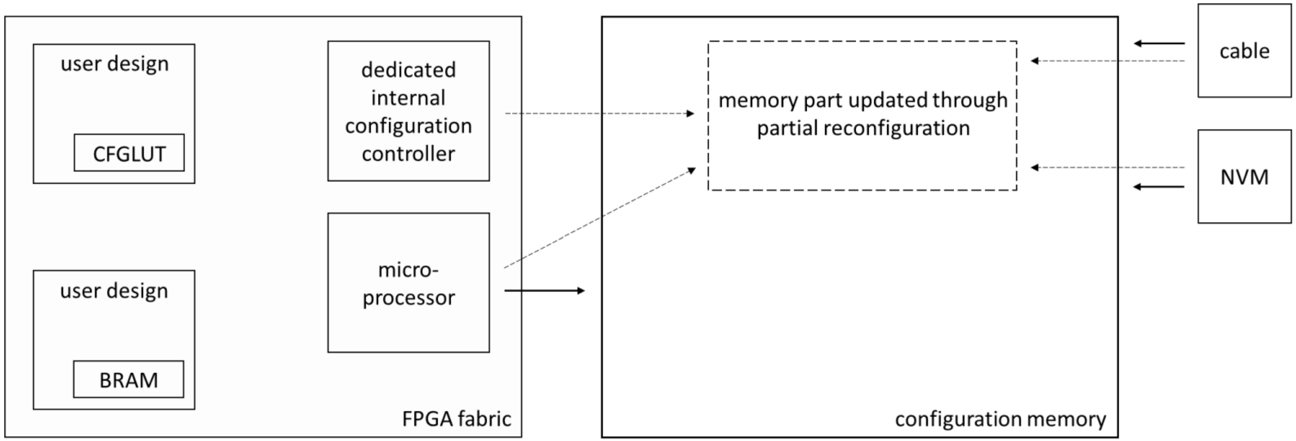


Morphing hardware architectures to protect against physical attacks

FPGA configuration mechanisms

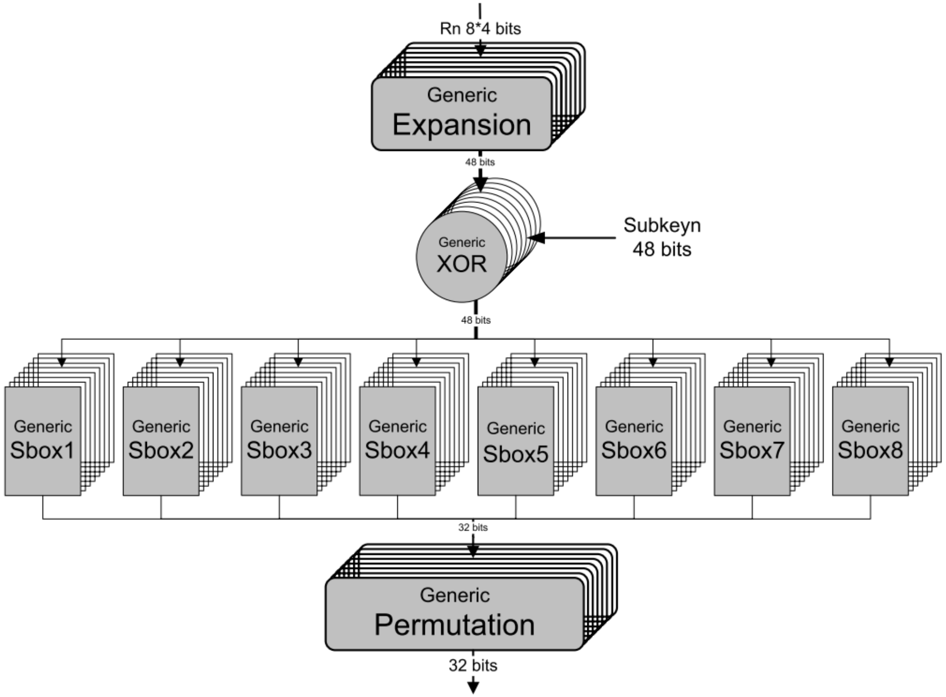
- Different configuration methods:
  - Full configuration
  - Partial configuration
  - Configuration through CFGLUTs
  - Configuration through BRAM updates





Morphing hardware architecture 1

- Randomized utilization of flip-flops and datapath elements (Poucheret et al., VLSI-SoC 2010)
  - Protection against local EM attacks
  - Applied to DES



Morphing hardware architecture 2

- Random shuffling of sub-operations (Stöttinger et al., Dynamically Reconfigurable Systems, 2010)
  - Protection against DPA attacks
  - Applied to elliptic curve cryptography

Short Weierstrass curve: $y^2 = x^3 + a*x + b$

Modified Jacobian coordinates: $x = X/Z_2$; $y = Y/Z_3$; $T = a * Z^4$

$(X_3,Y_3,Z_3,T_3) = (X_1,Y_1,Z_1,T_1) + (X_2,Y_2,Z_2,T_2)$

$ZZ_1 = Z_1^2$, $ZZ_2 = Z_2^2$

$U_1 = X_1*ZZ_2$, $U_2 = X_2*ZZ_1$

$S_1 = Y_1*Z_2*ZZ_2$, $S_2 = Y_2*Z_1*ZZ_1$

$H = U_2-U_1$, $I = (2*H)^2$

$J = H*I$, $r = 2*(S_2-S_1)$

$V = U_1*I$, $X_3 = r^2-J-2*V$

$Y_3 = r*(V-X^3)-2*S_1*J$

$Z_3 = ((Z_1+Z_2)^2-ZZ_1-ZZ_2)*H$

$ZZ_3 = Z_3^2$

$T_3 = a*ZZ_3^2$

$(X_3,Y_3,Z_3,T_3) = [2](X_1,Y_1,Z_1,T_1)$

$XX = X_1^2$

$A = 2*Y_1^2$

$AA = A^2$

$U = 2*AA$

$S = (X_1+A)^2-XX-AA$

$M = 3*XX+T_1$

$X_3 = M^2-2*S$

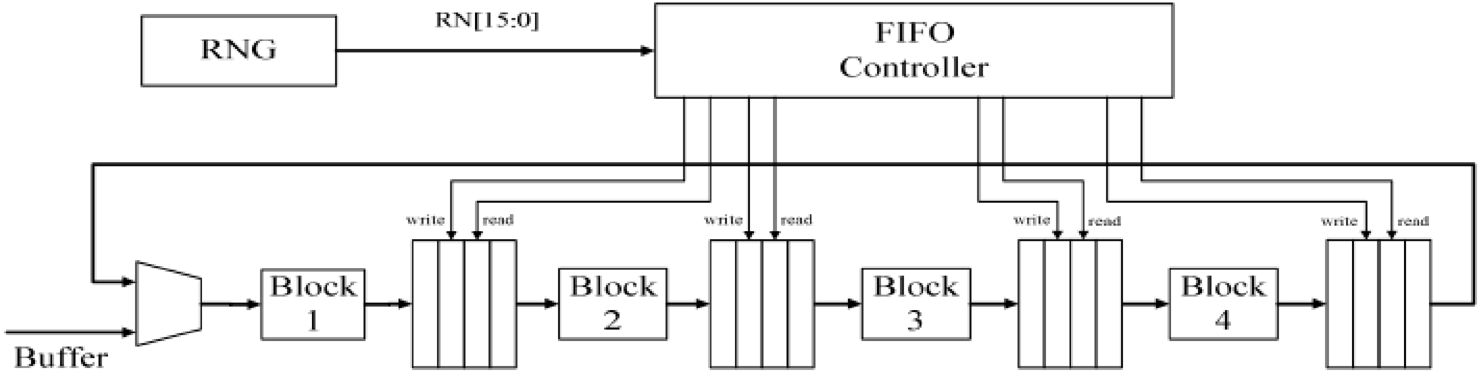
$Y_3 = M*(S-X_3)-U$

$Z_3 = 2*Y_1*Z_1$

$T_3 = 2*U*T_1$

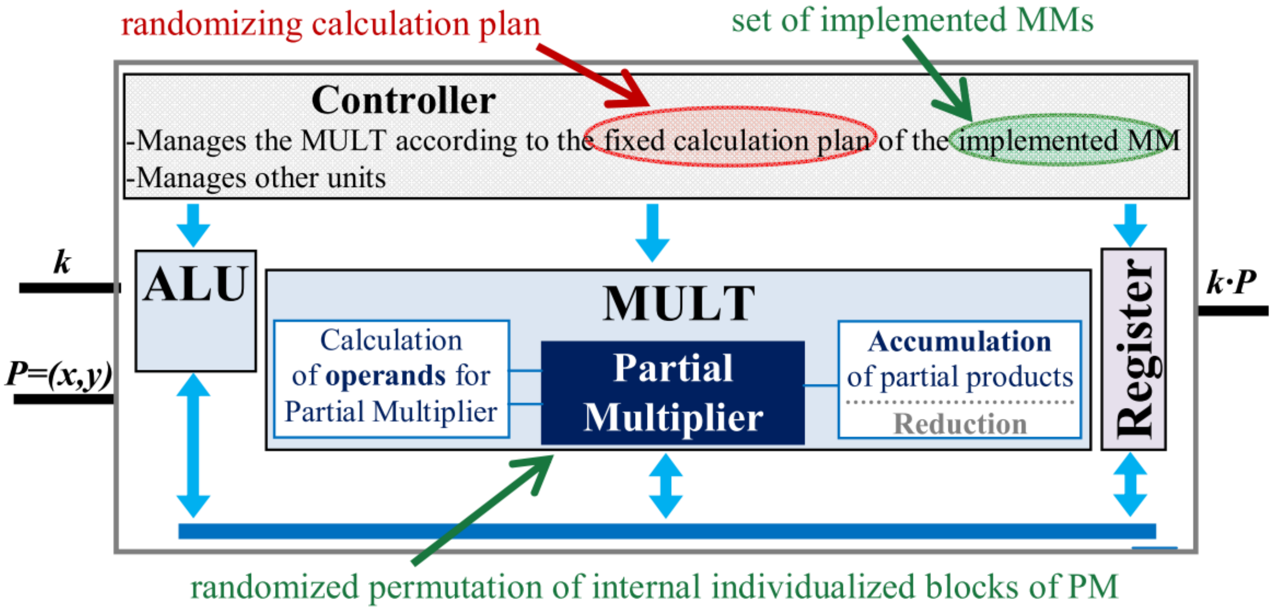
Morphing hardware architecture 3

- Insertion of random-delay FIFOs (Lin et al., IAS 2011)
    - Protection against power analysis attacks
    - Applied to AES



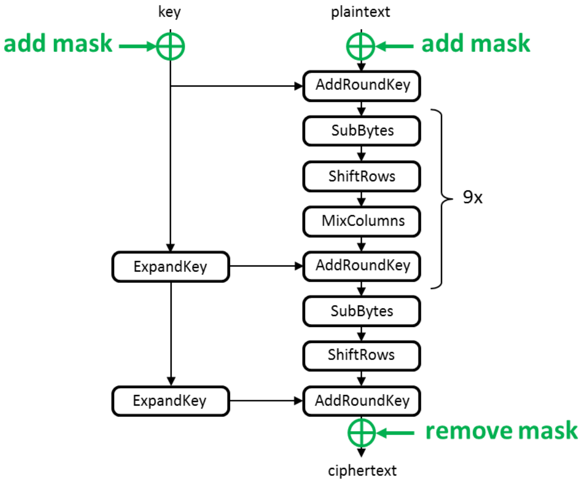
Morphing hardware architecture 4

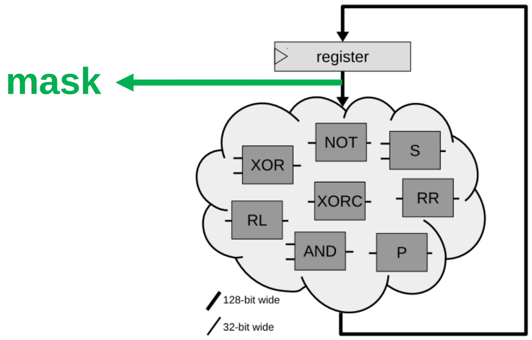
- Random selection and execution of functional units (Dyka et al., Euromicro 2015)
  - Protection against power analysis attacks
  - Applied to elliptic curve cryptography



Morphing hardware architecture 5

- For masking we need high-throughput pseudorandom number generators
- These PRNGs are susceptible to modelling/prediction attacks

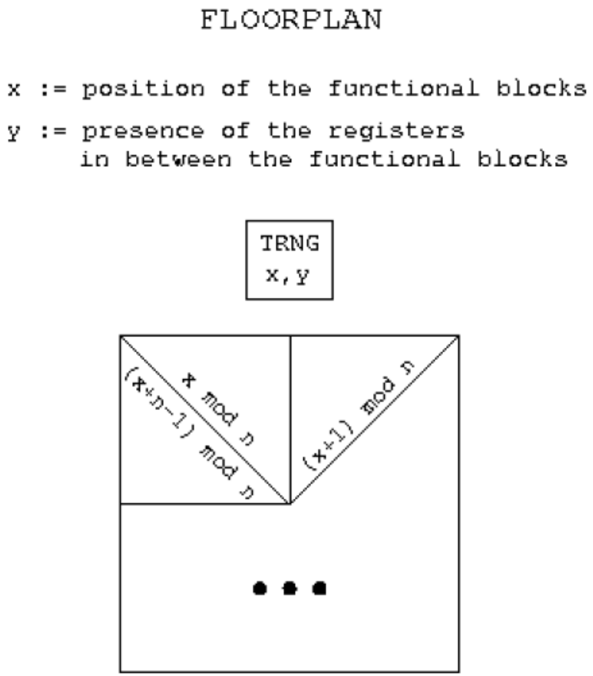


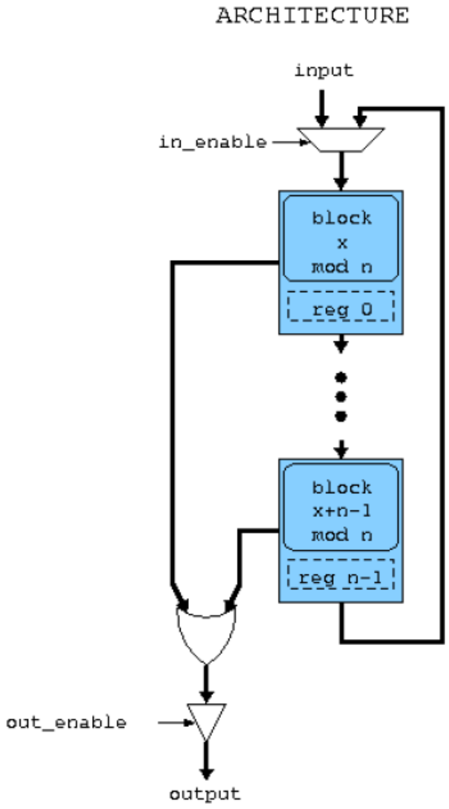


- Solution: Change the state update of a PRNG through run-time reconfiguration (Picek et al., CARDIS 2016)
- Generate new configurations that pass the NIST 800-22 statistical test suite through an evolutionary framework

Morphing hardware architecture 6

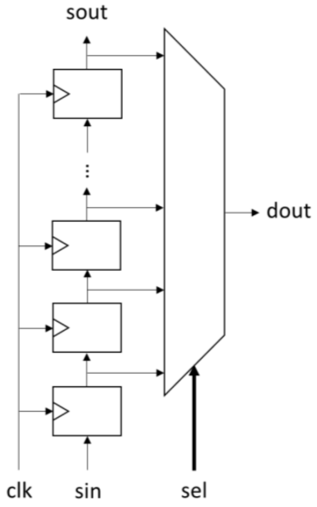
- Randomized location of pipeline registers and round functions (Mentens et al., CHES 2008)
  - Protection against power and fault analysis attacks
  - Applied to AES
  - Partial FPGA configuration





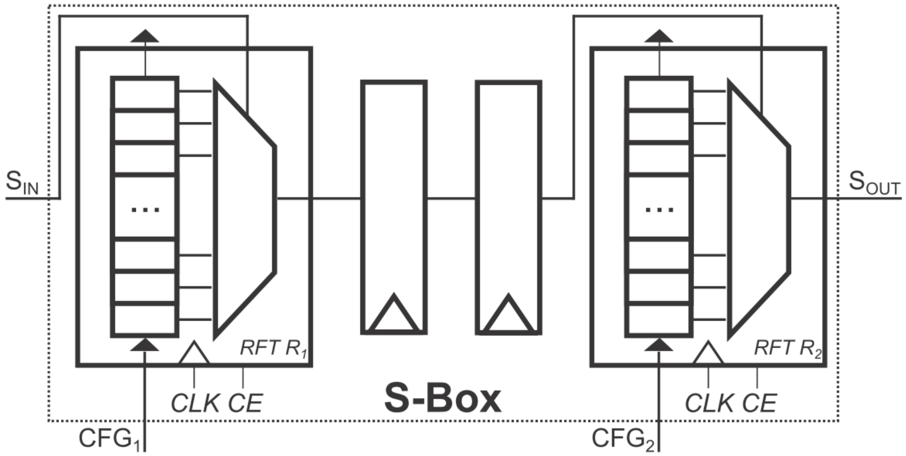
Morphing hardware architecture 7

- Addition of random power consumption (Güneysu et al., CHES 2011)
  - Protection against DPA attacks
  - Applied to AES
  - Using LUTs
    - Shift random values in shift-register LUTs
    - Create shortcuts in regular LUTs
  - Using Block RAM
    - Generate write collisions
    - Scramble the content for continuous remasking
  - Using on-chip digital clock managers (DCMs)
    - Create irregular clock cycle delays and phase shifts



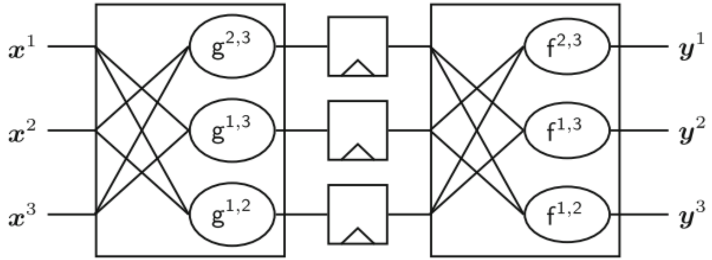
Morphing hardware architecture 8

- Randomized fine-grained configuration of LUTs, mask insertion and random register pre-loading (Sasdrich et al., HOST 2015)
  - Protection against DPA attacks
  - Applied to PRESENT
  - Using CFGLUTs



Morphing hardware architecture 9

- Random Substitution of Basic Elements and Random Encoding of Intermediate Connections (Sasdrich et al., CT-RSA 2017)
  - Protection against DPA attacks
  - Applied to a threshold implementation of PRESENT
  - Using BRAM updates



$$\mathsf{E}'_K = \underbrace{(\mathsf{f}^{r+1})^{-1} \circ \mathsf{E}^r_{k_r} \circ \mathsf{f}^r}_{table(s)} \circ \cdots \circ \underbrace{(\mathsf{f}^3)^{-1} \circ \mathsf{E}^2_{k_2} \circ \mathsf{f}^2}_{table(s)} \circ \underbrace{(\mathsf{f}^2)^{-1} \circ \mathsf{E}^1_{k_1} \circ \mathsf{f}^1}_{table(s)}$$