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

Yingchen Wang, *The University of Texas at Austin, Austin, TX, 78712, USA*

Riccardo Paccagnella and Elizabeth Tang He, *University of Illinois Urbana-Champaign, Urbana, IL, 61801, USA*

Hovav Shacham, *The University of Texas at Austin, Austin, TX, 78712, USA*

Christopher W. Fletcher, *University of Illinois Urbana-Champaign, Urbana, IL, 61801, USA*

David Kohlbrenner, *University of Washington, Seattle, WA, 98195, USA*

Power side-channel attacks exploit data-dependent variations in a CPU's power consumption to leak secrets. In this article, we show that on modern CPUs, power side-channel attacks can be turned into timing attacks that can be mounted without access to any power measurement interface. This discovery exploits how, under certain circumstances, the dynamic frequency scaling of modern x86 CPU depends on the current power consumption (and hence, data). We demonstrate that this "frequency side 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 small 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) supersingular isogeny key encapsulation that allows full key extraction via remote timing.

Power-analysis attacks have been known for decades to be a powerful source of side-channel information leakage.[1] Historically, these attacks were used to leak cryptographic secrets from embedded devices using physical probes. Recently, however, power-analysis attacks have been shown to be exploitable also via software power measurement interfaces.[2] Such interfaces, available on many of today's general-purpose processors, have been abused to fingerprint websites, recover RSA keys, break kernel address space layout randomization, and even recover AES keys.

Fortunately, software-based power-analysis attacks can be mitigated and easily detected by blocking (or restricting) access to power measurement interfaces. Up until today, such a mitigation strategy would effectively reduce the attack surface to physical power analysis, a significantly smaller threat in the context of modern general-purpose x86 processors.

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3274619
Date of publication 10 May 2023; date of current version 29 June 2023.

In this article, we show that, on modern Intel (and AMD) x86 CPUs, power-analysis attacks can be turned into timing attacks—effectively lifting the need for *any* power measurement interface. Our discovery is enabled by the aggressive dynamic voltage and frequency scaling (DVFS) of these CPUs. DVFS is a commonly used technique that 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). We find that, under certain circumstances, DVFS-induced CPU frequency adjustments depend on the current power consumption at the granularity of milliseconds. Therefore, since the power consumption is data dependent, it follows transitively that CPU frequency adjustments are data dependent too.

Making matters worse, we show that data-dependent frequency adjustments can be observed without the need for any special access, even by a *remote* attacker. The reason is that CPU frequency differences directly translate to execution time differences (as 1 Hz = 1 cycle per second). The security implications of this finding are significant. For example, they fundamentally undermine constant-time programming, which has been the bedrock defense against timing attacks since their discovery in 1996.[3] The premise behind constant-time programming is that by writing a program to only use "safe" instructions, whose latency is invariant to the data values, the program's execution time will be data-independent. With the frequency channel, however, timing becomes a function of *data*—even when only safe instructions are used.

Despite its theoretical power, it is not obvious how to construct practical exploits through the frequency side channel. This is because DVFS updates depend on the aggregate power consumption over *millions* of CPU cycles and only reflect coarse-grained program behavior. Yet, we show that the frequency side channel is a real threat to the security of cryptographic software, by 1) reverse engineering a precise *leakage model* for this channel on modern x86 CPUs and 2) showing that some cryptographic primitives admit *amplification* of single key bit guesses into thousands of high- or low-power operations, enough to induce a measurable timing difference.

FINDINGS AND IMPACT OF HERTZBLEED

We start by providing an overview of the key findings and significance of this article. We expand on these findings in the following sections.

Hertzbleed is the First CPU Analog Side Channel That Does Not Require Access to any Explicit Measurement Interface

Analog side-channel attacks analyze analog signals produced by a computer to leak secrets. These attacks are theoretically very powerful, but traditionally required access to physical equipment located in proximity to the victim. Recently, it was shown that analog side-channel attacks can also be performed using software-accessible power monitoring interfaces (e.g., RAPL). However, these attacks can be prevented by restricting software access to explicit power monitoring interfaces.

Hertzbleed is the first CPU analog side-channel attack that does not require access to any explicit measurement interface. Instead, Hertzbleed turns power side-channel attacks into remote timing attacks that can be carried out by simply measuring wall-clock time. We anticipate that this contribution will revive community interest in exploiting and mitigating power side-channel attacks in new contexts.

Differences as Seemingly Minute as a Set Bit's Position in a Word can Contribute to Measurable Differences in Power and Frequency on x86 CPUs

Power side-channel attacks rely on *leakage models* to approximate the dependency between data and power consumption. Two commonly used leakage models are the Hamming distance (HD) and the Hamming weight (HW) models. In the HD model, power consumption depends on the number of $1 \rightarrow 0$ and $0 \rightarrow 1$ bit transitions occurring in the data during a computation. In the HW model, power consumption depends on the number of bits that are 1 in the data being processed.

Our article reveals that power consumption and frequency on modern Intel x86 CPUs depend on both the HW and the HD of data. First, we found that these effects are distinct and additive. Second, perhaps most surprisingly, we found that the HW effect is non uniform. That is, we found that the bit position set in input data can be differentiated via power and frequency. We anticipate that this will aid development of new fine-grained attacks and countermeasures in the future, and that it will be useful towards building better, Intel-specific power leakage emulators.

Frequency Side-Channels Can Leak Keys Via Timing From Cryptography That Correctly Follows Constant-Time Programming Practices

Constant-time programming has been the bedrock defense against timing side-channel attacks since their discovery in 1996. The premise behind constant-time programming is that by writing a program to only use "safe" instructions, whose latency is invariant to the data values being operated on, the program's execution time will not depend on secrets.

Our article shows the first example of a remote timing attack that can leak secrets even from correctly implemented constant-time code. We also show how this—along with novel cryptanalysis that we develop—can be leveraged to perform remote key extraction from real constant-time cryptographic implementations. As a result, Hertzbleed fundamentally undermines the current practices for constant-time programming.

Hertzbleed is a Fundamentally New Type of Timing Attack

While our attack leaks information in the analog domain, it also represents a fundamentally new type of timing attack. Timing attacks are of foundational importance in computer architecture and have been setting the agenda for secure processor research in the past decade. Yet, despite the many timing attacks that have been reported, they all stem from just one of several phenomena. To see the taxonomy, consider the ubiquitous processor performance equation:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}.$$

All timing attacks up until Hertzbleed exploited data-dependent variations in the first two factors of this equation. Hertzbleed is the first timing attack that exploits variations in the third factor. As a result, Hertzbleed is fundamentally different from prior timing attacks. We anticipate that this contribution will spur the development of new software and hardware defenses against timing attacks which can also prevent data-dependent leakage in the third factor of the equation.

Community and Industry Response

The publication of our work has had immediate impact, including the following:

1) Cloudflare and Microsoft deployed patches to their libraries CIRCL and PQCrypto-SIDH.
2) Intel, AMD, and Ampere issued security advisories and assigned entries in the national vulnerability database: CVE-2022-23823, CVE-2022-24436, and CVE-2022-35888.
3) ARM and RedHat updated their public documentation and knowledge base to discuss Hertzbleed's implications.
4) Intel updated their guidance on mitigating timing side channels against cryptographic implementations[a] and released entirely new guidance on mitigating Hertzbleed.[b]
5) Community guidance pages for defending against timing attacks were updated to discuss how to mitigate Hertzbleed.[c]

BACKGROUND AND RELATED WORK

Intel P-States

In Intel processors, DVFS works at the granularity of P-states. P-states correspond to different operating points (voltage–frequency pairs) in 100-MHz frequency increments. The number of P-states varies across different CPU models. Our P-state naming convention follows the one used in Linux, where the lowest P-state corresponds to the lowest supported CPU frequency, and the highest P-state corresponds to the "max turbo" frequency for the processor. We use the term P-state and frequency interchangeably.

P-state management is also related to power management. Each Intel processor has a thermal design point (TDP), indicating the expected power consumption at steady state under a sustained workload. While in the max turbo mode, the processor can exceed its nominal TDP. However, if the CPU hits a certain power and thermal limit while in max turbo mode, the hardware will automatically downclock the frequency to stay at TDP for the duration of the workload.

Power Side-Channel Attacks

Power side-channel attacks against cryptosystems were first publicly discussed by Kocher in 1998.[1] Subsequent work demonstrated power-analysis attacks against several cryptographic algorithms including AES, DES, RSA, and ElGamal.[4] However, all these attacks were targeted against smart cards and required physical proximity to the device. More recently, power side-channel attacks have been applied to more complex devices such as smartphones and PCs. Some of these attacks rely only on software power measurement interfaces, meaning that they do not need physical proximity to the device. While some of these works use the HW and HD leakage models, none of them present a systematic reverse engineering of the dependency between power consumption and data on modern Intel x86 CPUs. Further, all these attacks can be blocked by restricting access to such power measurement interfaces.

CPU FREQUENCY LEAKAGE CHANNEL

We now show that, under certain circumstances, the distribution of a processor's frequencies leaks information about the instructions being executed as well as the data being processed. The reason is that CPU frequency depends on the current CPU power consumption, which is data dependent.

Distinguishing Instructions

First, we set out to understand how running different workloads affects the P-state selection logic of our CPUs. We pick two workloads from the `stress-ng` benchmark suite: `int32float` and `int32`. We run both benchmarks on all cores and starting from an idle

[a]https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html.
[b]https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/frequency-throttling-side-channel-guidance.html.
[c]https://timing.attacks.cr.yp.to/overclocking.html.

machine. We sample the CPU frequency and the (package domain) power consumption every 5 ms during the benchmark's execution, using the experimental setup detailed in our USENIX Security 2022 paper.[5]

Figure 1(a) shows the results for the `int32float` test on our i7-9700 CPU. The frequency starts at 4.5 GHz, the highest P-state available when all cores are active. This frequency is sustained for about 8 s, during which the power consumption is allowed to exceed the TDP. Then, the CPU drops to a lower P-state, bringing the power consumption down to TDP (65 W). From there onwards, the CPU remains in *steady state* and power stays around the TDP level for the duration of the workload. In this example, at steady state the frequency oscillates between the frequencies of 3.9 GHz and 4.0 GHz.

Figure 1(b) shows the results for the `int32` stress test. Here too, the frequency starts at 4.5 GHz and later drops to a lower P-state. However, compared to Figure 1(a), 1) the drop occurs later, after 10 s, and 2) the P-states used after the drop are higher, corresponding to 4.0 GHz and 4.1 GHz. This is because the power consumption of the `int32` test is lower. As a consequence, the processor can sustain the highest available P-state for longer and can also use higher P-states in steady state without exceeding the TDP.

The key takeaway is that both 1) the time that a processor can spend at the maximum available P-state and 2) the distribution of P-states at steady state depend on the CPU power consumption. Since the CPU power consumption depends on the workload, by the transitive property it follows that P-states depend on the workload too. This implies that dynamic scaling of P-states leaks information about the current workload running on the processor.

Distinguishing Data

We saw that P-state information leaks information about the instructions being executed (i.e., the workload). We now explore if the frequency leakage channel can leak information about the data being processed by instructions. Our question is motivated by the fact that processor power consumption is known to be data dependent. It is thus natural to ask: do data-dependent differences in power consumption show in the distribution of P-states?

To answer this question, we monitor the CPU frequency while executing the same instructions and only changing the content of the input registers. For example, we use the `shlx` instruction to continuously shift left the bits of a source register and write the result into different destination registers in a loop, while only varying the content of the source register. We run this experiment on all cores and compare the distribution of the P-states in steady state. Figure 2(a) shows the results when we set the content of the source register to have 16, 32, or 48 ones. In all cases the P-state oscillated between 4.3 GHz and 4.4 GHz. However, the larger the Hamming weight, the more the frequency stayed at the lower P-state. We also saw a data-dependent difference in terms of *when* the frequency drops to steady state if we start from idle [cf. Figure 2(b)]. The larger the Hamming weight, the quicker the frequency drops to steady state. This is because, as we show in the next section, processing data with larger Hamming weights consumes more power than processing data with smaller Hamming weights.

We get similar results with other instructions, too. For example, we observed data-dependent effects when running `or`, `xor`, `and`, `imul`, `add`, `sub`, as well as when computing on data loaded from memory.

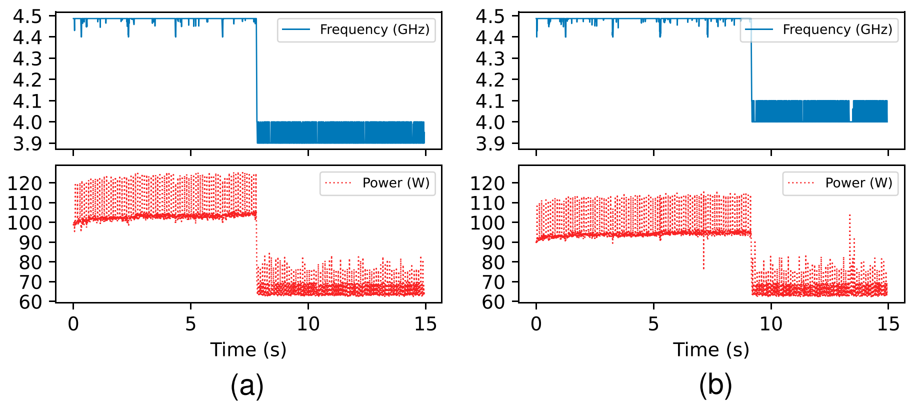


FIGURE 1. Example of distinguishing workloads using frequency traces on our i7-9700 CPU. The lighter workload (`int32`) allows for longer runtimes at higher frequencies than the heavier workload (`int32-float`). (a) `int32-float` test. (b) `int32` test.

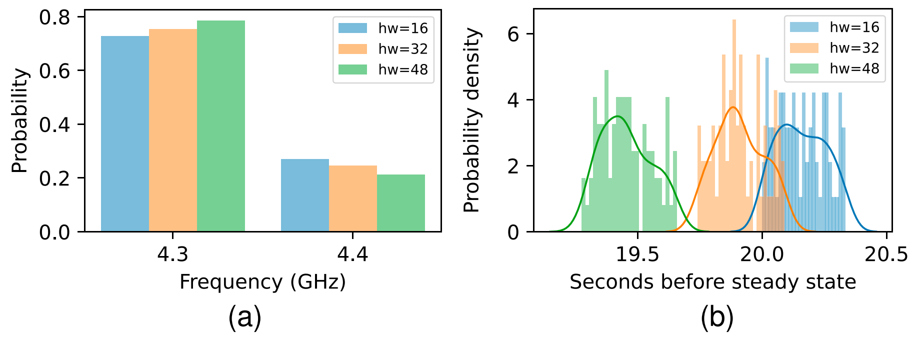


FIGURE 2. Distinguishing data (in the source register to a `shlx` instruction) using frequency traces on our i7-9700 CPU. (a) is over 30,000 samples. (b) is over 100 traces. (a) Steady-state frequency. (b) Time before steady state.

The key takeaway of the above results is that dynamic scaling of P-states leaks information about the data being processed. In the following sections, we use the distribution of P-states at steady state as our leakage channel.

FINE-GRAINED DATA LEAKAGE MODELS

We reverse engineer the dependency between power consumption/frequency and data on the arithmetic logic unit (ALU) of modern Intel CPUs and construct a detailed leakage model, to inform both mitigation strategies and future attacks. Along the way, we make unexpected discoveries about ALU power consumption. For example, we find that *different bit positions* in data values processed contribute *differently* to power consumption and CPU frequency.

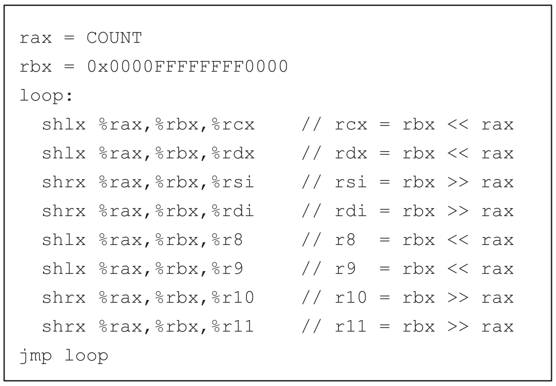
Methodology

In each experiment, we run a fixed set of ALU instructions (the *sender*) in a loop on all cores, while varying the input contents. We carefully design our senders to target specific behaviors and minimize side effects, as detailed in our USENIX Security 2022 paper.[5] We run each sender in two setups. In the first setup, we use the default system configuration, warm up the machine until it enters steady state, and monitor the frequency. In the second setup, we fix the CPU frequency to base frequency and monitor the (core domain) power consumption. We sample power/frequency every 1 ms, collect 30,000 data points for each experiment, and use their mean for our analyses.

HD Effect

Recall that the idea behind the HD model is that power consumption depends on the number of $1 \to 0$ and $0 \to 1$ bit transitions between consecutive data values being processed. To study the dependency between HD and power consumption/frequency, we design our sender to use interleaved `shlx` and `shrx` instructions, as shown in Figure 3. These instructions shift the bits of the second source register to the left or right by a COUNT value stored in the first source register. The result is written to a separate destination register. Since on our CPU both `shlx` and `shrx` can execute on either port 0 or port 6, we interleave them in groups of two. We fix the content of the second source register to `0x0000ffffffff0000`, corresponding to 16 zeros, followed by 32 ones, followed by 16 zeros. We then shift this register left and right by COUNT (with $0 \leq \mathrm{COUNT} \leq 16$).

By construction, the HD in the ALU output between a `shlx` and a `shrx` is $4 \times \mathrm{COUNT}$. For example, when $\mathrm{COUNT} = 8$, the output of each `shlx` is `0x00ffffffff000000`, and the output of each `shrx` is `0x000000ffffffff00`, translating to 4×8 bit transitions in the ALU output. Yet, the ALU input remains



```
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
```

FIGURE 3. Sender for our HD experiment to reverse engineer the dependency between data HD and power consumption/frequency on our CPUs.

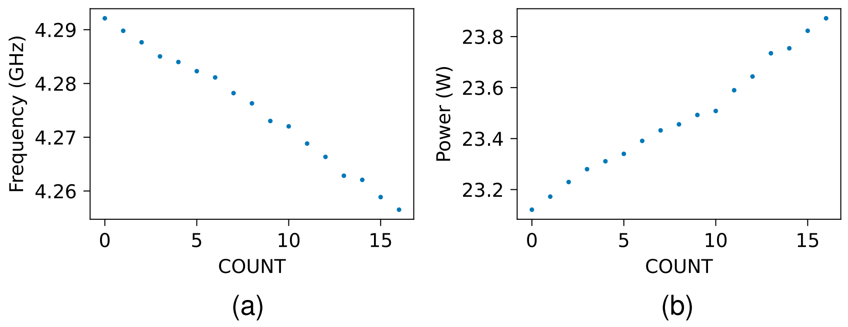


FIGURE 4. Effect of increasing COUNT in Figure 3's sender on our i7-9700 CPU. Higher COUNT values cause higher HDs in the ALU output. As the HD increases, the mean power consumption grows and the mean steady-state frequency drops. (a) Frequency versus COUNT. (b) Power versus COUNT.

unchanged and the number of 1s in the source and the destination registers is fixed.

Figure 4 shows the results when we vary the COUNT value. We see that the power consumption grows and the frequency drops when COUNT grows, confirming that the number of bit transitions directly affects power consumption and frequency. In in our USENIX Security 2022 paper,[5] we corroborate this observation with an additional experiment where transitions occur in the ALU input.

HW Effect

Recall that the idea behind the HW model is that power consumption depends on the number of 1s in the data being processed. To study the dependency between HW and power consumption/frequency, we design a sender that uses `or` instructions. These instructions perform a bitwise inclusive `or` operation between the source and the destination registers and store the result in the destination register. We always use the same input and output registers for all the `or` instructions in the loop. We fix the content of the source register to LEFT, and set the initial content of the output register to RIGHT.

As the `or` instructions use the same inputs and produce the same outputs, the number of bit transitions occurring on the ALU input and output during the execution of the above sender is zero. Hence, we can test different HW in the source registers without introducing HD effects.

Varying the Number of 1s

Our first experiment tests if the number of leading (e.g., `0xffffffff00000000`) or trailing (e.g., `0x00000000ffffffff`) 1s in matching $\mathrm{LEFT} = \mathrm{RIGHT}$ inputs affect power and frequency. We find that a larger number of 1s (a higher HW) directly correlates with higher power consumption and lower steady-state frequencies. More subtly, we find that the most significant 32 bits have a higher power consumption than the lower 32 bits.

Nonuniformity of the HW Effect

To analyze the impact of the *position* of 1s within the data, we run variants of our previous experiment where we set each byte independently to either `0x00` or `0xff`. We first test each byte in isolation: 7 bytes are fixed, and one is varied while power/frequency are measured. For each byte, we repeat this test with all the 2^7 combinations of the other 7 bytes. We compute the average and standard deviation of the deltas for each byte and show the result in Figure 5. From this, we find that the HW effect is nonuniform at byte granularity. At a high level, the four most significant bytes have a stronger HW effect than the four least significant bytes, and bytes closer to the 32nd bit have a weaker HW effect than bytes farther from the 32nd bit.

To sum up, the HW effect may be approximated as the inner product of two vectors. The first vector is the number of 1s per byte, and the second vector is the nonuniform power consumption/frequency "cost" of 1s in that byte (based on the deltas of Figure 5). In our USENIX Security 2022 paper,[5] we detail additional experiments in support of this model, show that the HW effect occurs on each operand independently, and find that the nonuniform "costs" per byte of the HW effect can be different across CPU models.

REMOTE TIMING ATTACK ON CONSTANT-TIME SIKE

Finally, we give a case study of how the frequency side channel can leak secrets from side-channel hardened cryptographic implementations. Our target is supersingular isogeny key encapsulation (SIKE), a post-quantum key encapsulation mechanism. For the full details of our cryptanalysis and our key recovery algorithm, we refer to our USENIX Security 2022 paper.[5]

Attack Model

The attacker's goal is to recover the secret key used by the server to decapsulate ciphertexts (i.e., to decrypt them and obtain ephemeral per-session keys). The attacker can send the server ciphertexts of their choice and measure how long the server takes to process them.

Attack Concept

We show that the attacker can construct ciphertexts that induce a large difference in the power drawn in decapsulation depending on the value of a server secret key bit. Specifically, if the attacker knows bits $m_0, m_1, \ldots, m_{i-1}$ of the secret key, they can construct a ciphertext such that:

- If $m_i \neq m_{i-1}$, then the ciphertext will cause the server's internal state to take on an anomalous all-0 value at step $i+1$ of decapsulation, and to get stuck, consuming and producing all-0 values in *all* subsequent steps.
- If, however, $m_i = m_{i-1}$, then the anomalous all-0 value is not produced at any step of decapsulation.

Because of the HD and HW effects described previously, decapsulating a ciphertext that induces an all-0 state will consume less power than otherwise. (Indeed, De Feo et al.[6] independently found the same property of SIKE in service of a traditional power side-channel attack.)

Under the frequency side channel, less power consumed means decapsulation takes a shorter wall time. This enables an adaptive chosen-ciphertext timing attack against SIKE implementations. Having extracted the $m_i = m_{i-1}$ least-significant bits of m, the adversary repeatedly queries the server with ciphertexts that should cause decapsulation to get stuck at step $(i+1)$. If the server responds faster than a baseline (established through profiling), the adversary concludes that bit m_i is the opposite of bit m_{i-1}; otherwise bit m_i is the same. The attacker then proceeds to the next bit.

SIKE Remote Key Recovery

The timing differences induced are large enough that the attack can be mounted across the network. Recovering the 378-bit SIKE-751 private key from Cloudflare's CIRCL library takes 36 h; from Microsoft's PQCrypto-SIDH library takes 89 h. Ensuring that the execution time does not depend on secrets was a design goal of both libraries.

In our experiments, the attacker and the target server are on separate machines on a LAN, with a 688-µs average round-trip time. The server has an Intel i7-9700 CPU.

For CIRCL, our server accepts a decapsulation request over HTTP and spawns a goroutine to handle it; for PQCrypto-SIDH, our server accepts a decapsulation request over TCP and spawns a pthread to handle it. The thread lets the client know when decapsulation completes.

The attacker sends a batch of requests (300 for CIRCL, 1000 for PQCrypto-SIDH) with a challenge ciphertext constructed as above and measures the time it takes for the batch to complete. We repeat the measurement 400 times, exclude outliers, compute the median of the remaining values, and compare to cutoffs determined in a separate profiling phase.

In Figure 6(a) and (b), we show the timing distribution of 300-connection batches for CIRCL and 1000-connection batches for PQCrypto-SIDH, grouped according to whether the challenge ciphertext of that

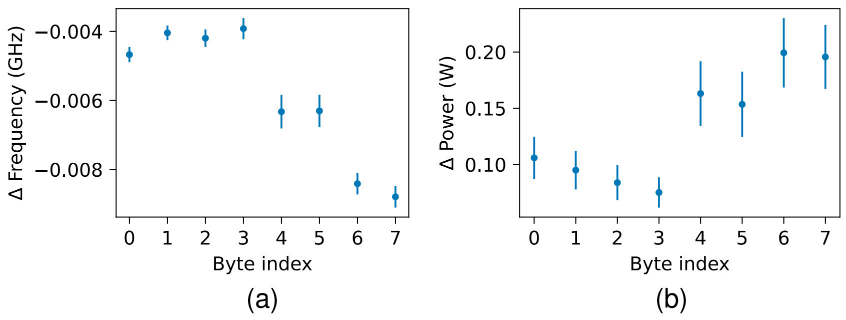


FIGURE 5. Effect of setting single bytes to 0xff in the data values being processed by our HW sender on our i7-9700 CPU. The effect varies depending on the position of 1s within the inputs. HW differences in the MSBs have the strongest effect; HW differences in the bits right below 32 have the weakest effect. (a) 0xFF effect to frequency. (b) 0xFF effect to power.

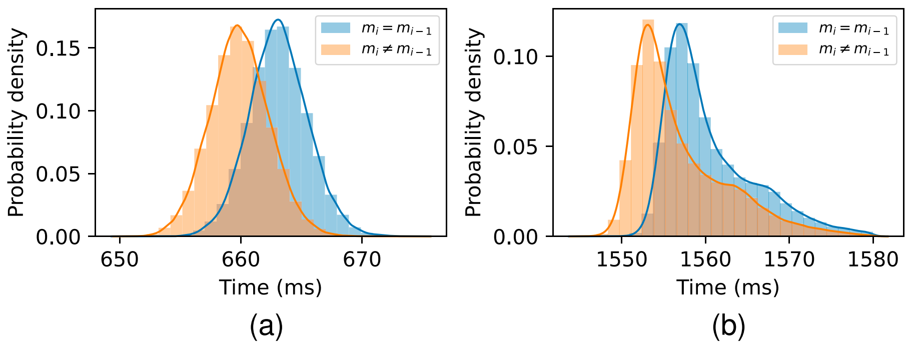


FIGURE 6. Distribution of the timings measured by the attacker during the remote key extraction attack, with the server running on an Intel i7-9700 CPU. The attacker makes 300 (CIRCL) and 1000 (PQCrypto-SIDH) connections all with the same challenge ciphertext and measures the time until the last connection completes. We group the execution time of each key bit extraction based on whether it should have triggered an anomalous all-0 decapsulation state (i.e., whether $m_i = 1 - m_{i-1}$). (a) CIRCL. (b) PQCrypto-SIDH.

run triggered an anomalous all-0 decapsulation state ($m_i \neq m_{i-1}$) or not ($m_i = m_{i-1}$).

SIKE Mitigations

Our attack was mitigated by Cloudflare and Microsoft by adding ciphertext consistency checks proposed by De Feo et al.;[6] these checks slowed decapsulation by about 5% in CIRCL and about 11% in PQCrypto-SIDH.

Castryck and Decru subsequently showed that SIKE is completely insecure.[7] It is no longer being considered for standardization or deployment.

MITIGATING THE FREQUENCY SIDE CHANNEL

There is extensive literature on hardware and software techniques for mitigating power side-channel leakage in cryptographic embedded systems. Those techniques, if translated to general-purpose computing systems, would likely incur an unacceptable performance cost. More efficient mitigation of the frequency side channel may take advantage of its coarse-grained nature: attackers learn the average power draw over some interval, not the power draw of each operation.

TAKEAWAY

Our findings expand the attack surface of power side-channel attacks by removing the need for power measurement interfaces or physical proximity. Current cryptographic best practices for writing constant-time code are no longer sufficient to guarantee constant time execution on modern, variable-frequency processors. Our group's follow-up work has already identified implementations of several cryptographic primitives beyond SIKE that are vulnerable to Hertzbleed.[8]

ACKNOWLEDGMENT

Yingchen Wang and Riccardo Paccagnella contributed equally to this work.

REFERENCES

1. P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptology Conf.*, 1999, pp. 388–397.
2. M. Lipp et al., "PLATYPUS: Software-based power side-channel attacks on x86," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2021, pp. 355–371, doi: 10.1109/SP40001.2021.00063.
3. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proc. Adv. Cryptology*, 1996, pp. 104–113.
4. S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, vol. 31. New York, NY, USA: Springer Science & Business Media, 2008.
5. Y. Wang, R. Paccagnella, E. He, H. Shacham, C. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into timing attacks on x86," in *Proc. USENIX Secur. Symp.*, 2022, pp. 679–697.
6. L. De Feo et al., "SIKE channels: Zero-value side-channel attacks on SIKE," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2022, no. 3, pp. 264–289, Jun. 2022, doi: 10.46586/tches.v2022.i3.264-289.
7. W. Castryck and T. Decru, "An efficient key recovery attack on SIDH," in *Proc. Adv. Cryptology Eurocrypt*, 2023, pp. 423–447.

8. Y. Wang et al., "DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2023.

YINGCHEN WANG is a Ph.D. student in the Computer Science Department, The University of Texas at Austin, Austin, TX, 78712, USA. Her research interests include exploring next-generation microarchitecture side-channels and studying their security implications on software-level applications, especially cryptography protocols. Contact her at yingchen@cs.utexas.edu.

RICCARDO PACCAGNELLA is a Ph.D. candidate in computer science at the University of Illinois Urbana-Champaign, Urbana, IL, 61801, USA. His research interests include system and hardware security. Contact him at rp8@illinois.edu.

ELIZABETH TANG HE received her master of computer science degree from the University of Illinois Urbana-Champaign, Urbana, IL, USA, in 2021. Contact her at ehe3@illinois.edu.

HOVAV SHACHAM is a professor of computer science at The University of Texas at Austin, Austin, TX, 78712, USA. His research interests include security, privacy, and tech policy. Shacham received his doctorate degree from Stanford University. Contact him at hovav@cs.utexas.edu.

CHRISTOPHER W. FLETCHER is an assistant professor of computer science at The University of Illinois Urbana-Champaign, Urbana, IL, 61801, USA. His research interests include architecture and security. Fletcher received his doctorate degree from the Massachusetts Institute of Technology. Contact him at cwfletch@illinois.edu.

DAVID KOHLBRENNER is an assistant professor of computer science at the University of Washington, Seattle, WA, 98195, USA. His research interests include hardware security and trusted systems design. Kohlbrenner received his doctorate degree from the University of California San Diego. Contact him at dkohlbre@cs.washington.edu.