Cache Timing Leakages in Zero-Knowledge Protocols

Shibam Mukherjee
Graz University of Technology, Austria
Know Center, Graz, Austria

Christian Rechberger
Graz University of Technology, Austria
TACEO, Graz, Austria

Markus Schofnegger
Fabric Cryptography, San Francisco, CA, USA

Abstract

The area of modern zero-knowledge proof systems has seen a significant rise in popularity over the last couple of years, with new techniques and optimized constructions emerging on a regular basis.

As the field matures, the aspect of implementation attacks becomes more relevant, however side-channel attacks on zero-knowledge proof systems have seen surprisingly little treatment so far. In this paper, we give an overview of potential attack vectors and show that some of the underlying finite field libraries, and implementations of heavily used components like hash functions using them, are vulnerable w.r.t. cache attacks on CPUs.

On the positive side, we demonstrate that the computational overhead to protect against these attacks is relatively small.

2012 ACM Subject Classification Security and privacy → Domain-specific security and privacy architectures

Keywords and phrases zero-knowledge, protocol, cache timing, side-channel, leakage

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.1

Related Version *ePrint Version*: `https://eprint.iacr.org/2024/1390`

Acknowledgements Special thanks to Andreas Kogler[1] for all the side-channel discussions we had. Also thanks to Erik Kraft and Lukas Giner[2] for the discussions on specific aspects of the attacks. We would also very much like to thank Roman Walch[3] and the members of TACEO and other groups for their useful feedback, especially on the current state of the art ZK scenario.

1 Introduction

Recent years have witnessed a significant development in the area of zero-knowledge proofs (ZKPs) [38] and it application in blockchain [94, 66, 32, 35, 108], decentralized apps [84, 31, 101], anonymous credentials [110], crypto-assets [124, 86, 91], verifiable computation [79, 127], and decentralized storage [109, 112, 104, 72], among others. A ZK protocol allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of some statement without revealing any additional information about the statement to $\mathcal{V}$. More specifically, most ZKP applications rely on the technique of zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) [19, 3, 39, 121] for sub-linear proof size and verification time. Since these properties are crucial for practicality, many recent developments focus on making these

[1] `https://andreaskogler.com/`
[2] `https://ginerlukas.com/`
[3] `https://rwalch.at/`

proofs even more efficient. This has also led to groups like Fabric [33], Ingonyama [56], and Supranational [111] working on dedicated hardware accelerators for ZK proof generation and verification.

At USENIX'20, Tramer, Boneh and Paterson [114] demonstrated attacks on two privacy-focused cryptocurrencies, namely, Zcash [124] and Monero [86]. They exploited the timing-based side-channel leakages occurring when executing the protocol, allowing an attacker $\mathcal{A}$ to de-anonymize the secret payee (prover $\mathcal{P}$), violating the fundamental purpose of using private transactions. In their attack, the timing leakage stemmed from the high-level protocol properties such as changed communication patterns, or additional operations like commitments being computed only in certain cases. More recently at USENIX'24, [24] took a broader look into the potential security vulnerabilities in SNARKs protocols, in particular, they discuss various threat models and system-level vulnerabilities.

This work looks into the lower level functionalities of the ZK protocols potentially vulnerable to timing attacks [62, 23], in particular *cache-based timing side-channel attacks* [17, 115]. We observed, leakages occurring in the fundamental components of ZK protocols like the underlying field arithmetic libraries used in various constructions like Merkle trees or operations like NTT, polynomial evaluation, among others. More specifically, we looked into the Merkle tree construction when using hash functions with leaky implementation become susceptible to cache timing attacks.

Being the first work in the literature to explore such timing leakages, we only rely on the well-known Flush+Reload (F+R) [122] and Prime+Probe (P+P) [4, 77, 55] attacks and conjecture that other advanced timing attacks such as the ones with pre-fetching [27] or attaining sub-cache level resolution [105, 123] will also work, potentially with even fewer assumptions and higher precision. We show that when constructing Merkle Trees for ZK membership proofs [16] with pairing-based [60, 28, 46] or FRI-based [11] approaches, a non-constant time hash implementation[4] may leak the preimage containing sensitive information about $\mathcal{P}$'s private witness or potentially forfeiting the zero-knowledge property of the ZK protocols.

On scrutinizing the standard field arithmetic libraries like Rust `ff`, `ff-ce`, `ark-ff` crates, Python `galois` library, Typescript `circomlibjs`, we found several potential timing leakages in various field arithmetic operations like modulo addition, multiplication, reduction, exponentiation, among others. The same leaky field arithmetic libraries are used in several public implementation of ZK-friendly hash functions like *Rescue* [11], MONOLITH [42], and even the de-facto industry-standard POSEIDON [43], carrying forward the timing vulnerabilities in the hash functions. It should be emphasized that the above hash functions themselves exhibit a constant-time algorithm, and the leakages occur only due to the use of underlying leaky field arithmetic libraries for the implementation. Additionally, we also looked into the family of lookup-based ZK-friendly hash functions, like REINFORCED CONCRETE (RC) [41] and to some extent Tip5 [113], which are inherently leaky (in the plain evaluation) due to the use of lookup-tables (LUTs) for efficient ZK proofs and verification. In such hash functions the timing leakage may originate both from the hashing algorithm itself (due to timing differences of LUT memory-access) and also the underlying field arithmetic library.

[4] Generally branched implementations are often faster than their constant time counterparts. This may serve as a motivation in the industry to use it for performance benefits.

Main Contributions

We summarize our main contributions in the following.

- For the first time in the literature, we discuss the implications of cache-based timing side-channel attacks on ZK protocols, forfeiting the zero-knowledge property of pairing-based and FRI-based ZK membership proofs.[5]
- We also discuss $\mathcal{P}$'s private witness recovery complexity by $\mathcal{A}$ when using P+P and F+R attacks on the building blocks of ZK protocols, in particular, ZK-friendly hash functions like Poseidon and other ZK-friendly designs like the ones with faster plain hashing performance due to use of lookup tables (Reinforced Concrete hash function is the first such proposal).
- As part of our larger survey, in Table 3, we provide a non-exhaustive list of potentially vulnerable ZK applications found in the wild, discuss mitigations against timing attacks[6] and motivate the need for expedited research in the direction of micro-architecture [76, 27, 85, 105] and power [63, 81, 98, 75, 1, 82, 25] based side channel attacks on ZK applications.
- Finally, we also provide a constant-time implementation effort for some of the standard field libraries (forked) used in various ZK applications discussed in this paper. On the bright side, we found that the constant-time implementations do not necessarily come with significant performance penalties.

2 Background

Here we briefly discuss the fundamentals of cache timing attacks like CPU caches, the concept of shared memory, and the two well-known cache timing attack strategies, namely P+P and F+R. Then we go through the fundamentals of zero-knowledge proofs and give a brief introduction on ZK-friendly hash functions and the role they play in pairing-based and FRI-based membership approaches.

2.1 Caches and Shared Memory

A standard CPU has a limited number of data registers, and thus cannot fit all the data it needs during a process execution. Any data that is required in the future is fetched from a low-latency memory into the CPU register before processing it. In most modern CPUs, caches provide this low-latency memory, however, they have a limited storage capacity and thus any large data must be fetched from the main memory using temporal and spatial data pre-fetching prediction algorithms. When executing a process, if the CPU finds the required data in the cache, we call it a *cache hit*. Otherwise, if the data needs to be fetched from the main memory, we call it a *cache miss*. The latter comes with a larger latency as the data needs to be fetched from the significantly slower main memory into the cache before the CPU can process it. For example, in a standard Intel CPU, the L1 cache has a latency of three to five clock cycles, whereas the L3 cache has a latency of around 30 to 40 clock cycles. In comparison, data fetching from the main memory may take up to 300 clock cycles.

[5] We also found some public discussions, like `https://github.com/facebook/winterfell/issues/9`, taking up the same concerns.
[6] Several micro-architectural measures have been proposed in the past, however, they come with their own trade-off. In this work, we do not focus on these mitigations as it is a topic on its own interest, but only discuss the best constant-time implementation practices.

Caches are usually partitioned into m smaller cache sets, which are further divided into n cache lines of b bytes each. We commonly refer to such caches as *n-way set-associative* caches, where the total cache size is $m \cdot n \cdot b$ bytes.[7] The L1 caches are shared between the sibling cores contained in a physical core, whereas the L3 cache is shared among all the physical cores and their respective sibling cores. Depending on the type of the cache-based timing attack, $\mathcal{A}$ and $\mathcal{P}$ may have to share the same physical core in the threat model.

The concept of shared memory is heavily used by the operating system due to its performance benefits. For instance, several processes relying on the same library can access the shared physical memory where the library is mapped, averting the need for having several copies of the same library in the main memory for each process. Standardized cryptographic implementations like OpenSSL [89], NSS [87], Libgcrypt [74], or even ZKP libraries can be examples of such shared libraries.

2.2 Cache Timing Attacks

Flush+Reload Attack

In 2016, Yarom and Falkner [122] presented the first Flush+Reload (F+R) attack to recover the RSA key. A F+R attack targets the shared instance, like a cryptographic library, mapped on the common cache accessible to both $\mathcal{A}$ and $\mathcal{P}$. $\mathcal{A}$ calls the `clflush` instruction to flush (remove) the cache lines containing the library data, for instance, the S-box lookup table (LUT) of AES or even Reinforced Concrete. $\mathcal{A}$ later reloads the flushed data with `maccess` and measures the reload latency. If the data is not present in the cache, reload requires more time as the data needs to be fetched from the main memory. If a particular section of the flushed data has a smaller reload latency, $\mathcal{A}$ learns that $\mathcal{P}$ accessed that section, as the data was already loaded, potentially leaking the secret input to the cryptographic function, for example, the input to the S-box. Similar to the F+R attack, Gruss, Maurice, Wagner, and Mangard [48] proposed the Flush+Flush (F+F) attack, a window-less low latency high resolution attack where reloading the data is not required as $\mathcal{A}$ can always flush the memory.

Prime+Probe Attack

In 2006, Osvik et al. [90] introduced the Prime+Probe (P+P) attack, followed by later improvements [4, 77]. Compared to F+R, this attack does not require access to shared cryptographic library between $\mathcal{A}$ and $\mathcal{P}$, reducing the security assumptions. In this attack, $\mathcal{A}$ primes (loads) all the cache sets with their eviction set (data) and then waits for $\mathcal{P}$ to load their data on the cache, replacing $\mathcal{A}$'s eviction set. $\mathcal{A}$ then sequentially probes the cache sets and looks for addresses where the eviction set was replaced. If the set was replaced by $\mathcal{P}$, $\mathcal{A}$'s access time to the set will be longer as the data needs to be fetched from the main memory into the cache. P+P attacks are generally more noisy and are limited only to the cache set resolution, giving low information leakage compared to F+R attacks which have a cache line resolution.

[7] For example, our test machine runs on an Intel i7-7600U CPU which has a 4 MB L3 cache with $m = 4096$, $n = 16$ and $b = 64$ bytes.

2.3 Zero-Knowledge Proof

A zero-knowledge proof [38] allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of a statement, using a public input (public data and the circuit) and potentially private witness data such that the public input reveals no information about the statement. For instance, $\mathcal{P}$ may want to convince $\mathcal{V}$ that they know a preimage x such that $y = H(x)$ for a publicly known value y and a cryptographic hash function H. A similar approach is also used for modern post-quantum signature schemes [26, 9, 30, 8], especially the ones based on MPCitH and VOLEitH ZKP paradigms. Here $\mathcal{P}$ proves the knowledge of their secret key *sk* (used for signing a message) tied to a particular public key *pk* (plaintext-ciphertext pair), where the randomness is generated with a seed and the message to be signed. Two crucial properties of any ZKP protocol are *completeness* and *soundness*. Informally, the former ensures that an honest $\mathcal{V}$ will be convinced by an honest $\mathcal{P}$ and the latter ensures that a dishonest $\mathcal{P}$ cannot convince an honest $\mathcal{V}$ except with some small probability (usually designed to be negligible in the security parameter). Additionally, any ZK protocol must also provide the *zero-knowledge* property.[8] Modern general-purpose ZK protocols can be essentially split into two categories, namely pairing-based [46, 29, 80, 28, 2, 73, 102] and FRI-based [11, 14, 18] approaches. With the former, hash functions are often used to prove membership in zero knowledge, e.g. to prove on-chain coin ownership [124, 86] or show group membership [101, 110]. In contrast, FRI-based approaches often use the hash function primitives internally in recursive proving approaches, which is a crucial building block of various ZK-rollup applications [52, 127, 116, 7].

2.4 ZK-Friendly Hash Functions

Modern ZK protocols often rely on hash functions which exhibit a property referred to as *circuit friendliness*, *ZK-friendliness*, or *arithmetization friendliness*, important for strong performance when proving particular statements in ZK. This essentially means that the algorithmic description of the hash function can be translated to a relatively small system of low-degree constraints, which makes these constructions efficient in proof systems for computational integrity. In the following, we give a brief overview of two such constructions, one of them being among the most popular choices and the other being a more modern approach using lookup arguments in the proof.

Poseidon

Poseidon [43] is a cryptographic hash function designed for efficient implementation as an arithmetic circuit, making it particularly suitable for zero-knowledge proof systems. It operates over a prime field p, where $p \approx 2^n$ and $n \geq 31$, mapping the input messages of arbitrary length to fixed-size digests. The underlying sponge permutation, Poseidon^{π}: $\mathbb{F}_p^*$ $\rightarrow \mathbb{F}_p^o$, where o is the fixed output length measured in p elements, is based on a substitution-permutation network (SPN) first used for HadesMiMC [44]. It consists of several rounds with two different types of layers, namely *full* (or *external*) and *partial* (or *internal*) rounds. The former includes full nonlinear layers where S-boxes are applied to every word in the state. The latter consists of a partial nonlinear layer where the S-box is only applied to the first word of the state. The idea behind this approach is to provide simple arguments against statistical attacks using consecutive full rounds, while achieving the same degree

[8] We note that in some practical settings the ZK property may not be needed, but instead only the computational integrity property of modern general-purpose proof systems is used.

growth using the more efficient partial rounds. The affine layer $M(\cdot)$ consists of a $t \times t$ MDS matrix multiplication and a round constant addition $\mathrm{ARC}(\cdot)$. The nonlinear S-box function is defined as $S(x) = x^{\alpha}$, where $\alpha \geq 3$ is a small positive integer for which $\gcd(p-1, \alpha) = 1$. An overview of the POSEIDON$^{\pi}$ permutation is shown in Figure 1.

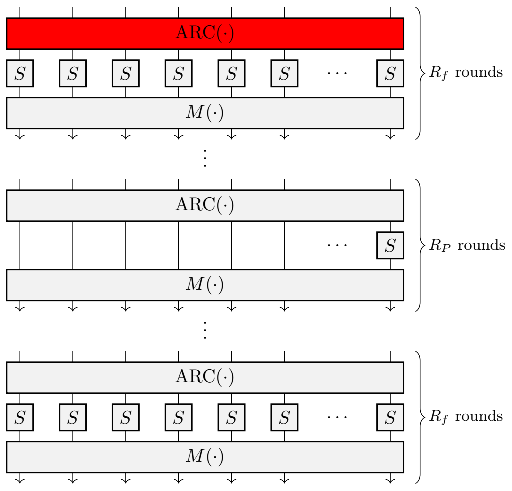


Figure 1 The POSEIDON$^{\pi}$ permutation.

In computational integrity proof systems, the advantages of POSEIDON stem from its low number of nonlinear operations, primarily due to the minimal use of S-boxes. This property makes it easier to express the hash function as a circuit and prove knowledge of its preimages within the proving framework. Indeed, nowadays POSEIDON is used in various ZK-related applications, including RISC Zero [99], Plonky2 [94], Plonky3 [94], and for many on-chain use cases such as Filecoin [72], Dusk Network [31], Sovrin [107], Loopring [78].[9] We refer the reader to POSEIDON [43] for more details.

Reinforced Concrete (RC)

REINFORCED CONCRETE [41] is another ZK-friendly sponge-based hash function which maps $\mathbb{F}_p^3 \to \mathbb{F}_p^3$ for some prime p, operating over two elliptic curves, namely BN254 and BLS12-381 and one special prime field (-ST) crafted for performance. REINFORCED CONCRETE uses lookup-based S-boxes to take advantage of proof systems which use the lookup argument for efficient set membership proofs. The underlying permutation consists of a modified 7-round SPN with CONCRETE, BRICKS and BARS layers, where CONCRETE $\circ$ BRICKS is one round, and a single BARS layer is applied in the middle of the permutation. The BRICKS function is a nonlinear permutation. The CONCRETE function is an MDS matrix multiplication with the state in $\mathbb{F}_p$. The BAR layer consists of composition/decomposition and S-box functions, where the BAR layer is implemented as a lookup table of the functions it contains. RC can be considered as an attractive drop-in replacement for POSEIDON due to its increased efficiency for lookup-based protocols like Arya [20], Plookup [36, 92], Halo2 [124], and logUp [49]. A graphical representation of the RC function is provided in Figure 2. We refer the reader to RC [41] for more details.

[9] By "on-chain use cases" we refer to operations and data executed and stored on the main blockchain directly.

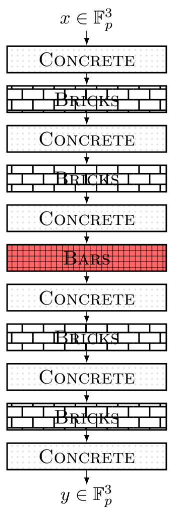


Figure 2 The Reinforced Concrete permutation.

3 Hash Functions in Zero-Knowledge Proofs

We distinguish between two types of general-purpose ZK proof constructions, namely pairing-based and FRI-based ones. Circuit-friendly hash functions are regularly used in both of them, and we give a summary of two main settings below.

For the purpose of this section, we utilize the side-channel technique as a blackbox component in order to describe the setting and the general approach of the attack. In the next section, we go into the details of the side-channel part of the attack specifically.

3.1 Hash Functions in Pairing-Based Proofs

ZK-friendly hash functions defined over pairing-friendly elliptic curves are a powerful tool in many real-world scenarios. In particular, many proof systems [28, 2, 46] (especially those directly employed on chains like Ethereum) are based on this type of elliptic curves in order to support efficient verification of proofs. In this context, a ZK-friendly hash function can be translated to a circuit with minimal overhead.

To give an example, let us consider a simple membership proof where $\mathcal{P}$ wants to convince $\mathcal{V}$ that they hold some secret input (say, a coin). The input is part of a publicly verifiable set fixed by a Merkle root (say, a block on the blockchain containing information about the ownership of the coin). In this case, a hash function can be used to prove knowledge of the secret input without trivially revealing it to $\mathcal{V}$. In order to prove the opening path in the Merkle tree efficiently, a ZK-friendly hash function is used to construct the Merkle tree, resulting in easily verifiable constraints when used together with a pairing-based proof. Here, $\mathcal{V}$ verifies if the path from the leaf to the root of the Merkle tree was computed correctly by checking the correct execution of all the d hash function calls on the path in ZK, where d is the depth of the Merkle tree. In the context of cache timing attacks, any information leakage from the hash function also directly leaks $\mathcal{P}$'s secret witness (leaf) values. Using classical hash functions like SHA-3 makes this approach significantly more expensive in general, as these often result in a more complex and less efficient constraint system when computing the membership proof (i.e., the path from a leaf to the public root).

3.2 Hash Functions in FRI

Similar to pairing-based approaches, ZK-friendly hash functions are also used in FRI-based approaches [10] for constructing Merkle trees during the commitment phase. However, unlike the pairing-based approach, FRI protocols do not directly take the secret witness as the input to the hash function, but instead, at a very high level, a polynomial f is constructed by interpolating the private witnesses, which is then evaluated at some special points (discussed later), and the outputs become the leaves of the Merkle tree fed to the hash functions for computing the Merkle root. Here $\mathcal{P}$ proves to $\mathcal{V}$ that they know a set of points (witnesses) which pass through f of a particular degree d (among other parts in the entire proof). In most applications, these witness points are execution traces of a program running on $\mathcal{P}$'s machine, where $\mathcal{P}$ proves the correct execution of the program to $\mathcal{V}$. The circuit friendliness of a given hash function is particularly important for efficient *recursive proofs*, which essentially include proving verifications of many other proofs.

Currently, FRI-based approaches are used in many popular projects widely adopted in the industry [100, 99, 108, 120, 93], mainly due to their advantages like simple setups and high prover performance. Indeed, many recent developments in the area of zero-knowledge protocols have focused on making FRI-based proving more efficient [51, 6].

Adding Zero Knowledge to FRI

While FRI is often used without the zero knowledge property (e.g., exploiting the succinctness property for arguments of computational integrity), in many cases ZK is needed and activated. For that, various other strategies have to be applied. When focusing on the trace in particular, one part consists of adding random values to an existing witness column in the trace, in order to change the length of the trace to a more suitable number (for example, a power of 2) and to mask the potentially private witness values. We refer to [50] for more details on this approach and a summary of different techniques.

Another step consists of masking the Merkle tree leaves with random paddings, in order to prevent leaking information in the opening part of the proving protocol. A more detailed description of this approach is given in [15]. Since our goal is to find private witnesses, we consider a scenario where zero knowledge is enabled.

Side-Channel Attack on FRI

Let us consider a trace domain $\mathcal{D}$, which we assume to be a multiplicative subgroup of order N generated by ω. The first step of $\mathcal{P}$ is to take the witnesses $\{w_i\}_{i=0}^{N-1}$ in a trace column and to find a polynomial f such that $f(\omega^i) = w_i$. This can be done by an inverse number-theoretic transform (NTT). Note that for simplicity, in this description we omit the random non-witness values added for zero knowledge. We will show that later this makes no difference in the context of our side-channel attack.

After this step, we define the evaluation domain to be a slightly larger domain $\mathcal{D}'$ of $\beta \cdot N$ elements, where β is called the *blowup factor*, generated by $\omega^{1/\beta}$ and shifted by s, i.e., $\mathcal{D}' = \{s \cdot \omega^{i/\beta}\}_{i=0}^{\beta \cdot N-1}$ and $|\mathcal{D}'| = \beta \cdot |\mathcal{D}|$. $\mathcal{P}$ then commits to the evaluations of f over $\mathcal{D}'$ (along with random padding, again added for zero knowledge) using a Merkle tree, where the Merkle root is the commitment. We refer to [10, 12] for more details.

In this scenario, the task of $\mathcal{A}$ is to find the potentially private witness values. This can be done if the evaluations $\{f(s \cdot \omega^{i/\beta})\}_{i=0}^{\beta \cdot N-1}$ are leaked. Indeed, note that the polynomial f originates from the witnesses and, with overwhelming probability, is of degree $N - 1$.

Hence, it suffices to choose any $N-1$ points from $\{(s \cdot \omega^{i/\beta}, f(s \cdot \omega^{i/\beta}))\}_{i=0}^{\beta \cdot N-1}$ and to find a polynomial $f^\star$ which goes through these points. With high probability, $f = f^\star$, and thus evaluating $f^\star$ on $\mathcal{D} = \{\omega^i\}_{i=0}^{N-1}$ is sufficient to find the witness values $\{w_i\}_{i=0}^{N-1}$.

Summarizing, the attack consists of the following steps.

1. Use side-channel leakage to find the evaluations of the trace polynomial f over $\mathcal{D}' = \{s \cdot \omega^{i/\beta}\}_{i=0}^{\beta \cdot N-1}$.
2. Use $|\mathcal{D}| = N$ of these evaluations to find a highly likely candidate $f^\star$ for f (this can be done by interpolation).
3. Evaluate $f^\star$ on $\mathcal{D}$ to find $\{f^\star(\omega^i) = w_i^\star\}_{i=0}^{N-1}$. Again, with high probability, $\{w_i^\star\}_{i=0}^{N-1} = \{w_i\}_{i=0}^{N-1}$, which corresponds to the original witness values.

The last two steps can be handled relatively easily from $\mathcal{A}$'s point of view. However, the first step requires more assumptions, in particular, using high-resolution cache timing attacks, like sub cache line attacks [105, 123], mounted on hash implementations with byte-level leakages, like when using LUTs for the S-boxes in RC hash function. This allows $\mathcal{A}$ to directly learn $\{f(s \cdot \omega^{i/\beta})\}_{i=0}^{\beta \cdot N-1}$ and reconstruct f with trivial complexity as discussed above.

When using low-resolution attacks like F+R and P+P, or if the hash implementation leaks only a few bits of information, $\mathcal{A}$ obtains several candidates for every $\{f(s \cdot \omega^{i/\beta})\}_{i=0}^{\beta \cdot N-1}$ preimage. $\mathcal{A}$ can then iterate through all these candidates in order to reconstruct f. In particular, this is possible if the evaluations of f over $\mathcal{D}'$ exhibit a strong structure, which however is mainly mitigated due to the diffusion properties of the NTT application. Indeed, the evaluation of f over $\mathcal{D}'$ results in a highly unstructured table even for small variations in $\{w_i\}_{i=0}^{N-1}$. An obvious exception occurs if all values in $\{w_i\}_{i=0}^{N-1}$ are the same (i.e., $f(x) = w_0$), which however is an unrealistic scenario both due to the practical use cases for proof systems and due to additional steps, such as padding applied to trace columns.

Zero Knowledge and Side-Channel Resistance

In the context of our cache timing attack, some of the additional techniques to achieve zero knowledge are no obstacle, as recovering the preimage includes both the evaluation points (evaluations of f at publicly known inputs) and the random paddings, which allows $\mathcal{A}$ to fully reconstruct f via interpolation. Once f is reconstructed, $\mathcal{A}$ can trivially retrieve the secret witness by evaluating f at the publicly known input points. The added random values to mask the interpolating polynomial are no obstacle here.

We also emphasize that the initial step of the interpolation is applied to potentially private witness data. It remains a future direction to investigate how this can be exploited in cache attacks with the high-resolution leakages and other side-channel approaches.

4 Cache Timing Attacks

Here we define the threat model under which we mount the F+R and P+P attacks and discuss the preimage recovery complexity when attacking public implementations [45] of Poseidon and RC. We also provide a non-exhaustive list of open source ZK libraries that were found to be using pairing-based and FRI-based ZK approaches with non-constant time Poseidon implementation. It should be emphasized that we only point out the preimage leakage originating from the hash implementation, especially critical for the secure pairing-based approaches. For FRI-based approaches, we are more interested in breaking the zero-knowledge property by recovering some bits of the preimage in the Merkle leaves.

Threat Models

- **First case.** $\mathcal{P}$ is generating the proof for their witness on their personal machine $\mathbf{M}$. We assume $\mathcal{A}$ can run side channel measurements on $\mathbf{M}$ through malware payload delivered to $\mathcal{P}$ or more sophisticated browser-based side-channel exploits [37, 103].
- **Second case.** Assume $\mathcal{P}$ is generating the proof for their witness on a shared $\mathbf{M}$ (like a cloud), allowing $\mathcal{P}$ to access powerful hardware for generating faster proofs while minimizing the cost by sharing resources[10]. $\mathcal{P}$ trusts cloud $\mathbf{M}$ to keep its plain unencrypted witness a secret from a third party $\mathcal{A}$. $\mathcal{A}$ (e.g., pretending to be another legit prover), runs their malicious side-channel programs on the shared $\mathbf{M}$ and learns some (if not all) bits about $\mathcal{P}$'s secret witness through cache-based timing attacks when $\mathcal{P}$ generates the ZK proof for $\mathcal{V}$.
- **Third case.** $\mathcal{P}$ does not trust cloud $\mathbf{M}$ and sends only encrypted witnesses to $\mathbf{M}$ which are privately decrypted in the secure enclaves, like Intel SGX, and then the proof is generated. Leaking the witness requires cloud $\mathbf{M}$ exploiting secure enclave vulnerabilities or performing cache timing attack on the application processor after the private decryption.

Due to the zero-knowledge property, we know that $\mathcal{V}$ cannot extract any new information about the secret witness by only reading the proof sent by $\mathcal{P}$. Likewise, if $\mathcal{A}$ gets access only to the same proof sent to $\mathcal{V}$, $\mathcal{A}$ cannot learn anything new about the witness either, and thus any side-channel attack performed on $\mathcal{V}$'s end is meaningless.

Both $\mathcal{A}$ and $\mathcal{P}$ are required to run on the same CPU, however, they may run on different physical cores as F+R and P+P attacks target the LLC shared by all cores. For the F+R attack we assume that $\mathcal{A}$ and $\mathcal{P}$ share the cryptographic library containing the ZK functionalities like hash functions and arithmetic field operations as a shared object. Also, the `clflush` instruction must be available to $\mathcal{A}$, essential for F+R attacks, otherwise $\mathcal{A}$ may have to perform other similar attacks like Evict+Time [90] or Evict+Reload [47]. All these attacks rely only on measuring the memory access timing difference. For some implementations, as a common practice in the community, when demonstrating proof-of-concept cache-based timing attacks, we use additional `nop` instructions to align the branches on different cache lines. Naturally, such an instruction is equivalent to calling functions from different branches which usually get aligned on different cache lines. Another example is compiler optimization which may put the secret-dependent branched code unfavorably on distinct cache lines, or even cache sets, leading to practical timing attack like in the case of the Kyber constant-time reference implementation.[11] When employing more modern side-channel attacks, one may not require including such `nop` instructions. Before independently verifying the leakages, we used the DATA framework [119][12] to search for any secret-dependent timing leakages in hash function implementations. A summary of the threat model can be found in Table 1.

4.1 Reinforced Concrete Cache Timing Attack

The RC hash function uses lookup tables (LUTs) in the Bars function. Any LUT implementation is vulnerable to cache timing attacks, especially when the LUT lies on multiple cache lines/sets. For the F+R attack, we assume that $\mathcal{A}$ and $\mathcal{P}$ (victim) share the RC

[10] The use-case in consideration mostly determines the computation power required to generate the proof irrespective of the succinct verification protocol. Use-cases like proving root of Merkle tree with $\geq 2^20$ hashes can be resource demanding.

[11] `https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/`

[12] `https://github.com/Fraunhofer-AISEC/DATA/tree/master`

Table 1 A summary of the threat model involving all the 3 parties, $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{V}$ in a ZKP protocol.

Parties	Machine	Cache SCA		Assumptions
		F+R	P+P	
Prover	Personal/Cloud	Yes	No	Shared ZKP library
		N/A	N/A	Additional instructions
		N/A	N/A	Attack resolution
Attacker	Prover's Personal/Cloud	Yes	No	Shared ZKP library
		`clflush,nop`	`nop`	Additional instructions
		Cache line level (64 bytes)	Cache set level (1024 bytes)	Attack resolution
Verifier	N/A	N/A	N/A	N/A

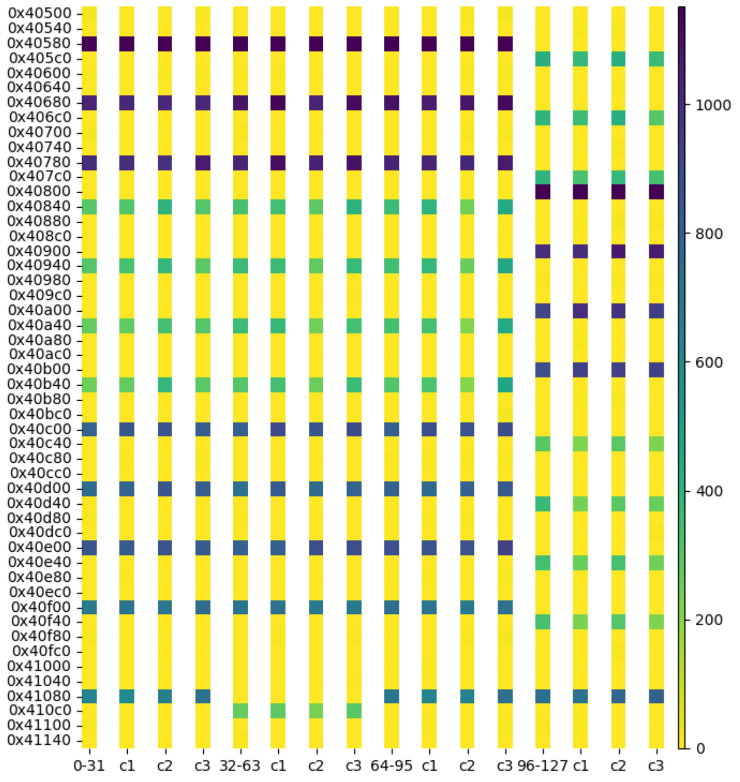
hash function as a shared library including the Bars layer LUT. In RC-BLS12-381, the $\mathbb{F}_p$ elements in the Bars layer are *decomposed* into 27 elements in $\mathbb{F}_{659}$ and then the S-box lookup is applied. Even though $\mathbb{F}_{659}$ can be represented with 10 bits, for efficiency purposes, the elements are mapped as 2-byte elements in the memory, making the LUT 1318 bytes in size, requiring 21 cache lines or 2 cache sets on the LLC of our test machine.

In the F+R attack, with every cache hit $\mathcal{A}$ learns $\approx$ 5 bits of equivalent (total input in bits – possible candidates in bits) S-box input information. In other words, out of 659 elements ($\approx 2^{9.32}$), $\mathcal{A}$ learns the possible $32 = 2^5$ or $19 \approx 2^{4.2}$ (the last cache line) input candidates for the S-box. In P+P attack, however, the recovery resolution is worse than F+R as the resolution is constrained to the cache set level only. In particular, $\mathcal{A}$ learns ≈ 2 bits of equivalent S-box input information, comparable to learning the possible $512 = 2^9$ or $147 \approx 2^7$ (last cache set) input candidates for the S-box. We observe somewhat similar results for RC-BN-254 and RC-ST. However, in the BN-254 implementation there exists a cache line with a single field element, allowing $\mathcal{A}$ to learn all 9.32 bits of equivalent input information in the best case F+R attack scenario occurring with a non-negligible probability of $(642)^{-1} \approx 2^{-9.3}$. We refer to Table 2 for an overview of the preimage recovery complexity. Refer to Figure 3 for illustration of the F+F[13] and P+P attack on the RC hash function.

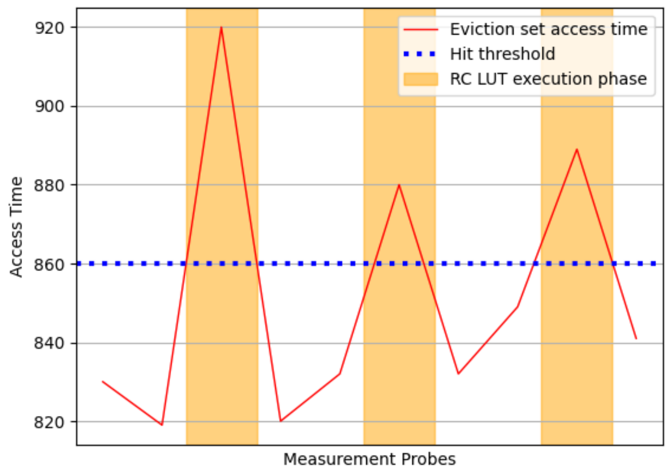
When considering stronger attacks, like the ones with sub-cache line resolution [105], $\mathcal{A}$ may learn all the S-box input bits, fully recovering the preimage without additional search operations. Moreover, for the F+R and P+P attacks, the preimage recovery complexity also decreases when the preimage has some structure to it. This is particularly the case in many ZK applications where the Merkle leaves contain, for example, personally identifiable information like name or dates of birth. Padding some randomness (salt) to the leaves may help against side-channel attacks where $\mathcal{A}$ merely observes the hashes of the tree. However, because the padding is only applied to specific positions of the leaves, this mitigation does not protect against cache timing attacks.

In the following subsections, we discuss some potentially vulnerable Merkle tree constructions used in real-world ZK applications.

[13] We use F+F instead of F+R for attacking RC as the former gives less noisy measurements. The recovery complexity remains the same in both the cases.



(a) F+F cache-hit and cache-miss ratio for Reinforced Concrete BLS-12-381 hash function. Darker blocks represent higher cache-hits in party containing S-box LUT. The input of each S-box starts from `0x0000` and sequentially increases by a multiple of `0x0020` probing the first 4 cache lines, containing 32 inputs each, accessed by $\mathcal{P}$ when generating the Merkle tree. We repeat the attack (c1..c3) demonstrating consistency in the timing leakage when calling S-box inputs contained in the same cache line.



(b) Probe measurements of Reinforced Concrete BLS-12-381 hash function with P+P attack. In the orange phases, the LUT S-box inputs 0-511 are called. We probe the eviction set contained in the cache-set where the 0-511 S-box inputs lie, giving slower probe time compared to not calling the 0-511 S-box inputs.

Figure 3 Access time for F+F and P+P attacks on the Reinforced Concrete hash function. Here $\mathcal{A}$ probes the S-box LUT memory addresses and measures the time to distinguish if the same address was accessed by $\mathcal{P}$ or not.

4.2 Poseidon Cache Timing Attack

As a demonstration, we only attack the Poseidon Goldilocks and Mersenne non-constant time implementations [45] and summarize their leakages.[14] In particular, we attack the ARC($\cdot$) function performing modulo reduction after the field addition between the *state* and the round constants rc. Depending on the branch, $\mathcal{A}$ can determine if the sum between a state and a round constant is larger than the modulo prime p, where extreme values of round constants (very small or large values in the field) give the most information about the secret state. Additionally, we can also attack functions like $M(\cdot)$ and $S(\cdot)$ containing non-constant time modulo reductions after multiplication operations. However, here we focus on the ARC($\cdot$) function, as this is the first layer that directly operates on the preimage. We assume that the `if-else` branches lie on two different cache lines or cache sets for the F+R or P+P attacks respectively.

In the F+R attack, similar to RC, we assume that $\mathcal{A}$ and $\mathcal{P}$ (victim) are sharing the Poseidon library as a shared object, however, $\mathcal{A}$ instead of flushing the LUT addresses, they flush the addresses of the `if-else` branches to detect which branch is accessed by $\mathcal{P}$.

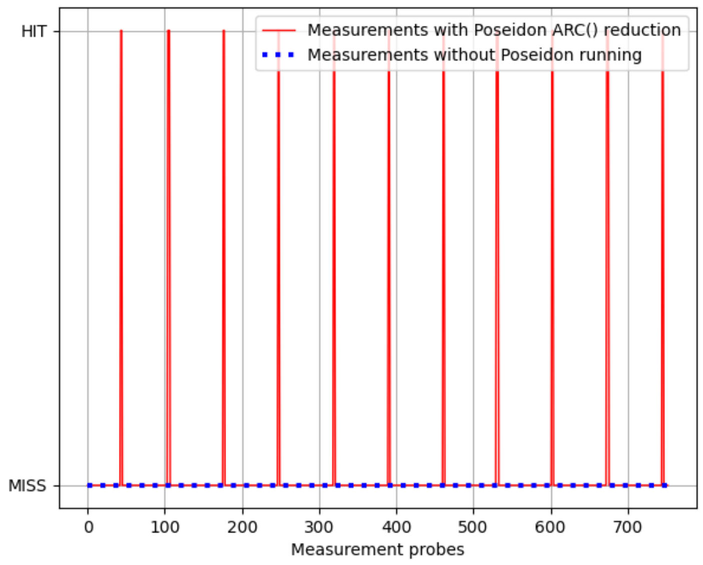
As an input to the Poseidon$^\pi$ permutation, Poseidon-(M, $t = 16$) takes an element in $\mathbb{F}_p^t$, where $p = 2^{31} - 1$ and $t = 16$, where the first layer of Poseidon$^\pi$ is the round constant addition. With the smallest round constant (`0x002f87c1`) in Poseidon-(M, $t = 16$), during a cache hit on the reduction branch, $\mathcal{A}$ learns the most significant ≈ 9 bits (all ones) of $\mathbb{F}_p$ input, where p is `0x7FFFFFFF`.[15] With the largest round constant (`0x7fc1a254`), $\mathcal{A}$ also learns the most significant ≈ 9 bits (all zeros) of the $\mathbb{F}_p$ input. If the round constant rc does not "deviate from random" in $\mathbb{F}_p$, then the leakage is only one bit equivalent information. We observed similar leakages when inspecting Poseidon-(M, $t = 24$) and refer to Table 2 for more details and an overview of timing leakages using different versions of the Poseidon hash function.

The generated round constants will be large with an overwhelming probability. However, for large fields like in BLS-12-381 or BN254, usually custom small round constants are preferred as they provide better plain performance (fewer instructions are necessary for the round constant additions). Refer to Figure 4 for the F+R and P+P attack on the Poseidon-GL hash function.

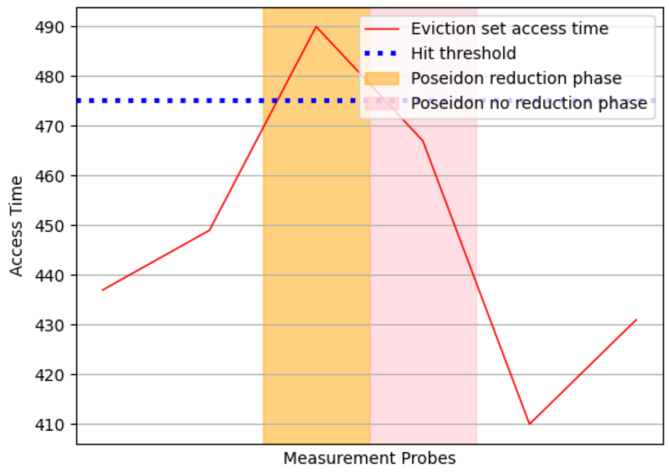
4.3 Other Hash Functions Cache Timing Attack

As the timing leakages mostly stem from the underlying arithmetic library, we also looked into other ZK-fiendly hash function candidates like Tip5, *Rescue*, and Monolith. Due to use of lookups operating over $\mathbb{F}_{2^8+1}$ in one of the Tip5 hash function S-boxes, it may also suffer from the same problems as RC. When performing the F+R attack, $\mathcal{A}$ learns 3 bits of equivalent information in the average case, and all 8 bits in the best case, occurring with a probability of $(2^8 + 1)^{-1}$. We also found some timing leakages in the public implementations of Monolith and *Rescue*. Depending on the *Rescue* implementation, for example, leakage in [67] may occur due to the use of crates like `ff-ce`, similar to Poseidon, or the leakage is only limited to the testing section of the framework like in [120] (`inv()` and `normalize()`). The non-constant time version of the Monolith implementation [45] relies on the same Goldilocks/Mersenne field implementations as Poseidon, thus it might be affected as well. However, Monolith also has a constant time version which is not affected.

[14] The 64-bit Goldilocks prime $2^{64} - 2^{32} + 1$ and the 31-bit Mersenne prime $2^{31} - 1$ define two popular prime fields which are often used for efficient zero-knowledge protocol implementations based on FRI.
[15] Here $\mathcal{A}$ has the same leakage resolution for both F+R and P+P attacks.



(a) Cache-hit and cache-miss ratio for Poseidon-GL hash function with F+R attack. We set the input to the hash function such that modulo reduction (cache hit) is called after every 100 hash calls. We probe the cache address containing the modulo reduction operation branched from the `if-else` check.



(b) Probe measurements of Poseidon-GL hash function with P+P attack. In the orange phase, the round constant addition is followed by the modulo reduction operation. In the pink phase no such reduction is performed. We probe the eviction set contained in the cache-set where the reduction operation instruction also lies, giving slower probe time compared to the phase not executing the modulo reduction.

Figure 4 Access time graph for F+R and P+P attacks on Poseidon-GL hash function. Here $\mathcal{A}$ probes the `if-else` branch memory addresses and measures the time to distinguish if the same addresses were accessed by $\mathcal{P}$.

Table 2 Preimage recovery complexity after F+R and P+P attacks on the Reinforced Concrete and Poseidon hash functions. Here we provide both the average and *best* case input recovery complexity, where the *best* case occurs with a fairly large probability. The preimage leakage is given in terms equivalents bits of information learnt from the full state of the hash function.

Hash function	Timing vulnerability	Preimage leaked (in bits)			
		F+R	% leaked	P+P	% leaked
Poseidon-(M, $t = 16$)	Reduction in ARC(·)	1×16	3.22	1×16	3.22
		$\approx 9 \times 16$	*29.03*	*$\approx 9 \times 16$*	*29.03*
Poseidon-(M, $t = 24$)	Reduction in ARC(·)	1×24	3.22	1×24	3.22
		$\approx 9 \times 24$	*29.03*	*$\approx 9 \times 24$*	*29.03*
Poseidon-(GL, $t = 8$)	Reduction in ARC(·)	1×8	1.56	1×8	1.56
		$\approx 9 \times 8$	*14.06*	*$\approx 9 \times 8$*	*14.06*
Poseidon-(GL, $t = 12$)	Reduction in ARC(·)	1×12	1.56	1×12	1.56
		$\approx 9 \times 12$	*14.06*	*$\approx 9 \times 12$*	*14.06*
RC-(BLS12-381, $n = 27$)	Bars S-box LUT	4.36×27	45.98	0.36×27	3.79
		5.12×27	*54*	*2.17×27*	*22.88*
RC-(BN-254, $n = 27$)	Bars S-box LUT	4.32×27	45.92	0.32×27	3.4
		9.32×27	*100*	*2.31×27*	*24.55*
RC-(ST, $n = 25$)	Bars S-box LUT	4.98×25	49.8	0.98×25	9.8
		5.59×25	*55.9*	*1.02×25*	*10.2*

4.4 Merkle Tree Leaf Distinguisher

Here we discuss the potential attacks that can be mounted on real-world applications, in particular, targeting the Merkle tree construction using non-constant time hash implementations like for Poseidon. One specific aspect we particularly focus on is the entropy of the private values in the Merkle leaves. Low-entropy values, even when padded with randomness at certain points (a common security practice [13, 104]) leak (partial) information about the preimage and hence the secret witness. For example, the Zerocash protocol [13], a foundational work of the Zcash cryptocurrency [124], hashes ($\mathcal{H}(\cdot)$) the coin values $v \in [0, 2^{64} - 1]$ with randomness $k \in [0, 2^{256} - 1]$ as $cm := \mathcal{H}(k \mid\mid 0^{192} \mid\mid v)$, where cm is the commitment value of the coin in the Merkle tree. The authors discuss using SHA-256 as the hash function, however in practice (Zcash), when proving Merkle tree commitments in ZK, Poseidon is the more efficient choice and any non-constant time implementation (Poseidon or other hash function like SHA-256) when constructing the Merkle tree, in this example, will leak the coin value v in accordance with Table 2. Similarly, Sia [104], a decentralized storage protocol, also pads its low-entropy preimage data used in the Merkle tree, such as *time lock*, the number of public keys, and signatures required, by adding a random nonce to the leaves along with the secret input data. Another example is Semaphore [101], a ZK signaling framework, where the public id id_{pub} is padded with a random sequence of bytes $id_{\text{nullifier}}$ when committing to the user identity for generating the Merkle tree membership proofs. These random paddings were adopted to prevent multiple signal broadcasts using the same public id id_{pub}.

In certain conditions, it is even sufficient if $\mathcal{A}$ is able to roughly distinguish the leaves instead of fully recovering the input. For example, a simple ZK use case, where $\mathcal{P}$ convinces $\mathcal{V}$ that they have telephone numbers from certain countries. Let's assume the above Poseidon-(M, $t \in \{16, 24\}$) case where a Merkle tree contains four leaves, the first two leaves contain short phone numbers from Falkland Islands (5 digits, at most 17 bits) and the next two leaves contain phone numbers from Norway (8 digits, at most 27 bits).With the smallest and largest Poseidon-(M, $t \in \{16, 24\}$) round constants, $\mathcal{A}$ can distinguish between the two groups of the leaves when $\mathcal{P}$ constructs the Merkle tree during the proof generation. Even though leaking only 1 bit of information, this holds a larger privacy implication for ZK protocols.

Table 3 We summarize the ZK frameworks (a non-exhaustive list) that may contain potential witness leakages due to F+R and P+P timing attacks on the various non-constant time Poseidon hash function implementations. We also state the source of the potential leakage affecting both the $ARC(\cdot)$ and $M(\cdot)$ layers.

ZK library (Application) - Source of leakage	Non-constant time implementation	Comments
Matter Labs[66] (Blockchain ZK) - rescue-poseidon [67] → ff-ce crate	Uses `ff-ce` crate with branched `reduce` in `add_assign`	`Readme.md` (`ff-ce`) - Does not provide constant-time guarantees
Lurk Labs [79] (Turing-complete ZK programming language) - neptune-poseidon [64] → ff crate [34]	Uses `ff` crate with branched `reduce` in `add_assign`	`Readme.md` (`neptune-poseidon`) - Has been audited `Readme.md` (`ff`) - Does not provide constant time guarantees
Ingonyama poseidon-hash [56] (Hardware acceleration for ZKP) - poseidon-hash [57] → galois [83]	Uses `Galois` library with branched `add_modular`	`Readme.md` (`galois`) - The library could be vulnerable to timing attacks. Not intended for production.
Sui [110] (ZK login and authentication) - Sui [68] → poseidon-lite [118] → jdk (BigInt) [88]	Uses `poseidon-lite` library with branched addition using `jdk/../BigInteger.java`	`Readme.md` (`poseidon-lite`) - The code has not been audited and the native js `BigInt` is vulnerable to timing attacks.
ZK-kit [125] (General ZK library for developing various applications) - ZK-kit [126]	Field arithmetic `f1-field.ts` contains branched addition.	–
Panther [91] (Virtual asset private trading) - Panther [97] → circomlibjs [53] → ffjavascript [54] → jdk (BigInt) [88]	Uses `circomlibjs` Poseidon implementation using `ffjavascript` for BN-128 and BLS12-381 with branched addition performed with `jdk/../BigInteger.java`.	–
Light Protocol [65] (ZK layer on Solana [106]) - Light Protocol [96] → light-poseidon [95] → ark-ff crate [5]	Uses `light-poseidon` using `ark-ff` crate for arithmetic operations. `add_assign` in `montgomery_backend.rs` has branching	`Readme.md` (`light-poseidon`) - Has been audited `Readme.md` (`ark-ff`) - Academic proof of concept, not ready for production use
Mina Protocol [84] (Decentralized Apps) - O1-labs/proof-systems [69] → ark-ff crate [5]	Uses ark-ff crate, `add_assign` in `montgomery_backend.rs` has branching	`Readme.md` (`proof-systems`) - Security audit missing. `Readme.md` (`ark-ff`) - Academic proof of concept, not ready for production use.
Plonky2/3 [70, 71] (Blockchain ZK) - Plonky2/3	`add` contains branching occurring with a small probability (similar to `reduce128`). In theory leaks the input to the hash function, and we believe SHOULD NOT lead to a practical attack with the attack vectors discussed in this work.	During the time of writing this work Plonky3 finished its security audit and became production ready. We did not find constant time implementation as a part of the threat model.

4.5 Leakages in Zero-Knowledge Protocols

Table 3 shows a non-exhaustive list of ZK frameworks, spread across various ZK applications using various non-constant time Poseidon implementations. Most of the vulnerabilities root from the underlying field arithmetic library, except for ZK-kit, where they implement their own field library. In case of Panther and Sui, the underlying jdk native BigInt implementation is the source of leakage, known since 2020 [22]. The poseidon-lite library acknowledges the timing leakage in its disclaimer, however, in both the Sui and the Panther ZK framework such declarations were found missing. Currently, we looked into only a handful of the ZK frameworks which use such leaky field libraries, however, we suspect that there are several other frameworks using them as well.

Even though most of the underlying field libraries state that they are actually vulnerable to timing attacks and thus should not be used for production, however, at the time of writing this paper, they were being used in many ZK libraries. Moreover, to the best of our knowledge, none of the ZK library security audits considered timing attacks in their threat models, which we believe in the future should be taken into consideration as a standard practice as is done in other areas of cryptographic implementations, including hardware designs.

We also looked into other ZK protocols for potential timing leakages, in particular Plonky2, Plonky3 and Halo2. We found branching in the addition and the subtraction operations in both Plonky2 and Plonky3 using Poseidon-GL. However, the authors claimed, and we also independently verified from their public implementation, that this branching happens with a low probability (however still large in the security parameter). In particular, for this to occur, there must exist a pair of a round constant and an input both $\geq (2^{64} - 1) - (2^{30} - 1)$, and hence it is more efficient to branch. In Halo2, we found the Poseidon S-box layer implementation to contain a non-constant time exponentiation `pow_vartime()`, but the exponent is already public and thus does not affect the security. Special care must be taken in the future to not use this function for any secret dependent operation. We also looked into Aztec [7], and to the best of our knowledge, at the time of writing this paper, we did not find any non-constant time Poseidon implementation.

At the time of writing this paper, we did not attack any ZK proof system in production and thus based on our particular findings we can only recommend switching to constant time implementation to mitigate any potential future risk associated with timing leakages, especially considering that the performance penalties are not significant [42] (Table 4). With this work, we are providing forked versions[16] of the `ff`, `ff-ce` and `ark-ff` crates which will include constant-time implementations of the functions discuss in Table 3. As a future collaboration with (but not limited to) the maintainers of the above libraries, we are currently looking into the possibility of integrating our constant-time proposal into the standard ZK libraries.

End-to-End Attacks

Any serious security mitigation step first requires assessing the practicality of a possible end-to-end attack. This work primarily focuses on hinting towards potential timing leakages in ZK protocols and its implications. No end-to-end attack has been demonstrated in this work, however, after discussing with the side channel community, we understand that once an attack works on an isolated target, that is, in our case the non-constant time field

[16] `https://extgit.isec.tugraz.at/krypto/ffconstzksca`

arithmetic libraries used to implement hash functions in zk protocols, a full attack is merely an engineering problem, similar to other side-channel challenges like handling noisy cloud environment during measurements, among other. As the ZK industry grows with more money at stake, the engineering motivation increases. On the contrary, an end-to-end side channel attack might never be even feasible, as in a personal conversation with a side-channel veteran, it is often cheaper to buy off-the-shelf zero-day exploits with system level or application level vulnerabilities. We believe, research on cache timing attacks, even if not financially practical, still plays an important role in identifying these threats and plug the gaps, like in the case of OpenSSL AES table look-ups [17].

Other Timing Attacks

While this work focuses on the well-known F+R and P+P timing attacks on ZKP protocols and their applications. We conjecture that more sophisticated attacks like [61, 76, 27, 105, 85, 40, 117] will reduce the preimage recovery complexity significantly, especially for lookup-based hash functions like Reinforced Concrete and potentially for Tip5. Power side-channel attacks might be another interesting direction to explore.

Disclosures

This work looks into the potential cache timing side-channel vulnerabilities in the current ZKP systems. Even though we do not show any real-world attack on the existing ZK applications, however, following our footsteps, in the future one may find real-world exploits, potentially causing financial loss and privacy violation, among other damages. We have already informed some of the industry groups about our findings, however, it is infeasible for us to inform every potentially affected group as the scope of our attack spreads across a wide range of ZK applications. The informed industry groups have already acknowledged our findings, and we hope mitigations are out soon.

Conclusion

Non-constant time implementations leak sensitive input data in ZK protocols. Even though attacks like P+P and F+R were introduced several years ago, one still finding potential exploits using them due to non-constant time implementations even in fairly recent protocols like ZKP and their applications. As ZK use-cases mature, side-channel protection will be more crucial. The first step in this direction could be to follow the Intel guidelines on mitigation against timing attacks [58, 59], including constant time implementation of any secret-dependent code. In our context, we recommend switching to a constant-time field arithmetic libraries for implementing constant time algorithm hash functions like Poseidon. While this can be challenging and may not result in the most performant implementation, even though the performance penalties are small, it is important from a security perspective and should be treated as a minimum requirement. In case of non-constant time hash algorithms like Reinforced Concrete and Tip5 (the ones relying on look-up-tables in plain), one can switch to newer generation of look-up-based hash functions like Monolith [42] and Skyscraper [21], with constant time plain S-box evaluation while still supporting look-up feature in ZK. Furthermore, one may also adopt free and easy-to-use tools like the DATA framework [119] for conducting initial checks for any secret-dependent cache-based timing leakages.

It is well-known that constant-time implementations often do not translate into constant-time assembly code due to compiler optimizations which are often unpredictable. The leakages may also originate from other sources like described in [27] and often for better constant time guarantees, protocols, or at least the secret dependent components, can be directly written in `asm`. Moreover, security audits of the ZK libraries may also take into consideration basic cache timing leakages in their security model. One can also switch to dedicated side-channel secure hardware for generating zero-knowledge proofs. Combining multi-party-computation with ZK, also known as coSNARKS can be another potential candidate to protect against side-channel attacks on ZK. Learning from the FRI protocols providing a natural protection against SCA due to the inverse-interpolation step, developing generic ZK side-channel counter measures on a similar path can also be an interesting direction to look into for other protocols.

Table 4 Runtime comparison between the constant time and non-constant time implementation of Poseidon-M and Poseidon-GL [45]. Benchmarked on Intel Core i5-12450H running WSL2.0 with Ubuntu 22.04.3 LTS.

Hash function	Non-constant time	Constant time	% Slower
Poseidon-(M, $t = 16$)	4.577 µs	4.996 µs	9%
Poseidon-(M, $t = 24$)	10.183 µs	10.929 µs	7%
Poseidon-(GL, $t = 8$)	1.997 µs	2.318 µs	16%
Poseidon-(GL, $t = 12$)	3.708 µs	4.057 µs	9%

References

1 Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002. doi:10.1007/3-540-36400-5_4.

2 Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. a*Plon*K: Aggregated *Plon*K from multi-polynomial commitment schemes. In Junji Shikata and Hiroki Kuzuno, editors, *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*, volume 14128 of *Lecture Notes in Computer Science*, pages 195–213. Springer, 2023. doi:10.1007/978-3-031-41326-1_11.

3 Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017. doi:10.1145/3133956.3134104.

4 Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. S$a: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 591–604. IEEE Computer Society, 2015. doi:10.1109/SP.2015.42.

5 ark ff. ark-ff. https://docs.rs/ark-ff/0.4.2/ark_ff/.

6 Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. *IACR Cryptol. ePrint Arch.*, page 390, 2024. URL: `https://eprint.iacr.org/2024/390`.

7 Aztec. Aztec. `https://aztec.network/`.

8 Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and owfs for post-quantum signatures. *IACR Cryptol. ePrint Arch.*, page 490, 2024. URL: `https://eprint.iacr.org/2024/490`.

9 Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 266–297. Springer, 2021. `doi:10.1007/978-3-030-75245-3_11`.

10 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPICS.ICALP.2018.14`.

11 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, page 46, 2018. URL: `http://eprint.iacr.org/2018/046`.

12 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, page 46, 2018. URL: `http://eprint.iacr.org/2018/046`.

13 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014. `doi:10.1109/SP.2014.36`.

14 Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019. `doi:10.1007/978-3-030-17653-2_4`.

15 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive Oracle Proofs. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. `doi:10.1007/978-3-662-53644-5_2`.

16 Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1993. `doi:10.1007/3-540-48285-7_24`.

17 Daniel J Bernstein. Cache-timing attacks on aes, 2005.

18 Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020. `doi:10.1145/3372297.3417893`.

19 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012. `doi:10.1145/2090236.2090263`.

20 Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018. doi:10.1007/978-3-030-03326-2_20.

21 Clémence Bouvier, Lorenzo Grassi, Dmitry Khovratovich, Katharina Koschatko, Christian Rechberger, Fabian Schmid, and Markus Schofnegger. Skyscraper: Fast hashing on big primes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):743–780, March 2025. doi:10.46586/tches.v2025.i2.743-780.

22 Tegan Brennan, Nicolás Rosner, and Tevfik Bultan. JIT leaks: Inducing timing side channels through just-in-time compilation. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1207–1222. IEEE, 2020. doi:10.1109/SP40000.2020.00007.

23 David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. URL: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical.

24 Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. Sok: What don't we know? understanding security vulnerabilities in snarks. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/chaliasos.

25 Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. doi:10.1007/3-540-36400-5_3.

26 Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017. doi:10.1145/3133956.3133997.

27 Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. Gofetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *Proc. USENIX Secur. Symp*, pages 1–21, 2024.

28 Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020. doi:10.1007/978-3-030-45721-1_26.

29 Electric Coin. Halo2. https://zcash.github.io/halo2/index.html.

30 Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 843–857. ACM, 2022. doi:10.1145/3548606.3559353.

31 Dusk. Dusk network. https://dusk.network/.

32 Ethereum. Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper. `https://ethereum.github.io/yellowpaper/paper.pdf`.

33 Fabric. Fabric cryptography. `https://www.fabriccryptography.com/`.

34 ff. ff crate. `https://docs.rs/ff/0.13.0/ff/`.

35 Nil Foundation. Nil;'s zkevm1. `https://cms.nil.foundation/uploads/zk_EVM_1_7d6f8caa16.pdf`.

36 Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, page 315, 2020. URL: `https://eprint.iacr.org/2020/315`.

37 Lukas Giner, Roland Czerny, Christoph Gruber, Fabian Rauscher, Andreas Kogler, Daniel De Almeida Braga, and Daniel Gruss. Generic and automated drive-by GPU cache attacks from the browser. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro Cardenas, editors, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*. ACM, 2024. `doi:10.1145/3634737.3656283`.

38 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985. `doi:10.1145/22145.22178`.

39 Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. `doi:10.1007/978-3-031-38545-2_7`.

40 Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 955–972. USENIX Association, 2018. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/gras`.

41 Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Reinforced concrete: A fast hash function for verifiable computation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1323–1335. ACM, 2022. `doi:10.1145/3548606.3560686`.

42 Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Hash functions monolith for ZK applications: May the speed of SHA-3 be with you. *IACR Cryptol. ePrint Arch.*, page 1025, 2023. URL: `https://eprint.iacr.org/2023/1025`.

43 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021. URL: `https://www.usenix.org/conference/usenixsecurity21/presentation/grassi`.

44 Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 674–704. Springer, 2020. `doi:10.1007/978-3-030-45724-2_23`.

45 IAIK TU Graz. Zk friendly hash zoo. `https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo`.

46 Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016. `doi:10.1007/978-3-662-49896-5_11`.

47 Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721 of *Lecture Notes in Computer Science*, pages 300–321. Springer, 2016. `doi:10.1007/978-3-319-40667-1_15`.

48 Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+flush: A fast and stealthy cache attack. In Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721 of *Lecture Notes in Computer Science*, pages 279–299. Springer, 2016. `doi:10.1007/978-3-319-40667-1_14`.

49 Ulrich Haböck. Multivariate Lookups Based on Logarithmic Derivatives. *IACR Cryptol. ePrint Arch.*, page 1530, 2022. URL: `https://eprint.iacr.org/2022/1530`.

50 Ulrich Haböck and Al Kindi. A note on adding zero-knowledge to STARKs. *IACR Cryptol. ePrint Arch.*, page 1037, 2024. URL: `https://eprint.iacr.org/2024/1037`.

51 Ulrich Haböck, David Levit, and Shahar Papini. Circle starks. *IACR Cryptol. ePrint Arch.*, page 278, 2024. URL: `https://eprint.iacr.org/2024/278`.

52 Hermez. Polygon hermez. `https://hermez.io/`.

53 Iden3. iden3/circomlibjs git. `https://github.com/iden3/circomlibjs`.

54 Iden3. iden3/ffjavascript git. `https://github.com/iden3/ffjavascript`.

55 Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 368–388. Springer, 2016. `doi:10.1007/978-3-662-53140-2_18`.

56 Ingonyama. Ingonyama. `https://www.ingonyama.com/`.

57 Ingonyama. ingonyama-zk/poseidon-hash git. `https://github.com/ingonyama-zk/poseidon-hash`.

58 Intel. Guidelines for mitigating timing side channels against cryptographic implementations. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html`.

59 Intel. Security best practices for side channel resistance. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/security-best-practices-side-channel-resistance.html`.

60 Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. `doi:10.1007/978-3-642-17373-8_11`.

61 Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: exploiting speculative execution. *Commun. ACM*, 63(7):93–101, 2020. `doi:10.1145/3399742`.

62 Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9.

63 Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. doi:10.1007/3-540-48405-1_25.

64 Lurk Lab. lurk-lab/neptune git. https://github.com/lurk-lab/neptune.

65 Light Protocol Labs. Light protocol. https://docs.lightprotocol.com/.

66 Matter Labs. Matter labs. https://matter-labs.io/.

67 Matter Labs. matter-labs/rescue-poseidon git. https://github.com/matter-labs/rescue-poseidon.

68 Mysten Labs. mystenlabs/sui git. https://github.com/MystenLabs/sui.

69 O1 Labs. o1-labs/proof-systems git. https://github.com/o1-labs/proof-systems.

70 Polygon Labs. 0xpolygonzero/plonky2 git. https://github.com/0xPolygonZero/plonky2.

71 Polygon Labs. Plonky3/plonky3 git. https://github.com/Plonky3/Plonky3.

72 Protocl Labs. Filecoin. https://filecoin.io/filecoin.pdf.

73 Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 2021. doi:10.1007/978-3-030-90453-1_1.

74 Libgcrypt. Libgcrypt. https://gnupg.org/software/libgcrypt/index.html.

75 Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

76 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

77 Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 605–622. IEEE Computer Society, 2015. doi:10.1109/SP.2015.43.

78 Loopring. Loopring. https://loopring.org/#/.

79 Lurk. Lurk. https://lurk-lang.org/.

80 Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2111–2128. ACM, 2019. doi:10.1145/3319535.3339817.

81 Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

82 Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999. doi:10.1007/3-540-48059-5_14.

83 mhostetter. galois git. https://github.com/mhostetter/galois.

84 Mina. Mina protocol. https://docs.minaprotocol.com/assets/technicalWhitepaper.pdf.

85 Ahmad Moghimi, Jan Wichelmann, Thomas Eisenbarth, and Berk Sunar. Memjam: A false dependency attack against constant-time crypto implementations. *Int. J. Parallel Program.*, 47(4):538–570, 2019. doi:10.1007/S10766-018-0611-9.

86 Monero. The monero project. https://www.getmonero.org/.

87 NSS. Nss. https://github.com/nss-dev/nss.

88 OpenJDK. openjdk/jdk git. https://github.com/openjdk/jdk/tree/master.

89 openSSL. openssl. https://www.openssl.org/.

90 Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006. doi:10.1007/11605805_1.

91 Panther. Panther protocol. https://www.pantherprotocol.io/.

92 Luke Pearson, Joshua Brian Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. *IACR Cryptol. ePrint Arch.*, page 86, 2022. URL: https://eprint.iacr.org/2022/086.

93 Polygon. Polygon miden technologies. https://polygon.technology/polygon-miden.

94 Polygon. Polygon technologies. https://polygon.technology/.

95 Light Protocol. Lightprotocol/light-poseidon git. https://github.com/Lightprotocol/light-poseidon.

96 Light Protocol. Lightprotocol/light-protocol git. https://github.com/Lightprotocol/light-protocol.

97 Panther Protocol. pantherprotocol/panther-core git. https://github.com/pantherprotocol/panther-core.

98 Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptogr.*, 4(2):15, 2020. doi:10.3390/CRYPTOGRAPHY4020015.

99 RISCZero. Risczero. https://www.risczero.com/.

100 Scroll. Scroll. https://scroll.io/.

101 Semaphore. Semaphore. https://semaphore.pse.dev/whitepaper-v1.pdf.

102 Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020. doi:10.1007/978-3-030-56877-1_25.

103 Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. Prime+probe 1, javascript 0: Overcoming browser-based side-channel defenses. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2863–2880. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/shusterman.

104 Sia. Sia. https://sia.tech/sia.pdf.

105 Florian Sieck, Zhiyuan Zhang, Sebastian Berndt, Chitchanok Chuengsatiansup, Thomas Eisenbarth, and Yuval Yarom. Teejam: Sub-cache-line leakages strike back. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):457–500, 2024. doi:10.46586/TCHES.V2024.I1.457-500.

106 Solana. Solana. https://solana.com/.

107 Sovrin. Sovrin. https://sovrin.org/.

108 Starkware. Starkware. https://starkware.co/.

109 Storj. Storj. https://www.storj.io/storjv3.pdf.

110 Sui. Sui. https://sui.io/zklogin.

111 Supranational. Supra national. https://www.supranational.net/.

112 Swarm. Swarm. `https://www.ethswarm.org/swarm-whitepaper.pdf`.

113 Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, and Bobbin Threadbare. The tip5 hash function for recursive starks. *IACR Cryptol. ePrint Arch.*, page 107, 2023. URL: `https://eprint.iacr.org/2023/107`.

114 Florian Tramèr, Dan Boneh, and Kenny Paterson. Remote side-channel attacks on anonymous transactions. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2739–2756. USENIX Association, 2020. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/tramer`.

115 Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptol.*, 23(1):37–71, 2010. `doi:10.1007/S00145-009-9049-Y`.

116 Tusima. Tusima. `https://www.tusima.network/`.

117 Stephan van Schaik, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Malicious management unit: Why stopping cache attacks in software is harder than you think. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 937–954. USENIX Association, 2018. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/van-schaik`.

118 Vimwitch. poseidon-lite git. `https://github.com/vimwitch/poseidon-lite`.

119 Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl. DATA - differential address trace analysis: Finding address-based side-channels in binaries. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 603–620. USENIX Association, 2018. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/weiser`.

120 Witnerfell. Winterfell git. `https://github.com/facebook/winterfell/tree/main`.

121 Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022. `doi:10.1007/978-3-031-15985-5_11`.

122 Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 719–732. USENIX Association, 2014. URL: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom`.

123 Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: A timing attack on openssl constant time RSA. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 346–367. Springer, 2016. `doi:10.1007/978-3-662-53140-2_17`.

124 Zcash. Zcash protocol specification. `https://zips.z.cash/protocol/protocol.pdf`.

125 ZK-kit. Zk-kit. `https://zkkit.pse.dev/`.

126 zk kit. zk-kit git. `https://github.com/privacy-scaling-explorations/zk-kit`.

127 ZKsync. Zksync. `https://zksync.io/`.