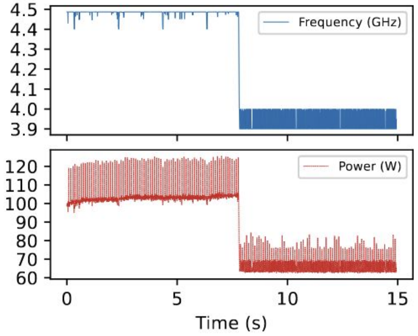
HertzBleed

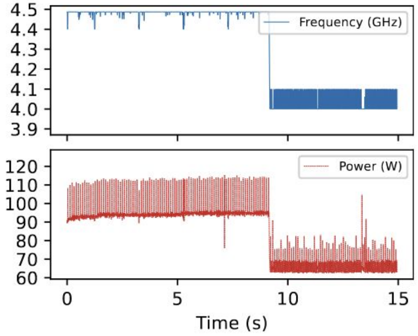
Andrew Jiang, Leo Wang

Key observation

- Dynamic Voltage and Frequency Scaling (DVFS) changes CPU frequency according to the power consumption (and other factor).
- When the workload is high, increase the CPU frequency for efficiency until the CPU is too hot.
- Power side channel → Timing side channel



(a) Run of the `int32-float` test



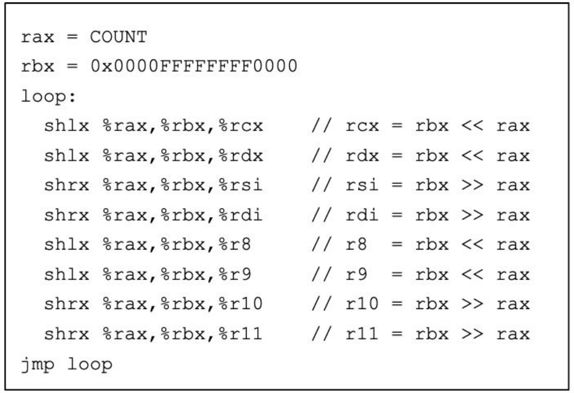
(b) Run of the `int32` test

Importance

- Power side channel attacks are powerful, but it’s also not hard to block attackers’ access to those information.
- The constant-time programming does not take CPU frequency into account.

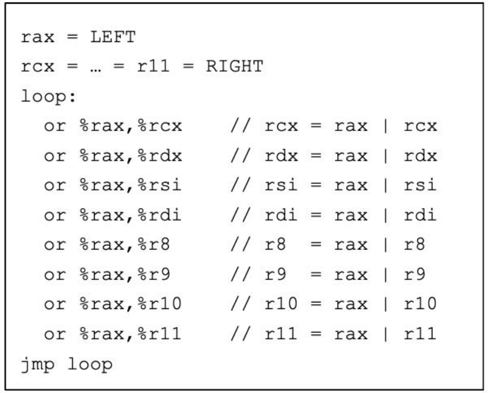
Power side channel

- Two common models: Hamming Weight, Hamming distance
- Consider the instruction $a \leftarrow a\ op\ b$
  - Hamming weight: number of 1s in a
  - Hamming distance: number of bits differed between old a and new a



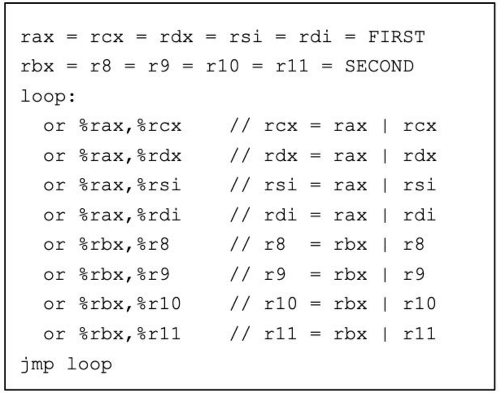
```
rax = COUNT
rbx = 0x0000FFFFFFFF0000
loop:
  shlx %rax,%rbx,%rcx    // rcx = rbx << rax
  shlx %rax,%rbx,%rdx    // rdx = rbx << rax
  shrx %rax,%rbx,%rsi    // rsi = rbx >> rax
  shrx %rax,%rbx,%rdi    // rdi = rbx >> rax
  shlx %rax,%rbx,%r8     // r8  = rbx << rax
  shlx %rax,%rbx,%r9     // r9  = rbx << rax
  shrx %rax,%rbx,%r10    // r10 = rbx >> rax
  shrx %rax,%rbx,%r11    // r11 = rbx >> rax
jmp loop
```

(a) Sender for our HD experiments.



```
rax = LEFT
rcx = … = r11 = RIGHT
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
  or %rax,%r8     // r8  = rax | r8
  or %rax,%r9     // r9  = rax | r9
  or %rax,%r10    // r10 = rax | r10
  or %rax,%r11    // r11 = rax | r11
jmp loop
```

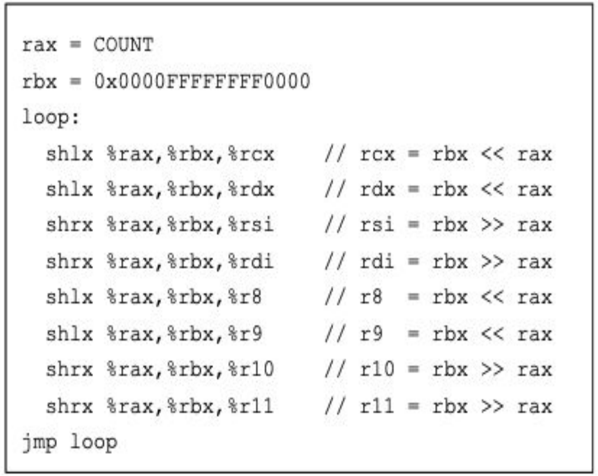
(b) Sender for our HW experiments.



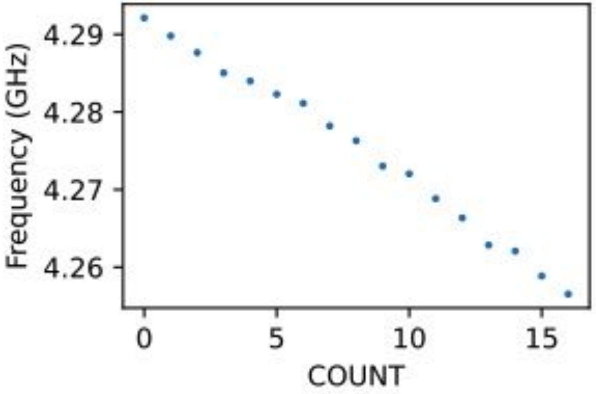
```
rax = rcx = rdx = rsi = rdi = FIRST
rbx = r8 = r9 = r10 = r11 = SECOND
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
  or %rbx,%r8     // r8  = rbx | r8
  or %rbx,%r9     // r9  = rbx | r9
  or %rbx,%r10    // r10 = rbx | r10
  or %rbx,%r11    // r11 = rbx | r11
jmp loop
```

(c) Sender for our HW+HD experiments.

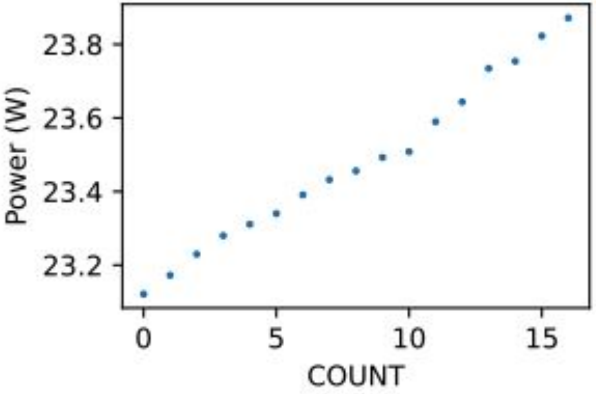
HD effect



```
rax = COUNT
rbx = 0x0000FFFFFFFF0000
loop:
  shlx %rax,%rbx,%rcx    // rcx = rbx << rax
  shlx %rax,%rbx,%rdx    // rdx = rbx << rax
  shrx %rax,%rbx,%rsi    // rsi = rbx >> rax
  shrx %rax,%rbx,%rdi    // rdi = rbx >> rax
  shlx %rax,%rbx,%r8     // r8  = rbx << rax
  shlx %rax,%rbx,%r9     // r9  = rbx << rax
  shrx %rax,%rbx,%r10    // r10 = rbx >> rax
  shrx %rax,%rbx,%r11    // r11 = rbx >> rax
jmp loop
```



(a) Frequency vs COUNT

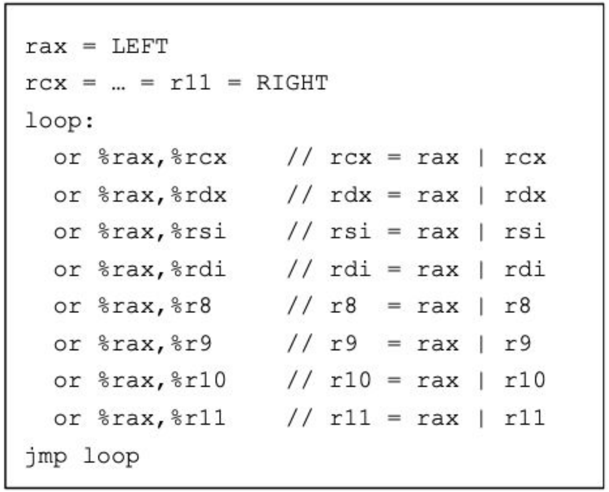


(b) Power vs COUNT

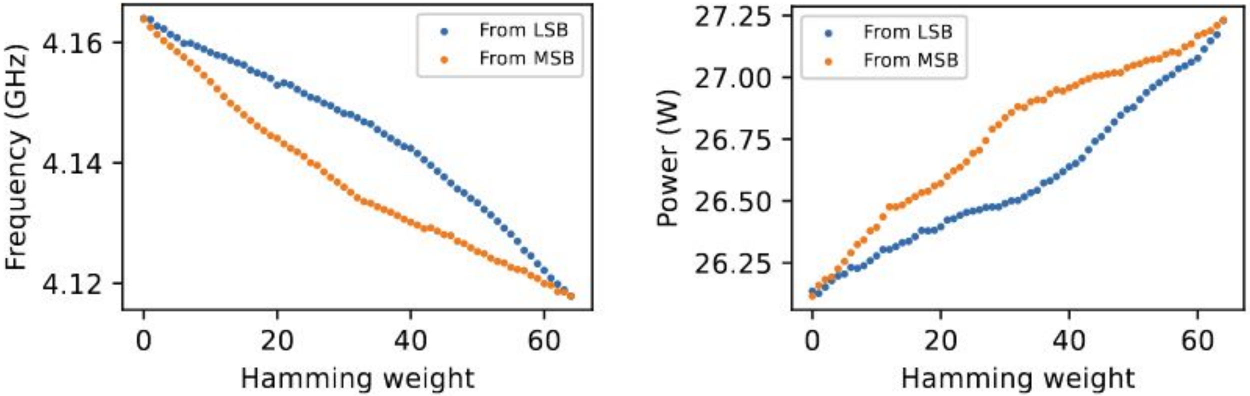
Input:

- rax = 0 <= COUNT <= 16
- rcx = 0x0000FFFFFFFF0000

HW effect



```
rax = LEFT
rcx = … = r11 = RIGHT
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
  or %rax,%r8     // r8  = rax | r8
  or %rax,%r9     // r9  = rax | r9
  or %rax,%r10    // r10 = rax | r10
  or %rax,%r11    // r11 = rax | r11
jmp loop
```



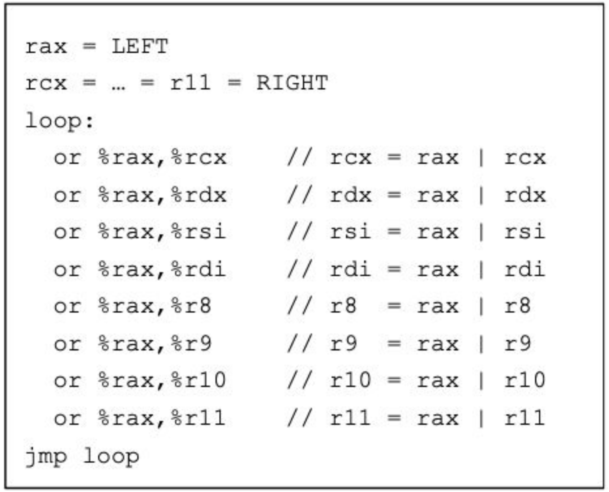
(a) Frequency vs HW

(b) Power vs HW

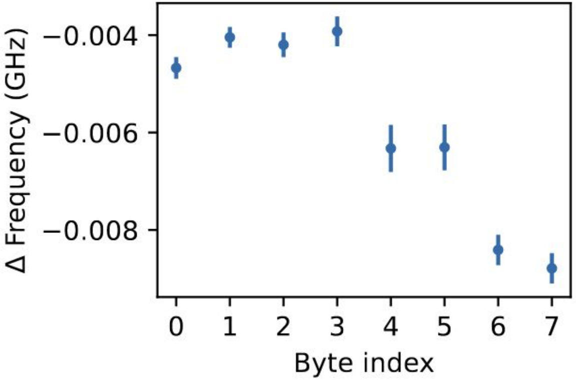
Input:

- LSB: LEFT = RIGHT = 0b0000011111
- MSB: LEFT=RIGHT = 0b1111100000

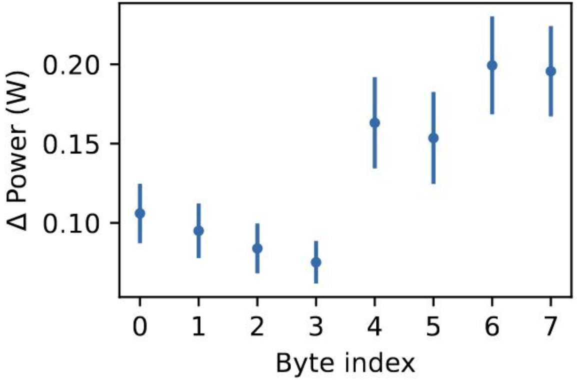
HW effect



```
rax = LEFT
rcx = … = r11 = RIGHT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
  or %rax,%r8      // r8  = rax | r8
  or %rax,%r9      // r9  = rax | r9
  or %rax,%r10     // r10 = rax | r10
  or %rax,%r11     // r11 = rax | r11
jmp loop
```



(a) Effect of 0xFF to frequency



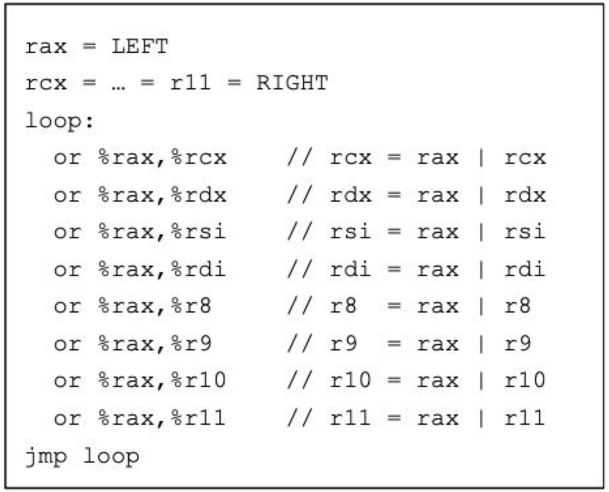
(b) Effect of 0xFF to power

Note: this effect (0.12W/byte)
is small compared to HW
(1.11W/byte)

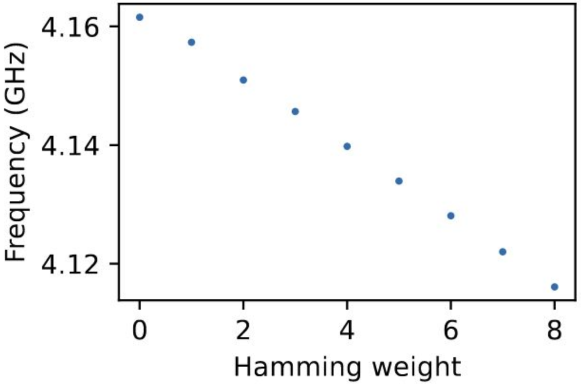
Input:

- LEFT = RIGHT = 0x????00????
- LEFT = RIGHT = 0x????FF????

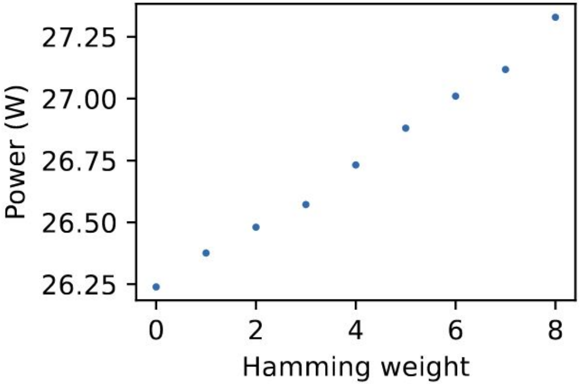
HW effect



```
rax = LEFT
rcx = … = r11 = RIGHT
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
  or %rax,%r8     // r8  = rax | r8
  or %rax,%r9     // r9  = rax | r9
  or %rax,%r10    // r10 = rax | r10
  or %rax,%r11    // r11 = rax | r11
jmp loop
```



(a) Frequency vs HW

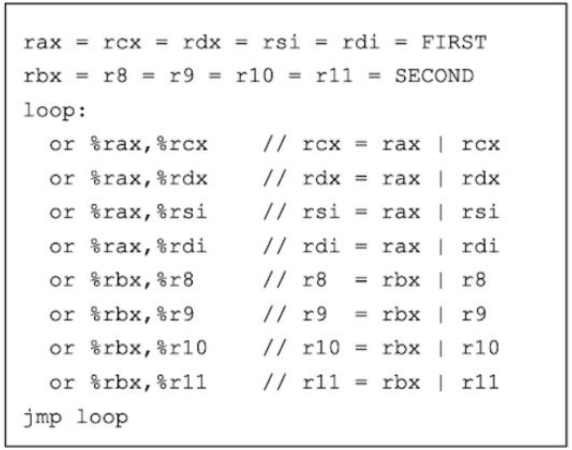


(b) Power vs HW

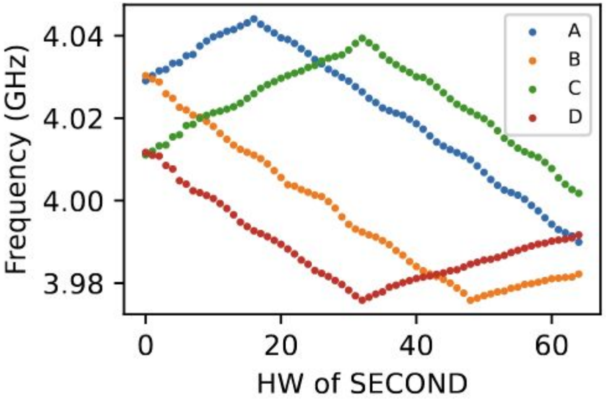
Input:

- LEFT = RIGHT = 0x01010101
- LEFT = RIGHT = 0x03030303 …

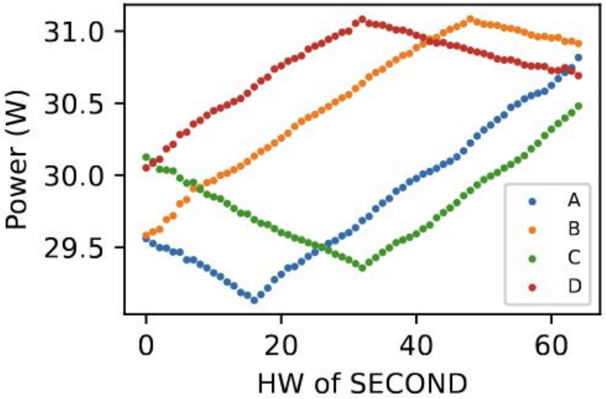
Additivity of HW and HD



```
rax = rcx = rdx = rsi = rdi = FIRST
rbx = r8 = r9 = r10 = r11 = SECOND
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
  or %rbx,%r8      // r8  = rbx | r8
  or %rbx,%r9      // r9  = rbx | r9
  or %rbx,%r10     // r10 = rbx | r10
  or %rbx,%r11     // r11 = rbx | r11
jmp loop
```



(a) Frequency vs HW



(b) Power vs HW

Input:

- A: FIRST = 0x000000000000FFFF
- B: FIRST = 0xFFFF000000000000
- C: FIRST = 0x00000000FFFFFFFF
- D: FIRST = 0xFFFFFFFF00000000

Summary

In our model, three things can change the power consumption and the CPU frequency:

- Hamming Distance
- Hamming Weight
- Position of 1 (not that significant)

Attacks

- Chosen Ciphertext Attack on SIKE
  - recover server's secret key through triggering and observing anomalous 0s
  - Attacker provides malicious P, Q
  - Server calculates P + [m]Q using Montgomery ladder
  - Server performs a few more steps, and then sanity check
  - In some case, P + [m]Q will results in (0, 0) (because of attacker's invalid input) which lowers the power consumption.
- Kernel ASLR break
  - Using the power consumption difference when prefetching mapped/unmapped address

Mitigations

- Disable DVFS
  - Turbo Boost, SpeedStep or Hardware Controlled Performance States(HWP) from BIOS
- Modify Cryptosystem
  - masking/blinding to limit individual operad leakage

Discussion

- What other cryptographic algorithms are at risk to this kind of attack besides SIKE?
- What is SIKE? Can the encryption scheme be explained again? How do we recover the secret key?
- Why do bits demonstrate a position dependency on power? For example, why does the MSB use more power than the LSB?
- How did the researchers select an algorithm for the attack model? Is there a general “repertoire” of common algorithms to expose data among secure hardware researchers/engineers or does this step require a lot of background research?
- Does performing the attack remotely vs. on a shared device affect how the SIKE decapsulation process works / how successful or efficient it is?
- Just like there is constant-time programming, would it be possible to implement programs that use a fixed amount of power so they'd be able to defend against HertzBleed?
- Can the same methodology be applied to other cryptographic algorithms in addition to SIKE?
- Besides simply turning off DVFS / Turbo Boost / etc, are there any other possible workload-independent defenses to this attack? Would it be possible to restrict access to the current CPU frequency (i.e. via the scaling_cur_freq interface from the cpufreq driver), or would that cause other problems? What about injecting noise into the CPU’s steady-state frequency, so that the exact P-state is unknown?
- How exactly were their experiments able to test HD and HW effects independently? I didn't really get why/how they’re experimental setups testing only one of these effects while leaving the other effect constant.
- Section 4 of this paper describes the scope of the CPU frequency leakage model to be limited to instructions involving the ALU. Could this approach be applied to / what experiments might be conducted to find if there is a frequency side channel for instructions involving main memory?
- I don't really understand the part where they group operations into 2 or 4. They claim that it's related to "port" and I don't know what that is.
- How do the masking/blinding techniques in the discussion of possible mitigations for the leakage in ciphers work?