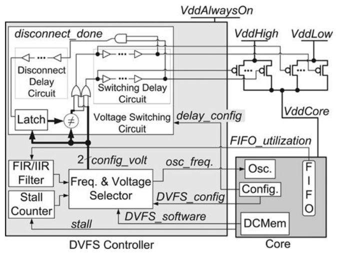
Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86

Selena Qiao

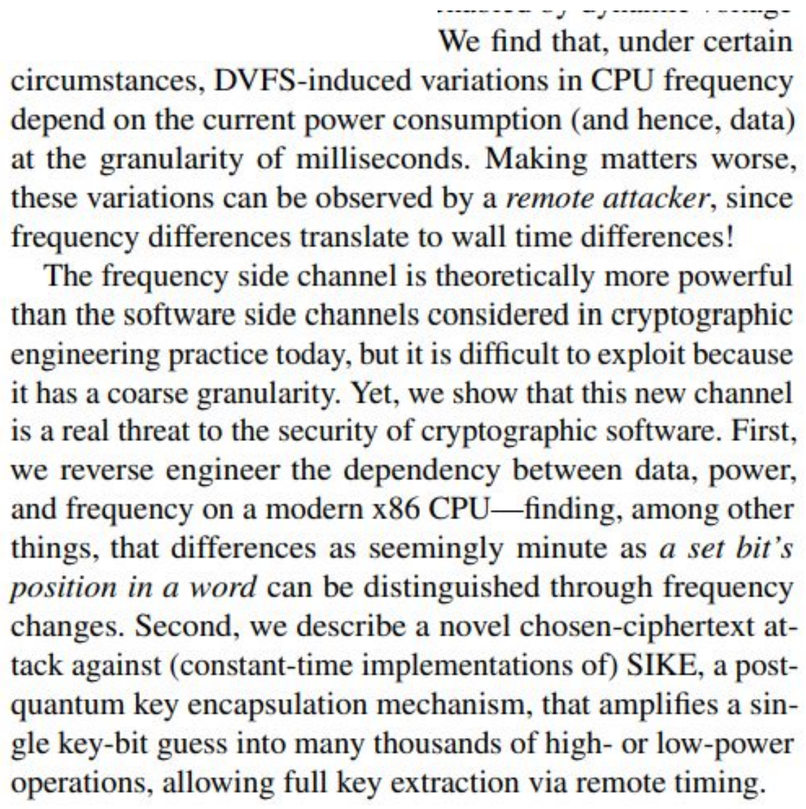
Background

Power side-channel attacks exploit data-dependent variations in a CPU's power consumption to leak secrets

Dynamic voltage and frequency scaling (DVFS) management is a technique used to switch the CPU core frequency based on load requirement. It consists of dynamically adjusting CPU frequency to reduce power consumption (during low CPU loads) and to ensure that the system stays below power and thermal limits (during high CPU loads). Under certain circumstances, DVFS-induced CPU frequency adjustments depend on the current power consumption at the granularity of milliseconds.



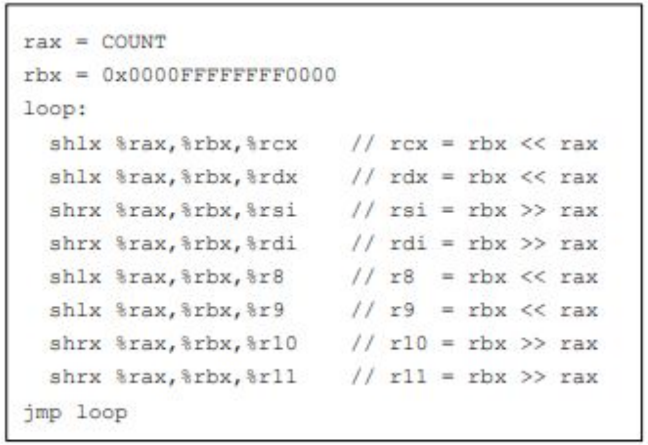
Key Contribution



We find that, under certain circumstances, DVFS-induced variations in CPU frequency depend on the current power consumption (and hence, data) at the granularity of milliseconds. Making matters worse, these variations can be observed by a *remote attacker*, since frequency differences translate to wall time differences!

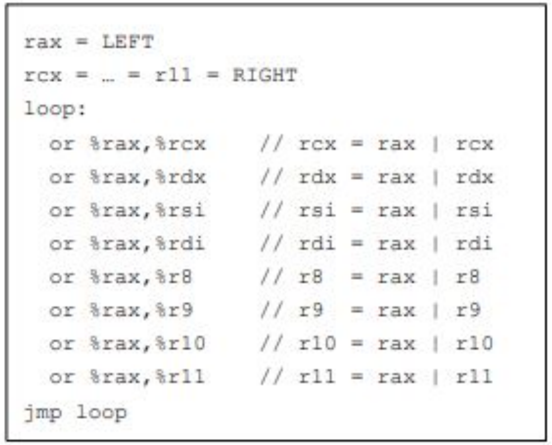
The frequency side channel is theoretically more powerful than the software side channels considered in cryptographic engineering practice today, but it is difficult to exploit because it has a coarse granularity. Yet, we show that this new channel is a real threat to the security of cryptographic software. First, we reverse engineer the dependency between data, power, and frequency on a modern x86 CPU—finding, among other things, that differences as seemingly minute as *a set bit's position in a word* can be distinguished through frequency changes. Second, we describe a novel chosen-ciphertext attack against (constant-time implementations of) SIKE, a post-quantum key encapsulation mechanism, that amplifies a single key-bit guess into many thousands of high- or low-power operations, allowing full key extraction via remote timing.

Key Contribution



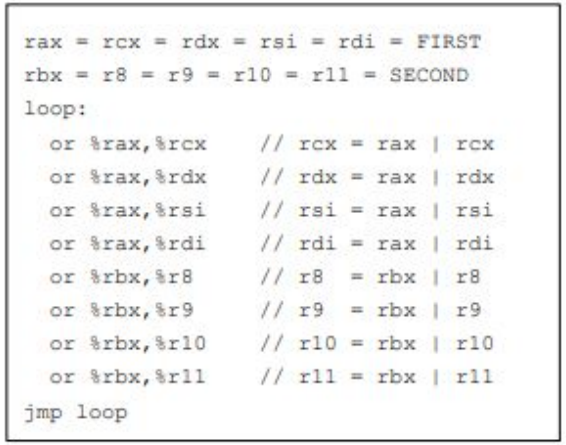
```
rax = COUNT
rbx = 0x0000FFFFFFFF0000
loop:
  shlx %rax,%rbx,%rcx    // rcx = rbx << rax
  shlx %rax,%rbx,%rdx    // rdx = rbx << rax
  shrx %rax,%rbx,%rsi    // rsi = rbx >> rax
  shrx %rax,%rbx,%rdi    // rdi = rbx >> rax
  shlx %rax,%rbx,%r8     // r8  = rbx << rax
  shlx %rax,%rbx,%r9     // r9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

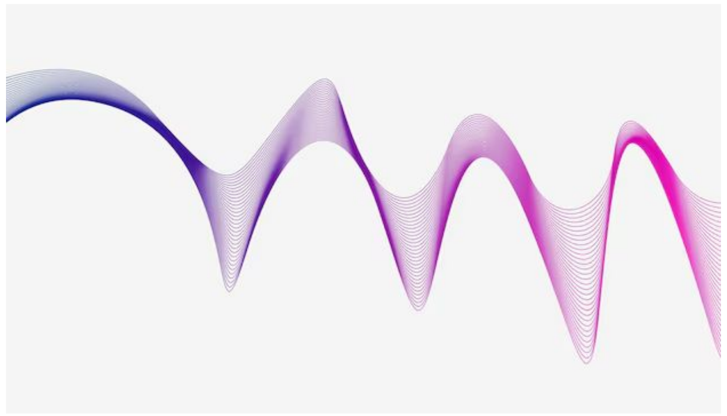
(b) Sender for our HW experiments.

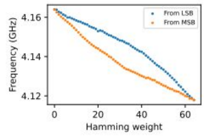


```
rax = rcx = rdx = rsi = rdi = FIRST
rbx = r8 = r9 = r10 = r11 = SECOND
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
  or %rbx,%r8     // r8  = rbx | r8
  or %rbx,%r9     // r9  = rbx | r9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

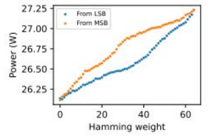
(c) Sender for our HW+HD experiments.

Power and Frequency

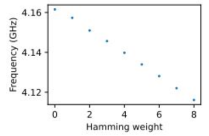




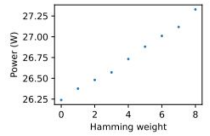
(a) Frequency vs HW



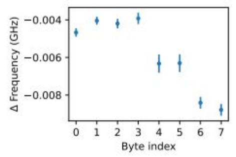
(b) Power vs HW



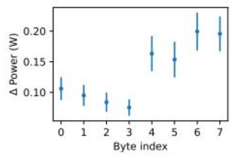
(a) Frequency vs HW



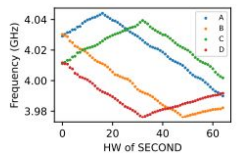
(b) Power vs HW



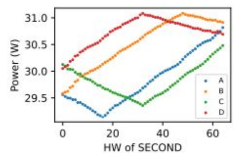
(a) Effect of 0xFF to frequency



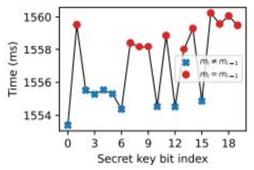
(b) Effect of 0xFF to power



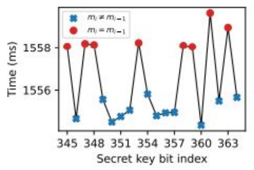
(a) Frequency vs HW



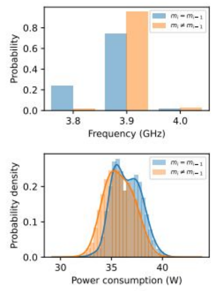
(b) Power vs HW



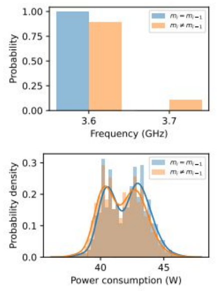
(a) PQCrypto-SIDH first 20 bits



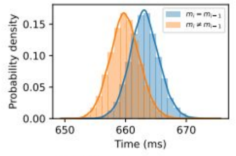
(b) PQCrypto-SIDH last 20 bits



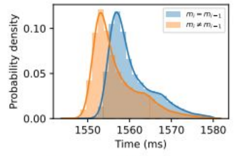
(a) CIRCL data



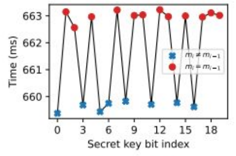
(b) PQCrypto-SIDH data



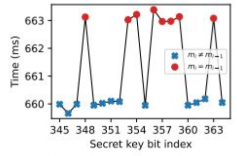
(a) CIRCL histogram



(b) PQCrypto-SIDH histogram



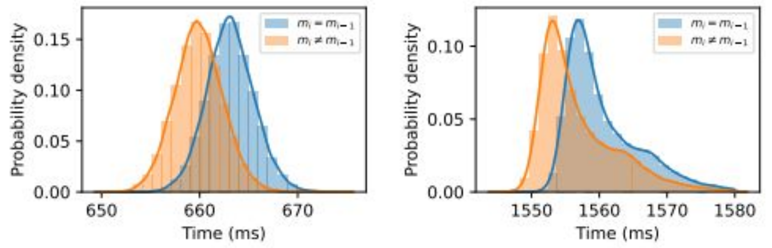
(a) CIRCL first 20 bits



(b) CIRCL last 20 bits

SIKE Attack Setup + Results

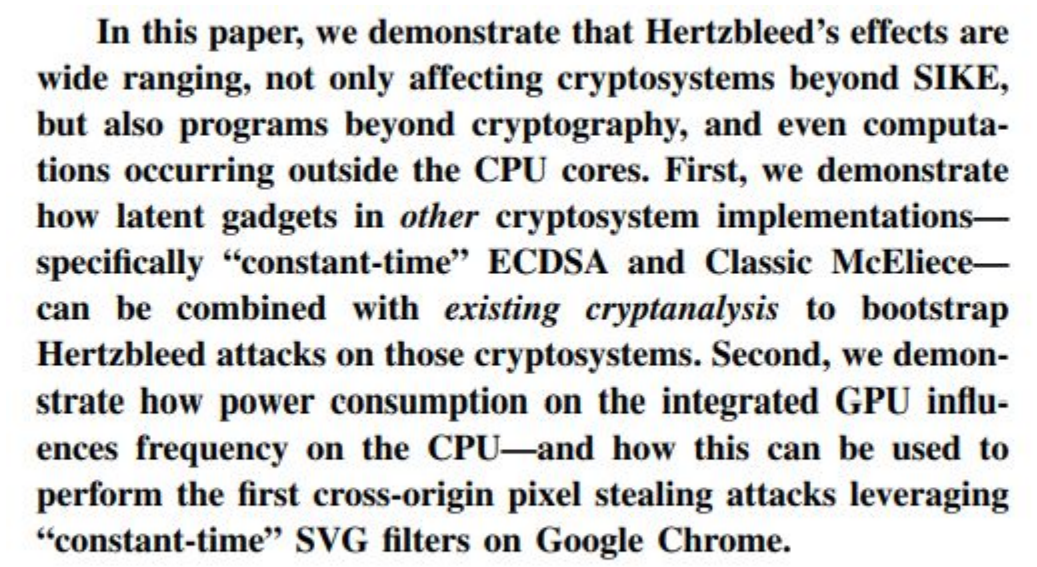
The SIKE timing attack was conducted on an i7-9700 CPU with two server configurations: an HTTP server using Go's `net.http` library for CIRCL and a TCP server using C with `pthread` for PQCrypto-SIDH. The attacker sent 300 concurrent requests for CIRCL and 1000 for PQCrypto-SIDH, all using the same crafted challenge ciphertext. Timings were measured until the last connection completed, with expected anomalies in CPU frequency leading to timing differences: CIRCL showed at most 660.2 ms versus at least 662.5 ms, while PQCrypto-SIDH showed at most 1556 ms versus at least 1558 ms. After repeating the measurements 400 times, excluding outliers, and computing medians, the key was extracted up to bit 364, with the last 14 bits recovered via brute force. Figures illustrate timing distributions for the first and last 20 bits, revealing differences depending on whether the ciphertext triggered an anomalous 0 (`mi ≠ mi-1`) or not (`mi = mi-1`).



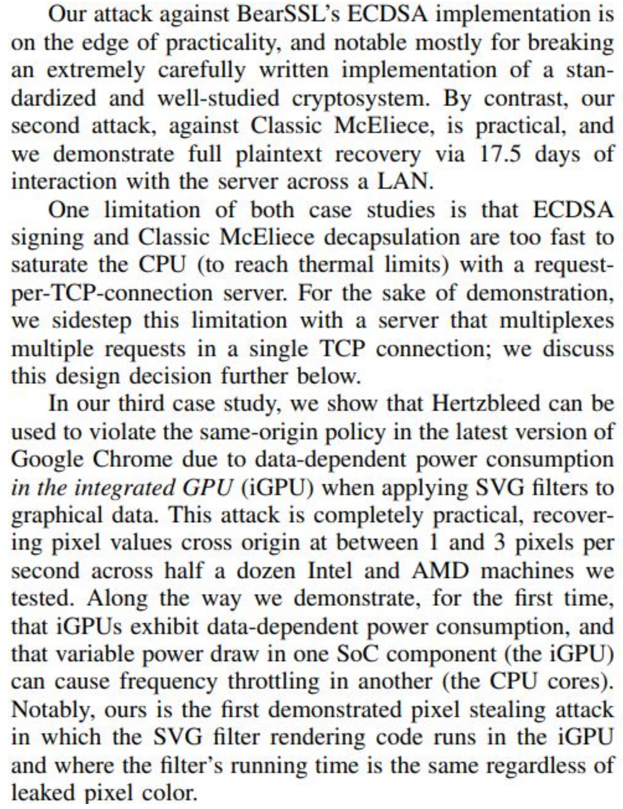
(a) CIRCL histogram (b) PQCrypto-SIDH histogram

Figure 10: Distribution of the timings measured by the attacker during the remote key extraction attack, with the server running on an i7-9700 CPU. The attacker makes 300 (CIRCL) and 1000 (PQCrypto-SIDH) connections (all with the same challenge ciphertext, constructed as per Section 5.3.2) and measures the time until the last connection completes. We group the execution time (filtered) of each key bit extraction based on whether it should have triggered an anomalous 0 in the Montgomery ladder (i.e., whether $m_i = 1 - m_{i-1}$ or not).

Effects (from follow-up paper)



In this paper, we demonstrate that Hertzbleed's effects are wide ranging, not only affecting cryptosystems beyond SIKE, but also programs beyond cryptography, and even computations occurring outside the CPU cores. First, we demonstrate how latent gadgets in *other* cryptosystem implementations—specifically "constant-time" ECDSA and Classic McEliece—can be combined with *existing cryptanalysis* to bootstrap Hertzbleed attacks on those cryptosystems. Second, we demonstrate how power consumption on the integrated GPU influences frequency on the CPU—and how this can be used to perform the first cross-origin pixel stealing attacks leveraging "constant-time" SVG filters on Google Chrome.



Our attack against BearSSL's ECDSA implementation is on the edge of practicality, and notable mostly for breaking an extremely carefully written implementation of a standardized and well-studied cryptosystem. By contrast, our second attack, against Classic McEliece, is practical, and we demonstrate full plaintext recovery via 17.5 days of interaction with the server across a LAN.

One limitation of both case studies is that ECDSA signing and Classic McEliece decapsulation are too fast to saturate the CPU (to reach thermal limits) with a request-per-TCP-connection server. For the sake of demonstration, we sidestep this limitation with a server that multiplexes multiple requests in a single TCP connection; we discuss this design decision further below.

In our third case study, we show that Hertzbleed can be used to violate the same-origin policy in the latest version of Google Chrome due to data-dependent power consumption *in the integrated GPU* (iGPU) when applying SVG filters to graphical data. This attack is completely practical, recovering pixel values cross origin at between 1 and 3 pixels per second across half a dozen Intel and AMD machines we tested. Along the way we demonstrate, for the first time, that iGPUs exhibit data-dependent power consumption, and that variable power draw in one SoC component (the iGPU) can cause frequency throttling in another (the CPU cores). Notably, ours is the first demonstrated pixel stealing attack in which the SVG filter rendering code runs in the iGPU and where the filter's running time is the same regardless of leaked pixel color.

Critiques

Strengths: new way of doing side-channel attack, has case studies

Weaknesses: none