Generalized Indifferentiable Sponge and its Application to Polygon Miden VM

Tomer Ashur[1,3] and Amit Singh Bhati[2]

[1] 3MI Labs, Leuven, Belgium
[2] COSIC, KU Leuven, Belgium
[3] Polygon Research

Abstract. Cryptographic hash functions are said to be the work-horses of modern cryptography. One of the strongest approaches to assess a cryptographic hash function's security is indifferentiability. Informally, indifferentiability measures to what degree the function resembles a random oracle when instantiated with an ideal underlying primitive. However, proving the indifferentiability security of hash functions has been challenging due to complex simulator designs and proof arguments. The Sponge construction is one of the prevalent hashing method used in various systems. The Sponge has been shown to be indifferentiable from a random oracle when initialized with a random permutation.

In this work, we first introduce GSponge, a generalized form of the Sponge construction offering enhanced flexibility in input chaining, field sizes, and padding types. GSponge not only captures all existing sponge variants but also unveils new, efficient ones. The generic structure of GSponge facilitates the discovery of two micro-optimizations for already deployed sponges. Firstly, it allows a new padding rule based on zero-padding and domain-separated inputs, saving one full permutation call in certain cases without increasing the generation time of zero-knowledge proofs. Secondly, it allows to absorb up to $c/2$ more elements (that can save another permutation call for certain message lengths) without compromising the indifferentiability security. These optimizations enhance hashing time for practical use cases such as Merkle-tree hashing and short message processing.

We then propose a new efficient instantiation of GSponge called Sponge2 capturing these micro-optimizations and provide a formal indifferentiability proof to establish both Sponge2 and GSponge's security. This proof, simpler than the original for Sponges, offers clarity and ease of understanding for real-world practitioners. Additionally, it is demonstrated that GSponge can be safely instantiated with permutations defined over large prime fields, a result not previously formally proven.

Keywords: Sponge · GSponge · Sponge2 · Hashing Mode · Algebraic · Miden VM · Indifferentiability · Random Oracle · Generic Attacks

1 Introduction

Cryptographic hash functions play a fundamental role in modern cryptography. Hash functions are used for converting input data of varying lengths into fixed-size outputs, ensuring data integrity, authenticating transactions, and facilitating digital signatures. These cryptographic primitives are integral to a wide range of applications, including but not limited to data storage, authentication protocols, and secure communication channels. In recent decades, hash functions have also found specialized applications in algebraic hashing, a field essential for Blockchain scaling via L2 rollups (e.g., Polygon Miden).

E-mail: tomer@3milabs.tech (Tomer Ashur), amitsingh.bhati@esat.kuleuven.be (Amit Singh Bhati)

Ideal primitives commonly used in hash function construction are random functions, random permutations, or ideal ciphers. In practice, such primitives are expected to generate pseudorandom outputs for given inputs, ensuring unpredictability and resistance against cryptographic attacks. By combining these ideal primitives within specific modes, hash functions can achieve desired security properties such as collision resistance, preimage resistance, and second preimage resistance. The selection of appropriate primitives is crucial in designing hash functions that meet the stringent security requirements of modern cryptographic applications.

Hashing Mode Strategies. The choice of hashing mode is as critical as the selection of primitives, impacting both the security and efficiency of the resulting hash function. In cryptographic literature, the two mainly used hashing strategies are tree hashing (a.k.a. parallel hashing) and cascade hashing (a.k.a. sequential hashing).

Tree hashing techniques, exemplified by Merkle trees [Bec08] and ABR trees [ABR21], involve the parallel computation of hash values across multiple branches or levels of a hash tree. These methods offer efficient verification and authentication of large datasets, making them suitable for applications requiring scalable and parallel processing; however, at a cost of larger (length dependent) state to be maintained.

In contrast, cascade hashing strategies, such as Merkle-Damgård [Mer89, Dam89] (MD) style hashes and sponge [BDPVA07] based hashes, process input data sequentially, iteratively updating a relatively small internal state to produce the final hash value. MD-style hashes such as Wide Pipe [Luc05], Fast Wide Pipe [NP10], Chop-MD [CDMP05] and pf-MD[CDMP05] by design rely of random functions as primitives, whereas sponge-based hashes such as Sponge [BDPVA07], Overwrite Sponge [GLP08] and JH [Wu11] can incorporate both random functions and permutations. Due to their versatility and wide applicability, the sponge-based approach has garnered huge attention in recent years. This approach, notably employed in SHA-3 and algebraic hashing systems such as Polygon Miden [Mid23], offers inherent flexibility, particularly in accommodating algebraic primitives as most of them are permutations.

Sponge Hashing. At a fundamental level, Sponge-based constructions are characterized by a state of b bits, comprising a c-bit inner state, known as the capacity, and an r-bit outer state, referred to as the rate, where the total state size b equals the sum of c and r. Traditionally, in Sponge-like modes, data absorbing operations occur via the rate part, processing r bits at a time.

A notable sponge variant documented in the literature is the overwrite sponge mode [GLP08]. This variant improves the efficiency of regular sponge by dropping the top r bits of each permutation output instead of cascading them, resulting in r-bit smaller state among the permutation calls and removal of all XOR operations during absorption. Originally, (overwrite) sponge was proposed for primitives operating on bits, i.e., binary fields. However, recent works such as Rescue [SAD20], Poseidon [GKR+21] and XHash [ABKM23] highlights that the applicability of sponge construction can be extended to prime field setting when algebraic primitives are preferred or required. The rate and capacity of sponge under algebraic primitive is measured in field elements.

Handling Arbitrary Lengths. By design, sponge variants can only handle messages that has lengths multiple of the rate size. Hence, to handle messages of arbitrary lengths, a sponge-compliant padding rule is used to preprocess the message into a rate-aligned message, i.e., a message whose length is a multiple of the rate size. One of the simple and popular padding rules is the pad10$*$ padding [BDPVA07]. This padding simply appends the input with a 1 followed by the minimum number of 0s such that the length of the result is a multiple of the rate. We note that despite the simple definition, this padding adds another r-bit pad to even already rate-aligned messages, and hence imposes an extra primitive call to process them. Such drawback of a padding can affect the performance

significantly in applications where reducing even one permutation call is highly beneficial such as applications that process small size messages.

Security Analysis. Security analysis of hash functions involves a two-step approach: Firstly, demonstrating the generic security of the hashing mode, initialized with an ideal primitive, by proving desired security properties such as collision resistance or indifferentiability. Subsequently, instantiating the ideal primitive with a concrete function that goes through multiple cryptanalysis for validation.

Generic Attacks. In a generic attack, adversaries exploit vulnerabilities in a crypto-algorithm by assuming perfect behavior of its underlying primitives. For instance, consider a hash function $H : \mathbb{F}_p^* \rightarrow \mathbb{F}_p^n$ utilizing a random permutation $\mathcal{P} : \mathbb{F}_p^m \rightarrow \mathbb{F}_p^m$ for some positive integers m and n. This creates an opportunity for generic attacks, which exploit weaknesses in H with fewer resources compared to a larger random oracle $\mathcal{RO} : \mathbb{F}_p^* \rightarrow \mathbb{F}_p^n$. Generic attacks on hash functions are widely documented in cryptographic literature. Examples include Joux's multi-collision attack [Jou04], the Kelsey–Schneier expandable-message second pre-image attack [KS05], and the Kelsey–Kohno herding attack [KK06], all targeting the prevalent Merkle–Damgård construction. Additionally, various generic attacks like pre-image, second pre-image, collision, multi-collision, and herding attacks have been identified for numerous other hash functions [ABF+08, GK08, Jou04, BMN10, HS06, HK06], expanding beyond the basic Merkle–Damgård structure. These attacks typically assume the hash function's fundamental primitive behaves optimally, fulfilling the criteria of generic attacks. Consequently, the prevalence of such attacks significantly influenced the security assessment of cryptographic hash functions, emphasizing the necessity to develop hash modes capable of withstanding such threats.

Security by Indifferentiability. The concept of indifferentiability, introduced by Maurer et al. [MRH04] in 2004, and later employed by Coron et al. [CDMP05] in 2005, offers a means to assess the resilience of hash modes against generic attacks. This framework measures how closely a hash function resembles a random oracle when instantiated with an underlying primitive behaving in an ideal manner. Many cryptographic protocols employ random oracles and therefore depend on the indifferntiability security of the underlying hash function for engineering purposes. In fact, by contemporary best practices, indifferentiability has become a common prerequisite for hash mode adoption, given its effectiveness in safeguarding against generic attacks.

Despite being a crucial requirement, proving and validating the indifferntiability security of hash functions has not been an easy task. All indifferentiability proofs rely on the concept of a simulator, which is used to simulate the idealized primitive when the hash function is replaced with a true random oracle. Simulators are defined carefully to be both indistinguishable from the ideal primitive on the one hand, and consistent with the outputs of the random oracle on the other. Due to these correlated requirements, the description of simulators in indifferentiability proofs sometime becomes very contrived which makes both the simulator design and the proof arguments less intuitive and hard to follow. See [BDPVA08, AMP12, MPST16, CN08] for examples of simulator descriptions, indifferentiability proof approaches and proof sizes.

Further, it is also sometimes hard to directly adapt a given indifferentiability proof of a hashing mode to even minor variations due to the specifically defined simulator definition or dedicated bad case analysis.

1.1 Our Contribution

Our contribution in this work is twofold: (1) We generalize sponge mode into GSponge over input chaining type (capturing regular sponge and overwrite), field type (capturing binary and algebraic setting) and padding type (capturing all injective paddings). In

comparison with the regular sponge, GSponge provides extra rate of $\mathsf{r}_0 < \mathsf{c}$ elements for the first permutation call where c denotes its capacity. These elements can also be used for domain separation to directly achieve multi-rate and multi-protocol security.

We show that GSponge provides same security as any efficient instantiation of it. We then propose Sponge2, as an efficient instantiation of GSponge.

(2) We provide an intuitive yet formal proof of indifferentiability for Sponge2. To reduce the design complexity of the simulator and to improve the proof intuition, we introduce a new ideal object for sponges called capacity-collision-free random functions. Idealizing primitives as capacity-collision-free random functions naturally makes the indifferentiability proof simple and easily verifiable.

We note that by design, Sponge2 saves one full permutation call in cases where the unpadded message's length is already an integral multiple of the rate (e.g., in 2-to-1 Merkle-tree hashing) without increasing the proof generation time in zkVMs like Polygon Miden. Further, it also allows to absorb up to $\mathsf{c}/2$ more elements in the first permutation call, again resulting in saving a permutation call for some message lengths, without any loss in the security size, i.e., still $\approx \mathsf{c} \log_2 p/2$ bits.

2 Related Work

The Parazoa family, proposed by Andreeva et al. [AMP12], can also be seen as a generalization of the sponge construction. However, it is limited to injective padding functions and provides the same rate for all primitive calls. Consequently, it cannot accommodate interesting sponge instantiations like Sponge2, which works with zero padding (can be seen as non-injective) to save one full permutation call and provides an extra rate for the first primitive call.

Furthermore, the provable security results of Parazoa says nothing about its multi-rate and multi-protocol security. As mentioned earlier, the simulator descriptions and proof approaches of existing indifferentiability proofs are lengthy and difficult to follow, and this complexity is exacerbated by the generalization of Parazoa. For GSponge, we achieve multi-rate and multi-protocol security with a simple and intuitive proof.

Naito et al. improved the indifferentiability result of PHOTON's hashing mode in [NO14]. As a side result, they also claimed that the rate of the first permutation call in PHOTON's sponge-like hashing mode could be improved by half of the capacity size, although they omitted the proof, relying on their main indifferentiability result of PHOTON's hashing mode. While one may indeed be able to derive this proof for PHOTON's hashing mode in the binary setting, we have considered it for a generalized sponge that captures various sponge-like modes in both binary and algebraic settings.

Khovratovich et al. [KBM23] presented a hash indifferentiability result involving large prime fields, focusing on a dedicated sponge variant known as SAFECore. However, their proof is limited to the SAFECore design and includes specific restrictions. One such restriction is the necessity for an additional hash function, modeled as a random oracle, to process the capacity input of the first permutation call. These constraints make the proof non-generic and inapplicable to other popular sponge variants, such as the overwrite mode or the sponge mode with $\mathsf{c}/2$ extra rate, which injects $\mathsf{c}/2$ message elements 'directly' into the capacity part of the first permutation.

The security result in [KBM23] can be seen as a generalization of the security result in [NO14], with large prime field support but under the restriction that the extra rate of the first permutation is limited to outputs of a random oracle. Consequently, the indifferentiability of the regular sponge and other variants, including our proposed Sponge2 mode with zero padding, remained an open problem under large prime fields.

3 Paper Organization

We provide the preliminaries in Section 4. We then propose the sponge generalization GSponge and argue its security in Section 5. In Section 6, we provide Sponge2 as an efficient instantiation of GSponge and formally prove its security in Section 8. In Section 7, we discuss some concrete applications of Sponge2 in Miden VM. Finally, we conclude the paper in Section 9.

4 Preliminaries

4.1 Notation

Vectors. We let $\mathbb{F}_p$ to a denote a finite field of order $p = a^b$ with a a prime and b a positive integer. A vector S of size n is denoted by $S = (S[0], \ldots, S[n-1])$. We use $\oplus$ is used to denote addition over the finite field $\mathbb{F}_p$. For simplicity, we refer to a vector of $n > 0$ many $\mathbb{F}_p$ elements as an (n, p)-vector. Note that in particular $(n, 2)$-vectors are analogous to n-bit binary strings. We denote the set of all (n, p)-vectors by $\mathbb{F}_p^n$. The set of vectors of any possible length i.e., arbitrary n is denoted by $\mathbb{F}_p^*$. The set of all permutations of $\mathbb{F}_p^n$ is denoted by $\mathrm{Perm}_p(n)$ and the set of all functions/maps from $\mathbb{F}_p^m$ (respectively, $\mathbb{F}_p^*$) to $\mathbb{F}_p^n$ is denoted by $\mathrm{Func}_p(m, n)$ (respectively, $\mathrm{Func}_p(*, n)$). For any (n, p)-vector A, $|A|$ is the length of A in elements i.e., $|A| = n$. For any two sets A and B, their Cartesian product is defined as $A \times B = \{(i, j) \mid i \in A, j \in B\}$ and the term $A \backslash B$ denotes the largest subset of A that shares no element with B.

Partitions. Given a vector A and an integer $n > 0$ such that $|A| = an + d$, where a is a positive integer and $0 < d \leq n$, the notation $A_1, A_2, \ldots, A_{a+1} \xleftarrow{n} A$ is used to indicate the partitioning of A into a maximum number of (n, p)-vectors a.k.a. block vectors. Each block vector A_i has a length of n for $1 \leq i \leq a$, and the last block vector A_{a+1} has a length of d. If $d = n$, we say A is n-aligned. Similarly, we also use $[A_1, A_2, \ldots, A_{a+1}]$ to denote the ordered union a.k.a. concatenation of these vectors i.e., A.

Miscellaneous. We use $\langle i \rangle$ to denote a bijective encoding of $0 \leq i < p$ in $\mathbb{F}_p$. The notation $x \xleftarrow{\$} \mathcal{X}$ indicates the random sampling of an element x from a finite set $\mathcal{X}$ with a uniform distribution. The symbol $\perp$ is used to represent an undefined value or an error.

4.2 Statistical Distance

The statistical distance between two random variables (or distributions) is defined as follows

Definition 1 (Statistical Distance)**.** Let X and Y be two random variables taking values from a finite set $\mathcal{X}$. The statistical distance between X and Y is defined as

$$\mathsf{SD}(X, Y) = \frac{1}{2} \sum_{x \in \mathcal{X}} |\Pr[X = x] - \Pr[Y = x]| .$$

4.3 Indifferentiability

We now recall the main indifferentiability theorem, due to Maurer et al. [MRH04] for hash functions under the random permutation model. If a hash function H, based on a public permutation $\mathcal{P}$ (with inverse $\mathcal{P}^{-1}$) is indifferentiable from a random oracle $\mathcal{RO}$, then a cryptosystem $\mathcal{C}$ based on $\mathcal{RO}$ (in the random oracle model) is at least as secure as $\mathcal{C}$ based on H (in the random permutation model). This highlights the importance

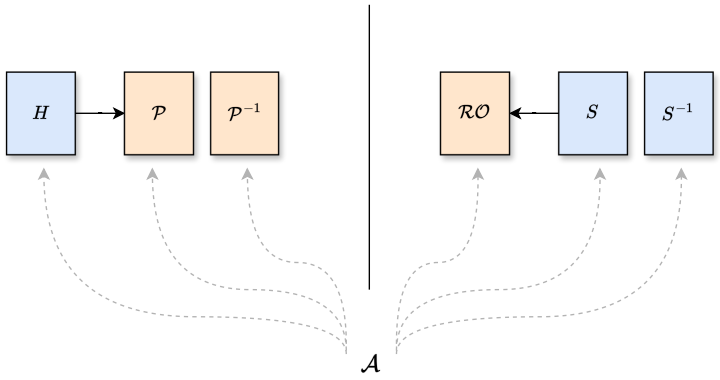


Figure 1: The indifferentiability notion

of indifferentiability in ensuring the security of cryptographic protocols that use hash functions to instantiate random oracles.

We now generalize and formally define the indifferentiability notion as of [CDMP05] for hash functions that use random public permutations (or any other ideal public primitives with both forward and backward oracles) and are defined over $\mathbb{F}_p$ with arbitrary prime power p.

Definition 2 (Indifferentiability from an RO)**.** Let $H : \mathbb{F}_p^* \to \mathbb{F}_p^n$ for some integer $n > 0$ and prime power p be a hash function that internally uses a random permutation $\mathcal{P} : \mathbb{F}_p^m \to \mathbb{F}_p^m$ for some integer $m > 0$ and let $\mathcal{RO} : \mathbb{F}_p^* \to \mathbb{F}_p^n$ be a random oracle. Let $\mathcal{A}$ be a computationally unbounded adversary $\mathcal{A}$ with triple oracle access - the hash function, its underlying primitive and primitive's inverse. The advantage of $\mathcal{A}$ against the indifferentiability of H from $\mathcal{RO}$ is defined as

$$\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{H[\mathcal{P}]}(\mathcal{A}) = \min_{S} |\Pr[\mathcal{A}^{H,\mathcal{P},\mathcal{P}^{-1}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathcal{RO},S,S^{-1}} \Rightarrow 1]|$$

where $S : \mathbb{F}_p^m \to \mathbb{F}_P^m$ is a simulator algorithm simulating $\mathcal{P}$ for $\mathcal{RO}$.

The simulator S has oracle access to $\mathcal{RO}$, but it cannot directly observe past queries made to $\mathcal{RO}$ by $\mathcal{A}$ (refer to Fig. 1 for a visual representation of the indifferentiability concept).

5 GSponge: A Generalized Sponge Mode

In this section, we provide a generalization of the popular sponge [BDPVA07] mode to capture all existing sponge variants such as overwrite sponge, prepadded sponge, domain-separated sponge, etc. We then argue its security by its indifferentiability from a random oracle (RO).

$\mathsf{GSponge}_{u,\mathsf{r}_0}[\pi] : \mathbb{F}_p^* \to \mathbb{F}_p^{\mathsf{r}}$ is based on a permutation $\pi : \mathbb{F}_p^{\mathsf{b}} \to \mathbb{F}_p^{\mathsf{b}}$ with size $\mathsf{b} = \mathsf{r} + \mathsf{c}$ for some positive integers r and c called the *rate* and *capacity* of π, respectively and an injective padding function $\mathsf{pad}_{\mathsf{r},\mathsf{r}_0} : \mathbb{F}_p^* \to \mathbb{F}_p^*$ that for a given $\mathsf{r}_0 < \mathsf{c}$ maps any arbitrary length vector M to a unique $a\mathsf{r}+\mathsf{r}_0+1$-length vector $[x, M, y]$ for some $a \geq 1$ and non-empty x (a pre-pad) and y (a post-pad). An input $M \in \mathbb{F}_p^*$ to $\mathsf{GSponge}$ is first padded to $P = \mathsf{pad}_{\mathsf{r},\mathsf{r}_0}(M)$ and then processed using π as shown in Fig. 2. Here $u \in \{0, 1\}$ that decides if the rate output of a permutation is chained to the next rate input or not.

Deriving Popular Sponge Variants. Recall that $\mathsf{GSponge}$ is parameterized with u and r_0. When $u = 1$, $\mathsf{r}_0 = 0$, $p = 2^n$ for some positive integer n and $\mathsf{pad}_{\mathsf{r},0}(M) =$

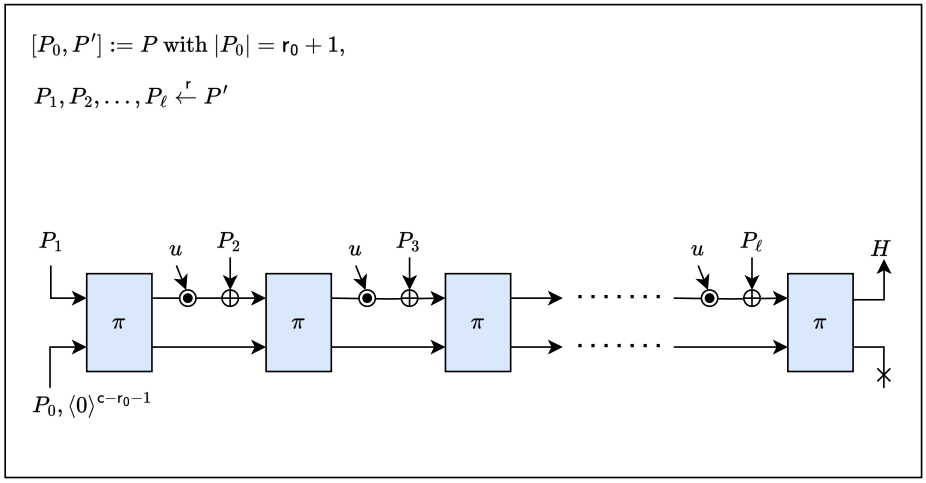


Figure 2: GSponge (block diagram). Here $\odot$ represents field multiplication with $u \in \{0, 1\}$ which returns zero or the input itself, depending on the value of u. The upper input (or output) part of π corresponds to its rate whereas the lower input (or output) part corresponds to its capacity.

$[\langle 0 \rangle, M, \langle 1 \rangle, \langle 0 \rangle^{\mathsf{r}-1-(|M| \mod \mathsf{r})}]$, we get the original sponge mode [BDPVA07]. Further, when under the same setting u is replaced to 0, we get the original overwrite mode [GLP08]. Finally, when p is not fixed to 2^n, we get sponge mode and overwrite mode under prime a.k.a. algebraic setting as used in [SAD20, GKR+21].

We highlight that the formal security by indifferentiability of the original sponge mode is proven in [BDPVA08], however, to the best of our knowledge, the other popular variants as derived above do not have formal proofs in literature. We also note that a formal proof for GSponge will cover the formal security of these variants as well as other possible instantiations of GSponge.

5.1 Security of GSponge

We target the security of GSponge as its indifferentiability from an RO. We highlight that indifferentiability under sufficiently large digest size implicitly provides basic cryptographic hash security properties such as (second) pre-image resistance and collision resistance.

We note that the input space of GSponge is invariant of u as every unique input under $u = 0$ matches with some unique input under $u = 1$. This implies that the outputs of GSponge are differently ordered a.k.a. permuted for different values of u, however, with the same output multiset and thus same output distribution.

Thus, the indifferentiability of GSponge is independent of u and therefore, it is sufficient to argue its indifferentiability with $u = 0$, i.e., as overwrite GSponge mode.

Further, we also note that the indifferentiability of $\mathsf{GSponge}_{0,\mathsf{r}_0}$ remains same over all choices of the injective paddings. This holds because injective paddings are bijective to each other and when modeled as an RO, the statistical distance between the output distributions of $\mathsf{GSponge}_{0,\mathsf{r}_0}$ under two different injective paddings is zero. Therefore, the indifferentiability of $\mathsf{GSponge}_{0,\mathsf{r}_0}$ can be argued by proving the same for any efficient instantiation of it. In the next section, we provide such an efficient instantiation of GSponge and formally prove its security. We emphasize that this identical output distribution argument only works when $\mathsf{GSponge}_{0,\mathsf{r}_0}$ (for any injective padding) is shown indifferentiable from an RO.

6 Sponge2: An Efficient GSponge Instantiation

In this section, we provide a new sponge variant called Sponge2 as an efficient instantiation of GSponge. $\mathsf{Sponge2} : \mathbb{F}_p^* \to \mathbb{F}_p^{\mathsf{r}}$ is based on a permutation $\pi : \mathbb{F}_p^{\mathsf{b}} \to \mathbb{F}_p^{\mathsf{b}}$ with size $\mathsf{b} = \mathsf{r} + \mathsf{c}$ for some positive integers r and c called the rate and capacity of π, respectively. An input $M \in \mathbb{F}_p^*$ to Sponge2 is processed using π as shown in Fig. 3 (b).

Domain Separation in Sponge2. For a message M of length $0 < |M| \leq \mathsf{r}_0$, the domain separator is defined as $i = \mathsf{r} + \mathsf{r}_0 - |M|$ and otherwise, it is defined as $i = (\mathsf{r} - ((|M| - \mathsf{r}_0) \bmod \mathsf{r}))) \bmod \mathsf{r}$. It is easy to see that i is a unique value that varies from 0 to $\mathsf{r} + \mathsf{r}_0 - 1$ for messages with different last block lengths and is same as the number of zero elements padded after the message to make it rate-aligned i.e., the number of zeros used in $\langle 0 \rangle^*$ defines i for every message.

Sponge2 differs from the popular overwrite-style sponge hash in the padding rule and the underlying first primitive's input as shown in Fig. 3. This difference allows Sponge2 to 1) process $\mathsf{r}_0 < \mathsf{c}$ many more $\mathbb{F}_p$ elements and to 2) additionally reduce the overall cost by one π call than the regular overwrite sponge mode. More specifically, unlike Sponge, no extra π call is required at the end to finalize rate-aligned messages. This gain could be reasonably large for applications that work with stringent resource constraints and/or typically hash small size messages. The targeted application of Miden VM in this work fits to both of these categories which makes Sponge2 a viable choice for it.

6.1 Security of Sponge2

We state the formal claim about the indifferentiability (from an RO) of Sponge2 in Theorem 1.

Theorem 1. *Let* $\mathsf{Sponge2}[\pi]$ *be the hash function as defined above in Fig. 3 with* $\pi \xleftarrow{\$} \mathit{Perm}_p(\mathsf{b})$ *and let* $\mathsf{r}_0 < \mathsf{c}$ *be some fixed integer. Then for any adversary* $\mathcal{A}$ *who makes at most* q_P π *queries, we have*

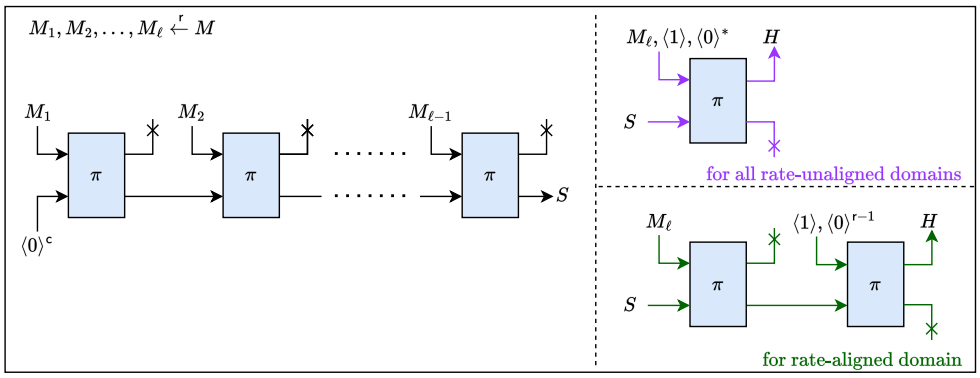
$$\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}[\pi]}(\mathcal{A}) \leq \frac{q_P(q_P-1)}{2p^{\mathsf{b}}} + \frac{q_P}{p^{\mathsf{c}-\mathsf{r}_0}}\left(1 + \frac{1}{p-1}\right) + \frac{q_P^2}{p^{\mathsf{c}}}\left(1 + \frac{p^{-\mathsf{c}+\mathsf{r}_0+1}}{p-1-p^{-\mathsf{c}+\mathsf{r}_0+1}}\right).$$

We defer the proof of Theorem 1 to Section 8. We now define a corollary of Theorem 1 that provides the indifferentiability bound for Sponge2 with fields of odd characteristic i.e., when $p > 2$ and when r_0 is fixed to $\mathsf{c}/2$. In simple words, Corollary 1's bound states that in the AO context, Sponge2 provides at least $(\mathsf{c} \cdot \log_2 p - 4)/2$ bits of security. This results in a concrete security of at least ≈ 126 bits when $p \approx 2^{64}$ and $\mathsf{c} = 4$ i.e., π has 4 $\mathbb{F}_p$ elements as the capacity.

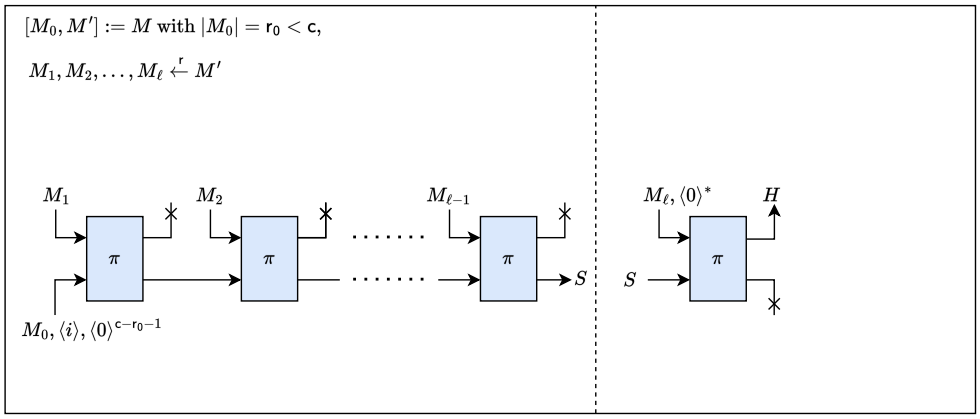
Corollary 1. *Let* $\mathsf{Sponge2}[\pi]$ *be the hash function as defined above in Fig. 3 with* $\pi \xleftarrow{\$} \mathit{Perm}_p(\mathsf{b})$ *and let* $\mathsf{r}_0 = \mathsf{c}/2$*, and* $p > 2$*. Then for any adversary* $\mathcal{A}$ *who makes at most* q_P π *queries, we have*

$$\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}[\pi]}(\mathcal{A}) \leq \frac{3q_P}{p^{c/2}}.$$

Proof of Corollary 1. The proof of Corollary 1 follows from the result of Theorem 1. More



(a) Overwrite Sponge Hashing Mode.



(b) Sponge2 Hashing Mode.

Figure 3: Overwrite Sponge vs Sponge2 (block diagrams). Here $\langle 0 \rangle^*$ represents a zero vector with sufficient number of $\langle 0 \rangle$s to fill the corresponding input of π. In Sponge2, the domain separator i is defined by the number of zeros used in the corresponding $\langle 0 \rangle^*$.

specifically, with $r_0 = c/2$, $b \geq c$, $p > 2$, $q_P \leq p^{c/2}/3$ and Theorem 1, we get

$$\begin{aligned}
\mathbf{Adv}^{\text{ro-indiff}}_{\text{Sponge2}[\pi]}(\mathcal{A}) &\leq \frac{q_P^2}{2p^c} + \frac{q_P}{p^{c/2}}\left(1 + \frac{1}{p-1}\right) + \frac{q_P^2}{p^c}\left(1 + \frac{1}{p-2}\right) \\
&\leq \frac{q_P}{p^{c/2}}\left(\frac{1}{6}\right) + \frac{q_P}{p^{c/2}}\left(\frac{3}{2}\right) + \frac{q_P}{p^{c/2}}\left(\frac{2}{3}\right) \\
&\leq \frac{3q_P}{p^{c/2}}\,.
\end{aligned}$$

Finally, we drop the assumption $q_P \leq p^{c/2}/3$ as for $q_P > p^{c/2}/3$, this bound becomes void anyways. This completes the proof of Corollary 1. □

6.2 Sponge2 under Multi-rate and Multi-protocol Setting

In real-world applications, the same hash function is sometimes used under the same input space but with different rate sizes or different protocols. In such situations, we need domain separations to ensure the security of the hash function across these use cases.

Sponge2 provides security for multi-rate and multi-protocol applications by simply adding a domain separator in the r_0 elements of the first call. This domain separator can be seen as an identifier of the used rate size or protocol. The security of Sponge2 (in bits) under multi-rate and multi-protocol applications is upper bounded by the security of Sponge2 (in bits) under the maximum allowed rate r as captured in Theorem 1.

7 Applications of Sponge2 in Polygon Miden VM

In this section, we present some direct applications of Sponge2 in Polygon Miden VM and discuss the specific improvements. It is worth noting that similar applications of Sponge2 can also be shown for other STARK-based zkVMs.

Miden VM currently uses Rescue-Prime Optimized (RPO) as the underlying hash function. RPO is a sponge function instantiated with a permutation over $\mathbb{F}_p^{\mathsf{b}}$, where p is a 64-bit prime and $\mathsf{b} = \mathsf{r} + \mathsf{c}$, with $\mathsf{r} = 2\mathsf{c} = 8$. In Miden VM, RPO is used in various contexts and with different types of inputs. We categorize these hash calls into three main types based on their input:

1. 2-to-1 hashing with metadata
2. Hashing for leaf computation
3. Variable-input-length (VIL) hashing.

2-to-1 Hashing with Metadata. Miden VM currently performs 2-to-1 compression calls in many places with input sizes as small as 8 $\mathbb{F}_p$ elements. However, these inputs are accompanied by additional metadata, such as left-or-right hashing type information, increasing the total input size to 9 elements. RPO (a vanilla sponge with a rate of 8 elements per call) requires two permutation calls to hash these 9 elements, whereas Sponge2 (which processes up to 10 elements in the first call) requires only one permutation call. This results in a 50% improvement in the hashing cost.

Hashing for Leaf Computation. Currently, in Miden VM, each leaf of the Merkle-tree is derived from hashing a sequence of 72 field elements. Miden plans to shift to 64 columns in the Merkle tree, reducing this cost to a sequential hash of 64 elements. Using the sponge, we will need 9 permutation calls to process these elements, whereas Sponge2 can achieve this in 8 permutation calls, resulting in a 12.5% improvement in the rate.

VIL Hashing. Apart from the aforementioned hash applications, Miden also uses the same RPO instantiation to hash different length inputs. We know that for any message M, its length can be expressed as $\mathsf{r}\alpha_M + i_M$ for some non-negative integer α_M and $0 \leq i_M \leq \mathsf{r} - 1$. For all messages M' with $0 \leq i_{M'} \leq \mathsf{r}_0 - \mathsf{r}$ and $\alpha_{M'} > 0$, Sponge2 requires one less permutation call than the sponge (due to our two micro optimizations), resulting in a $100/\alpha_{M'}\%$ improvement in the rate.

Notably, for a uniformly sampled message (from the space of all messages with size at least 8 elements) there is a $100(\mathsf{r}_0 - \mathsf{r} + 1)/\mathsf{r}\%$ chance of encountering a message M' with $0 \leq i_{M'} \leq \mathsf{r}_0 - \mathsf{r}$. For Miden VM with $\mathsf{r}_0 = \mathsf{r} + \mathsf{c}/2 = 10$, it is 37.5%.

8 Security Analysis

In this section, we provide the deferred proof of Theorem 1.

Proof of Theorem 1. **Replacing π.** We treat duplicate queries and cross-oracle known response queries (i.e., querying the π^{-1} oracle with an output of previously queried π

or vice versa) as *trivial* queries and the rest as *non-trivial* queries. We note that trivial queries cannot help $\mathcal{A}$ in increasing its advantage as their output is already known and thus independent of the queried oracle. Hence, we can assume w.l.o.g., that $\mathcal{A}$ only makes non-trivial queries.

We now recall that as per the standard RP-RF switching lemma [BR06], a randomly sampled (π, π^{-1}) with $\pi \xleftarrow{\$} \mathrm{Perm}_p(\mathsf{b})$ is indistinguishable up to the birthday bound (in the output size) from a randomly sampled function pair (f_1, f_2) for non-trivial oracle queries where $(f_1, f_2) \xleftarrow{\$} \mathrm{Func}_p(\mathsf{b}, \mathsf{b}) \times \mathrm{Func}_p(\mathsf{b}, \mathsf{b})$.

More formally, for any adversary $\mathcal{A}$ that makes at most q_P many non-trivial π queries (to both forward and inverse oracles in total), we have that

$$\left| \Pr[\mathcal{A}^{\pi,\pi^{-1}} \Rightarrow 1] - \Pr[\mathcal{A}^{f_1,f_2} \Rightarrow 1] \right| \leq \frac{q_P(q_P-1)}{2p^{\mathsf{b}}} .$$

Let us denote $\mathsf{Sponge2}[\pi]$ by $\mathsf{Sponge2}'[\pi, \pi^{-1}]$. Then, with the above inequality we get

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}[\pi]}(\mathcal{A}) &= \mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[\pi,\pi^{-1}]}(\mathcal{A}) \\ &\leq \mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[f_1,f_2]}(\mathcal{A}) + \frac{q_P(q_P-1)}{2p^{\mathsf{b}}} . \end{aligned} \tag{1}$$

Blacklisting Outputs for f_1 and f_2. For a smooth indifferentiability proof, we will later need to restrict the primitive to not return a particular form of outputs, hence we hereby update the primitive to easily achieve this restriction later. Let $\mathcal{L}$ be a set of restricted outputs in $\mathbb{F}_p^{\mathsf{b}}$ with size $|\mathcal{L}| = \Delta$ and let g_1 and g_2 be uniform random functions from $\mathbb{F}_p^{\mathsf{b}}$ to $\mathbb{F}_p^{\mathsf{b}} \backslash \mathcal{L}$. Observing that the statistical distance between f_1 and g_1 (similarly, between f_2 and g_2) is Δ/p^{b} and since we evaluate the primitives f_1 and f_2 in total q_P many times, we get

$$\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[f_1,f_2]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[g_1,g_2]}(\mathcal{A}) + \frac{q_P\Delta}{p^{\mathsf{b}}} . \tag{2}$$

Making g_1 and g_2 Capacity-collision-free. Now, since g_1 and g_2 are uniform random functions over $\mathbb{F}_p^{\mathsf{b}} \backslash \mathcal{L}$, they can still have random collisions in the last c elements a.k.a. the capacity or the inner part. Such collisions are undesirable as they can be used to construct hash outputs for not-yet-queried inputs and thus make the hash differentiable from an RO. We therefore now replace the primitive with a capacity-collision-free variant.

We also note that for every input of the form $[x_1, \ldots, x_{\mathsf{r}}, x_{\mathsf{r}+1}, \ldots, x_{\mathsf{r}+\mathsf{c}}]$ with $x_i \in \mathbb{F}_p$, the outputs of g_1 and g_2 functions are sampled uniformly at random from $(\mathbb{F}_p^{\mathsf{r}} \times \mathbb{F}_p^{\mathsf{c}}) \backslash \mathcal{L}$ with form $[y_1, \ldots, y_{\mathsf{r}}, y_{\mathsf{r}+1}, \ldots, y_{\mathsf{r}+\mathsf{c}}]$ where $y_i \in \mathbb{F}_p$.

We now define a new class of functions called *capacity-collision-free* or CCF. CCF functions are defined in pairs.

Definition 3. A CCF pair $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ is a pair of $\mathbb{F}_p^{\mathsf{r}} \times \mathbb{F}_p^{\mathsf{c}} \to (\mathbb{F}_p^{\mathsf{r}} \times \mathbb{F}_p^{\mathsf{c}}) \backslash \mathcal{L}$ functions. Each of these functions takes $\mathsf{r} + \mathsf{c}$ elements in $\mathbb{F}_p$ as input and maps them to $\mathsf{r} + \mathsf{c}$ elements in $\mathbb{F}_p$ with the added restriction that the capacity i.e., the last c elements of any output never collides/matches with the capacity of any previously queried input or output of both of the CCF functions.

One can notice that a CCF pair (i.e., both functions in total) cannot be exhausted in fewer than $p^{\mathsf{c}}/2$ queries and is necessarily exhausted after at most $p^{\mathsf{c}} - 1$ queries as by then all $\mathbb{F}_p^{\mathsf{c}}$ many capacity values are exhausted. Further, a CCF pair can be constructed plainly by keeping a mapping table that requires a memory of at least $p^{\mathsf{c}}/2$ (when all queried

inputs and outputs have different capacities) to at most $p^{\mathsf{c}} - 1$ (when only the queried outputs have different capacities) input-output pairs.

A randomly sampled CCF pair is defined as a CCF pair where for every input query (to either of the two functions) the output is sampled uniformly at random from the remaining space of possible outputs. We replace (g_1, g_2) with a random CCF pair $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ and get

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[g_1,g_2]}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}'[f_1^{\mathsf{ccf}},f_2^{\mathsf{ccf}}]}(\mathcal{A}) \\ & + \left|\Pr[\mathcal{A}^{f_1^{\mathsf{ccf}},f_2^{\mathsf{ccf}}} \Rightarrow 1] - \Pr[\mathcal{A}^{g_1,g_2} \Rightarrow 1]\right|. \end{aligned} \tag{3}$$

Note that the above probability difference is specific to the adversary $\mathcal{A}$ and its strategy. We know that this difference can never be higher than the actual statistical distance between the input-output distributions of the q_P queries to $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ and (g_1, g_2).

Let $\mathcal{A}$ be a query bounded adversary making q_1 many queries to the first oracle and q_2 many queries to the second such that $q_1 + q_2 = q_P$. Let $\Theta_f = (\Theta_X, \Theta_{Y,f}) = (\{X_i\}_{i=1}^{q_P}, \{Y_i^f\}_{i=1}^{q_P})$ and $\Theta_g = (\Theta_X, \Theta_{Y,g}) = (\{X_i\}_{i=1}^{q_P}, \{Y_i^g\}_{i=1}^{q_P})$ be the random variables for the input-output distributions of the q_P queries with any adversarial choice of oracle order to $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ and (g_1, g_2), respectively. Here X_i and Y_i represent the input and output of the i^{th} query to the corresponding oracle, respectively. Θ_X takes values from $\mathcal{S}_X = (\mathbb{F}_p^{\mathsf{b}})^{q_P}$ whereas $\Theta_{Y,f}$ and $\Theta_{Y,g}$ take values from $\mathcal{S}_Y = (\mathbb{F}_p^{\mathsf{b}} \backslash \mathcal{L})^{q_P}$. Let $\mathcal{S} = \mathcal{S}_X \times \mathcal{S}_Y$ and $\mathcal{S}_{\mathsf{ccf}} \subseteq \mathcal{S}$ be the set of all possible q_P input-output tuples for oracles $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ i.e., the set of all possible capacity-collision-free query-response tuples. We get $\left|\Pr[\mathcal{A}^{f_1^{\mathsf{ccf}},f_2^{\mathsf{ccf}}} \Rightarrow 1] - \Pr[\mathcal{A}^{g_1,g_2} \Rightarrow 1]\right|$

$$\begin{aligned} &\leq \mathsf{SD}(\Theta_f, \Theta_g) = \frac{1}{2}\sum_{\theta \in \mathcal{S}} |\Pr[\Theta_f = \theta] - \Pr[\Theta_g = \theta]| \\ &= \frac{1}{2}\sum_{\theta \in \mathcal{S}_{\mathsf{ccf}}} |\Pr[\Theta_f = \theta] - \Pr[\Theta_g = \theta]| \\ &\qquad + \frac{1}{2}\sum_{\theta \in \mathcal{S} \backslash \mathcal{S}_{\mathsf{ccf}}} |0 - \Pr[\Theta_g = \theta]|. \end{aligned}$$

Here the last equality holds as by definition, Θ_f only returns outputs from $\mathcal{S}_{\mathsf{ccf}}$. Now, since by design $\mathcal{A}$ does not make trivial queries we know that the q_P outputs of g_1 and g_2 are each sampled uniformly and independently at random from $\mathbb{F}_p^{\mathsf{b}} \backslash \mathcal{L}$ and hence for $\theta = (\theta_X, \theta_Y)$, we get $\Pr[\Theta_g = \theta] = \Pr[\Theta_X = \theta_X \wedge \Theta_{Y,g} = \theta_Y] = \Pr[\Theta_X = \theta_X]/|\mathcal{S}_Y|$. Similarly, for a given query-tuple θ_X, we also know that the corresponding q_P response-tuple θ_Y w.r.t. f_1^{ccf} and f_2^{ccf} is sampled uniformly at random from the set of all possible capacity-collision-free response tuples that correspond to the query tuple θ_X, i.e., from the set $\mathcal{S}_{\mathsf{ccf},Y}^{\theta_X} \subseteq \mathcal{S}_Y$ defined as $\{j \mid (i,j) \in \mathcal{S}_{\mathsf{ccf}} \text{ and } i = \theta_X\}$ which implies that $\Pr[\Theta_f = \theta] = \Pr[\Theta_X = \theta_X] \cdot \Pr[\Theta_{Y,f} = \theta_Y \mid \Theta_X = \theta_X] = \Pr[\Theta_X = \theta_X]/|\mathcal{S}_{\mathsf{ccf},Y}^{\theta_X}|$. This gives us

$$
\begin{aligned}
&\frac{1}{2}\sum_{\theta\in\mathcal{S}_{\mathsf{ccf}}}|\Pr[\Theta_f=\theta]-\Pr[\Theta_g=\theta]|+\frac{1}{2}\sum_{\theta\in\mathcal{S}\setminus\mathcal{S}_{\mathsf{ccf}}}|0-\Pr[\Theta_g=\theta]|\\
&=\frac{1}{2}\Big(\sum_{\theta\in\mathcal{S}_{\mathsf{ccf}}}\Pr[\Theta_X=\theta_X]\cdot\Big(\frac{1}{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}-\frac{1}{|\mathcal{S}_Y|}\Big)\\
&\qquad+\sum_{\theta\in\mathcal{S}\setminus\mathcal{S}_{\mathsf{ccf}}}\frac{\Pr[\Theta_X=\theta_X]}{|\mathcal{S}_Y|}\Big)\\
&=\frac{1}{2}\sum_{\theta_X\in\mathcal{S}_X}\Pr[\Theta_X=\theta_X]\cdot\Big(\sum_{\theta_Y\in\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}}\Big(\frac{1}{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}-\frac{1}{|\mathcal{S}_Y|}\Big)\\
&\qquad+\sum_{\theta_Y\in\mathcal{S}_Y\setminus\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}}\frac{1}{|\mathcal{S}_Y|}\Big)\\
&=\frac{1}{2}\sum_{\theta_X\in\mathcal{S}_X}\Pr[\Theta_X=\theta_X]\cdot\Big(|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|\Big(\frac{1}{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}-\frac{1}{|\mathcal{S}_Y|}\Big)\\
&\qquad+\Big(|\mathcal{S}_Y|-|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|\Big)\frac{1}{|\mathcal{S}_Y|}\Big)\\
&=\sum_{\theta_X\in\mathcal{S}_X}\Pr[\Theta_X=\theta_X]\cdot\Big(1-\frac{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}{|\mathcal{S}_Y|}\Big)\\
&\le\max_{\theta_X\in\mathcal{S}_X}\Big(1-\frac{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}{|\mathcal{S}_Y|}\Big)\sum_{\theta_X\in\mathcal{S}_X}\Pr[\Theta_X=\theta_X]
\end{aligned}
$$

$$
=\max_{\theta_X\in\mathcal{S}_X}\Big(1-\frac{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|}{|\mathcal{S}_Y|}\Big)\,. \tag{4}
$$

Let $Q=|\mathbb{F}_p^{\mathsf{b}}\backslash\mathcal{L}|=p^{\mathsf{b}}-\Delta$ and therefore, $|\mathcal{S}_Y|=Q^{q_P}$. Further, since $\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}$ is a set of capacity-collision-free response tuples with each tuple containing q_P many elements from $\mathbb{F}_p^{\mathsf{b}}\backslash\mathcal{L}$, we have $\min_{\theta_X\in\mathcal{S}_X}\{|\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}|\}=(Q-1\cdot p^{\mathsf{r}})(Q-3\cdot p^{\mathsf{r}})(Q-5\cdot p^{\mathsf{r}})\cdots(Q-(2q_P-1)\cdot p^{\mathsf{r}})$. This bound comes from the case when the capacity part of every input and output of q_P queries is unique which reduces $\mathcal{S}^{\theta_X}_{\mathsf{ccf},Y}$ to its minimum size. Combining these results with Exp. 4, we get

$$
\begin{aligned}
\mathsf{SD}(\Theta_f,\Theta_g)&\le 1-\frac{\prod_{i=1}^{q_P}(Q-(2i-1)\cdot p^{\mathsf{r}})}{Q^{q_P}}\\
&=1-\prod_{i=1}^{q_P}\Big(1-\frac{2i-1}{Qp^{-\mathsf{r}}}\Big)\\
&\le 1-\Big(1-\sum_{i=1}^{q_P}\frac{2i-1}{Qp^{-\mathsf{r}}}\Big)=\frac{q_P^2}{Qp^{-\mathsf{r}}}\,.
\end{aligned} \tag{5}
$$

Here the last inequality holds from the following observation: For two positive numbers a_1 and a_2,

$$
\prod_{i=1}^{2}(1-a_i)=(1-a_1)(1-a_2)=1-a_1-a_2+a_1a_2\ge 1-\sum_{i=1}^{2}a_i\,.
$$

Now, combining Exp. 1, 2, 3 and 5 together gives us for $q_P\le p^{\mathsf{c}}/2$

$$\mathbf{Adv}^{\text{ro-indiff}}_{\mathsf{Sponge2}[\pi]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ro-indiff}}_{\mathsf{Sponge2'}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]}(\mathcal{A}) + \frac{q_P(q_P-1)}{2p^{\mathsf{b}}} + \frac{q_P\Delta}{p^{\mathsf{b}}} + \frac{q_P^2 p^{\mathsf{r}}}{p^{\mathsf{b}}-\Delta}\,. \tag{6}$$

We are now left with bounding the term $\mathbf{Adv}^{\text{ro-indiff}}_{\mathsf{Sponge2'}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]}(\mathcal{A})$. We note that the above inequality holds for any value of Δ; however, we are only interested in the smallest value of Δ for which the above upper bound minimizes.

Reordering Input Space. We recall that in $\mathsf{Sponge2'}$, the inputs are post-padded with minimum number of zeros to make the input rate aligned as shown in Fig. 3 (b). We define a new variant of $\mathsf{Sponge2'}$ named $\mathsf{Sponge2}^{\dagger}$ where the padded zeros are pulled to the start of the message (see Fig. 4). For example, a message $M = [x_1, x_2, \ldots, x_{\mathsf{r}+\mathsf{r}_0+2}]$ of size $\mathsf{r}+\mathsf{r}_0+2$ elements is padded as $[x_1, x_2, \ldots, x_{\mathsf{r}_0}, x_{\mathsf{r}_0+1} \ldots, x_{\mathsf{r}+\mathsf{r}_0+2}, \langle 0\rangle, \langle 0\rangle, \ldots, \langle 0\rangle]$ with $\mathsf{r}-2$ postpadded zeros when processed under $\mathsf{Sponge2'}$ whereas the same message would become $[\langle 0\rangle, \langle 0\rangle, \ldots, \langle 0\rangle, x_1, x_2, \ldots, x_{\mathsf{r}_0}, x_{\mathsf{r}_0+1} \ldots, x_{\mathsf{r}+\mathsf{r}_0+2}]$ with $\mathsf{r}-2$ prepadded zeros when processed under $\mathsf{Sponge2}^{\dagger}$.

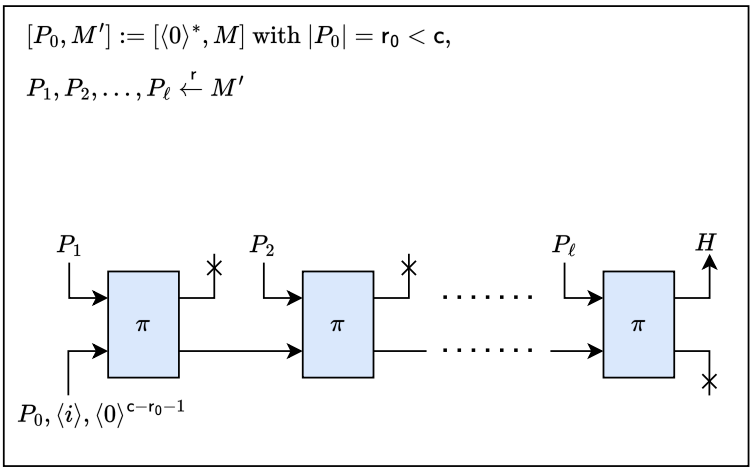


Figure 4: $\mathsf{Sponge2}^{\dagger}$ (block diagram) with prepadding.

We stress that the hash functions $\mathsf{Sponge2'}$ and $\mathsf{Sponge2}^{\dagger}$ only differ in the padding rules and there is a bijective mapping between both of these padding rules. This implies that the output distributions of these two hash functions are identical and hence they provide the same security (by indifferentiability from an RO). More concretely, we have

$$\begin{aligned}\mathbf{Adv}^{\text{ro-indiff}}_{\mathsf{Sponge2'}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]}(\mathcal{A}) &= \mathbf{Adv}^{\text{ro-indiff}}_{\mathsf{Sponge2}^{\dagger}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]}(\mathcal{A}) \\ &\leq |\Pr[\mathcal{A}^{\mathsf{Sponge2}^{\dagger}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}], f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}} \Rightarrow 1] \\ &\qquad - \Pr[\mathcal{A}^{\mathcal{RO}, S_1, S_2} \Rightarrow 1]|\end{aligned} \tag{7}$$

where S_1 and S_2 are two oracles defined to simulate adversarial-query response for $f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}$ in the ideal world. Note that the above expression is an inequality instead of an equality as S_1 and S_2 may not necessarily be the best simulators to define the advantage tightly.

Defining Simulators. In order to show that Sponge2† is indifferentiable from an RO, we need to construct two simulators S_1 and S_2 to model f_1^{ccf} and f_2^{ccf} in the ideal world, respectively.

Let for any b-element vector X with $\mathsf{b} = \mathsf{r} + \mathsf{c}$, X^{r} and X^{c} represent the first r and the last c elements (as a vector) of X. In other words, $X = [X^{\mathsf{r}}, X^{\mathsf{c}}]$. We define $S_2 = f_2^{\mathsf{ccf}}$ and S_1 as shown in Fig. 5. In the algorithmic description of S_1, $\mathcal{U}$ is a set of pairs of the form (α, β) where α is a c-element vector and β is an arbitrary size vector. We set $\mathcal{U} = \{\}$ before $\mathcal{A}$ makes its first query.

For a fresh primitive query $S_1(X) = Y$ of $\mathcal{A}$, an entry (α, β) is added to $\mathcal{U}$ where α represents the last c elements of the output Y whereas β represents the longest message whose hash output can be constructed using the already made S_1 queries (including the current query) with a condition that the final S_1 call of this hash is $S_1(X)$.

```
1: function S_1(X)
2:     Y ← f_1^ccf(X)
3:     β ← [ ]
4:     for i ← 0 to r + r_0 − 1 do
5:         if [X[r + r_0 − i], ..., X[b − 1]] = [⟨0⟩^i, ⟨i⟩, ⟨0⟩^(c−r_0−1)] then      ▷ case when X is
6:             β ← [X[0], ..., X[r + r_0 − i − 1]]                                  the first primitive
7:             Y^r ← RO(β)                                                          input in a hash call
8:         end if
9:     end for
10:    if ∃ (α', β') ∈ U with α' = X^c then                                         ▷ case when X is
11:        β ← [β', X^r]                                                            some primitive
12:        Y^r ← RO(β)                                                              input in a hash call
13:    end if
14:    α ← Y^c
15:    U ← U ∪ {(α, β)}
16:    Y ← [Y^r, Y^c]
17:    return Y
18: end function
```

Figure 5: The Simulator Algorithm S_1.

Note that S_1 and f_1^{ccf} differs in the generation of top r elements of the output, however, in both cases these r elements are sampled uniformly at random from $\mathbb{F}_p$ for each value of X. This implies (S_1, S_2) is indistinguishable from $(f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$.

Indifferentiability from an RO. We first define the set $\mathcal{L}$ consisting restricted outputs for both primitives (introduced above for Exp. 2) to simplify the remaining analysis as the set of all possible first primitive call inputs of Sponge2†. More concretely, $\mathcal{L} = \{X \in \mathbb{F}_p^{\mathsf{b}} \mid \text{ for some } i \in [0, \mathsf{r} + \mathsf{r}_0 - 1], [X[\mathsf{r} + \mathsf{r}_0 - i], \ldots, X[\mathsf{b} - 1]] = [\langle 0 \rangle^i, \langle i \rangle, \langle 0 \rangle^{\mathsf{c} - \mathsf{r}_0 - 1}]\}$. With $\mathcal{L}$ defined, we get $\Delta = |\mathcal{L}| = \sum_{i=0}^{\mathsf{r}+\mathsf{r}_0-1} p^{\mathsf{r}+\mathsf{r}_0-i} = p(p^{\mathsf{r}+\mathsf{r}_0} - 1)/(p-1)$.

Let us refer the first oracle in both worlds (i.e., Sponge2† in the real and $\mathcal{RO}$ in the ideal world) as the hash oracle and the second and third oracles in both worlds as the forward and backward primitive oracles, respectively. It is easy to notice that when the forward and backward primitives are restricted to return outputs only from $\mathbb{F}_p^{\mathsf{b}} \backslash \mathcal{L}$ and are capacity-collision-free, $\mathcal{A}$ can compute a hash oracle output only by either constructing the same using the old forward primitive query-response pairs or by making a fresh hash oracle query. We call the former type of queries —*constructed* queries and the latter type of queries —*direct* queries.

We emphasize here that the backward primitive calls become useless for $\mathcal{A}$ as they can't be used to replace the forward primitive calls in any hash query. This holds as backward primitive queries never return outputs that belong to $\mathcal{L}$ (i.e., the set of restricted outputs for primitives) or have capacity collisions with forward primitive query outputs (as they are capacity-collision-free).

Note that in the real world, all *constructed* and *direct* hash queries are consistent as per the definition of $\mathsf{Sponge2}^{\dagger}$ i.e., on same message they both return same outputs. Similarly, in the ideal world, with the defined simulator S_1, all *constructed* and *direct* hash queries are consistent as per the definition of S_1. See Fig. 6 for an example showing how a constructed query in the ideal world is consistent with its corresponding direct query. The example shows that for some input M, the constructed query returns $\mathcal{RO}(\mathsf{unpad}_i([P_0, \ldots, P_\ell]))$ whereas the direct query returns $\mathcal{RO}(M)$ and since by definition, $M = \mathsf{unpad}_i([P_0, \ldots, P_\ell])$, we have $\mathcal{RO}(\mathsf{unpad}_i([P_0, \ldots, P_\ell])) = \mathcal{RO}(M)$. Here, for any vector X, $\mathsf{unpad}_i([\langle 0 \rangle^i, X]) = X$ and $\perp$, otherwise.

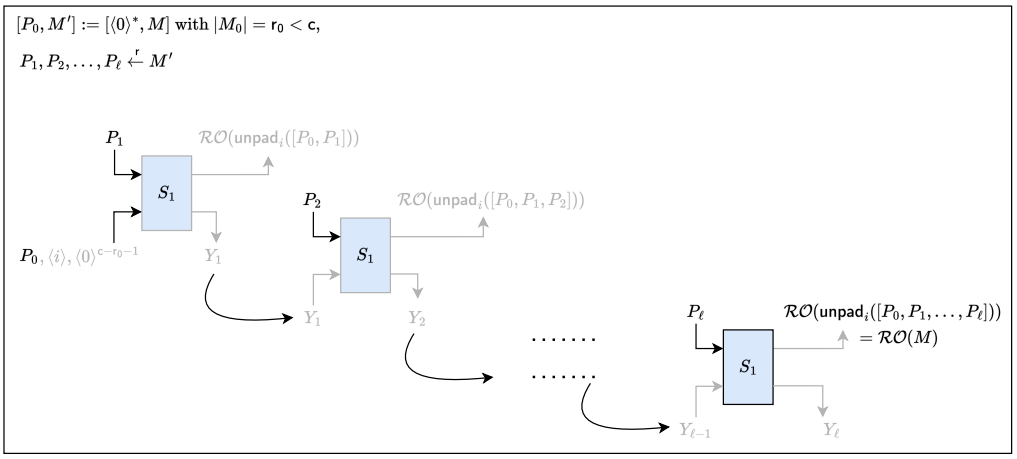


Figure 6: Simulator S_1 returning consistent outputs with $\mathcal{RO}$ queries. Here for any vector X, $\mathsf{unpad}_i([\langle 0 \rangle^i, X]) = X$ and $\perp$, otherwise.

This means the only way left for $\mathcal{A}$ to differentiate $(\mathsf{Sponge}^{\dagger}, f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}})$ from $(\mathcal{RO}, S_1, S_2)$ is by distinguishing the direct query outputs of $\mathsf{Sponge}^{\dagger}$ (i.e., hash outputs in the real world) from the direct query outputs of $\mathcal{RO}$ (i.e., hash outputs in the ideal world). Now, since f_1^{ccf} is a capacity-collision-free function, we have that for distinct direct queries to $\mathsf{Sponge}^{\dagger}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]$, the last f_1^{ccf} calls will always have a unique input and thus the rate part of its output (which is the final hash output) will be sampled uniformly at random from $\mathbb{F}_p^{\mathsf{r}}$. Similarly, since the outputs of $\mathcal{RO}$ are also sampled uniformly at random from $\mathbb{F}_p^{\mathsf{r}}$ for distinct direct queries, we have that

$$\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}^{\dagger}[f_1^{\mathsf{ccf}}, f_2^{\mathsf{ccf}}]}(\mathcal{A}) = 0 \tag{8}$$

We finalize the indifferentiability bound by combining Exp. 6, 7 and 8 with $\Delta = (1 + (p-1)^{-1})(p^{\mathsf{r}+\mathsf{r}_0} - 1)$ and get

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ro\text{-}indiff}}_{\mathsf{Sponge2}[\pi]}(\mathcal{A}) \leq \frac{q_P(q_P-1)}{2p^{\mathsf{b}}} + \frac{q_P}{p^{\mathsf{c}-\mathsf{r}_0}}\left(1 + \frac{1}{p-1}\right) \\ + \frac{q_P^2}{p^{\mathsf{c}}}\left(1 + \frac{p^{-\mathsf{c}+\mathsf{r}_0+1}}{p-1-p^{-\mathsf{c}+\mathsf{r}_0+1}}\right)\end{aligned} \tag{9}$$

and thus the result of Theorem 1. □

9 Discussion and Conclusion

Arithmetization oriented hash functions are a crucial building block in ZKP systems deployed in the real world (e.g., blockchain L2 rollups). In many cases, the efficiency of

the hash function is one of the major cost drivers for the entire system. Consequently, optimizing certain aspects of the hash function results in a net improvement to the entire system. Work in this field employed the *permutation based cryptography* paradigm to construct Sponge-based hash functions. When employing this paradigm, a designer comes up with a suitable permutation; i.e., a carefully crafted permutation admitting no distinguishable properties that is efficient to evaluate on the target architecture. Then, the permutation is used to instantiate a Sponge-function, resulting in a versatile algorithm that can be used as a hash function.

Indeed, most work in this domain focused on designing more efficient permutations (for some definition of efficiency). In this work we take a different approach and focus instead on the Sponge construction itself. As a first contribution, which is of independent interest, we generalize the sponge construction to GSponge. We then argue the security of the generalized construction with a formal indifferentiability proof. While the proof alone does not yet provide any efficiency gain to the concrete system, it is much simpler than the original proof for Sponges [BDPVA08] and, in our opinion, simpler to grasp by real-world practitioners. As an additional benefit, we show that the Generalized Sponge can be safely instantiated with permutations defined over large prime fields. This latter result has been known as folklore for the original Sponge and overwrite construction; however, we are not aware of any paper formally proving it.

Thanks to GSponge's generic structure, we found two micro-optimizations for deployed sponges. First, we introduce a new type of padding rule based on zero-padding and domain-separated inputs. This padding rule never extends the message length by a full rate size block, and consequently, saves one full permutation call in cases where the unpadded message's length is already an integral multiple of the rate (e.g., in 2-to-1 Merkle-tree hashing) without increasing the proof generation time in zkVMs like Polygon Miden. Secondly, we show that in the first permutation call it is possible to absorb up to $\mathsf{c}/2$ more elements, again resulting in saving a permutation call for some message lengths, without any loss in the security size, i.e., still $\approx \mathsf{c} \log_2 p/2$ bits.

While not asymptotic, these micro-optimizations can be used to improve the hashing time of practical use-cases (e.g., Merkle-tree hashing, short messages, etc.). As a vision for future work, we hope that this paper will inspire further work on modes of operation tailored for permutations defined over large prime fields.

Acknowledgements

Amit Singh Bhati was supported by CyberSecurity Research Flanders with reference number VR20192203, in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by the Flemish Government through FWO Project G.0835.16 A security Architecture for IoT.

References

[ABF+08] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 270–288. Springer, 2008.

[ABKM23] Tomer Ashur, Amit Singh Bhati, Al Kindi, and Mohammad Mahzoun. XHash8 and XHash12: Efficient STARK-friendly Hash Functions. *Cryptology ePrint Archive*, 2023.

[ABR21] Elena Andreeva, Rishiraj Bhattacharyya, and Arnab Roy. Compactness of hashing modes and efficiency beyond Merkle tree. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT*, pages 92–123. Springer, 2021.

[AMP12] Elena Andreeva, Bart Mennink, and Bart Preneel. The parazoa family: generalizing the sponge hash functions. *International Journal of Information Security*, 11:149–165, 2012.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007, 2007.

[BDPVA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT*, pages 181–197. Springer, 2008.

[Bec08] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 12:19, 2008.

[BMN10] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security analysis of the mode of JH hash function. In *Fast Software Encryption: 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers 17*, pages 168–191. Springer, 2010.

[BR06] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. *EUROCRYPT 2006*, 2006.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*, pages 430–448. Springer, 2005.

[CN08] Donghoon Chang and Mridul Nandi. Improved indifferentiability security analysis of chopMD hash function. In *Fast Software Encryption - FSE*, pages 429–443. Springer, 2008.

[Dam89] Ivan Bjerre Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

[GK08] Praveen Gauravaram and John Kelsey. Linear-XOR and additive checksums don't protect Damgård-Merkle hashes from generic attacks. In *Cryptographers' Track at the RSA Conference*, pages 36–51. Springer, 2008.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium*, pages 519–535, 2021.

[GLP08] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide attacks on a class of hash functions. In *Advances in Cryptology-ASIACRYPT*, pages 143–160. Springer, 2008.

[HK06] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *Annual International Cryptology Conference*, pages 41–59. Springer, 2006.

[HS06] Jonathan J Hoch and Adi Shamir. Breaking the ICE–finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In *Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers 13*, pages 179–194. Springer, 2006.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In *Annual International Cryptology Conference*, pages 306–316. Springer, 2004.

[KBM23] Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink. Generic Security of the SAFE API and Its Applications. In *International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT*, pages 301–327. Springer, 2023.

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 183–200. Springer, 2006.

[KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2 n work. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 474–490. Springer, 2005.

[Luc05] Stefan Lucks. A failure-friendly design principle for hash functions. In *Advances in Cryptology-ASIACRYPT*, pages 474–494. Springer, 2005.

[Mer89] Ralph C Merkle. One way hash functions and DES. In *Conference on the Theory and Application of Cryptology*, pages 428–446. Springer, 1989.

[Mid23] Polygon Miden. https://github.com/0xPolygonMiden/crypto/blob/next/benches/README.md, 2023.

[MPST16] Dustin Moody, Souradyuti Paul, and Daniel Smith-Tone. Indifferentiability security of the fast wide pipe hash: Breaking the birthday barrier. *Journal of Mathematical Cryptology*, 10(2):101–133, 2016.

[MRH04] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of Cryptography: First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004. Proceedings 1*, pages 21–39. Springer, 2004.

[NO14] Yusuke Naito and Kazuo Ohta. Improved indifferentiable security analysis of PHOTON. In *International Conference on Security and Cryptography for Networks*, pages 340–357. Springer, 2014.

[NP10] Mridul Nandi and Souradyuti Paul. Speeding up the wide-pipe: Secure and fast hashing. In *International Conference on Cryptology in India*, pages 144–162. Springer, 2010.

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). *Cryptology ePrint Archive*, 2020.

[Wu11] Hongjun Wu. The hash function JH. *Submission to NIST (round 3)*, 6, 2011.