Secure Implementation of Cryptographic Algorithms

Chapter 3
Timing Attacks on Cryptographic Algorithms and their Implementation

Wieland Fischer & Berndt Gammel
Infineon Technologies

(Slide set: Courtesy Stefan Mangard, TU Graz)

Content of Chapter 3

Chapter 3

3.1. Timing Attacks on AES and Countermeasures

Chapter 3

Timing Dependencies in Software Implementations of AES

3.1 TA on AES and Countermeasures

- Timing attacks are in particular relevant for asymmetric cryptography.
- However, attacks on symmetric cryptography is possible, if either
  - the software implementation is not done carefully.
  - the hardware adds data-depended behavior.

Are there any operations in AES that are critical in the context of timing attacks?

3.1 TA on AES and Countermeasures
Timing Dependencies of Data within AES

- Timing dependencies that do not (directly or indirectly) depend on the secret AES key seem to be uncritical.
- Timing dependencies that depend on the secret key might be used to decide whether the key (or parts of it) has a certain value or not.
- Question 1: Which data depend on the secret key k?
  Answer: All intermediary data of AES:
  - $r^{(i)}$
  - $sb^{(i)}$
  - $sr^{(i)}$
  - $mc^{(i)}$

  → All operations on these data might be a source of data dependent timings.

3.1 TA on AES and Countermeasures
Timing Dependencies of Data within AES

- Question 2: Which operations (and their timing) might depend on these intermediary data and therefore on the secret key k?
  Answer:

	implemented via:	
☐ RK_i	XOR-operation	→ no
☐ SB_i	table-lookup	→ probably not
☐ SR_i	operation independent of data	→ no
☐ MC_i	xtime in software with if-else	→ yes

3.1 TA on AES and Countermeasures
Implementation of the xtime Operation

- An efficient implementation of `xtime(s)` was already shown:

```
x = s << 1;
if (s & 0x80)
  x ^= 0x1B;
```

- shift left
- reduce by p(x)

x, s are 8-bit unsigned integer variables

¬ Timing dependency on input data: If the MSB of `s` is 1, the operation of `xtime` takes longer than if the MSB of `s` is 0.

- The following implementation has constant timing, if the hardware provides constant timing of the instructions:

```
x = s << 1;
t = -MSB(s)
t = t & 0x1B;
x = x ^ t
```

- where `-1 = 0xFF`

3.1 TA on AES and Countermeasures Attack Settings

- The MixColums operation is carelessly implemented, e.g., using the data dependent `xtime` operation from above. I.e., every byte of $sr^{(i)}$ will be input of (exactly) one `xtime` operation and the timing of this operation is shorter or longer whether the MSB of $sr^{(i)}_{j\ (j=0,\ldots,15)}$ is set or not.
- The AES implementation encrypts n random data blocks (d_i), $i=1,\ldots,n$.
- The attacker knows the plaintexts (d_i) entering AES and is able to observe the timing (t_i) for each encryption.
- Everything else has constant timing. (Not really necessary, but it makes things easier.)

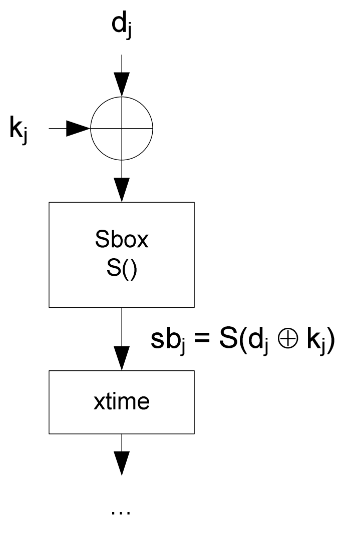
Plaintext	Timing
d_1	t_1
d_2	t_2
…	…
d_{n-1}	t_{n-1}
d_n	t_n

3.1 TA on AES and Countermeasures

- For each byte $(d)_j$ of the plaintext d the following steps are performed at the beginning of the AES encryption:
  1. The byte $(d)_j$ is xored with a key byte k_j.
  2. The result is sent through the AES Sbox.
  3. The output sb_j of the Sbox is then sooner or later multiplied by x using the xtime function.

- Depending on the fact whether the MSB of the Sbox output is 1 or not, the AES encryption takes longer or shorter, say by the value of δ.

- What can we learn from this?



3.1 TA on AES and Countermeasures
Simple Example

Assume we could observe the timing of a single `xtime` operation (this is equivalent to knowing the MSB of the Sbox output)

- As the input data is known it is easy to calculate the Sbox output for all 256 possible candidates for this byte of the key.
- The table on the left side shows the Sbox output sb_j for different inputs $(d)_j$ and different keys k_j; the table on the right shows the MSB of the Sbox output;
- The column t shows the execution time for the observed xtime operation (0 means short, 1 means long)

		KEY					
d	t	00	01	02	03	04	...
BE	0	AE	08	65	7A	F4	...
CA	1	74	1F	E8	DD	8B	...
4B	0	B3	D6	3B	52	84	...
B5	0	D5	8D	A9	4E	C8	...
81	0	0C	CD	EC	13	97	...

		KEY					
d	t	00	01	02	03	04	...
BE	0	1	0	0	0	1	...
CA	1	0	0	1	1	1	...
4B	0	1	1	0	0	1	...
B5	0	1	1	1	0	1	...
81	0	0	1	1	0	1	...

3.1 TA on AES and Countermeasures
Simple Example

Assume we could observe the timing of a single `xtime` operation (this is equivalent to knowing the MSB of the Sbox output)

- As the input data is known it is easy to calculate the Sbox output for all 256 possible key candidates for this byte
- Based on this the key byte can be revealed easily

Only this key leads to the observed timing behavior
→ this is the key of the device

		KEY					
d	t	00	01	02	03	04	...
BE	0	AE	08	65	7A	F4	...
CA	1	74	1F	E8	DD	8B	...
4B	0	B3	D6	3B	52	84	...
B5	0	D5	8D	A9	4E	C8	...
81	0	0C	CD	EC	13	97	...

		KEY					
d	t	00	01	02	03	04	...
BE	0	1	0	0	0	1	...
CA	1	0	0	1	1	1	...
4B	0	1	1	0	0	1	...
B5	0	1	1	1	0	1	...
81	0	0	1	1	0	1	...

3.1 TA on AES and Countermeasures Observations

1. All parts of the implementation with constant timing do not change the attack (it just adds an offset to the overall timing).

2. All data-dependent timings except for the attacked one can be viewed as noise; the consumed time for the overall AES is on average still smaller if the MSB of the considered Sbox output is 0 and longer, if it is 1.

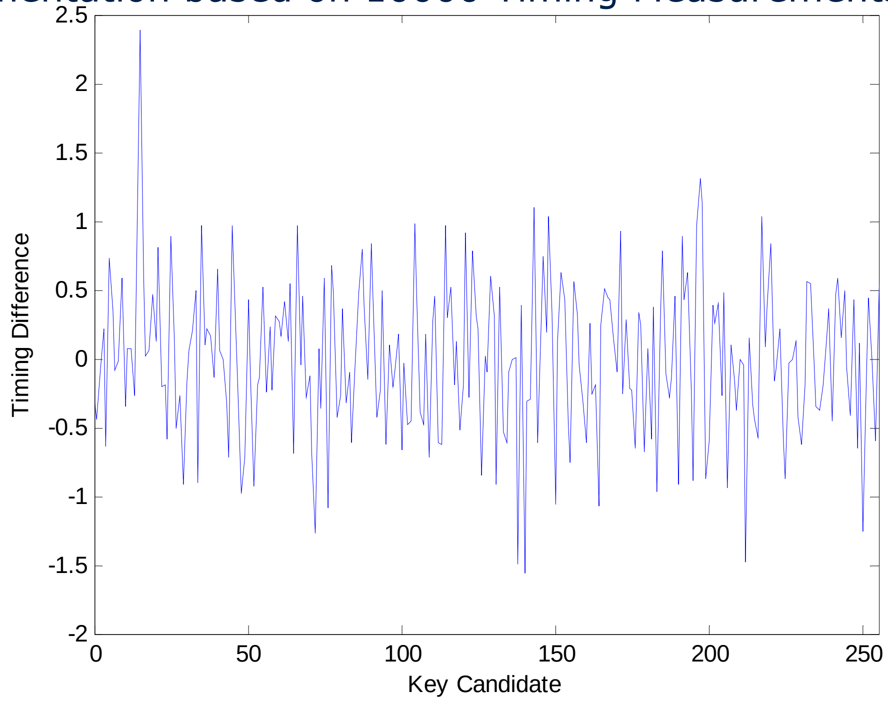
This leads to the following attack strategy: (See also [KQ99])

3.1 TA on AES and Countermeasures Attack Description

- Observe timings t_i of n encryptions with random input d_i $(i=1,\ldots,n)$.
- Repeat the following for each 16 Sbox output $sb^{(1)}_j$ in round 1:
  Select one Sbox output $sb^{(1)}_J$ in round 1 as attack target.
  - For each candidate k_J^{hyp} of the key byte k_J do:
    - For each input d_i of this Sbox calculate the MSB x_{Ji} of the Sbox output $sb^{(1)}_J(d_i) = S(k_J^{hyp}\ xor\ (d_i)_J)$ of the J-th Sbox.
    - Calculate the average timing $\underline{t}^+$ for the encryptions where the MSB x_{Ji} is 1, and the average timing $\underline{t}^-$ for the encryptions where the MSB x_{Ji} is 0.
  - For all key candidates k_J^{hyp} for k_J compare the difference $\underline{t}^+ - \underline{t}^-$ between these average timings.
  - Select the key byte value as best guess for the key byte k_J that leads to the largest difference.
  - If no significant peak occurs, perform more encryptions to reduce the noise further.

3.1 TA on AES and Countermeasures

- Result for $(\underline{t^+} - \underline{t^-})(k_J)$ of an Attack on an AES Software Implementation based on 10000 Timing Measurements:



Insertion: Statistics Recapitulation I

Definition:

Ω	any probability space, i.e., any set with a probability measure P.
$f, f_i: \Omega \to \mathbf{R}$	real valued random variables, random functions.
$\mathbf{E}$	expected value.

- $\mathbf{E}(f) = \sum_{x \in f(\Omega)} x \cdot P(f = \{x\})$ (discrete case)
- $\mathbf{E}(f) = \int_\Omega f \cdot dP$ (continuous case)

Var, σ	variance, standard deviation

- $\mathrm{Var}(f) = \mathbf{E}((f - \mathbf{E}(f))^2) = \mathbf{E}(f^2) - \mathbf{E}(f)^2$, $\sigma(f) = \sqrt{\mathrm{Var}(f)}$.

Properties:

- For any random functions f, f_i :
  - $\mathbf{E}(f_1 + \ldots + f_n) = \mathbf{E}(f_1) + \ldots + \mathbf{E}(f_n)$.
  - $\mathbf{E}(af) = a\,\mathbf{E}(f)$, for any $a \in \mathbf{R}$.
  - $\mathrm{Var}(af) = a^2\,\mathrm{Var}(f)$, $\mathrm{Var}(a + f) = \mathrm{Var}(f)$.
- If the random functions are independent:
  - $\mathrm{Var}(f_1 + \ldots + f_n) = \mathrm{Var}(f_1) + \ldots + \mathrm{Var}(f_n)$.

Example:

Let	$f: \Omega \to \{0,1\}$,
with	$P(f = \{1\}) = p$
and	$P(f = \{0\}) = 1 - p =: q$.
Then	$\mathbf{E}(f) = 1 \cdot p + 0 \cdot q = p$
and	$\mathrm{Var}(f) = \mathbf{E}(f^2) - \mathbf{E}(f)^2 = p - p^2 = p\,q$

3.1 TA on AES and Countermeasures
Why does this attack work? I

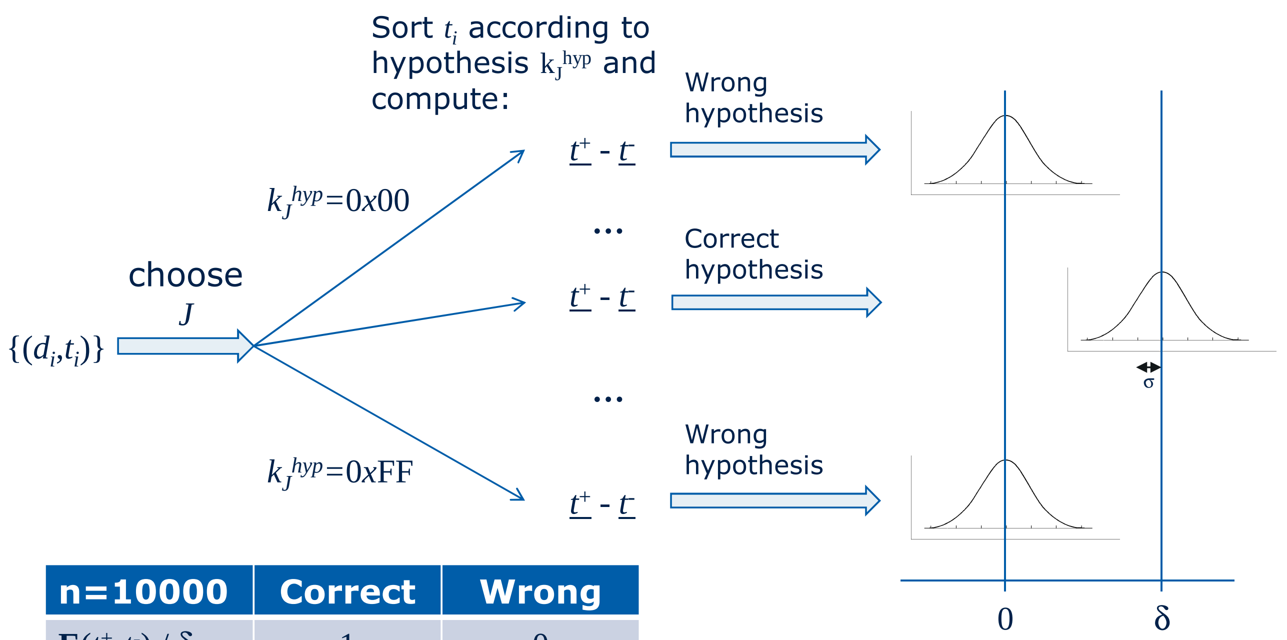
Theorem: Assume that the messages $(d_i)_{i=1,\ldots,n}$ are chosen randomly, modeled by the random variable

$$d : \Omega \to (\mathbf{F}_2^{128})^n,\ \omega \to (d_i),$$

and assume, one can neglect other timing variations besides the one resulting from xtime, then:

- For any choice of $J \in \{0,\ldots,15\}$ and $k_J^{hyp} \in \{0x00,\ldots,0xFF\}$, the functions t_i and the thereof derived functions $\underline{t}^+$, $\underline{t}^-$, and $(\underline{t}^+ - \underline{t}^-)$ are random functions which are (nearly) normally distributed.
- If $k_J^{hyp} = k_J$ (correct hypothesis), then
  - $\mathbf{E}(\underline{t}^+ - \underline{t}^-) = \delta$
  - $\mathrm{Var}(\underline{t}^+ - \underline{t}^-) = \delta^2 \cdot (143/n)$.
- If $k_J^{hyp} \neq k_J$ (wrong hypothesis), then
  - $\mathbf{E}(\underline{t}^+ - \underline{t}^-) = 0$
  - $\mathrm{Var}(\underline{t}^+ - \underline{t}^-) = \delta^2 \cdot (144/n)$.

3.1 TA on AES and Countermeasures
Why does this attack work? II


Sort t_i according to hypothesis k_J^{hyp} and compute:
$\{(d_i,t_i)\}$
choose J
k_J^{hyp}=0x00
k_J^{hyp}=0xFF
$\underline{t}^+ - \underline{t}^-$
…
$\underline{t}^+ - \underline{t}^-$
…
$\underline{t}^+ - \underline{t}^-$
Wrong hypothesis
Correct hypothesis
Wrong hypothesis
σ
0
δ

n=10000	Correct	Wrong
$\mathbf{E}(\underline{t}^+ - \underline{t}^-) / \delta$	1	0
$\mathbf{Var}(\underline{t}^+ - \underline{t}^-) / \delta^2$	≈1/70	≈1/69
$\sigma(\underline{t}^+ - \underline{t}^-) / \delta$	≈1/8	≈1/8

For each J, k_J^{hyp}: $\underline{t}^+ - \underline{t}^-$ is a random function, generated from the random functions t_i, and hence normally distributed.

3.1 TA on AES and Countermeasures
Proof of the Theorem I

- Computation of t_i:

  Define the MSB of every Sbox output (which goes into the MixColumn operation)

  $$x_{ji}^{(h)} \quad := \mathrm{MSB}(sb^{(h)}_j (d_i)) \text{ , for } i=1,\dots,n;\ j=0,\dots,15;\ h=1,\dots,9.$$

  These functions $x_{ji}^{(h)}$ can be seen as random functions (since they depend on the randomly chosen input (d_i)) with values in $\{0,1\}$, as in the example above for $p=q=1/2$.

  Furthermore one may make the heuristic assumption that all these random functions are practically independent.

  Then, we have

  $$t_i \quad = C + \sum_{j,h} (x_{ji}^{(h)} \cdot \delta)$$

  C is some constant that depend on the implementation. This includes also the timings for all xtime-operations, but only counted as short operations.

  $$\mathbf{E}(t_i) \quad = C + \delta \sum_{j,h} \mathbf{E}(x_{ji}^{(h)}) \quad = C + \delta \cdot (144/2)$$

  $$\mathrm{Var}(t_i) \quad = \delta^2 \sum_{j,h} \mathrm{Var}(x_{ji}^{(h)}) \quad = \delta^2 \cdot (144/4)$$

3.1 TA on AES and Countermeasures
Proof of the Theorem II

- Computation of $\underline{t}^+$: (for a fixed Sbox J)

  Let $\underline{t}^+$ be the average of all t_i, where $x_{Ji}^{(1)}=1$, (with a long `xtime`). Assume, their number is n^+. Then

  $$\underline{t}^+ \quad = (1/n^+) \sum^+_i t_i$$

  Where the sum $\sum^+_i$ runs only over the n^+ indices i with $x_{Ji}^{(1)}=1$.

  $$\mathbf{E}(\underline{t}^+) \quad = (1/n^+) \sum^+_i \mathbf{E}(t_i)$$

  $$= C + \delta \cdot (145/2) \text{ ,}$$

  because in this case

  $$\mathbf{E}(t_i) \quad = C + \delta \sum_{j,h} \mathbf{E}(x_{ji}^{(h)}) = C + \delta \cdot (143/2 + 1) \text{ ,}$$

  where $\mathbf{E}(x_{Ji}^{(1)})=1$ and $\mathbf{E}(x_{ji}^{(h)})=1/2$, for all other indices.

  $$\mathrm{Var}(\underline{t}^+) = (1/n^+)^2 \sum^+_i \mathrm{Var}(t_i)$$

  $$= (1/n^+) \cdot \delta^2 \cdot (143/4) \text{ ,}$$

  since in this case

  $$\mathrm{Var}(t_i) = \delta^2 \cdot \sum_{j,h} \mathrm{Var}(x_{ji}^{(h)}) = \delta^2 \cdot (143/4) \text{ ,}$$

  where $\mathrm{Var}(x_{Ji}^{(1)}) = 0$ and $\mathrm{Var}(x_{ji}^{(h)}) = 1/4$, for all other.

3.1 TA on AES and Countermeasures Proof of the Theorem III

- Analogously for $\underline{t}^-$: (for a fixed Sbox J)

$$\mathbf{E}(\underline{t}^-) = (1/n^-) \sum^-_i \mathbf{E}(t_i) =$$
$$= C + \delta \cdot (143/2)\,,$$

since in this case

$$\mathbf{E}(t_i) = C + \delta \cdot \sum_{j,h} \mathbf{E}(x_{ji}^{(h)}) = C + \delta \cdot (143/2)\,,$$

where $\mathbf{E}(x_{Ji}^{(1)}) = 0$ and $\mathbf{E}(x_{ji}^{(h)}) = 1/2$, for all other indices. Also

$$\mathrm{Var}(\underline{t}^-) = (1/n^-) \cdot \delta^2 \cdot (143/4)\,,$$

- Then one has for $\underline{t}^+ - \underline{t}^-$:

$$\mathbf{E}(\underline{t}^+ - \underline{t}^-) = \mathbf{E}(\underline{t}^+) - \mathbf{E}(\underline{t}^-) = \delta$$
$$\mathrm{Var}(\underline{t}^+ - \underline{t}^-) = \mathrm{Var}(\underline{t}^+) + \mathrm{Var}(\underline{t}^-)$$
$$= ((1/n^+) + (1/n^-)) \cdot \delta^2 \cdot (143/4)\,.$$

If one assumes $n^+ = n^- = n/2$, then

$$\mathrm{Var}(\underline{t}^+ - \underline{t}^-) = (4/n) \cdot \delta^2 \cdot (143/4) = \delta^2 \cdot (143/n)$$

This describes the properties of the value $(\underline{t}^+ - \underline{t}^-)$ if it is computed from the measurements (t_i) with the **correct** key (hypothesis) for k_J.

3.1 TA on AES and Countermeasures Proof of the Theorem IV

- For a **wrong** hypothesis, all the functions $x_{ji}^{(h)}$ (including $x_{Ji}^{(1)}$) are random functions and the computed values for $\underline{t}^+$ and $\underline{t}^-$ have the properties

$$\mathbf{E}(\underline{t}^+) = (1/n^+) \sum^+_i \mathbf{E}(t_i) = C + \delta \cdot (144/2)\,,$$
$$\mathrm{Var}(\underline{t}^+) = (1/n^+)^2 \sum^+_i \mathrm{Var}(t_i) = (1/n^+) \cdot \delta^2 \cdot (144/4)\,,$$
$$\mathbf{E}(\underline{t}^-) = (1/n^-) \sum^-_i \mathbf{E}(t_i) = C + \delta \cdot (144/2)\,,$$
$$\mathrm{Var}(\underline{t}^-) = (1/n^-)^2 \sum^-_i \mathrm{Var}(t_i) = (1/n^-) \cdot \delta^2 \cdot (144/4)\,,$$

Therefore

$$\mathbf{E}(\underline{t}^+ - \underline{t}^-) = \mathbf{E}(\underline{t}^+) - \mathbf{E}(\underline{t}^-) = 0$$
$$\mathrm{Var}(\underline{t}^+ - \underline{t}^-) = \mathrm{Var}(\underline{t}^+) + \mathrm{Var}(\underline{t}^-)$$
$$= ((1/n^+) + (1/n^-))\, \delta^2\, (144/4)$$
$$= (4/n)\, \delta^2\, (144/4)$$
$$= \delta^2\, (144/n).$$

If one assumes $n^+ = n^- = n/2$.

3.1 TA on AES and Countermeasures Remarks

- The described attack can be improved by first characterizing the timing of the encryption or by using other statistical tests. (see power analysis attacks)

- The attack on the AES xtime implementation can be conducted in a similar way on any other block cipher implementations that implement certain transformations on secret-dependent data in a data-dependent way.

- Remember for the implementation of all cryptographic algorithms: Each intermediate data value carries information about the key!

3.1 TA on AES and Countermeasures Countermeasures

- Execution time of the algorithm must not depend on secret information, i.e., on information derived from the secret (key)!
  - **Constant timing**:
    - Do not use “while” statements that use secret values in the loop condition.
    - Do not perform conditional branches based on secret information.
    - If possible, write code without conditional branches (example: `xtime`)
  - **Independent Timing** by randomization: instead of computing $\mathrm{xtime}(s)$, choose a random byte r and compute:

    $$\mathrm{xtime}(s \oplus r) \oplus \mathrm{xtime}(r).$$

    This is equal to $\mathrm{xtime}(s)$, since xtime is linear and the timing of this new operation is now independent of s on average.
    - Why?
    - In which situations should one use this countermeasure?

Chapter 3

Timing Dependencies in Hardware Implementations

3.1 TA on AES and Countermeasures Data-Dependencies in Hardware

- Throughput is one of the main optimization goals when building hardware.
- Throughput optimization is not something static that only takes place during the hardware development.
- Typically, there are several mechanisms built into the hardware to optimize the throughput during the execution of the software.
- Not every instruction always takes the same amount of time to execute! So even if there are no data-dependencies in the software, the hardware might add data-dependencies.
- Architectural components that often lead to data-dependent behavior on embedded devices:
  - Functional Units
  - Cache

3.1 TA on AES and Countermeasures

Remark on Data-Dependency in more Complex Processors

- Usually, the more complex a processor is, the more data-dependencies exist.
- On processors running multiple tasks in parallel a “spy task” can be used to attack a “security task” that processes secret data.
- Basic idea of such attacks:
  - There are several hardware components shared between different tasks (e.g. branch prediction unit, …).
  - The spy task can setup the shared resource in such a way that it is possible to check whether the security task has used this resource or not.
  - If the access on the resource depends on secret data, the spy process can collect information about this secret.

More information and references on this can for example be found in Chapter 18 of [K09].

3.1 TA on AES and Countermeasures

Example: Data-Dependency of Functional Units in an ARM Cortex M3 Processor

Typical examples of functional units with data-dependent execution time are complex arithmetic operations such as multiplication and division.

Data-dependent functional units of the ARM Cortex M3 processor (a low-power 32-bit processor):

- Multiplication: two 32 bit multiplicands, 64 bit result

  “UMULL/SMULL/UMLAL/SMLAL use early termination depending on the size of source values. […] MLAL versions take four to seven cycles and MULL versions take three to five cycles. For MLAL, the signed version is one cycle longer than the unsigned.”

- Division: 32 bit dividend, 32 bit divisor, 32 bit result

  “DIV timings depend on dividend and divisor. […] When dividend and divisor are similar in size, divide terminates quickly. Minimum time is for cases of divisor larger than dividend and divisor of zero. A divisor of zero returns zero (not a fault), although a debug trap is available to catch this case.”

Quoted from the Cortex M3 Technical Reference Manual: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf

3.1 TA on AES and Countermeasures
Example: Data-Dependency of Functional Units in an ARM Cortex M3 Processor

- Conditional Branches

  "Branches take one cycle for instruction and then pipeline reload for target instruction. Non-taken branches are 1 cycle total. Taken branches with an immediate are normally 1 cycle of pipeline reload (2 cycles total). Taken branches with register operand are normally 2 cycles of pipeline reload (3 cycles total). Pipeline reload is longer when branching to unaligned 32-bit instructions in addition to accesses to slower memory. A branch hint is emitted to the code bus that permits a slower system to pre-load. This can reduce the branch target penalty for slower memory, but never less than shown here."

  → Even if exactly the same instruction sequence is executed at each branch target, the timing potentially still depends on the tested condition!

- Load/Store

  "Generally, load-store instructions take two cycles for the first access and one cycle for each additional access. Stores with immediate offsets take one cycle."

Quoted from the Cortex M3 Technical Reference Manual: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf

3.1 TA on AES and Countermeasures
Summary on Data-Dependencies of Functional Units

Know your processor!

- For secure implementations of cryptographic algorithms it is necessary to know the timing behavior of the underlying hardware.

- Problem for programmers in practice: Things such as the exact timing behavior under all possible circumstances are not so critical for normal applications. Hence, not always the hardware reference manual contains all necessary information → double check timing behavior when you rely on certain properties.

3.1 TA on AES and Countermeasures The Cache

- The basic idea of a cache is to buffer memory content inside the processor. Caches are used for two reasons:
  - Performance: accessing a data block in memory is significantly slower than a cache access.
  - Power consumption: accessing a data block in memory consumes more energy than a cache access.
- Caches can be implemented as:
  - data-cache.
  - instruction cache.
  - unified data and instruction cache.
- Cache size: 1kB up to several MB.
- On PCs there are multiple (typically 3) levels of caching.

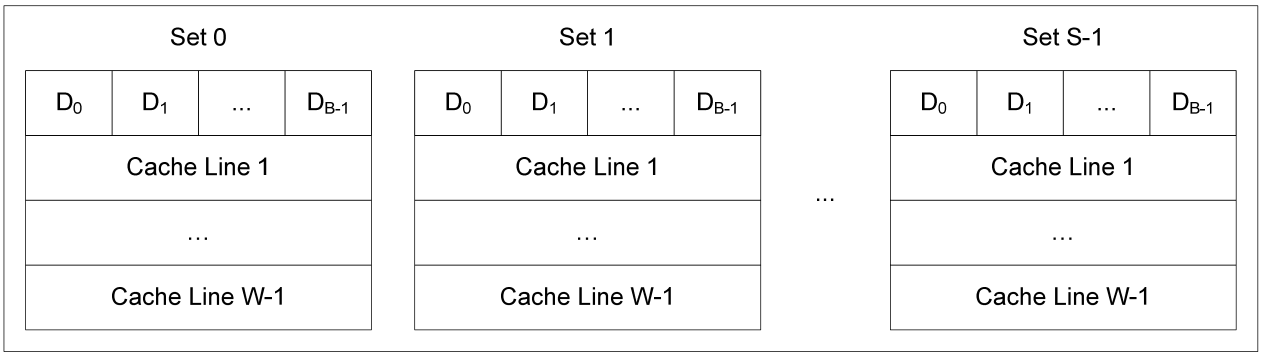
3.1 TA on AES and Countermeasures Set-Associative Cache

- Building a cache, where each data word of the memory can be stored at each location in the cache is usually too expensive in practice (all the cache would need to be searched in parallel upon an access).
- In practice *set-associative caches* are used as a compromise between hardware effort and cache efficiency.
- Structure of a set-associative cache:
  - Cache: the cache is split into S *cache sets*.
  - Cache sets: each cache set consists of W *cache lines*.
    (W is referred to as the number of *ways* of the cache – e.g. in a 4 way cache one has W=4.)
  - Cache line: each cache line contains B data bytes.
- The total cache size is S*W*B bytes.

3.1 TA on AES and Countermeasures Set-Associative Cache

- Each data word can only be stored in one of the sets (typically determined by the lowest bits of the memory address) → there are W positions where the data word can be stored.

- Upon an access to the cache only one set of the cache needs to be searched.



3.1 TA on AES and Countermeasures Cache Operations

Upon a cache access, the following things can happen:

- *Cache hit*: the requested data block is in the cache and no memory access is necessary.
- *Cache miss*: the requested data block is not in the cache.
  - In this case a block of the cache is replaced by the requested data block. (Most common replacement policy: replace the least recently used block.)
  - The replaced block is called *victim*.
  - If the victim has been changed in cache it is *dirty*.
  - Before storing the new block in the cache, the old one needs to be written back to memory if it is dirty.

A cache hit is handled significantly faster than a cache miss!

3.1 TA on AES and Countermeasures Cache based Timing Attacks

- Cache-based attacks exploit the fact that cache hits/misses depend on some secret value.
- Typical target of cache-based timing attacks are table lookups.
- The time to access a certain element of a table is only independent of the index of the accessed element, if the entire table is stored in the cache.
- If some elements are not in the cache and the attacker knows which elements are not in the cache, this can be exploited in a timing attack.

3.1 TA on AES and Countermeasures Categories of Timing Attacks

- **Trace-driven attacks**: the attacker is able to obtain a full trace of cache hits/misses.
  - Examples: MHMMHH, MHHHMM, …
  - Such traces can for example be obtained by a spy process that runs on the same machine as the attacked process.
  - Another alternative is measuring the power consumption (see power analysis attacks).
- **Time-driven attacks**: the attacker can only observe the total numbers of cache hits/misses;
  - This can for example be determined by measuring the execution time.
  - This attack can potentially be conducted even remotely.
- **Access-driven attacks**: the attacker is able to obtain information about which sets of the cache are accessed by the attacked process
  - Such attacks are typically conducted by a spy process with significantly relaxed requirements compared to trace-driven attacks; Essentially the spy process evicts certain sets with garbage and then checks whether this set is changed by the attacked process.

3.1 TA on AES and Countermeasures Time-Driven Attacks on AES

Assumption:

- Not the entire AES Sbox is (or T-Tables are) stored in the cache. This can for example be achieved by the attacker by executing other operations between two encryptions.

This leads to the following cache-collision assumption:

- For any pair of lookups i, j, given a large number of random AES encryptions with the same key, the average time when $l_i = l_j$ will be less than the average time when $l_i \neq l_j$. [BM06] (l_i and l_j denote the indices of the lookups i, j.)

3.1 TA on AES and Countermeasures Time Driven Attack on the First Round

- In the first round, table lookups are done after the initial key XOR; the table index of the ith lookup is $d_i \oplus k_i$.
- For any two lookups i, j the average time is shorter, if $S(d_i \oplus k_i) = S(d_j \oplus k_j)$. This implies $d_i \oplus k_i = d_j \oplus k_j$, because S is bijective. Set

$$dk_{i,j} := d_i \oplus d_j = k_i \oplus k_j.$$

- An attacker can determine the values $dk_{i,j}$ as follows:
  - Perform AES encryptions with random plaintexts.
  - Determine the average execution time $\underline{t}(i,j, dk_{i,j})$ for each pair of lookups i, j and a given difference $dk_{i,j}$.(256 possible values)
  - The timing $\underline{t}(i,j, dk_{i,j})$ that is significantly lower than all other ones identifies the value $dk_{i,j}$.
- In principle, in this way the difference between all keys that are used for the same table lookup can be determined.

3.1 TA on AES and Countermeasures
Time Driven Attack on the First Round

Note: The description of the attack from above is simplified. It is not possible to reveal some $\log_2(B)$ bits of each delta key byte due to the fact that the cache is organized in cache lines.

- There are B bytes in each cache line.
- All inputs of a table lookup that only differ in the lowest $\log_2(B)$ bits access the same cache line.

Consequence:

- It is possible to determine only $(8\text{-}\log_2(B))$ bit of the deltas $dk_{i,j}$ between all 16 lookups. Equivalently, one evaluates a set of B values for each $dk_{i,j}$.
- The attacker needs to guess one key byte and the remaining bits of the deltas.
- Attack complexity: $8 + 15\cdot\log_2(B)$. (I.e., $2^8\cdot B^{15}$ key candidates.)
- So, revealing the full key is typically impractical in this way and the attack needs be extended to the second round (see for example [TOS10]).

3.1 TA on AES and Countermeasures
Summary of Cache Timing Attacks

Summary:

- Cache timing attacks on AES are possible. (This holds for implementations with Sbox Tables as well as for implementations with T-tables.)
- The attack in the first round does not lead to the full key directly due to cache structure → an extension to the second round is necessary.

Countermeasures:

- Ensure that table lookups are completely in the cache before executing an algorithm and stay in cache during the execution. (hardware support necessary!)
- Alternatively, avoid table lookups and compute the lookup tables. But this also has to be implemented secure against timing analysis. (See [KS09] for an AES implementation)

3.1 TA on AES and Countermeasures Summary: Timing Attacks on Symmetric Ciphers

Summary:

- Pure timing attacks on symmetric cryptography is rather a niche topic in practice.
- Reason: there is only a limited number of scenarios, where resistance against timing attacks is required, but no resistance against power analysis is necessary.
- Timing attacks are highly critical in connection with power analysis attacks (this allows trace-driven attacks) → hence, timing effects are usually discussed in the context of power analysis attacks.

Countermeasures:

- Security-aware programming:
  - no branch operations depending on secret data.
  - avoid instructions with data-dependent timing when processing secret data.
- Use Dedicated Hardware
  - The 2010 Intel® Core™ processor family in 32nm implements dedicated instructions for AES (see [I10]);
  - Together with performance, timing attacks had been the driver to implement these instructions!

Chapter 3

3.2. Timing Attacks on RSA and Countermeasures

3.2 TA on RSA and Countermeasures Timing Dependence in RSA Implementations

- Even if only the timing information of complete exponentiations is known, then a timing attack is possible, if Montgomery multiplication was used for the implementation of the exponentiation
  $m^d \bmod N$.

- Most implementations of RSA use Montgomery multiplication. There, a squaring can not be distinguished from a multiplication. Mainly, the operation A*B is timely constant with one exception:

  The extra reduction (er) at the end is data dependent!

Insertion: Statistics Recapitulation II

Definition:

- **cov** covariance
  - $\mathbf{cov}(f, g) = \mathbf{E}[(f - \mathbf{E}f)\cdot(g - \mathbf{E}g)]$, with $f, g: \Omega \rightarrow \mathbf{R}$ two random functions.
- **corr** correlation coefficient.
  - $\mathbf{corr}(f, g) = \mathbf{cov}(f,g)/(\sigma(f)\,\sigma(g))$.

Properties:

- The correlation coefficient measures the linear dependency between two random functions.
  - $-1 \leq \mathbf{corr}(f,g) \leq 1$
  - If f and g are independent random functions then $\mathbf{corr}(f,g)=0$.
  - If $f = a\cdot g + b$, $a > 0$ then $\mathbf{corr}(f,g) = 1$.
  - If $f = a\cdot g + b$, $a < 0$ then $\mathbf{corr}(f,g) = -1$.
  - $\mathbf{corr}(a\cdot f+b, c\cdot g+d) = \mathbf{corr}(f, g)$, if $a, c > 0$.

Computation of corr:

If $x_1,\ldots,x_n$ and $y_1,\ldots,y_n$ are independent f and g-distributed measurements/ samples that are. Then a good estimate of the correlation coefficient $\mathbf{corr}(f,g)$ is given by:

$$\frac{\sum_{i=1}^{n} (x_i-\underline{x})\cdot(y_i-\underline{y})}{\left(\sum_{i=1}^{n} (x_i-\underline{x})^2 \cdot \sum_{i=1}^{n} (y_i-\underline{y})^2\right)^{1/2}}$$

where, $\underline{x} = (\sum_{i=1}^{n}(x_i))/n$, $\underline{y} = (\sum_{i=1}^{n}(y_i))/n$.

3.2 TA on RSA and Countermeasures
Timing of Montgomery Multiplication

- **Assumtion:**
  A Montgomery multiplication modulo N needs the time
  τ, if no extra reduction is needed, and
  $\tau+\delta$, if an extra reduction is needed.
  Then an exponentiation needs the time

$$t = (n + \mathrm{hw}(d)) \cdot \tau + (\#\mathrm{er's}) \cdot \delta.$$

```
A * B:
C := A · B;        // $C \in [0, N^2[$
D := C · N′ mod Z; // $D \in [0, Z[$
E := C + D · N;    // $E \in [0, N^2+ZN[$ & $E \equiv 0 \bmod Z$
F := E div Z;      // $F \in [0, N+N[$
if F>=N then
          F:=F-N;  // extra reduction step
return F;
```

- Define the following function er:
  - $\mathrm{er}\,(m) = 1$, if an extra reduction is needed during the operation $m * m$.
  - $\mathrm{er}(m) = 0$, if no extra reduction is needed.

3.2 TA on RSA and Countermeasures
Timing Attack on RSA with Montgomery Multiplication

- Let t_j be the time the exponentiation needs for the input m_j.
- Attack by induction on i:
  1. Assume $D_{i\text{-}1} = (d_{n\text{-}1}\, d_{n\text{-}2} \ldots d_{n\text{-}i+1})_2$ is already known to the attacker.
  2. Make hypothesis on $d_{n\text{-}i} = 0$ or 1, i.e., $D_i = D'$ or D''.
  3. Compute $\mathrm{er}_{0j} = \mathrm{er}\,(m_j^{D'} \cdot Z \bmod N\,)$ and $\mathrm{er}_{1j} = \mathrm{er}\,(\,m_j^{D''} \cdot Z \bmod N\,)$.
     This reflects the timing (short vs. long) of the squaring during the lrSM in loop $(n\text{-}i)$.
  4. Evaluate the correlation coefficients
     $e_0 := \mathbf{corr}(\mathrm{er}_{0j}, t_j)$ and $e_1 := \mathbf{corr}(\mathrm{er}_{1j}, t_j)$
  5. If $e_0 > e_1$, then $D_i = D'$,
     else if $e_0 < e_1$, then $D_i = D''$.
  6. Increment i and go to (1.) until $i = n\text{-}1$.
  7. Guess the last digit d_0.

```
M′ := M * Z²;
X  := Z;
for i:=n-1 to 0 by -1 do
    X := X * X;
    if $d_i$=1 then X := X * M′; end;
    // $X = m^{D_{n-i}} \cdot Z \bmod N$
end;
S  := X * 1;
return S;
```

3.2 TA on RSA and Countermeasures Countermeasures

Generic countermeasures:

Implement Montgomery multiplication with constant timing. This can be done by implementing a dummy (extra reduction)-step if no extra reduction is necessary.

Algorithmic countermeasures:

Hinder attacker to being able to compute intermediary values like $m^{Di} \bmod N$, by:

- Randomization of N: $(m^d \bmod (N \cdot r_1)) \bmod N$
- Randomization of m: $((m+N \cdot r_2)^d \bmod (N \cdot r_1)) \bmod N$
- Randomization of d: $m^{d+r \cdot \varphi(N)} \bmod N$
- Randomization of m: take random (r,s) with $1=s \cdot r^d \bmod N$
  $((m \cdot r)^d \bmod N) \cdot s \bmod N$
  update $r := r^2$ and $s := s^2$.

References

[BM06] J. Bonneau and I. Mironov: *Cache-Collision Timing Attacks against AES*, Workshop on Cryptographic Hardware and Embedded Systems — CHES 2006, LNCS, Springer Verlag.

[DR99] Joan Daemen and Vincent Rijmen: AES Proposal: *Rijndael*, available online at http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[DR02] Joan Daemen and Vincent Rijmen: *The Design of Rijndael: AES - The Advanced Encryption Standard*, Springer Verlag 2002.

[I10] Intel Corporation: *Intel® Advanced Encryption Standard (AES) Instructions Set*, a vailable online at: http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/.

[K09] Cetin Kaya Koc, *Cryptographic Engineering*, Springer Verlag, 2009.

[KS09] Emilia Käsper and Peter Schwabe: *Faster and Timing-Attack Resistant AES-GCM*, Workshop on Cryptographic Hardware and Embedded Systems — CHES 2009, LNCS, Springer Verlag, 2009.

[KQ99] Francois Koeune, Jean-Jacques Quisquater: *A Timing Attack against Rijndael*. UCL Louvain, Crypto Group, Technical report CG-1999/1.

[NIST99] National Institute of Standards and Technology (NIST): *FIPS-46-3: Data Encryption Standard*, 1999.

[NIST01] National Institute of Standards and Technology (NIST): *FIPS-197: Advanced Encryption Standard*, 2001.

[TOS10] Eran Tromer, Dag Arne Osvik, Adi Shamir: *Efficient Cache Attacks on AES, and Countermeasures*, Journal of Cryptology, Volume 23, No. 1, Springer Verlag, 2010.