Comp 590-184: Hardware Security and Side-Channels

Lecture 7: Cache Side-Channel Defenses

February 3, 2026
Andrew Kwong

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Slides adapted from Mengjia Yan (shd.mit.edu)

Outline

- How to mitigate side-channel attacks
- Non-interference property
- Constant-time programming

Attack Examples

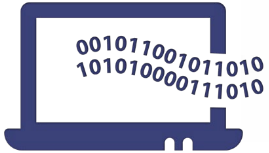
Example #1: termination time vulnerability

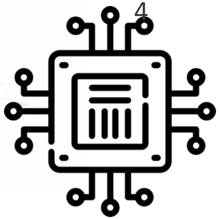
```
def check_password(input):

  size = len(password);

  for i in range(0,size):
    if (input [i] == password[i]):
      return ("error");


  return (“success”);
```

Example #2: RSA cache vulnerability

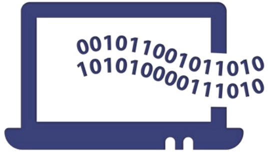
```
for i = n-1 to 0 do
  r = sqr(r)
  r  = r mod n
   if e_i == 1 then
    r = mul(r, b)
    r  = r mod n
  end
end
```

Who to blame? Who should fix the problem?





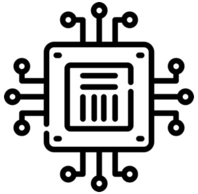
Break SW and HW Contract



Software

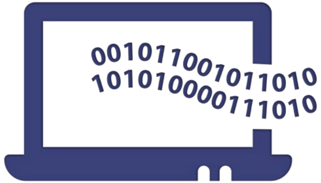
Instruction Set Architecture (ISA)

The contract for functional correctness.



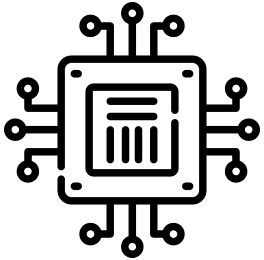
Hardware

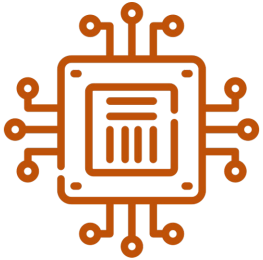
Software Developer's Problem

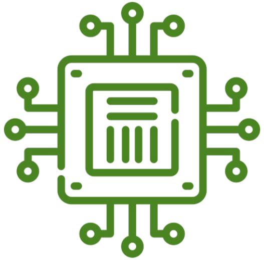


Software developers:

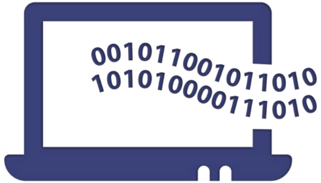
- Need to write software for devices with unknown design details.
- How can I know whether the program is secure running on different devices?

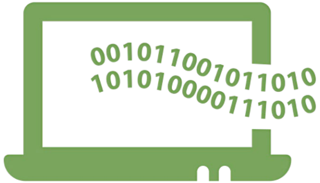


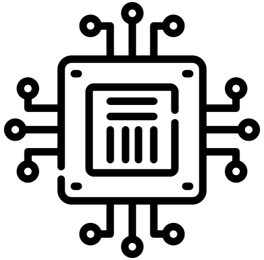




Hardware Designer's Problem







Hardware designer:

- Need to design processors for arbitrary programs.
- How to describe what kind of programs can run securely on my device?

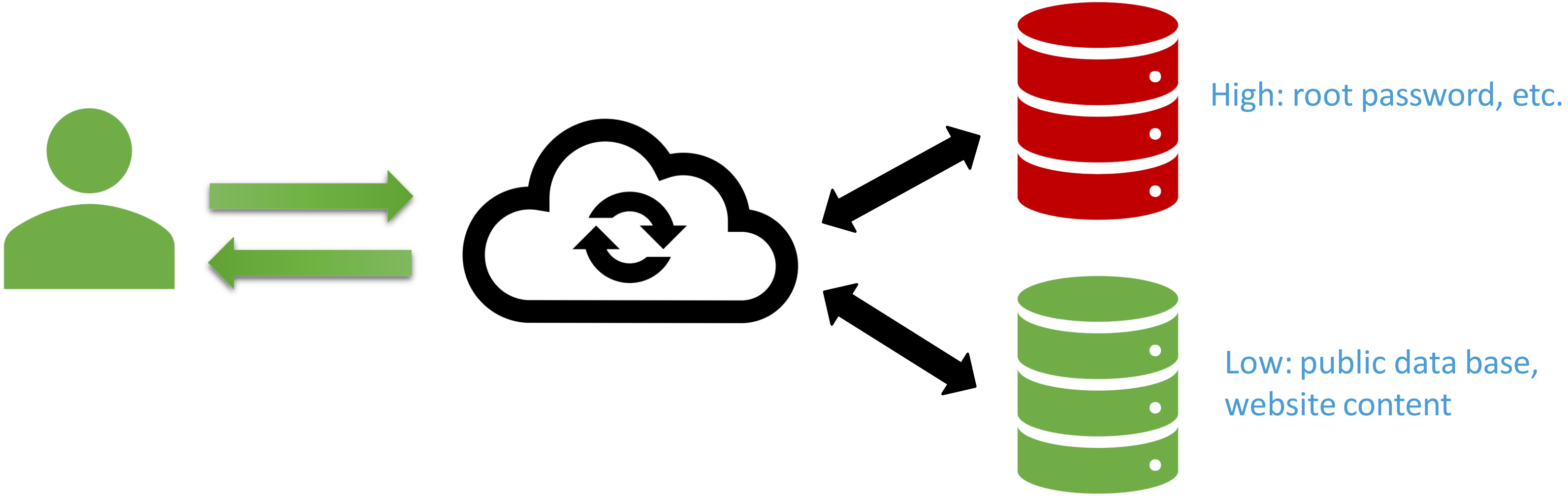
Example: Termination Time Vulnerability

- How to fix it?

```
def check_password(input):

  size = len(password);

  for i in range(0,size):
    if (input [i] != password[i]):
      return ("error");


  return (“success”);
```

Make the computation time **independent** from the secret (password)

Non-Interference Example



- Intuitively: not affecting
- Any sequence of **low** inputs will produce the same **low** outputs, regardless of what the **high** level inputs are.

Non-Interference: A Formal Definition

- The definition of noninterference for a deterministic program P

$$\forall\, M1, M2, P$$

$$M1_L = M2_L \quad \wedge \quad (M1, P) \rightarrow^* M1' \quad \wedge \quad (M2, P) \rightarrow^* M2'$$

$$\Longrightarrow \quad M1_L' = M2_L'$$

Non-Interference for Side Channels

- The definition of noninterference for a deterministic program P

$$\forall\ M1, M2, P$$

$$M1_L = M2_L \quad \wedge \quad (M1, P) \xrightarrow{O1}{}^{*} M1' \quad \wedge \quad (M2, P) \xrightarrow{O2}{}^{*} M2'$$

$$\Longrightarrow \quad O1 = O2$$

What should be included in the observation trace?

Understanding the Property

$$\forall\, M1, M2, P$$

$$M1_L = M2_L \;\wedge\; (M1, P) \xrightarrow{O1}{}^{*} M1' \;\wedge\; (M2, P) \xrightarrow{O2}{}^{*} M2'$$

$$\Longrightarrow \quad O1 = O2$$

```
def check_password(input):

  size = len(password);

  for i in range(0,size):
    if (input [i] == password[i]):
      return ("error");


  return ("success");
```

Consider input as part of M

- What is M_L ?
- What is M_H ?
- What is O ?

Constant-Time Programming

Think about whether the statement below is a reasonable definition that follows non-interference

- For any inputs, secret values, and machines, a program always takes the same amount of time to execute.
- For any inputs and secret values, a program always takes the same amount of time when executing on the same machine.
- For any secret values, a program always takes the same amount of time for the same input when executing on the same machine.
- For any secret values, a program always takes the same amount of time for the same input when executing on the same machine, and this holds for arbitrary inputs.

Data-oblivious/Constant-time programming

- How do we deal with conditional branches/jumps?
- How do we deal with memory accesses?
- How do we deal with arithmetic operations: division, shift/rotation, multiplication?

Your Code

Compiler

Hardware

For details on real-world constant-time crypto, check this out: https://www.bearssl.org/constanttime.html

```python
def check_password(input):

  size = len(password);

  for i in range(0,size):
    if (input [i] != password[i]):
      return ("error");


  return ("success");
```

→

```python
def check_password(input):

  size = len(password);
  dontmatch = false;
  for i in range(0,size):
    if (input [i] != password[i]):
      dontmatch = true;

return dontmatch;
```

```
def check_password(input):

  size = len(password);
  dontmatch = false;
  for i in range(0,size):
    if (input [i] != password[i]):
      dontmatch = true;

return dontmatch;
```

```
def check_password(input):

  size = len(password);
  dontmatch = false;
  for i in range(0,size):
    dontmatch |= (input [i] != password[i])

  return dontmatch;
```

Real-world Crypto Code

from libsodium cryptographic library:

```
for (i = 0; i < n; i++)
    d |= x[i] ^ y[i];
return (1 & ((d - 1) >> 8)) - 1;
```

Compare two buffers x and y, if match, return 0, otherwise, return -1.

Examples from Cauligi et al. FaCT: A DSL for Timing-Sensitive Computation. PLDI'19

Another Example

From the “donna” Curve25519 implementation

```
for (i = 0; i < 5; ++i)
{
    if (swap) {
        tmp = a[i];
        a[i] = b[i];
        b[i] = tmp;
    }
}
```

➡

```
for (i = 0; i < 5; ++i) {
    const limb x = swap & (a[i] ^ b[i]);
    a[i] ^= x;
    b[i] ^= x;
}
```

swap is a mask, either 0 or 0xFFFFFFFF

Eliminate Secret-dependent Branches

- An instruction: `cmov_`
  - Check the state of one or more of the status flags in the EFLAGS register (`cmovz`: moves when ZF=1)
  - Perform a move operation if the flags are in a specified state
  - Otherwise, a move is not performed and execution continues with the instruction following the `cmov` instruction

Conditional Branches

```
if (secret) x = e

x = (-secret & e) | (secret - 1) & x


test secret, secret // set ZF=1 if zero
cmovz r2, r1 // r2 for x, r1 for e
```

What do we assume about the hardware here?

More Conditional Branches

```
if (secret)
  res = f1();
else
  res = f2();
```

```
r1 ← f1();
r2 ← f2();
mov r3, r1
test secret, secret
cmovz r3, r2
// res in r3
```

Potential problems:

- What if we have nested branches?
- What if when `secret==0`, `f1` is not executable, e.g., causing page fault or divide by zero?
- What if `f1` or `f2` needs to write to memory, perform IO, make system calls?
- **Hardware assumption:** what if cmovz will be executed as soon as the flag is known (e.g., speculative execution)?

Memory Accesses

```
a = buffer[secret]
```

⬇

```
for (i=0; i<size; i++)
{
    tmp = buffer[i];
    xor secret, i
    cmovz a, tmp
}
```

- Performance overhead.
- Techniques such as ORAM can reduce the overhead when the buffer is large

An Optimization

- We can reduce the redundant accesses by only accessing one byte in each cache line.

```
for (i=0; i<size; i++)
{
    tmp = buffer[i];
    xor secret, i
    cmovz a, tmp
}
```

```
offset = secret % 64;
for (i=0; i<size; i+=64)
{
    index = i+offset;
    tmp = buffer[index];
    xor secret, index
    cmovz a, tmp
}
```

OpenSSL Patches Against Timing Channel

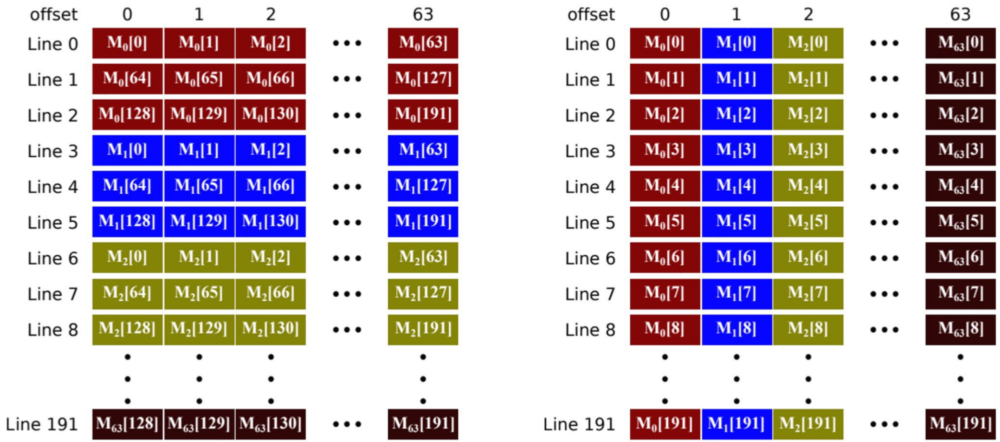


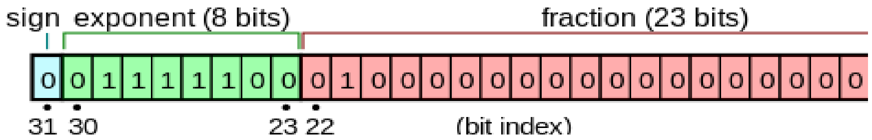
Fig. 1. Conventional (left) vs. scatter-gather (right) memory layout.

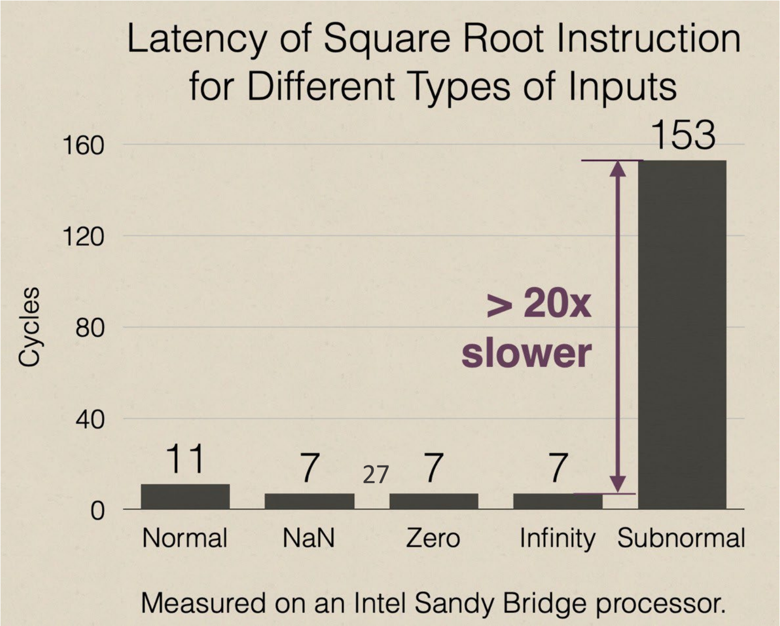
CacheBleed, an attack leaks SSL keys via **L1 cache bank conflict**.

Yarom et al. CacheBleed: A Timing Attack on OpenSSL Constant Time RSA. https://faculty.cc.gatech.edu/~genkin/cachebleed/index.html

Arithmetic Operations

Subnormal floating point numbers

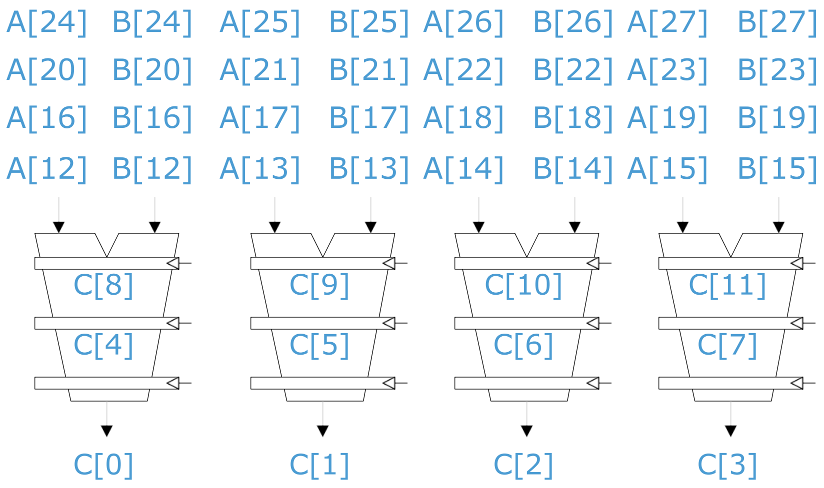




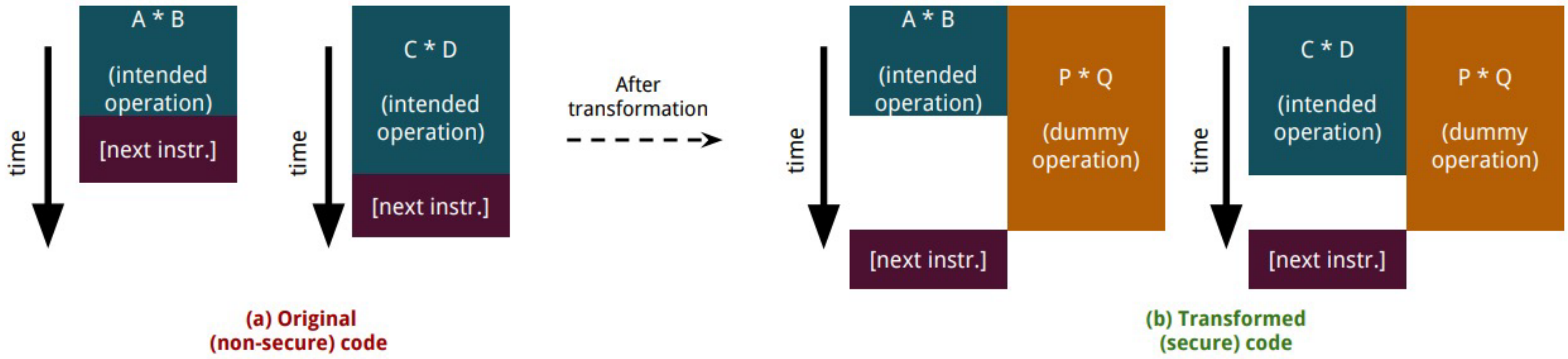
SIMD Hardware Implementation

```
# Vector code
LI VLR, 64
LV V1, R1
LV V2, R2

SV V3, R3
```

Example: 4 pipelined functional units



The Problem and A Solution



Rane et al. Secure, Precise, and Fast Floating-Point Operations on x86 Processors. USENIX'16

Constant-time ISA

- Some efforts:
  - ARM Data Independent Timing (DIT)
  - Intel Data Operand Independent Timing (DOIT)

ARM DIT: https://developer.arm.com/documentation/ddi0601/2020-12/AArch64-Registers/DIT--Data-Independent-Timing
Intel DOIT: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/data-operand-independent-timing-isa-guidance.html

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL