Countermeasures Against Power Analysis

Side-Channel Security

Rishub Nagpal

May 22, 2024

IAIK – Graz University of Technology

Overview

Recap

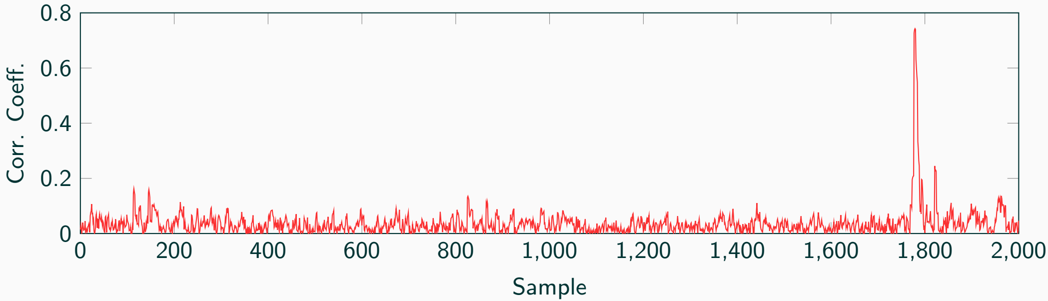
Constant Time/Control-flow Algorithms

Protocol Countermeasures

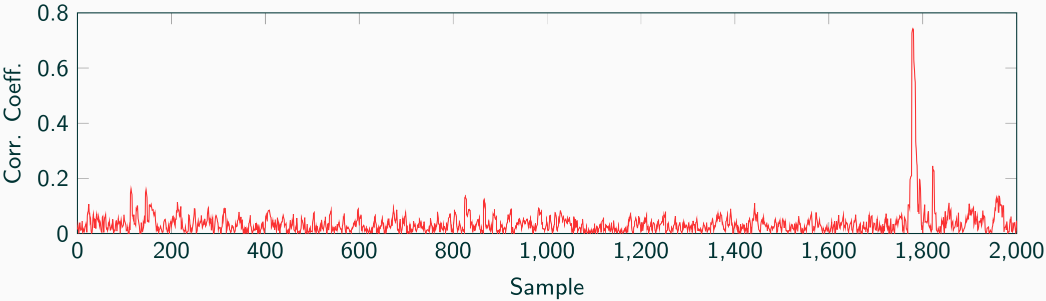
Algorithm-level Countermeasures

Case-study: Asymmetric Crypto

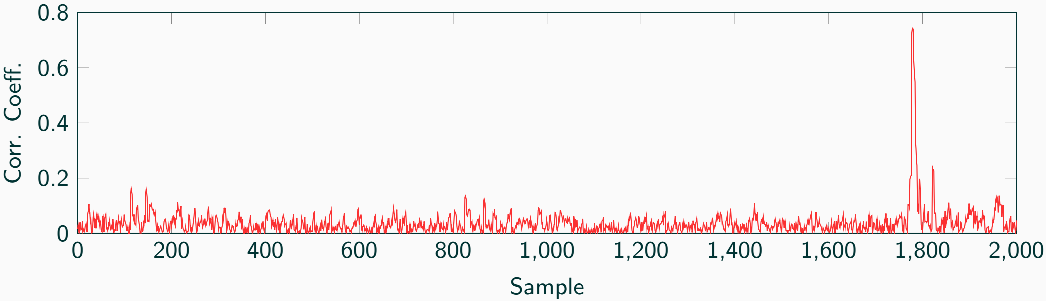
Recap



- Power analysis of symmetric crypto implementations



- Power analysis of symmetric crypto implementations
- DPA: Generic, yet powerful



- Power analysis of symmetric crypto implementations
- DPA: Generic, yet powerful
- Templates: More assumptions, but even stronger attacks

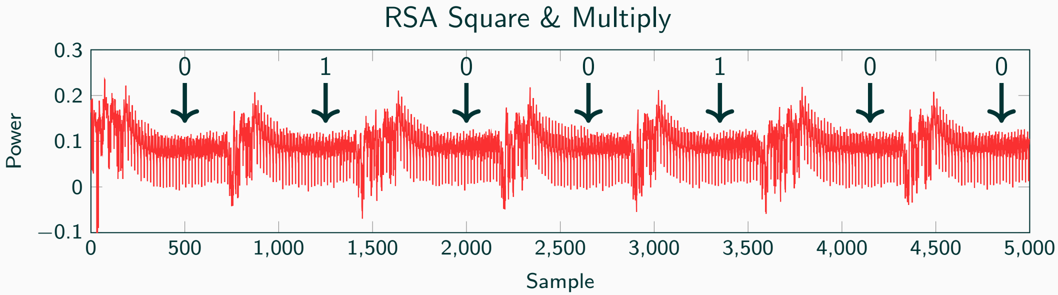
But we want to be the good guys!

- We want to build secure devices
- Protect against all sorts of side-channels (and fault attacks)
- Understanding and designing attacks is necessary
- Only then we can construct countermeasures

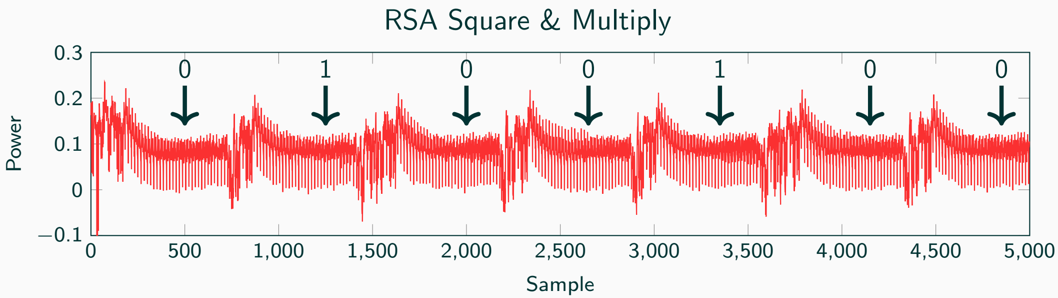
Recall: Scenario

- Device running crypto implementation
- Attacker wants to recover key
- Now: Countermeasures for crypto implementations
  - Tailored for crypto
  - To some extent applicable to non-crypto

Constant Time/Control-flow Algorithms



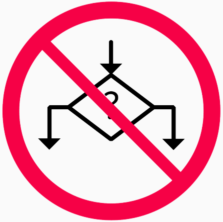
- Constant runtime algorithms
  - Defeates timing attacks



- Constant runtime algorithms
  - Defeates timing attacks
- Constant control flow
  - Defeates timing/cache attacks

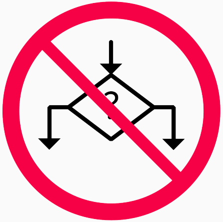
Constant Runtime & Control Flow

- No branching on secret data
  - Constant instruction sequence but different data



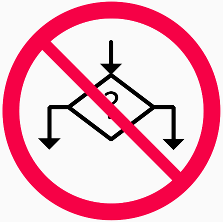
Constant Runtime & Control Flow

- No branching on secret data
  - Constant instruction sequence but different data
- Mind your hardware!
  - Table lookups depending on secret data
    → Cache attacks! Hardware inserts "implicit" branch!
  - Jump between idential code blocks with different constants
    → Pipeline flush!



Constant Runtime & Control Flow

- No branching on secret data
  - Constant instruction sequence but different data
- Mind your hardware!
  - Table lookups depending on secret data
    → Cache attacks! Hardware inserts "implicit" branch!
  - Jump between idential code blocks with different constants
    → Pipeline flush!
- No trivial "dummy" operations to compensate
  - E.g. insert NOPs to pad out
  - Detectable with power consumption



Pipelining

- Exact same instructions in `block1`, `block2`:
- But different constants
- Pipelining causes variable-time behavior

```
  :
  cmp eax, 1
  jne block2
block1:
  mov eax, 1
  shr ebx, 4
  xor dax, ebx
  :
  jmp end
block2:
  mov eax, 2
  shr ebx, 4
  xor dax, ebx
  :
  jmp end
end:
  :
```

- No table lookups depending on secrets, e.g.: SubBytes
  - At least on devices with cache (even some μC can have them...)

- No table lookups depending on secrets, e.g.: SubBytes
  - At least on devices with cache (even some μC can have them...)
- Alternative: Arithmetic descriptions of SubBytes

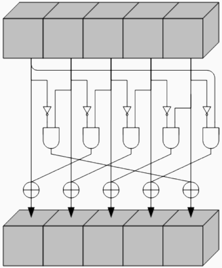
- No table lookups depending on secrets, e.g.: SubBytes
  - At least on devices with cache (even some μC can have them...)
- Alternative: Arithmetic descriptions of SubBytes
- AES: SubBytes $\approx$ Inversion in GF(2^8)
  - Can be done in constant time with e.g., Little Fermat: $x^{-1} = x^{254}$

Constant Runtime & Symmetric Crypto

- No table lookups depending on secrets, e.g.: SubBytes
  - At least on devices with cache (even some μC can have them...)
- Alternative: Arithmetic descriptions of SubBytes
- AES: SubBytes $\approx$ Inversion in GF(2^8)
  - Can be done in constant time with e.g., Little Fermat: $x^{-1} = x^{254}$
- Bitslicing: Find representation using bitwise operations
  - AND, XOR, ...
  - Can be executed in parallel for multiple state bytes
  - More on that next time!

- More recent cryptographic schemes:
  → S-box (SubBytes) already described that way
- Keccak hash function
  (Winner in the SHA3 competition)
- Ascon AEAD scheme
  (Winner in the CAESAR competition)

Keccak S-box



Constant-time crypto prevents many attacks (caches, timing,...)

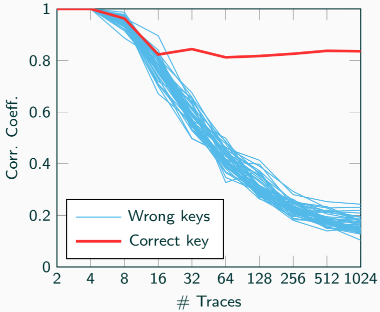
Constant-time crypto prevents many attacks (caches, timing,...)

... but not data-leakage $\rightarrow$ Power Analysis

Protocol Countermeasures

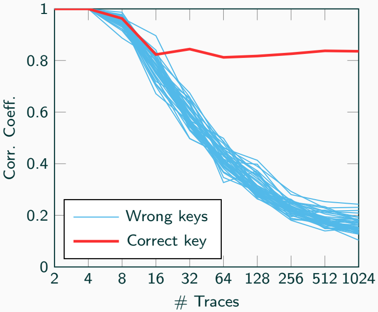
Protocol-Level Countermeasure: Frequent Key Update

- DPA requires multiple encryptions with constant key

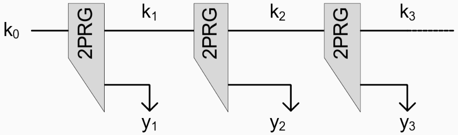


Protocol-Level Countermeasure: Frequent Key Update

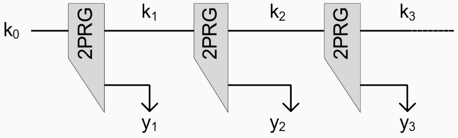
- DPA requires multiple encryptions with constant key
- Idea: Use key only for very small number of encryptions!
  - 2 encryptions per key:
    1. encrypt data
    2. generate new key
  - → not enough information for DPA



- Problems:
  - Usually requires synchronization of sender and receiver
  - Protocol and use-case specific

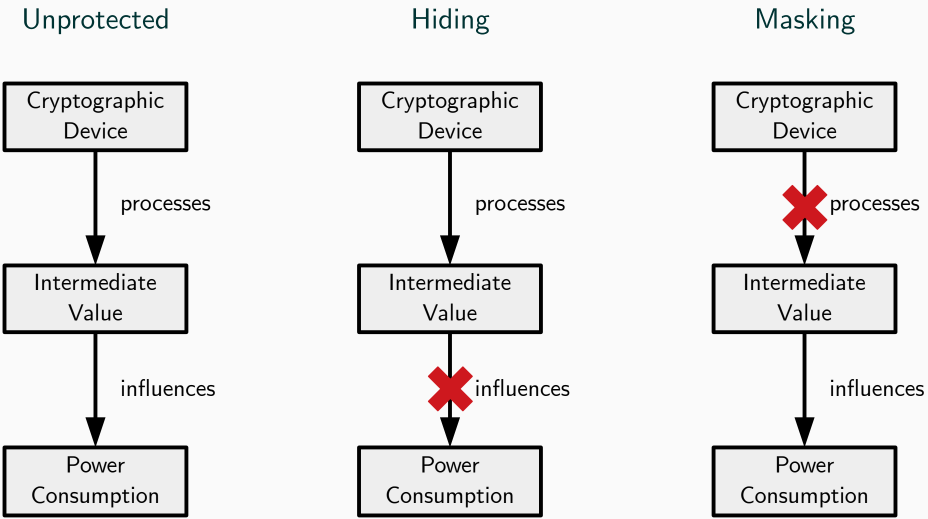


- Problems:
  - Usually requires synchronization of sender and receiver
  - Protocol and use-case specific
- Exceptions exist:
  - ISAP AEAD scheme:
    (NIST LWC standardization finalist)
  - Out-of-the-box DPA protection without further countermeasures
  - Standard AEAD interface (no synchronization needed)

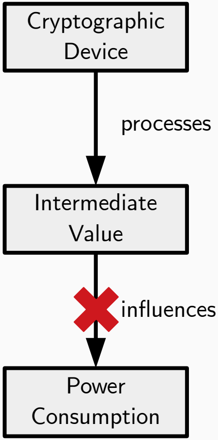


Algorithm-level Countermeasures

Scenarios


Unprotected
Hiding
Masking
Cryptographic Device
processes
Intermediate Value
influences
Power Consumption
Cryptographic Device
processes
Intermediate Value
influences
Power Consumption
Cryptographic Device
processes
Intermediate Value
influences
Power Consumption

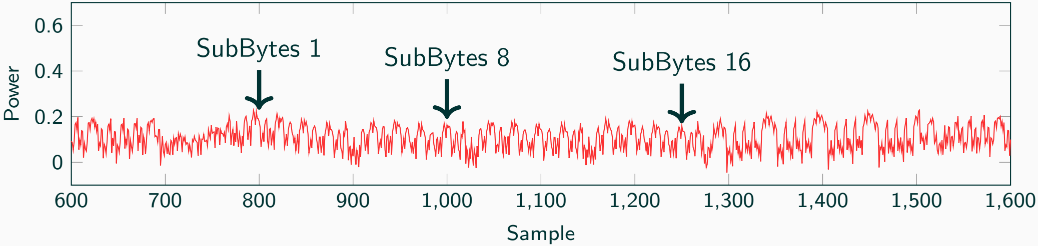
- Hide (reduce) the data-dependent power consumption



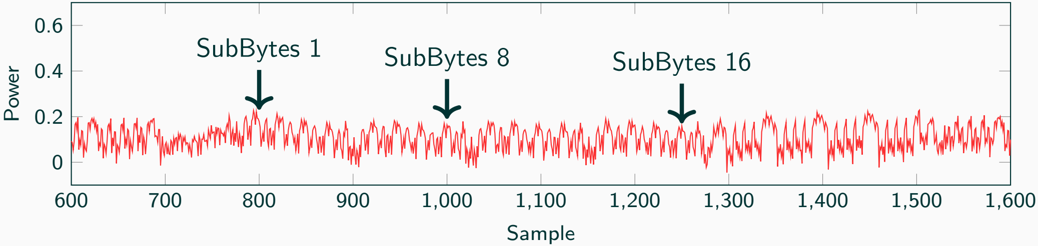
- Dedicated logic styles (Dual-Rail Precharge)
    - Precharge: Set wires to a fixed value (e.g. 0)
    - Dual-Rail: Evaluate both f and $\neg f$
    - $\rightarrow$ Overall switching activity is constant

Make Power Consumption Constant

- Dedicated logic styles (Dual-Rail Precharge)
  - Precharge: Set wires to a fixed value (e.g. 0)
  - Dual-Rail: Evaluate both f and $\neg f$
  - $\rightarrow$ Overall switching activity is constant
- Vastly improved security, but still problems
  - Expensive (chip size, runtime, development)
  - No perfect balancing possible (manufacturing variations)
  - Highly localized measurements might still work

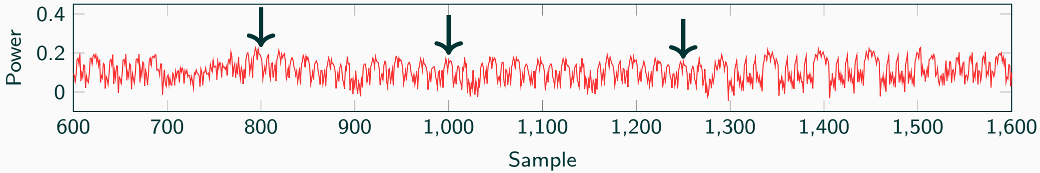


- Assumption of DPA:
  - Same operation at same instant in time

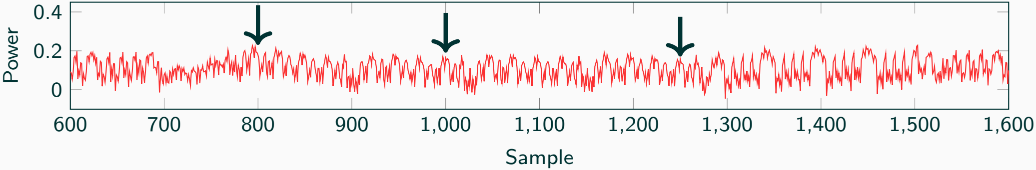


- Assumption of DPA:
  - Same operation at same instant in time
- Break assumption!

Hiding in Time - Methods

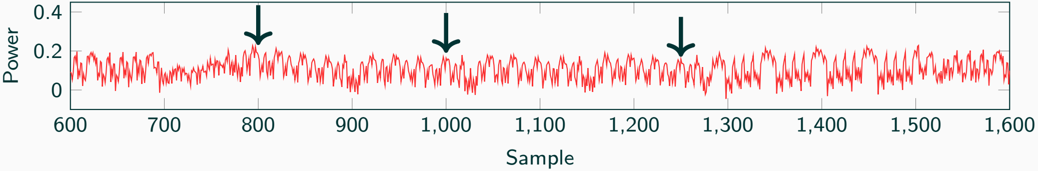


- Insert random number of dummy operations
  - Caution: must be the same basic operation
    (NOP and true S-box lookup are distinguishable → perform S-box on dummy data)
  - Total number of operations constant for all invocations
    (otherwise runtime gives away information)



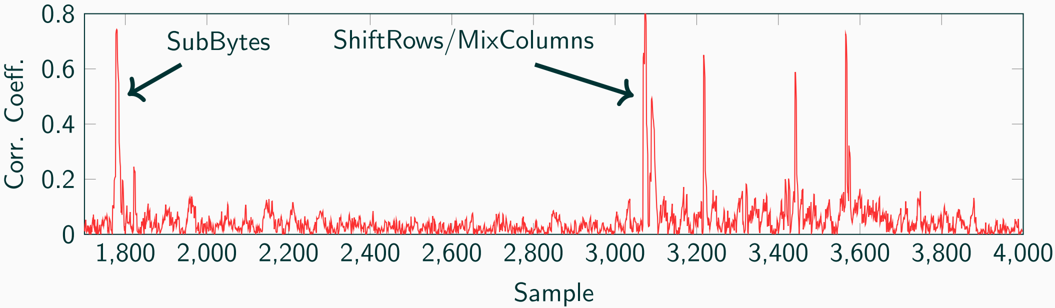
- Insert random number of dummy operations
  - Caution: must be the same basic operation (NOP and true S-box lookup are distinguishable $\rightarrow$ perform S-box on dummy data)
  - Total number of operations constant for all invocations (otherwise runtime gives away information)
- Shuffle the operations!
  - Process bytes in different order each time (in SubBytes, MixColumns...)
  - Limited by implemented algorithm (AES: 16 positions for SB, 4 positions for MC)

Hiding in Time - Methods



- Insert random number of dummy operations
  - Caution: must be the same basic operation
    (NOP and true S-box lookup are distinguishable → perform S-box on dummy data)
  - Total number of operations constant for all invocations
    (otherwise runtime gives away information)
- Shuffle the operations!
  - Process bytes in different order each time (in SubBytes, MixColumns...)
  - Limited by implemented algorithm (AES: 16 positions for SB, 4 positions for MC)
- Combination of both
  - e.g., 8 dummy S-boxes, then shuffle all 24 (dummy + real) S-boxes

Data leaks twice! (at least)



- Data leaks at each access
- AES: Compute S-box output, change address in ShiftRows, input to MixColumns
- Protecting just one operation (e.g., shuffling S-boxes) is pointless!
- Beware: 16x S-box, but only 4x MixColumns

Implementation of Shuffling

- Goal: compute 16 S-boxes in random order (each round)

Implementation of Shuffling

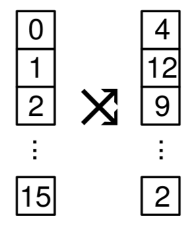
- Goal: compute 16 S-boxes in random order (each round)
- Random starting index (RSI)
  - Sample random index $r \in [0, 15]$
  - For $i = 0 \ldots 15$: compute S-box at $(r + i \mod 16)$
  - Problem: only 16 possibilities
  - Recover most likely r with template attack (attack addresses)

Implementation of Shuffling

- Goal: compute 16 S-boxes in random order (each round)
- Random starting index (RSI)
  - Sample random index $r \in [0, 15]$
  - For $i = 0 \ldots 15$: compute S-box at $(r + i \mod 16)$
  - Problem: only 16 possibilities
  - Recover most likely r with template attack (attack addresses)
- Random Permutation (RP)
  - Generate a random 16-permutation (vector $\mathbf{p}$ containing $0 \ldots 15$ in random order)
  - $16! \approx 2^{44}$ possibilities
  - For $i = 0 \ldots 15$: compute S-box at $\mathbf{p}(i)$
  - Recover most likely r with template attack (attack addresses)

- Efficient algorithm for generating a random permutation:

```
Initialize p with 0...15
for i from n-1 downto 1 do:
    j = random integer in [0,i]
    exchange p[j] and p[i]
```

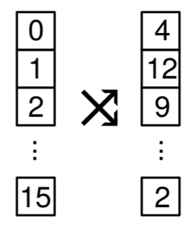


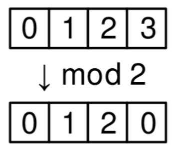
The Fisher-Yates Shuffle

- Efficient algorithm for generating a random permutation:

```
Initialize p with 0...15
for i from n-1 downto 1 do:
    j = random integer in [0,i]
    exchange p[j] and p[i]
```

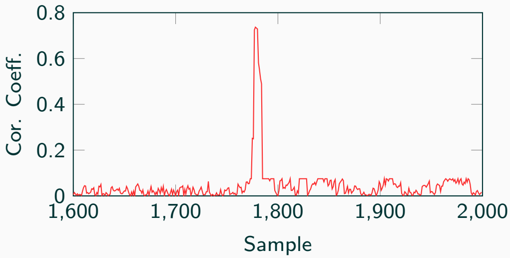
- Sampling in $[0, i]$ can be tricky...
  - $r \mod (i+1) \rightarrow$ mod not constant time!
  - Replace with $[0, n-1]$
    much faster but bias $\rightarrow$ side-channel leak

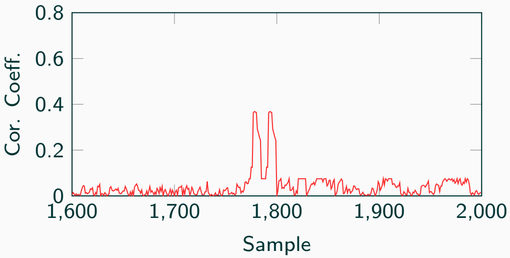




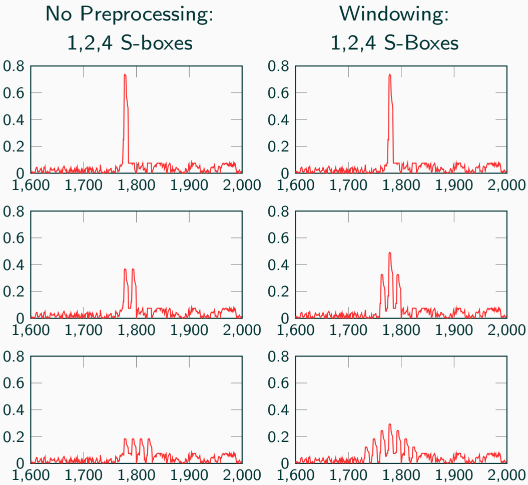
Effect on DPA

- DPA still possible, but increased noise
- ρ goes down linearly with #possible positions
- $\rightarrow$ #traces grows quadratically





- Windowing
  - Sum up power consumption over all possible positions
  - Perform DPA on processed traces
  - Result: ρ goes down with square root of #possible positions
  - $\rightarrow$ Only linearly more traces!
- But still. . .
  - Finding all positions might not be easy
  - Still effective in combination with other countermeasures

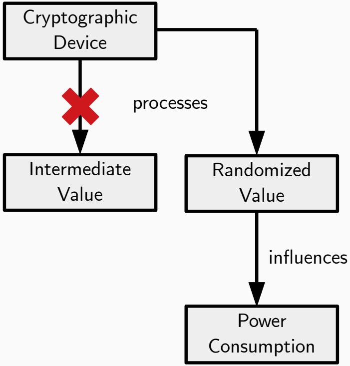


- Algebraic/Analytical Attacks
  - Perform template attack
  - Plug values into equations describing AES
  - Solve equations (e.g., SAT solvers, graphical models,. . .)
  - Often just 1 (averaged) trace
- Examples:
  - Measure Hamming weights in AES key schedule to reduce keyspace to bruteforce complexity
  - Collision attacks: Detect that two S-boxes have same input by comparing traces and building equations that can be solved

- Algebraic attacks are very noise sensitive
  - Some need perfect Hamming weights, collisions with 100% certainty,...
  - Single error → attack fails
  - Others can deal with some errors
- Shuffling is very effective against algebraic attacks!
  - 2 S-boxes collide → but which?
  - Hamming weights of round keys → but in which order?

Masking (Secret Sharing)

- Operate on randomized intermediates
- Side-channel information on randomized intermediate does not help attacker
- But still require correct algorithm output



Secret Sharing Intuition

- We want to compute f on input x and secret s ...
  - But avoid using s directly

$$f(x, s) = y$$

Secret Sharing Intuition

- We want to compute f on input x and secret $s\ldots$
  - But avoid using s directly
- Idea: Split s into e.g. 3 shares s_1, s_2, s_3 such that:
  - Individual shares do not reveal s
  - Each 2-combination of shares does not reveal s
  - The computed y_1, y_2, y_3 can be combined to y

$$f(x, s) = y$$

$$f(x_1, s_1) = y_1$$

$$f(x_2, s_2) = y_2$$

$$f(x_3, s_3) = y_3$$

$$y = y_1 \circ y_2 \circ y_3$$

Secret Sharing Intuition

- We want to compute f on input x and secret $s\dots$
  - But avoid using s directly
- Idea: Split s into e.g. 3 shares s_1, s_2, s_3 such that:
  - Individual shares do not reveal s
  - Each 2-combination of shares does not reveal s
  - The computed y_1, y_2, y_3 can be combined to y
- For technical reasons:
  - Split x into 3 shares x_1, x_2, x_3 as well

$$f(x, s) = y$$

$$f(x_1, s_1) = y_1$$

$$f(x_2, s_2) = y_2$$

$$f(x_3, s_3) = y_3$$

$$y = y_1 \circ y_2 \circ y_3$$

- Application to crypto operations:
  - Split key k into k_1, k_2, k_3
  - Split plaintext x into x_1, x_2, x_3
  - Compute ciphertext $y = y_1 \circ y_2 \circ y_3$
  - (Use new shares for each encryption!)

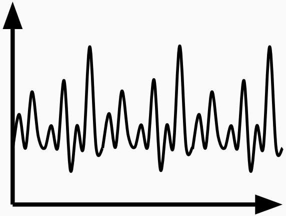
$$\text{enc}(x_1, k_1) = y_1$$
$$\text{enc}(x_2, k_2) = y_2$$
$$\text{enc}(x_3, k_3) = y_3$$
$$y = y_1 \circ y_2 \circ y_3$$

Secret Sharing Intuition

- Application to crypto operations:
  - Split key k into k_1, k_2, k_3
  - Split plaintext x into x_1, x_2, x_3
  - Compute ciphertext $y = y_1 \circ y_2 \circ y_3$
  - (Use new shares for each encryption!)

$$\text{enc}(x_1, k_1) = y_1$$
$$\text{enc}(x_2, k_2) = y_2$$
$$\text{enc}(x_3, k_3) = y_3$$
$$y = y_1 \circ y_2 \circ y_3$$

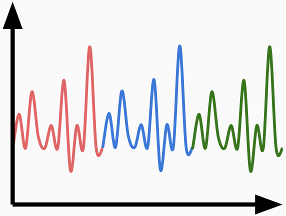
- But we cannot distribute the computation over multiple devices...
  - So we do secret sharing with ourself!?



- Application to crypto operations:
  - Split key k into k_1, k_2, k_3
  - Split plaintext x into x_1, x_2, x_3
  - Compute ciphertext $y = y_1 \circ y_2 \circ y_3$
  - (Use new shares for each encryption!)

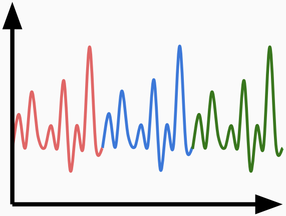
$$\mathrm{enc}(x_1, k_1) = y_1$$
$$\mathrm{enc}(x_2, k_2) = y_2$$
$$\mathrm{enc}(x_3, k_3) = y_3$$
$$y = y_1 \circ y_2 \circ y_3$$

- But we cannot distribute the computation over multiple devices...
  - So we do secret sharing with ourself!?
  - Yeah! Remember: $k_1, k_2, k_3 \rightarrow$

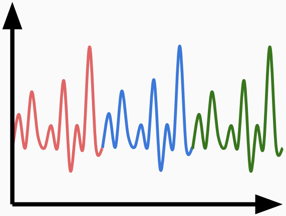


Masking Prerequisite

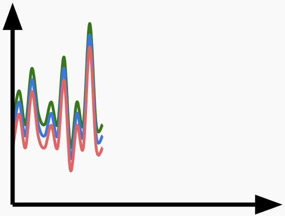
- Required condition for masking to protect against DPA:
  - No joint power consumption of shares
  - (No errors in masking scheme and in implementation)
- For attack: Force violation by joining power consumptions up again



- Sequential processing of shares (typical in software)



- Sequential processing of shares (typical in software)



- Parallel processing of shares (typical in hardware)
- Faster but comes with a catch...

Can we still attack that?

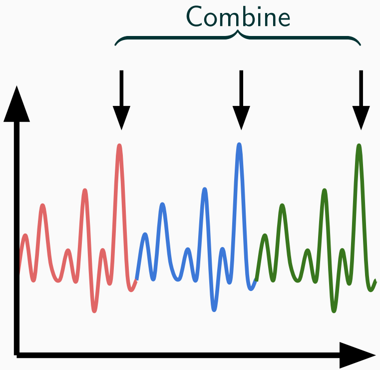
- Sequential computation
  - Shares processed at different time

Can we still attack that?

- Sequential computation
  - Shares processed at different time
- Attack: Combine them again
  - Identify times of processing
  - Cut traces and get multiple power measurements
  - Combine them together

Can we still attack that?

- Sequential computation
  - Shares processed at different time
- Attack: Combine them again
  - Identify times of processing
  - Cut traces and get multiple power measurements
  - Combine them together
- Combination function
  - Chosen s.t. we learn about k



Higher-Order DPA

1. Preprocessing
    - Use a combination function to combine points in trace
2. DPA attack
    - Perform a standard (first-order) DPA on preprocessed traces

1. Preprocessing
    - Use a combination function to combine points in trace
2. DPA attack
    - Perform a standard (first-order) DPA on preprocessed traces

- Nomenclature:
    - "Higher-order": prediction and/or dependence on both multiple shares
    - Just using multiple points is not necessarily "higher order" (e.g., template attacks)

- First hurdle:
  - Attacker usually does not know a-priori when shares are processed
  - "Solution": Pair-wise combination of large range of points in trace
  - $\rightarrow$ Quadratic growth of computational complexity
  - Designer: Use that, make it hard to find out when shares are processed

- First hurdle:
    - Attacker usually does not know a-priori when shares are processed
    - "Solution": Pair-wise combination of large range of points in trace
    - → Quadratic growth of computational complexity
    - Designer: Use that, make it hard to find out when shares are processed
- Combination of two points
    - Assumption: Hamming-weight leakage
    - → At correct point combination, we get noisy leakage of the shares
    - Want: Combination correlates with HW(v)

- Addition?
    - Correlation $\rho(\mathrm{HW}(v), \mathrm{HW}(v_1) + \mathrm{HW}(v_2)) = 0$
    - $\rightarrow$ Attack fails

Higher-Order DPA: Combination Function

- Addition?
  - Correlation $\rho(\mathrm{HW}(v), \mathrm{HW}(v_1) + \mathrm{HW}(v_2)) = 0$
  - $\rightarrow$ Attack fails
- Better (ideal) Centered Product Combination
  - Centered: Subtract mean from each point in time
  - Product: Multiply sample values with each other

Higher-Order DPA: Combination Function

- Addition?
  - Correlation $\rho(\text{HW}(v), \text{HW}(v_1) + \text{HW}(v_2)) = 0$
  - $\rightarrow$ Attack fails
- Better (ideal) Centered Product Combination
  - Centered: Subtract mean from each point in time
  - Product: Multiply sample values with each other
- Example with 8 bits (no noise)
  - Mean $m = (HW(0 \ldots 255)) = 4$
  - Combined power $p_c = (HW(v_1) - m) \times (HW(v_2) - m)$
  - $\rho(HW(v), p_c) = -0.35$

Higher-Order Template Attacks

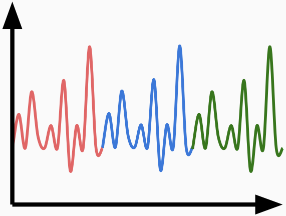
- What about template attacks?

- What about template attacks?
- Either: Templates on preprocessed traces
    - Both profiling and attacking

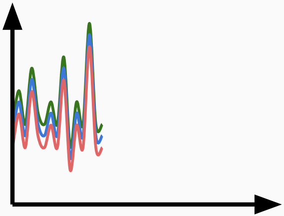
- What about template attacks?
- Either: Templates on preprocessed traces
  - Both profiling and attacking
- Or: Templates on each share
  - Get two probability vectors: $p(v_1|t)$, $p(v_2|t)$ for all values of v_1, v_2
  - Combine probabilities:

$$p(v|t) = \sum_{(v_1,v_2):v_1 \oplus v_2 = v} p(v_1|t)p(v_2|t)$$

Masking: Two Cases



- Sequential processing of shares (typical in software)



- Parallel processing of shares (typical in hardware)
- What about this case?

Higher-Order DPA: Parallel Case

- Good for attacker: Shares already leak at same time
  - → No need to combine all possible points

Higher-Order DPA: Parallel Case

- Good for attacker: Shares already leak at same time
  - $\rightarrow$ No need to combine all possible points
- Bad for attacker: Power consumption adds up...
  - Power $\approx HW(v_1) + HW(v_2)$
  - But correlation $\rho(\mathrm{HW}(v), \mathrm{HW}(v_1) + \mathrm{HW}(v_2)) = 0 \ldots$

Higher-Order DPA: Parallel Case

- Good for attacker: Shares already leak at same time
  - $\rightarrow$ No need to combine all possible points
- Bad for attacker: Power consumption adds up...
  - Power $\approx HW(v_1) + HW(v_2)$
  - But correlation $\rho(\text{HW}(v), \text{HW}(v_1) + \text{HW}(v_2)) = 0 \ldots$
- Solution: Squaring traces
  - $\rho(HW(v), (HW(v_1) + HW(v_2))^2) = -0.04$
  - A lot lower, but it works...

Higher-Order Masking

- Masked: harder to attack, but still possible...
  - Add more masks!
  - Same attacks still apply, but even harder

Higher-Order Masking

- Masked: harder to attack, but still possible...
  - Add more masks!
  - Same attacks still apply, but even harder
- d-th order masking
  - Withstands d-th order attacks (e.g., combine d points, take trace to d-th power)
  - Needs (at least) $d + 1$ shares

Higher-Order Masking

- Masked: harder to attack, but still possible...
  - Add more masks!
  - Same attacks still apply, but even harder
- d-th order masking
  - Withstands d-th order attacks (e.g., combine d points, take trace to d-th power)
  - Needs (at least) $d + 1$ shares
- Security gain: exponential in d
- (More information next lecture)

Countermeasures - Recap

- Main techniques
  - Constant time/control-flow implementations
  - Protocol-level: Key update
  - Algorithm-level: Hiding, Masking

Countermeasures - Recap

- Main techniques
    - Constant time/control-flow implementations
    - Protocol-level: Key update
    - Algorithm-level: Hiding, Masking
- Ideal: Mixture of countermeasures

Countermeasures - Recap

- Main techniques
  - Constant time/control-flow implementations
  - Protocol-level: Key update
  - Algorithm-level: Hiding, Masking
- Ideal: Mixture of countermeasures

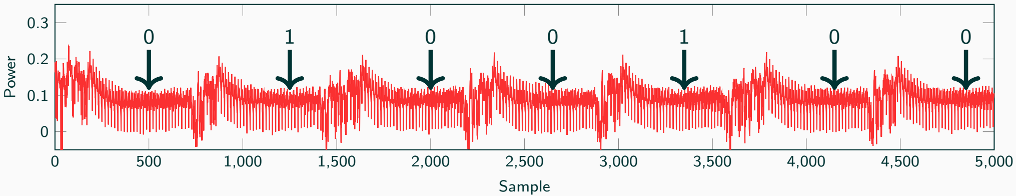
Remember : Each countermeasure can be broken!

just a matter of effort...

Make sure that attack effort greater than value of asset

Case-study: Asymmetric Crypto

Exponentiation Algorithms



- RSA decryption: $m = c^d \mod n$ (d = private key)
- Left-to-right square-and-multiply exponentiation:

```
m = c                       //init
for i = log2(d)-1...0       //loop over bits
    m = m*m mod n           //square
    if di == 1              //if bit is set
        m = m*c mod n       //multiply
return m
```

Constant-Time Exponentiation

- Montgomery ladder

```
R0 = 0, R1 = c              //init
for i = log2(d)-1...0       //loop over bits
    t = di                  //get the value of the bit
    R1-t = R0 * R1 mod n    //always multiply
    Rt = Rt * Rt mod n      //always square
return R0
```

Constant-Time Exponentiation

- Montgomery ladder

```
R0 = 0, R1 = c             //init
for i = log2(d)-1...0      //loop over bits
    t = di                 //get the value of the bit
    R1-t = R0 * R1 mod n   //always multiply
    Rt = Rt * Rt mod n     //always square
return R0
```

- Always same operations, just different operands (addresses)

Constant-Time Reductions

$$s = a + b \mod q$$

```
int s = a + b;
if (s >= q)
    s -= q;
```

```
int s = a + b;
int m = s - (q + 1);
m >>= 31;
s -= q & (!m);
```

- Dedicated algorithms for efficient reductions after multiplications
    - Make them constant time using similar tricks
- Still does not help against data leakage (DPA etc.)...

Asymmetric Crypto and Blinding

- Blinding: Similar to masking
- RSA exponent blinding (additive):
  - $d' = d + x(p-1)(q-1) = d + x\phi(n)$
  - $c^{d'} = c^d \mod n$
- RSA message blinding (multiplicative):
  - Message c, mask $x \rightarrow c' = c + x^e$
  - $(c')^d = c^d x \mod n$

Takeaways

1. Public-key crypto can have different side-channel challenges
    - Constant-time very important
    - Attacker often limited to single execution
    - Even without blinding, many protocols use one-time keys
    - But longer traces, intermediates used very often
    - Somewhat different protection techniques

1. Public-key crypto can have different side-channel challenges
   - Constant-time very important
   - Attacker often limited to single execution
   - Even without blinding, many protocols use one-time keys
   - But longer traces, intermediates used very often
   - Somewhat different protection techniques
2. There are many attacks outside of DPA / Templates
   - Algebraic attacks, horizontal attacks, collision correlation attacks,...
   - "Simple" side-channel analysis can be anything but ...

Thank you!

Questions:

rishub.nagpal@iaik.tugraz.at

Discord

Countermeasures Against Power Analysis

Side-Channel Security

Rishub Nagpal

May 22, 2024

IAIK – Graz University of Technology