Hardware Security

Side-channel attacks

R. Pacalet

Telecom Paris / EURECOM

Outline

1 Introduction
2 Timing attacks
3 Power attacks
4 Solutions of exercises

Section 1

Introduction

Hardware leaks information

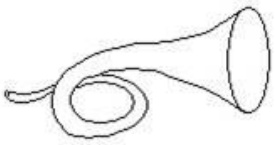
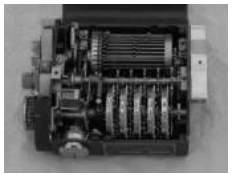
Eventually security always implemented in hardware

- Electronic devices:
  - Consume energy
  - Take time to compute
  - Emit electromagnetic radiations
  - Plus temperature, noise...
- These *side-channels* usually correlated with processing
- In security applications side-channels can be used to retrieve embedded secrets
- Few hundreds power traces can be sufficient to retrieve secret key
  - Even from theoretically unbreakable system
- Unlike in quantum cryptography information leakage usually undetectable
  - True for time
  - Almost true for power

A bit of history (1/2)

An old idea

- 1956: MI5/GCHQ against Egyptian Embassy in London
- Communications embassy $\leftrightarrow$ Cairo encrypted (Hagelin crypto machine)
- Suez crisis, MI5/GCHQ want to read the Egyptian cipher
- They play the old faulted phone system trick to plant microphones
- They record the clicks during the machine reset every morning...

"*Spy Catcher: The Candid Autobiography of a Senior Intelligence Officer*", Peter M. Wright, July 1, 1988, Dell Publishing

A bit of history (2/2)

An old idea

- 1996: P. Kocher timing attacks on RSA, DH, DSS
  - Applied with success in 2003 against OpenSSL 0.9.6
- 1999: P. Kocher power-attacks DES, AES, etc. (SPA, DPA)
  - Successful against smart cards, FPGAs...
- 2000-: New attacks (SEMA, DEMA, TPA).
- Importance of hardware security increases (CHES)
  - Huge scientific literature on side-channel attacks
- Software and hardware implementations consider them
  - Not always
  - Not always seriously enough
- Certification authorities take them into account

History repeats itself (1/5)

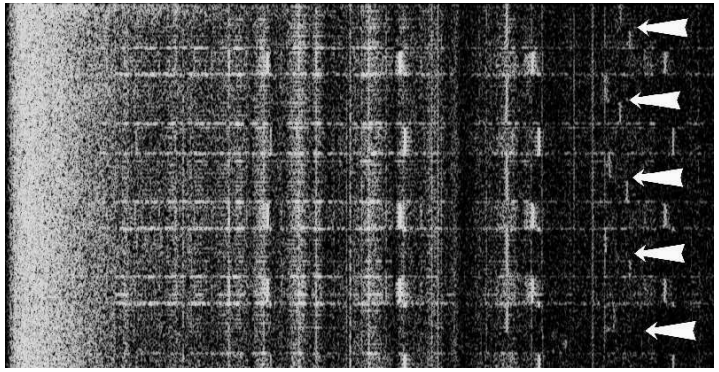
Ultrasounds

- 2013, 18^{th} of December
  Daniel Genkin (Technion and Tel Aviv University), Adi Shamir (Weizmann Institute of Science), Eran Tromer (Tel Aviv University)
  RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis.
- Target: regular computer computing RSA
  - GnuPG's current implementation
- Vibrations of electronic components (capacitors and coils)
  - Parts of voltage regulation circuit
  - Regulate voltage across large fluctuations in power consumption
- http://www.cs.tau.ac.il/~tromer/acoustic/

History repeats itself (2/5)

The attack can extract full 4096-bit RSA decryption keys from laptop computers (of various models), within an hour, using the sound generated by the computer during the decryption of some chosen ciphertexts. We experimentally demonstrate that such attacks can be carried out, using either a plain mobile phone placed next to the computer, or a more sensitive microphone placed 4 meters away.


History repeats itself (3/5)



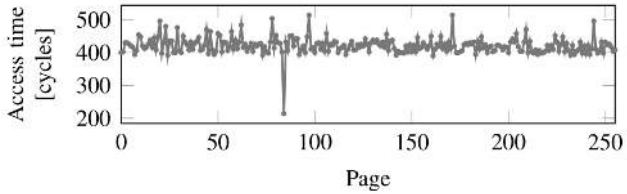
http://www.cs.tau.ac.il/~tromer/acoustic/img/gnupg-manykeys-downshifted.mp3
https://perso.telecom-paris.fr/pacalet/HWSec/misc/gnupg-manykeys-downshifted.mp3 (the RSA *Paso Doble*)

History repeats itself (4/5)

CVE-2023-0361 debian

Name	CVE-2023-0361
Description	A timing side-channel in the handling of RSA ClientKeyExchange messages was discovered in GnuTLS. This side-channel can be sufficient to recover the key encrypted in the RSA ciphertext across a network in a Bleichenbacher style attack. To achieve a successful decryption the attacker would need to send a large amount of specially crafted messages to the vulnerable server. By recovering the secret from the ClientKeyExchange message, the attacker would be able to decrypt the application data exchanged over that connection.
Source	CVE (at NVD; CERT, LWN, oss-sec, fulldisc, bugtraq, EDB, Metasploit, Red Hat, Ubuntu, Gentoo, SUSE bugzilla/CVE, Mageia, GitHub advisories/code/issues, web search, more)
References	DLA-3321-1, DSA-5349-1

History repeats itself (5/5)



```
Even if a memory location is only accessed
during out-of-order execution, it remains
cached. Iterating over the 256 pages of
probe array shows one cache hit, exactly
on the page that was accessed during the
out-of-order execution.
```

https://meltdownattack.com/

Section 2

Timing attacks

History and notations

History

- First published by Paul Kocher (CRYPTO'96)
- Possibly known before that
- Implemented by Dhem, Quisquater, et al. (CARDIS'98)
- Used by Canvel, Hiltgen, Vaudenay, and Vuagnoux to attack OpenSSL
  - CRYPTO'03

Notations

- Unless otherwise stated all numbers are natural integers
- $v[i]$: i^{th} component of vector v
- $w[i,j]$: element of row i, column j of matrix w
- $b_{n-1}b_{n-2}\dots b_1 b_0$: binary representation of number b on n bits
  - b_0: Least Significant Bit (LSB)
  - b_i: i^{th} bit
  - b_{n-1}: Most Significant Bit (MSB)

Example 1 (1/4)

Exponentiation: $m, d \mapsto m^d$

- $d = d_{w-1}d_{w-2}\dots d_2 d_1 d_0$, w-bits exponent
- $x^{a+b} = x^a \times x^b$; $x^{a\times b} = (x^a)^b = \left(x^b\right)^a$
- $d = d_0 + d_1 \times 2 + d_2 \times 4 + d_3 \times 8 + \dots + d_{w-2} \times 2^{w-2} + d_{w-1} \times 2^{w-1}$
- $m^d = m^{d_0 + d_1\times 2 + d_2 \times 4 + d_3 \times 8 + \dots + d_{w-2}\times 2^{w-2} + d_{w-1}\times 2^{w-1}}$
- $m^d = m^{d_0} \times m^{d_1\times 2} \times m^{d_2\times 4} \times m^{d_3\times 8} \times \dots \times m^{d_{w-2}\times 2^{w-2}} \times m^{d_{w-1}\times 2^{w-1}}$
- $m^d = m^{d_0} \times \left(m^{d_1}\right)^2 \times \left(m^{d_2}\right)^4 \times \left(m^{d_3}\right)^8 \times \dots \times \left(m^{d_{w-2}}\right)^{2^{w-2}} \times \left(m^{d_{w-1}}\right)^{2^{w-1}}$
- $m^d = m^{d_0} \times \left(m^{d_1} \times \left(m^{d_2}\right)^2\right)^2 \times \left(m^{d_3}\right)^8 \times \dots \times \left(m^{d_{w-2}}\right)^{2^{w-2}} \times \left(m^{d_{w-1}}\right)^{2^{w-1}}$
- $m^d = m^{d_0} \times \left(m^{d_1} \times \left(m^{d_2} \times \left(m^{d_3}\right)^2\right)^2\right)^2 \times \dots \times \left(m^{d_{w-2}}\right)^{2^{w-2}} \times \left(m^{d_{w-1}}\right)^{2^{w-1}}$

Example 1 (2/4)

Exponentiation: $m, d \mapsto m^d$

$$m^d = m^{d_0} \times \left(m^{d_1} \times \left(m^{d_2} \times \left(m^{d_3} \times \left(\dots \times \left(m^{d_{w-2}} \times \left(m^{d_{w-1}}\right)^2\right)^2 \dots\right)^2\right)^2\right)^2\right)^2$$

Example 1 (3/4)

Multiply and Square algorithm, 1 exponent bit per iteration (MS1)

- MSB first, iterations $k = w-1$ **down to** 0, temporary variables a, b

Algorithm MS1 exponentiation

1: $a \leftarrow 1$
2: **for** $k \leftarrow w-1$ **down to** 0 **do** ▷ From MSB to LSB of d
3: **if** $d_k = 1$ **then** ▷ k^{th} bit of d
4: $b \leftarrow a \times m$ ▷ Multiplication
5: **else**
6: $b \leftarrow a$
7: **end if**
8: $a \leftarrow b^2$ ▷ Square
9: **end for**
10: **return** b ▷ $b = m^d$

Algorithm Variant: SM1

1: $a \leftarrow 1$
2: **for** $k \leftarrow w-1$ **down to** 0 **do**
3: $a \leftarrow a^2$
4: **if** $d_k = 1$ **then**
5: $a \leftarrow a \times m$
6: **end if**
7: **end for**
8: **return** a

Example 1 (4/4)

Modular exponentiation, pseudo-code

- TA prone? Why? What can we expect from TA?

Algorithm MS1 modular exponentiation

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do                    ▷ From MSB to LSB of d
3:      if d_k = 1 then                            ▷ k^th bit of d
4:          b ← a × m mod n                        ▷ Modular multiplication
5:      else
6:          b ← a
7:      end if
8:      a ← b^2 mod n                              ▷ Modular square
9:  end for
10: return b                                       ▷ b = m^d mod n
```


Example 2

What if computation time of modular product and square is data-dependent?

- What can we expect from TA?

Algorithm MS1 modular exponentiation

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do                    ▷ From MSB to LSB of d
3:      if d_k = 1 then                            ▷ k^th bit of d
4:          b ← a × m mod n                        ▷ Modular multiplication
5:      else
6:          b ← a
7:      end if
8:      a ← b^2 mod n                              ▷ Modular square
9:  end for
10: return b                                       ▷ b = m^d mod n
```


Example 2: P. Kocher attack

```
1:  a ← 1
2:  for
        k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b      ▷ b = m^d mod n
```

Principle of attack

- Target computes x `modexp`
- Known plaintexts $m[i]; 1 \leq i \leq x$
- $\forall 1 \leq i \leq x$ attacker measures **total** `modexp` computation time $t[i] = \texttt{TIME}\left(m[i]^d \bmod n\right)$
- W.l.o.g. attacker already knows leading bits of d
  - $d_{w-1} \dots d_{r+1}$, for some $0 \leq r \leq w-1$
  - 👉 None if $r = w-1$
- Attacker extracts next unknown bit d_r:
  - $\forall m[i]$ attacker computes input a of iteration r
  - $\forall m[i]$ attacker **measures or estimates** computation time of modular multiplication at iteration r: $\widehat{t_\times}[i, r] = \texttt{TIME}\left(a \times m[i] \bmod n\right)$

Example 2: P. Kocher attack

```
1:  a ← 1
2:  for
        k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b      ▷ b = m^d mod n
```

Principle of attack

- Assume for some $m[i]$, $\widehat{t_\times}[i, r]$ *significantly* greater/smaller than average (attacker can distinguish "slow" and "fast" cases from "average")
- In average, if $t[i]$ large (slow) when $\widehat{t_\times}[i, r]$ large (slow) $\Rightarrow d_r = 1$ (probably)
  - Victim probably computed $b \leftarrow a \times m[i] \bmod n \Rightarrow t \uparrow$
- Else $d_r = 0$ (probably)
  - Victim probably skipped $b \leftarrow a \times m[i] \bmod n \Rightarrow t \rightsquigarrow$
- The larger the difference, the higher the attacker's confidence
- Do you understand difference with example 1?

Example 2: P. Kocher attack

```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Let's formalize a bit

- $\forall 1 \leq i \leq x, m[i]$: i^{th} plaintext
- $t[i] = \texttt{TIME}\left(m[i]^d\right)$ (measured)
- $t[i] = e[i] + \sum_{k=0}^{k=w-1} t[i,k]$
  - $e[i]$: measurement error
  - $t[i,k]$: computation time of iteration k of $m[i]^d$
  - Attacker knows $t[i]$
  - Attacker ignores $e[i]$ and $t[i,k]$
- Compute $w-1-r$ first iterations
  - $m[i], n, d_{w-1} \ldots d_{r+1}$ known
- $\widehat{t_\times}[i,r]$: attacker's estimate for $\texttt{TIME}\,(a \times m[i] \bmod n)$ at iteration r

Example 2: P. Kocher attack

```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Let's formalize a bit

- $\mathcal{S}$: indexes of 10% slowest cases
  - $\forall i \in \mathcal{S}, \forall j \notin \mathcal{S}, \widehat{t_\times}[i,r] > \widehat{t_\times}[j,r]$
  - $|\mathcal{S}| = x/10$
  - $t_{\mathcal{S}} = \frac{\sum_{i \in \mathcal{S}} t[i]}{x/10}$
- $\mathcal{F}$: indexes of 10 % fastest cases
  - $\forall i \in \mathcal{F}, \forall j \notin \mathcal{F}, \widehat{t_\times}[i,r] < \widehat{t_\times}[j,r]$
  - $|\mathcal{F}| = x/10$
  - $t_{\mathcal{F}} = \frac{\sum_{i \in \mathcal{F}} t[i]}{x/10}$
- Difference of computation time averages: $\Delta = t_{\mathcal{S}} - t_{\mathcal{F}}$

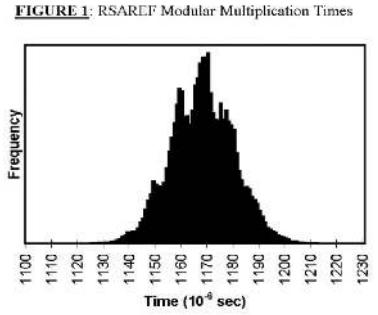
Example 2: P. Kocher attack

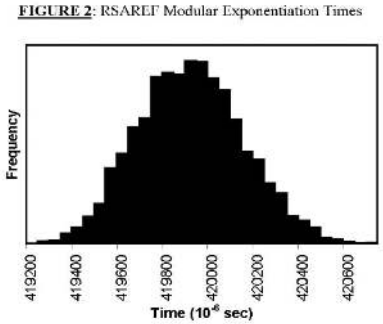
```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Let's formalize a bit

- τ: threshold
- $\Delta > \tau \Rightarrow d_r = 1$
- $\Delta < \tau \Rightarrow d_r = 0$
- Continue with next bit (d_{r-1}) until all bits of d "*known*"
- Attack targets one bit at a time
- Attack efficiency depends on:
  - Number x of acquisitions
  - Variability of $\widehat{t_\times}[i,r]$ (data dependency)
  - Noise $e[i]$
  - $|\mathcal{S}|$ and $|\mathcal{F}|$ (10% in our example)
  - Threshold τ
  - ...

P. Kocher attack on RSAREF (1/2)

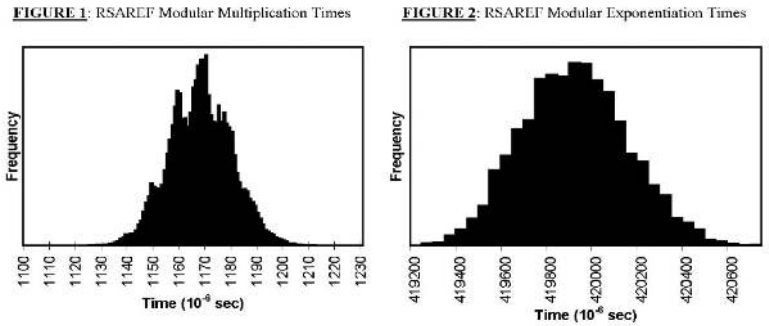




- $t_\times = \texttt{TIME}\,(a \times b \bmod n)$
- $\mathbb{E}\,(t_\times) \approx 1167.8 \times 10^{-6}$ sec.
- $\sigma\,(t_\times) \approx 12.01 \times 10^{-6}$ sec.

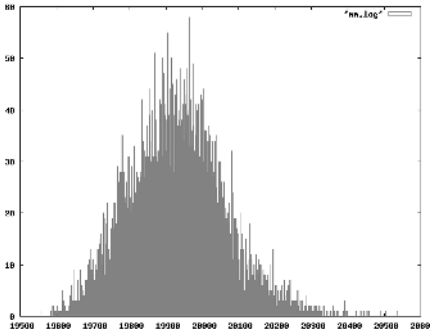
- $t_{exp} = \texttt{TIME}\left(m^d \bmod n\right)$
- $\mathbb{E}\,(t_{exp}) \approx 419901 \times 10^{-6}$ sec.
- $\sigma\,(t_{exp}) \approx 235 \times 10^{-6}$ sec.

P. Kocher attack on RSAREF (2/2)

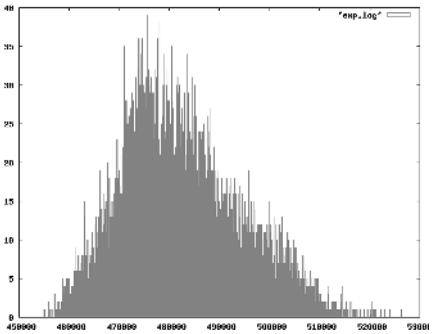


- RSAREF (functional) reference software library
- 512 bits exponentiation, 256 bits exponent (speed up)
- With 250 measurements probability of correct decision at any step...
  - ✔ 0.885

OpenSSL BN library



- $t_{\times} = \texttt{TIME}(a \times b \bmod n)$
- $\mathbb{E}(t_{\times}) \approx 19929$ clock cycles
- $\sigma(t_{\times}) \approx 130$ clock cycles



- $t_{exp} = \texttt{TIME}\left(m^d \bmod n\right)$
- $\mathbb{E}(t_{exp}) \approx 482206$ clock cycles
- $\sigma(t_{exp}) \approx 11453$ clock cycles

Optimizations

```
1:  a ← 1
2:  for
        k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b      ▷ b = m^d mod n
```

Cross-check on modular square computation time

- $\widehat{t_2}[i, r]$: attacker's estimate of $\texttt{TIME}(b^2 \bmod n)$ at iteration r
- Once d_r "*known*", verification on $\widehat{t_2}[i, r]$
- In average, is total computation time $t[i]$ large (small) when $\widehat{t_2}[i, r]$ large (small)?
  - Yes: better confidence in decision
  - No: doubt about decision

Optimizations

```
1:  a ← 1
2:  for
        k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b      ▷ b = m^d mod n
```

Detect wrong decisions and fix them

- d_r hypothesis wrong $\Rightarrow \forall k \leq r, \widehat{a} \neq a \wedge \widehat{b} \neq b$
- Correlation with measured computation time not observable any more
- Attack improvement
  - Keep list of decisions
  - Keep likelihood $t_{\mathcal{S}} - t_{\mathcal{F}}$
  - Likelihood-driven back-tracking
  - Hard decisions → soft decisions
  - Resembles channel decoding
  - More memory and CPU usage
  - Reduce number of acquisitions

Optimizations

```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Observe variance of computation time residue

- Following works iff absolute estimates
  - Not with relative estimates...
  - ...But relative / absolute ratio can be estimated
- $t[i,k] = \texttt{TIME}\left(\text{iteration } k \text{ of } m[i]^d \bmod n\right)$
- $\widehat{t}[i,k]$: attacker's estimate for iteration r:
  - $\widehat{t_\times}[i,r] + \widehat{t_2}[i,r]$ ($d_r = 1$) or $\widehat{t_2}[i,r]$ ($d_r = 0$)
- $\Delta[i] = t[i] - \sum_{k=r+1}^{k=w-1} \widehat{t}[i,k]$
- $\Delta[i] = e[i] + \sum_{k=0}^{k=w-1} t[i,k] - \sum_{k=r+1}^{k=w-1} \widehat{t}[i,k]$
- $\Delta[i] = e[i] + \sum_{k=0}^{k=r} t[i,k] + \sum_{k=r+1}^{k=w-1} \left(t[i,k] - \widehat{t}[i,k]\right)$

Optimizations

```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Observe variance of computation time residue

- If $d_{w-1} \dots d_{r+1}$ correct
  - $\widehat{t}[i, k > r] \approx t[i, k > r]$
  - $\Delta[i] = e[i] + \sum_{k=0}^{k=r} t[i,k] + \sum_{k=r+1}^{k=w-1} \left(t[i,k] - \widehat{t}[i,k]\right)$
  - $\Delta[i] \approx e[i] + \sum_{k=0}^{k=r} t[i,k]$
  - $\mathbb{V}(\Delta[.]) \approx \mathbb{V}(e[.]) + \mathbb{V}\left(\sum_{k=0}^{k=r} t[.,k]\right)$
  - $\mathbb{V}(\Delta[.]) \approx \mathbb{V}(e) + (r+1) \times \mathbb{V}(t)$
- $\mathbb{V}(\Delta[.]) \downarrow$ when $r \downarrow$

Optimizations

```
1:  a ← 1
2:  for
    k ← w − 1 down to r + 1 do
3:      ...
4:      a ← b^2 mod n
5:  end for
6:  if d_r = 1 then
7:      b ← a × m mod n
8:  else
9:      b ← a
10: end if
11: a ← b^2 mod n
12: for k ← r − 1 down to 0 do
13:     ...
14:     a ← b^2 mod n
15: end for
16: return b        ▷ b = m^d mod n
```

Observe variance of computation time residue

- If $d_{w-1} \dots d_{s+1}$ correct but...
- $\dots d_s \dots d_{r+1}$ wrong for some $s \geq r+1$
  - $\widehat{t}[i, k > s] \approx t[i, k > s]$
  - $\widehat{t}[i, s \geq k > r] \neq t[i, s \geq k > r]$
  - $\Delta[i] = e[i] + \sum_{k=0}^{k=r} t[i,k] + \sum_{k=r+1}^{k=w-1} \left(t[i,k] - \widehat{t}[i,k]\right)$
  - $\Delta[i] \approx e[i] + \sum_{k=0}^{k=r} t[i,k] + \sum_{k=r+1}^{k=s} \left(t[i,k] - \widehat{t}[i,k]\right)$
  - $\Delta[i] \approx e[i] + \sum_{k=0}^{k=s} t[i,k] - \sum_{k=r+1}^{k=s} \widehat{t}[i,k]$
  - $\mathbb{V}(\Delta[.]) \approx \mathbb{V}(e) + (s+1) \times \mathbb{V}(t) + (s-r) \times \mathbb{V}(t)$
  - $\mathbb{V}(\Delta[.]) \approx \mathbb{V}(e) + (2 \times s - r + 1) \times \mathbb{V}(t)$
- $\mathbb{V}(\Delta[.]) \uparrow$ when $r \downarrow$

Wrap up on TA (1/3)

Where does it come from?

- Time: processing data takes time

How does it work?

- Acquisition phase
  - Same secret
  - *Sufficient* number of acquisitions with different input messages
  - Build database of {input, computation time} pairs (may also work with outputs, how?)
- Analysis phase (usually off-line)
  - Attacker tries to retrieve part s of secret (e.g., 1 bit)
  - Attacker builds "computation time models" $TM_g(i)$
    - — Of one part of computation with input $m[i]$ (e.g., 1 `modmul`)
    - — Under assumption that $s = g$
  - Attacker estimates correlations between TM_g and measured total times
  - TM_g with best correlation $\Rightarrow$ g best candidate for s

Wrap up on TA (2/3)

Principle

- Our example attacks one exponent bit at a time
- For each attacked bit d_k, 2 computation time models:
  - $TM_0(i) = 0$ ($d_k = 0$ case)
  - $TM_1(i) = \texttt{TIME}(a \times m[i] \bmod n)$ at iteration r ($d_r = 1$ case)
- $t_S - t_{\mathcal{F}}$: estimator of correlation between TM_1 and t
- Note: there are better correlation estimators
  - Pearson correlation coefficient, Kolmogorov-Smirnov test...
- Correlation$(TM_1, t) > \tau \Rightarrow d_r = 1$ (τ: threshold)
- Correlation$(TM_1, t) < \tau \Rightarrow d_r = 0$

Wrap up on TA (3/3)

Principle

- Other examples with $TM_0(i) \neq 0$:
  - $TM_0(i) = \texttt{TIME}\left(b \leftarrow a; a \leftarrow b^2 \bmod n\right)$ at iteration r
  - $TM_1(i) = \texttt{TIME}\left(b \leftarrow a \times m \bmod n; a \leftarrow b^2 \bmod n\right)$ at iteration r
- Then correlations can be compared (no need for threshold)
  - Correlation$(TM_1, t) >$ Correlation$(TM_0, t) \Rightarrow d_r = 1$
  - Correlation$(TM_1, t) <$ Correlation$(TM_0, t) \Rightarrow d_r = 0$
- Whatever the statistical tool, principle remains the same:
  - TM_g with best correlation $\Rightarrow g$ best candidate for s
- Analysis usually off-line
  - But interactive, adaptive attacks also exist

Pearson correlation coefficient

A better statistical tool than partitioning

- s: portion of the secret under attack
- x: number of computation time measurements
- $t[i], 1 \leq i \leq x$: computation time measurements
- $m[i], 1 \leq i \leq x$: input messages
- $TM_g(i)$: attacker's computation time estimate
  - Of part of $m[i]^d$ (e.g. one iteration)
  - With hypothesis g on s
- $PCC(TM_g, t)$: estimator of correlation between the $t[i]$ and $TM_g(i)$ $(1 \leq i \leq x)$
- g with highest $PCC(TM_g, t) \Rightarrow$ best candidate for s

Exercises on timing attacks (1/2)

Check your understanding

- ⍰ Ex. 1: List the hypotheses for a timing attack to be practical
- ⍰ Ex. 2: Do you think the modified implementation below is protected?

Algorithm Modified MS1 modular exponentiation, version 1

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do                ▷ From MSB to LSB of d
3:      f ← a × m mod n                       ▷ Modular multiplication
4:      if d_k = 0 then                       ▷ k^th bit of d
5:          b ← a
6:      else
7:          b ← f
8:      end if
9:      a ← b^2 mod n                         ▷ Modular square
10: end for
11: return b                                  ▷ b = m^d mod n
```

Exercises on timing attacks (2/2)

Check your understanding

? Ex. 3: Do you think the modified implementation below is protected?

Algorithm Modified MS1 modular exponentiation, version 2

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do                ▷ From MSB to LSB of d
3:      f ← a × m mod n                       ▷ Modular multiplication
4:      g ← a^2 mod n                         ▷ Modular square
5:      h ← f^2 mod n                         ▷ Modular square
6:      if d_k = 0 then                       ▷ k^th bit of d
7:          b ← a
8:          a ← g
9:      else
10:         b ← f
11:         a ← h
12:     end if
13: end for
14: return b                                  ▷ b = m^d mod n
```


Homework on timing attacks

For next time (please do it)

- Imagine countermeasures, evaluate their cost and efficiency
- Study and understand P. Kocher paper
  - Especially the blinding countermeasure he proposes (section 10)
- Prepare questions

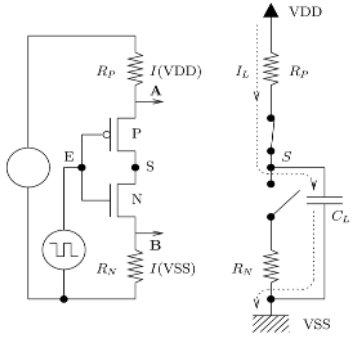
Section 3

Power attacks

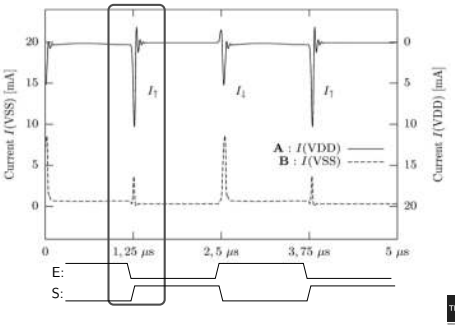
Power in CMOS logic (1/2)

Energy and cell output transitions are correlated

- E falling edge $\Rightarrow$ S rising edge



Current observed across R_p resistor: I(VDD)

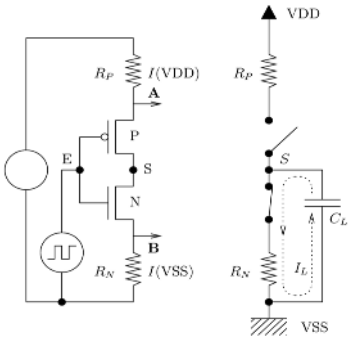


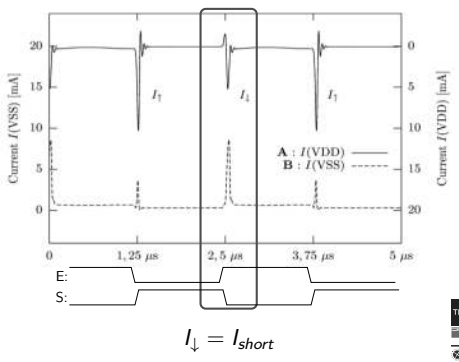
$I_\uparrow = I_{short} + I_L$

Power in CMOS logic (2/2)

Energy and cell output transitions are correlated

- E rising edge $\Rightarrow$ S falling edge

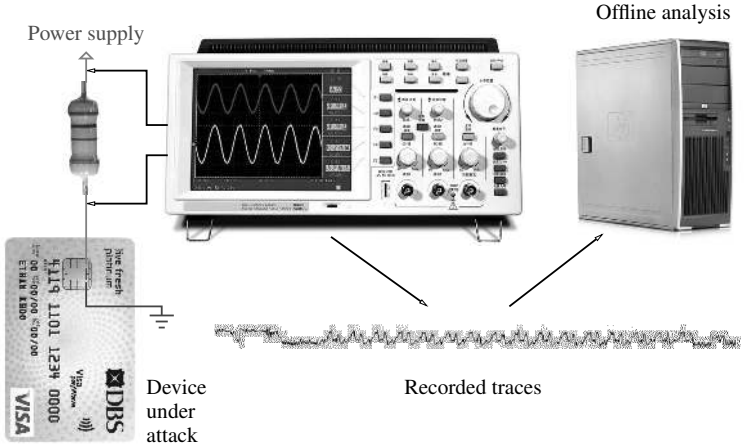




Current observed across R_p resistor: I(VDD)

$I_\downarrow = I_{short}$

Power analysis setup (1/2)



Power analysis setup (2/2)

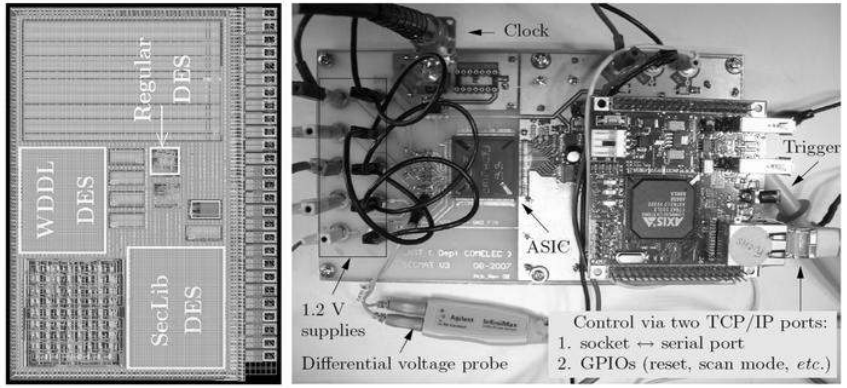
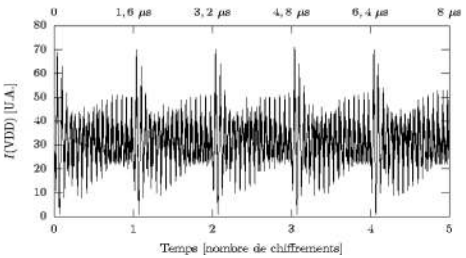


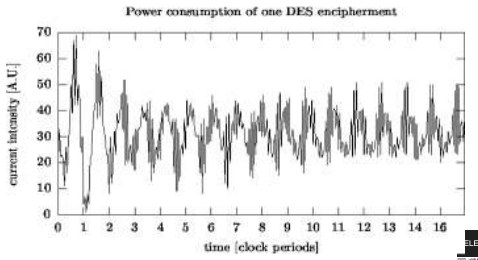
Figure 1: Floorplan and acquisition board of the SecMat v3 ASIC.

Simple power analysis, example 1

The power trace of a "naive" DES hardware implementation leaks a lot of information

- CMOS structures consume energy when switching
- Hamming distance of register transitions
- Hamming weights in some implementations
- Clock spikes

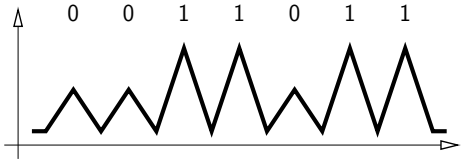




Simple power analysis, example 2

Multiply and square exponentiation

- Iterate on exponent bits
- From MSB to LSB
- $c \leftarrow m^d$



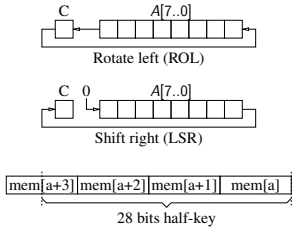
Algorithm MS1 exponentiation

1: $a \leftarrow 1$
2: **for** $k \leftarrow w-1$ **down to** 0 **do**
3: **if** $d_k = 1$ **then**
4: $b \leftarrow a \times m$ ▷ Mult.
5: **else**
6: $b \leftarrow a$
7: **end if**
8: $a \leftarrow b^2$ ▷ Square
9: **end for**
10: **return** b ▷ $b = m^d$

Mount SPA of DES key schedule

6502: 8 bits CPU

- 8b accumulator A, 1b carry flag C
- Left rotate 28-bits half-key
- Channel: energy per instruction
  - Transitions in C, A
  - $0 \leq n_t \leq 9$
- ? Ex. 4: What's your attack?

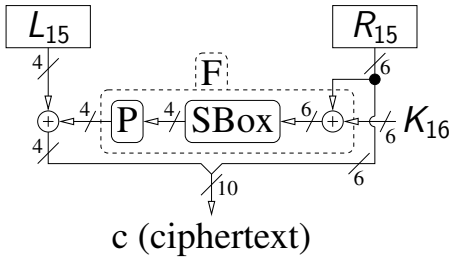



```
1  CLC       # C <- 0
2  LDA  a    # A <- mem[a]
3  ROL       # C|A <- A|C
4  STA  a    # mem[a] <- A
5  LDA  a+1  # A <- mem[a+1]
6  ROL       # C|A <- A|C
7  STA  a+1  # mem[a+1] <- A
8  LDA  a+2  # A <- mem[a+2]
9  ROL       # C|A <- A|C
10 STA  a+2  # mem[a+2] <- A
11 LDA  a+3  # A <- mem[a+3]
12 ROL       # C|A <- A|C
13 STA  a+3  # mem[a+3] <- A
14 AND  0x1f # A <- 000|A[4..0]
15 LSR       # C|A <- A[0]|0|A[7..1]
16 LSR       # C|A <- A[0]|0|A[7..1]
17 LSR       # C|A <- A[0]|0|A[7..1]
18 LSR       # C|A <- A[0]|0|A[7..1]
19 ORA  a    # A <- A or mem[a]
20 STA  a    # mem[a] <- A
```


P. Kocher attack on DES (1/2)

Last round of DES, 1 SBox at a time

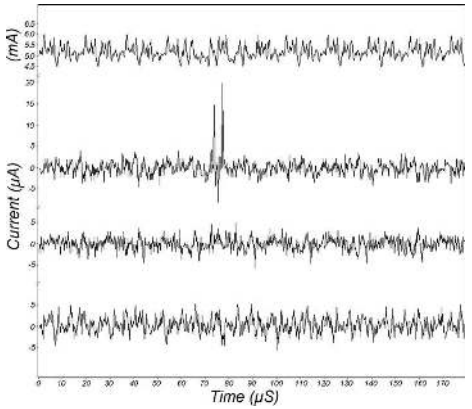
- n {power trace, ciphertext} pairs
  - $\{\boldsymbol{T}[i], c[i]\}$
- $\forall 1 \leq i \leq n, R_{15} = L_{16} \in c[i]$ known
- Add hypothesis g on 6 bits of K_{16}
  - ☞ "*Compute*" 4 bits of L_{15}
  - As a function of g and $c[i]$
- Focus on 1 bit only: $b[i](g)$
- Split traces in 2 sets
  - $\boldsymbol{S_0}(g) = \{\boldsymbol{T}[i] \mid b[i](g) = 0\}$
  - $\boldsymbol{S_1}(g) = \{\boldsymbol{T}[i] \mid b[i](g) = 1\}$
- If guess g correct sets should exhibit different energy consumptions
  - $\boldsymbol{score}(g) = \mathbb{E}(\boldsymbol{S_0}(g)) - \mathbb{E}(\boldsymbol{S_1}(g))$



P. Kocher attack on DES (2/2)

Differential Power Analysis (DPA)

- The larger the score (difference), the more likely the hypothesis
- Hence the name "*Differential Power Analysis*" (DPA)



Wrap up on DPA (1/4)

Foundations

- ? Where does it come from?
- Power: computing consumes energy

Power attack principles

- Target small portion s of secret
  - 6 bits of last round key of DES
  - 1 bit of secret exponent of RSA...
- Use intermediate value v as oracle
  - One bit of L_{15}
  - Modular product or square...
  - Computed (stored) by **same** hardware element
- Based on hypothesis of data-dependent power consumption
  - Compute (store) different v values $\Rightarrow$ consume different energy

Wrap up on DPA (2/4)

Acquisition phase

- Same secret, different input messages
- *Sufficient* number of acquisitions $\Rightarrow$ database of acquisitions
  - $\{\boldsymbol{T}[i], m[i]\}$
  - Pairs of {power trace, input (output) message}

Analysis phase (usually off-line)

- Attacker build "*power models*" PM_g of target implementation
  - g hypothesis (guess) on portion of secret
- $PM_g(i)$ = estimate of energy of specific operation (computation, storage...)
  - For input (output) $m[i]$
  - If g correct
- As many PM_g models as hypotheses g
- PM_g that best correlates with actual power traces $\Rightarrow$ most likely hypothesis g

Wrap up on DPA (3/4)

Ones do not consume more than zeros

- CMOS logic has (almost) no static power consumption
- It is only transitions $v_0 \rightarrow v_1$ that consume energy (dynamic power)

From states to transitions

- What if only v_1 can be "*computed*"?
- Assume rising and falling transitions consume differently: $I_{\uparrow} - I_{\downarrow} = \epsilon \neq 0$
- Assume v_0 uncorrelated
  - $v_1 = 0 \Rightarrow$ falling transition with probability $1/2$, no transition with probability $1/2$
  - $v_1 = 1 \Rightarrow$ rising transition with probability $1/2$, no transition with probability $1/2$
  - On large number of traces average difference $\approx \epsilon/2$

Wrap up on DPA (4/4)

Two types of attacks

- "*Compute*" only $v_1 \Rightarrow$ Hamming weight of v_1
- "*Compute*" v_0 and $v_1 \Rightarrow$ Hamming distance $v_0 \rightarrow v_1$

Exercises on DPA

Check your understanding

- Two power models
  - Hamming weight (based on current state only)
  - Hamming distance (based on previous and current states)
  - ⍰ Ex. 5: Most efficient power model?
- ⍰ Ex. 6: Differences with timing attacks?
- ⍰ Ex. 7: DPA easier or more difficult than TA? Why?
- ⍰ Ex. 8: List the hypotheses for a DPA to be practical
- ⍰ Ex. 9: Design countermeasures
- ⍰ Ex. 10: What if energy depends only on key?
- ⍰ Ex. 11: What if energy depends only on input messages?
- ⍰ Ex. 12: What if energy depends neither on key nor on input messages?

Homework on power attacks

For next time (please do it)

- Read and understand every detail of original paper by P. Kocher
- Imagine attack against hardware DES implementation
  - Ciphertexts are known
  - $L_0R_0, \ldots, L_{15}R_{15}, R_{16}L_{16}$ values stored successively in **same** 64 bits register
  - Attacker monitors current on power supply side
  - Describe in deep details your algorithm
- Search ways to blind DES

Prepare first lab (please do it)

- Read directions
- Look at provided software libraries
- Imagine what you will do, why and how

Section 4

Conclusion

Further Reading

In EURECOM library

- *The Hardware Hacking Handbook, Breaking Embedded Security with Hardware Attacks*, Jasper van Woudenberg and Colin O'Flynn, No Starch Press, 2022, 515 p.
- *Security of Ubiquitous Computing Systems, Selected Topics*, Avoine, Gildas ; Hernandez-Castro, Julio, Springer, 2021, 268 p.
- *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Stefan Mangard and Elisabeth Oswald and Thomas Popp, Springer, 2007, 337 p. (PDF available)
- *Smart Card Handbook*, Wolfgang Rankl and Wolfgang Effing, John Wiley and Sons, 2004, 1088 p.
- *Embedded Cryptographic Hardware: Design & Security*, Nadia Nedjah and Luiza de Macedo Mourelle, Nova Publishers, 2005, 255 p.
- And a lot more...

Questions?

Section 5

Solutions of exercises

List the hypotheses for a timing attack to be practical (1/3)

Ex. 1: H1: Cryptosystem takes different time to process different input data

- Performance optimizations (bypass useless operations)
- Branching, conditional statements
- Cache misses

⚠ Note: intuition suggests leak of small amount of information
  - Like Hamming weight of key
  - ⚠ Intuition usually wrong (same in probabilities and security)
  - ⚠ Never trust your intuition

List the hypotheses for a timing attack to be practical (2/3)

H2: Attacker can build models of timing based on

- Knowledge of inputs (or outputs)
- Guesses on secret key
- Knowledge of implementation
- Attacker can redo computations and measure time
  - With similar timing characteristics
- Or use another estimate
  - Hamming weight of operand...

👍 Relative estimate sufficient

H3: "*Sufficient*" number of different inputs, same secret key

- Victim runs algorithm
- Attacker records timing

List the hypotheses for a timing attack to be practical (3/3)

H4: Total number of guesses on secret key "*manageable*"

- Such that attacker can check them all for correctness

☺ 1 bit ⇒ 2 different guesses

☹ 128 bits secret key ⇒ $2^{128} \approx 3.4 \times 10^{38}$ different guesses

⚠ One billion per second ⇒ 720 billion times age of universe (brute force attack)

👉 If 128 bits can be split in 16 bytes ⇒ $16 \times 2^8 = 4096$ guesses to check...

H5: SNR *good enough*

✔ Timing model of right guess match actual measurements "*significantly*" better

Do you think the modified implementation below is protected?

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do
3:      f ← a × m mod n
4:      if d_k = 0 then
5:          b ← a
6:      else
7:          b ← f
8:      end if
9:      a ← b^2 mod n
10: end for
11: return b
```

Ex. 2: No, it is not better protected

☹ Can even be worse

👉 Modular multiplication always computed ⇒ higher signal

☺ branches of `if` perfectly balanced but...

☹ ...Timing of modular square still depends on b

⚠ That is, on taken branch, that is, on d_r

- Attacker can build 2 timing models
  - $TM_0 = \texttt{TIME}\left(a^2 \bmod n\right)$
  - $TM_1 = \texttt{TIME}\left(f^2 \bmod n\right)$

✔ Timing model that better correlates with measured total times ⇒ most likely d_r value

Do you think the modified implementation below is protected?

```
1:  a ← 1
2:  for k ← w − 1 down to 0 do
3:      f ← a × m mod n
4:      g ← a^2 mod n
5:      h ← f^2 mod n
6:      if d_k = 0 then
7:          b ← a
8:          a ← g
9:      else
10:         b ← f
11:         a ← h
12:     end if
13: end for
14: return b
```

Ex. 3: No, it is not better protected

☹ Can even be worse

👉 3 modular operations ⇒ twice higher signal

☺ branches of `if` perfectly balanced but...

☹ ...Timing of modular operations at iteration $r-1$ depend on a at end of iteration r

⚠ That is on taken branch, that is on d_r

- Attacker can build 2 timing models
  - $TM_0 = \texttt{TIME}\left(\text{iteration } r-1 \mid d_r = 0\right)$
  - $TM_1 = \texttt{TIME}\left(\text{iteration } r-1 \mid d_r = 1\right)$

✔ Timing model that better correlates with measured total times ⇒ most likely d_r value

Mount SPA of DES key schedule

Ex. 4: Consumption of last `LSR` (line 18)

- At beginning, before call number 1
  - Memory $= k_1 \dots k_{28}$
- Before call $2 \leq i \leq 28$
  - Memory $= k_i \dots k_{28} k_1 \dots k_{i-1}$
- Before last `LSR` operation of call i
  - $A = 000000 k_i k_{i+1}$, $C = k_{i+2}$
- After last `LSR` operation of call i
  - $A = 0000000 k_i$, $C = k_{i+1}$
- $n_t = k_i + k_{i+1} \oplus k_i + k_{i+2} \oplus k_{i+1}$
  - $n_t = 0 \Leftrightarrow k_i k_{i+1} k_{i+2} \in \{000\}$
  - $n_t = 1 \Leftrightarrow k_i k_{i+1} k_{i+2} \in \{001, 011, 111\}$
  - $n_t = 2 \Leftrightarrow k_i k_{i+1} k_{i+2} \in \{010, 100, 110\}$
  - $n_t = 3 \Leftrightarrow k_i k_{i+1} k_{i+2} \in \{101\}$

✔ 28 independent equations of 28 variables

```
1  CLC        # C <- 0
2  LDA  a     # A <- mem[a]
3  ROL        # C/A <- A/C
4  STA  a     # mem[a] <- A
5  LDA  a+1   # A <- mem[a+1]
6  ROL        # C/A <- A/C
7  STA  a+1   # mem[a+1] <- A
8  LDA  a+2   # A <- mem[a+2]
9  ROL        # C/A <- A/C
10 STA  a+2   # mem[a+2] <- A
11 LDA  a+3   # A <- mem[a+3]
12 ROL        # C/A <- A/C
13 STA  a+3   # mem[a+3] <- A
14 AND  0x1f  # A <- 000|A[4..0]
15 LSR        # C/A <- A[0]|0|A[7..1]
16 LSR        # C/A <- A[0]|0|A[7..1]
17 LSR        # C/A <- A[0]|0|A[7..1]
18 LSR        # C/A <- A[0]|0|A[7..1]
19 ORA  a     # A <- A or mem[a]
20 STA  a     # mem[a] <- A
```

Most efficient power model?

Ex. 5: Hamming distance more efficient

- I_{short}, I_L: short circuit and charging currents
- Assume attacker monitors current on power supply side only
- With Hamming distance:
  - If no transition ($0 \to 0$ or $1 \to 1$), $I_{\to} = 0$
  - If rising transitions ($0 \to 1$), $I_{\uparrow} = I_{short} + I_L$
  - If falling transitions ($1 \to 0$), $I_{\downarrow} = I_{short}$
  - 👉 Average difference between transition and no transition (can probably do better): $d_{hd} = \frac{I_{short}+I_L+I_{short}}{2} - I_{\to} = I_{short} + \frac{I_L}{2}$
- With Hamming weight:
  - If ending state $= 1$ ($? \to 1$), in average, $I_{\uparrow ?} = \frac{I_{\to}+I_{short}+I_L}{2} = \frac{I_{short}+I_L}{2}$
  - If ending state $= 0$ ($? \to 0$), in average, $I_{\downarrow ?} = \frac{I_{\to}+I_{short}}{2} = \frac{I_{short}}{2}$
  - ✔ Difference between ending states 1 and 0 (average): $d_{hw} = \frac{I_{short}+I_L}{2} - \frac{I_{short}}{2} = \frac{I_L}{2} < d_{hd}$

Differences with timing attacks?

Ex. 6: Pros and cons

- Richer side channel (full vector instead of single scalar)
  - 💡 Statistics within one single power trace?
- Remote attacks not practical any more (almost)
- More expensive acquisition phase
- More processing in analysis phase
- Attack could be detectable
- More difficult to compute with constant power than with constant time
  - ⍰ How would you compute with constant power?
  - ✔ More on this later

DPA easier or more difficult than TA? Why?

Ex. 7: Pros and cons

- More difficult to monitor energy than time
- More data to process
- More information in side channel
- Extra difficulties like perfect synchronization, for instance
- More difficult to compute with constant power than with constant time
  - ✔ More on this later

List the hypotheses for a DPA to be practical (1/2)

Ex. 8: H1: Power consumption depends on processed data

- Activity of CMOS logic (number of node switches) data-dependent
- Dynamic power of CMOS logic is driven by activity

H2: Attacker can build models of power, based on

- Knowledge of inputs (or outputs)
- Guesses on secret key
- Knowledge of implementation
- Attacker can "*simulate*" some internal states (or transitions) of victim
  - With similar power consumptions
- Or use another estimate
  - Hamming weight or distance...

👉 Relative estimate sufficient

List the hypotheses for a DPA to be practical (2/2)

H3: "*Sufficient*" number of different inputs, same secret key

- Victim runs algorithm
- Attacker records power

H4: Total number of guesses on secret key "*manageable*"

- Same reasons as for timing attacks

H5: SNR *good enough*

✔ Power models of right guess match actual power traces "*significantly*" better

Design countermeasures (1/4)

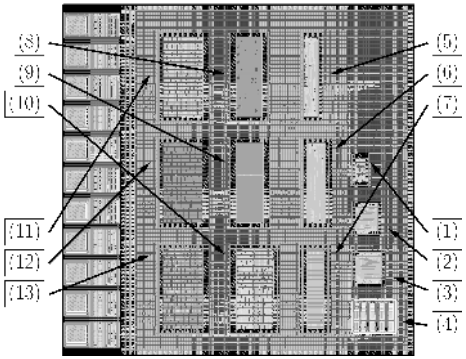
Ex. 9: H1: Power consumption depends on processed data

- Compute with constant power consumption
- ☹ Much more difficult than constant timing
  - But the data dependency can be reduced
- ☹ Rather expensive (slower, energy hungry, larger silicon area)
- More on this next slide

H2: Attacker can build models of power, based on knowledge of inputs (outputs)

- Prevent attacker from accessing inputs or outputs
- ☹ Not Always Practical (NAP)
- More on this later

Constant power computation



13 hardware implementations of AES SBox

1. Std cells, decomposed in $GF(16)$
2. Std cells, lookup table
3. Std cells, decode-permute-encode
4. ROM
5. WDDL basic
6. + Symmetric placement
7. + Symmetric routing + dummies
8. + Shielding
9. + Symmetric gates
10. SecLib, symmetric placement
11. + Symmetric routing + dummies
12. + Shielding
13. + EMA coating plane

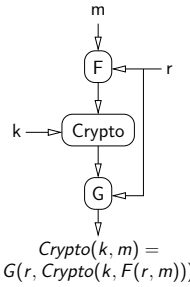
Prevent attacker from accessing inputs or outputs (1/4)

H2: Attacker can build models of power, based on knowledge of inputs (outputs)

- Blinding

Mix input with random seed before processing

- Compute complementary transform after processing
- ☺ Simple with math-based algorithms (RSA)
- ☹ Difficult with symmetric block ciphers (DES)
- ☹ Has a cost, may be sensitive to higher order DPA



$Crypto(k, m) = G(r, Crypto(k, F(r, m)))$

Prevent attacker from accessing inputs or outputs (2/4)

Example of blinding on a public key algorithm

- RSA, $c = m^d \bmod n = (m \times r)^d \times s \bmod n$ where $r^d \times s \bmod n = 1$

Algorithm Without blinding

```
1:  procedure RSA-PURE(m, n, d)
2:      a ← 1
3:      for k ← w − 1 down to 0 do
4:          a ← a² mod n
5:          if d_k = 1 then
6:              a ← a × m mod n
7:          end if
8:      end for
9:      return a
10: end procedure
```

Algorithm With blinding

```
1:  procedure RSA-BLIND(m, n, d)
2:      let r^d × s mod n = 1 in
3:      a ← 1
4:      m' ← m × r mod n
5:      for k ← w − 1 down to 0 do
6:          a ← a² mod n
7:          if d_k = 1 then
8:              a ← a × m' mod n
9:          end if
10:     end for
11:     return a × s mod n
12: end procedure
```


Prevent attacker from accessing inputs or outputs (3/4)

Example of blinding on a public key algorithm

- RSA, $c = m^d \bmod n = (m \times r \bmod n)^d \times s \bmod n$ where $r^d \times s \bmod n = 1$

⚠ $m \times r \bmod n$ is prone to timing attacks

$\Rightarrow$ r could be revealed

💡 Change (r, s) after each modular exponentiation

☹ Computing $r^d \bmod n$ and $s = \left(r^d \bmod n\right)^{-1} \bmod n$ time consuming

💡 $(r, s) \leftarrow (r^2 \bmod n, s^2 \bmod n)$

- ... or any other update preserving $r^d \times s \bmod n = 1$

Prevent attacker from accessing inputs or outputs (4/4)

Masking

- Notations: $\neg$ (NOT), $\vee$ (OR), $\wedge$ (AND), $\oplus$ (XOR), . (product in $GF(2^n)$)

💡 Randomly split data in two (or more) parts, process sub-parts, merge results

💡 Applied at level of elementary boolean operations

- Example of first order boolean masking
  - 👉 $x \rightarrow (x_0, x_1)$ with $x_0 = x \oplus m, x_1 = m$ $(x = x_0 \oplus x_1)$
  - 👉 $z = x \oplus y \rightarrow (z_0, z_1) = (x_0 \oplus y_0, x_1 \oplus y_1)$
  - 👉 $z = x \wedge y \rightarrow (z_0, z_1) = ((x_0 \wedge y_0) \oplus (x_0 \vee \neg y_1), (x_1 \wedge y_0) \oplus (x_1 \vee \neg y_1)) \ldots$

👉 Kind of blinding with similar advantages and drawbacks

☺ Works also with symmetric block ciphers

☹ Has a cost, may be sensitive to higher order DPA
  - 💡 Higher order masking
  - 👉 $x \rightarrow (m_1.x) \oplus m_0$ (second order affine masking)

Design countermeasures (2/4)

H2: Attacker can build models of power, based on guesses on secret key

❓ How to prevent somebody from guessing?

H2: Attacker can build models of power, based on knowledge of implementation

⚠ Auguste who, did you say?

H3: "*Sufficient*" number of different inputs, same secret key

💡 Prevent access to the device (NAP)

💡 Authentication and limited number of tries; same as PIN codes (NAP)

💡 Variant: if authentication fails add exponential delay before next try (NAP)

💡 Limited lifetime of secret keys
  - $\Rightarrow$ Secret keys used only for small number of processing
  - ☺ Not enough traces with same secret key
  - ☹ Key management much more complex than with long lifetime secret (NAP)

Design countermeasures (3/4)

H4: Total number of guesses on secret key "*manageable*"

- Increase number of required hypothesis
- DPA "*easy*" against DES or AES
  - Because small width of last (first) logic cones
- But combinational functions with large number of inputs...
  - ☹ ...Expensive
  - ☹ ...Slow
- ☹ And attacks could also be mounted against internal nodes of logic cone

Design countermeasures (4/4)

H5: SNR *good enough*

- Desynchronize traces
- Random clock drift
  - ☹ Clock spikes still there
- Random dummy clock cycles
  - Dummy clock cycles could be found out
- Add some power noise
- ✔ Increases number of required traces

What if energy depends only on key?

Ex. 10: No statistical analysis any more

- Equivalent to Simple Power Analysis
  - ☹ One trace only
  - ☺ But several traces can be averaged to improve SNR
- As with SPA, in most cases, best attacker can expect...
  - • ...≈ Hamming weight of secret key...
  - ✔ ...but not always (see SPA examples and exercise)

What if energy depends only on input messages?

Ex. 11: Energy does not depend on secret

- ⇒ Energy contains no information on secret
- ⇒ There is nothing to expect about secret from energy
- ✔ Power attacks (simple and differential) do not work any more

What if energy depends neither on key nor on input messages?

Ex. 12: Energy is constant

⇒ Energy contains no information
⇒ There is nothing to expect about anything from energy
✔ Power attacks (simple and differential) do not work any more