MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

High-assurance crypto – Part II: Security

Peter Schwabe

January 30, 2023

Optimizing Crypto vs. optimizing “something”

- So far there was nothing crypto-specific in this lecture
- Is optimizing crypto the same as optimizing any other software?

Optimizing Crypto vs. optimizing "something"

- So far there was nothing crypto-specific in this lecture
- Is optimizing crypto the same as optimizing any other software?

No – we must not leak secret data to an attacker!

Hello World – with a secret

```
#include <sys/random.h>
#include <unistd.h>
#include <stdio.h>

int main(void)
{
  unsigned char secret;
  getrandom(&secret, sizeof(secret), 0);
  secret &= 1;

  if(secret) sleep(3);

  printf("Hello World!\n");
}
```

Timing leakage part I

- Consider the following piece of code:

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

Timing leakage part I

- Consider the following piece of code:

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- General structure of any conditional branch
- A and B can be large computations, r can be a large state

Timing leakage part I

- Consider the following piece of code:

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- General structure of any conditional branch
- A and B can be large computations, r can be a large state
- This code takes different amount of time, depending on s
- Obvious timing leak if s is secret

Timing leakage part I

- Consider the following piece of code:

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- General structure of any conditional branch
- A and B can be large computations, r can be a large state
- This code takes different amount of time, depending on s
- Obvious timing leak if s is secret
- Even if A and B take the same amount of cycles this is *generally not* constant time!
- Reasons: Branch prediction, instruction-caches
- **Never use secret-data-dependent branch conditions**

Eliminating branches

- So, what do we do with this piece of code?

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

Eliminating branches

- So, what do we do with this piece of code?

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- Replace by

$$r \leftarrow sA + (1 - s)B$$

Eliminating branches

- So, what do we do with this piece of code?

  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- Replace by

$$r \leftarrow sA + (1 - s)B$$

- Can expand s to all-one/all-zero mask and use XOR instead of addition, AND instead of multiplication

Eliminating branches

- So, what do we do with this piece of code?

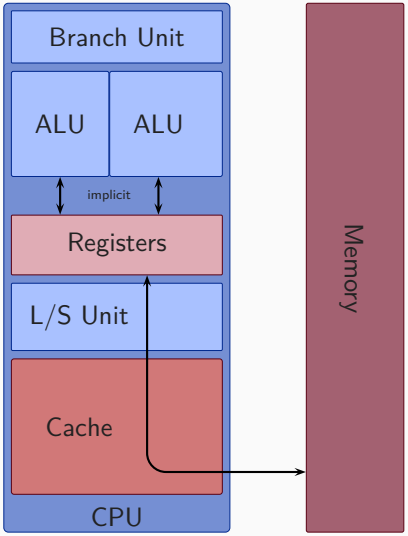
  if s **then**
  $r \leftarrow A$
  else
  $r \leftarrow B$
  end if

- Replace by

$$r \leftarrow sA + (1 - s)B$$

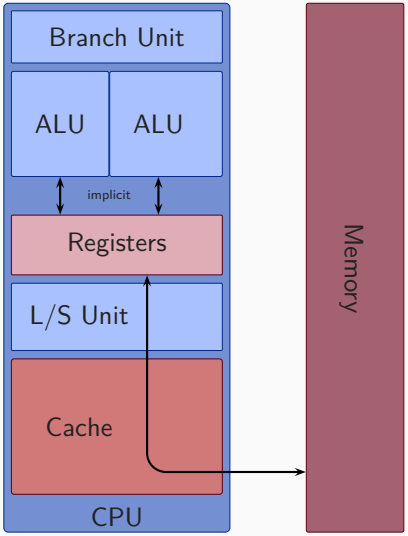
- Can expand s to all-one/all-zero mask and use XOR instead of addition, AND instead of multiplication
- For very fast A and B this can even be faster

Cached memory access



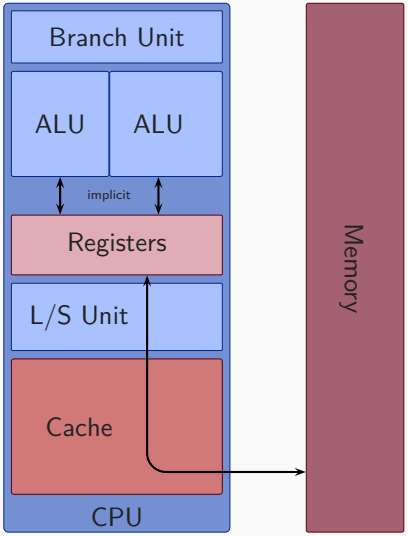
- Memory access goes through a **cache**
- Small but fast transparent memory for frequently used data

Cached memory access



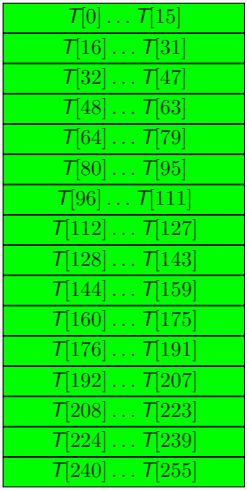
- Memory access goes through a **cache**
- Small but fast transparent memory for frequently used data
- A load from memory places data also in the cache
- Data remains in cache until it's replaced by other data

Cached memory access



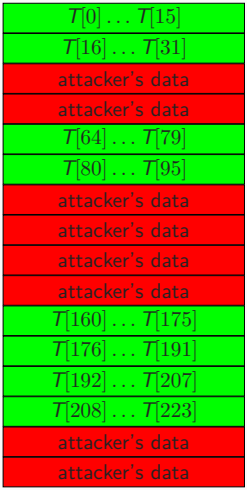
- Memory access goes through a **cache**
- Small but fast transparent memory for frequently used data
- A load from memory places data also in the cache
- Data remains in cache until it's replaced by other data
- Loading data is fast if data is in the cache (**cache hit**)
- Loading data is slow if data is not in the cache (**cache miss**)

Timing leakage part II



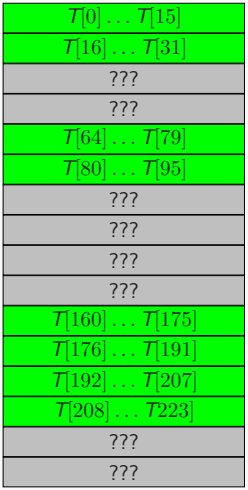
- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache

Timing leakage part II



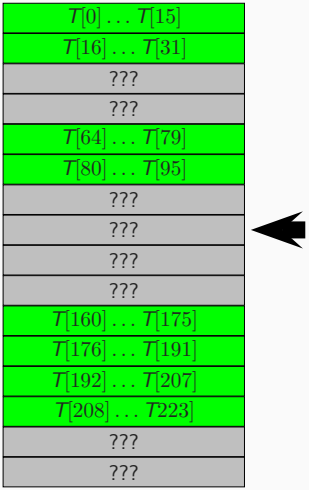
- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache
- The attacker's program replaces some cache lines

Timing leakage part II



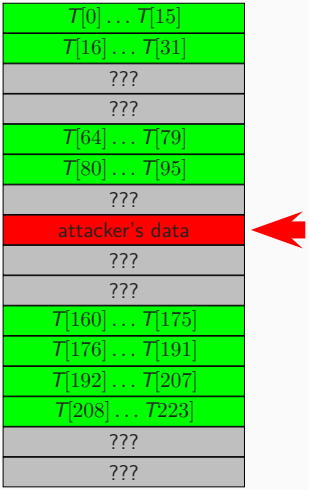
- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache
- The attacker's program replaces some cache lines
- Crypto continues, loads from table again

Timing leakage part II



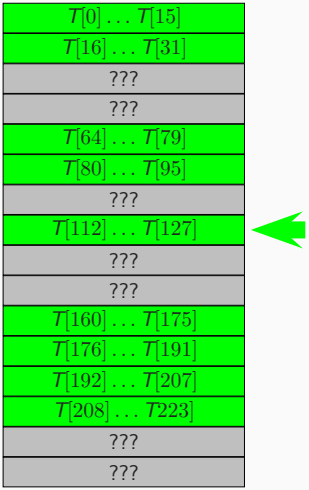
- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache
- The attacker's program replaces some cache lines
- Crypto continues, loads from table again
- Attacker loads his data:

Timing leakage part II



- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache
- The attacker's program replaces some cache lines
- Crypto continues, loads from table again
- Attacker loads his data:
  - Fast: cache hit (crypto did not just load from this line)

Timing leakage part II



- Consider lookup table of 32-bit integers
- *Cache lines* have 64 bytes
- Crypto and the attacker's program run on the same CPU
- Tables are in cache
- The attacker's program replaces some cache lines
- Crypto continues, loads from table again
- Attacker loads his data:
  - Fast: cache hit (crypto did not just load from this line)
  - Slow: cache miss (crypto just loaded from this line)

Some comments on cache-timing

- This is only the *most basic* cache-timing attack

Some comments on cache-timing

- This is only the *most basic* cache-timing attack
- Non-secret cache lines are not enough for security
- Load/Store addresses influence timing in many different ways
- **Do not access memory at secret-data-dependent addresses**

Some comments on cache-timing

- This is only the *most basic* cache-timing attack
- Non-secret cache lines are not enough for security
- Load/Store addresses influence timing in many different ways
- **Do not access memory at secret-data-dependent addresses**
- Timing attacks are practical:
  Osvik, Tromer, Shamir, 2006: 65 ms to steal a 256-bit AES key used for Linux hard-disk encryption

Some comments on cache-timing

- This is only the *most basic* cache-timing attack
- Non-secret cache lines are not enough for security
- Load/Store addresses influence timing in many different ways
- **Do not access memory at secret-data-dependent addresses**
- Timing attacks are practical:
  Osvik, Tromer, Shamir, 2006: 65 ms to steal a 256-bit AES key used for Linux hard-disk encryption
- *Remote* timing attacks are practical:
  Brumley, Tuveri, 2011: A few minutes to steal ECDSA signing key from OpenSSL implementation

Eliminating lookups

- Want to load item at (secret) position p from table of size n

Eliminating lookups

- Want to load item at (secret) position p from table of size n
- Load all items, use arithmetic to pick the right one:

for i from 0 to $n-1$ **do**
    $d \leftarrow T[i]$
    if $p = i$ **then**
        $r \leftarrow d$
    end if
end for

Eliminating lookups

- Want to load item at (secret) position p from table of size n
- Load all items, use arithmetic to pick the right one:

for i from 0 to $n-1$ **do**
  $d \leftarrow T[i]$
  if $p = i$ **then**
    $r \leftarrow d$
  end if
end for

- Problem 1: if-statements are not constant time (see before)

Eliminating lookups

- Want to load item at (secret) position p from table of size n
- Load all items, use arithmetic to pick the right one:

for i from 0 to $n-1$ **do**
  $d \leftarrow T[i]$
  if $p = i$ **then**
    $r \leftarrow d$
  end if
end for

- Problem 1: if-statements are not constant time (see before)
- Problem 2: Comparisons are not constant time, replace by, e.g.:

```
static unsigned long long eq(uint32_t a, uint32_t b)
{
  unsigned long long t = a ^ b;
  t = (-t) >> 63;
  return 1-t;
}
```

Is that all? (Timing leakage part III)

Lesson so far

- Avoid all data flow from secrets to branch conditions and memory addresses
- This can *always* be done as long as input size is public (or at least upper bounded)
- Cost highly depends on the algorithm

Is that all? (Timing leakage part III)

Lesson so far

- Avoid all data flow from secrets to branch conditions and memory addresses
- This can *always* be done as long as input size is public (or at least upper bounded)
- Cost highly depends on the algorithm

Is that all? (Timing leakage part III)

Lesson so far

- Avoid all data flow from secrets to branch conditions and memory addresses
- This can *always* be done as long as input size is public (or at least upper bounded)
- Cost highly depends on the algorithm

"In order for a function to be constant time, the branches taken and memory addresses accessed must be independent of any secret inputs. (That's assuming that the fundamental processor instructions are constant time, but that's true for all sane CPUs.)"
—Langley, Apr. 2010

Is that all? (Timing leakage part III)

Lesson so far

- Avoid all data flow from secrets to branch conditions and memory addresses
- This can *always* be done as long as input size is public (or at least upper bounded)
- Cost highly depends on the algorithm

"In order for a function to be constant time, the branches taken and memory addresses accessed must be independent of any secret inputs. (That's assuming that the fundamental processor instructions are constant time, but that's true for all sane CPUs.)"
—Langley, Apr. 2010

"So the argument to the `DIV` instruction was smaller and `DIV`, on Intel, takes a variable amount of time depending on its arguments!"
—Langley, Feb. 2013

Dangerous arithmetic (examples)

- `DIV`, `IDIV`, `FDIV` on pretty much all Intel/AMD CPUs
- Various math instructions on Intel/AMD CPUs (`FSIN`, `FCOS`...)

Dangerous arithmetic (examples)

- `DIV`, `IDIV`, `FDIV` on pretty much all Intel/AMD CPUs
- Various math instructions on Intel/AMD CPUs (`FSIN`, `FCOS`...)
- `MUL`, `MULHW`, `MULHWU` on many PowerPC CPUs
- `UMULL`, `SMULL`, `UMLAL`, and `SMLAL` on ARM Cortex-M3.

Dangerous arithmetic (examples)

- `DIV`, `IDIV`, `FDIV` on pretty much all Intel/AMD CPUs
- Various math instructions on Intel/AMD CPUs (`FSIN`, `FCOS`...)
- `MUL`, `MULHW`, `MULHWU` on many PowerPC CPUs
- `UMULL`, `SMULL`, `UMLAL`, and `SMLAL` on ARM Cortex-M3.

Solution

- Avoid these instructions
- Make sure that inputs to the instructions don't leak timing information

Is “constant-time” enough?

- **Local** attacker may see much more than just timing (see Thursday lectures):
  - Power consumption
  - Electromagnetic radiation
  - Acoustic emissions
  - ...

Is “constant-time” enough?

- **Local** attacker may see much more than just timing (see Thursday lectures):
  - Power consumption
  - Electromagnetic radiation
  - Acoustic emissions
  - ...
- Sometimes constant-time also too weak against **remote** attackers (see Friday lecture):
  - Spectre and Meltdown
  - Hertzbleed
  - ...

Is “constant-time” enough?

- **Local** attacker may see much more than just timing (see Thursday lectures):
  - Power consumption
  - Electromagnetic radiation
  - Acoustic emissions
  - ...
- Sometimes constant-time also too weak against **remote** attackers (see Friday lecture):
  - Spectre and Meltdown
  - Hertzbleed
  - ...
- Constant-time remains important **base-line defense**

Summary

- Crypto software must avoid timing leaks
- Mostly two rules:
  1. **Never branch depending on secret data**
  2. **Never access memory at secret locations**
- Additionally: avoid variable-time arithmetic instructions

Summary

- Crypto software must avoid timing leaks
- Mostly two rules:
  1. **Never branch depending on secret data**
  2. **Never access memory at secret locations**
- Additionally: avoid variable-time arithmetic instructions
- This is **necessary base-line defense** for essentially all crypto software
- Does not protect against physical side-channel attacks
- Helps, but does not protect against advanced microarchitectural attacks