Aborting Random Oracles: How to Build them, How to Use them

Gottfried Herold [1] Dmitry Khovratovich [1] Mikhail Kudinov [2]
Stefano Tessaro [3] Benedikt Wagner [1]

July 17, 2026

[1] Ethereum Foundation
{gottfried.herold,dmitry.khovratovich,benedikt.wagner}@ethereum.org
[2] Blockstream Research
mishel.kudinov@gmail.com
[3] University of Washington
tessaro@cs.washington.edu

Abstract

In this work, we initiate the study of *aborting hash functions*, i.e., hash functions that may abort on a non-negligible fraction of inputs. We introduce the aborting random oracle model (aROM), an idealized framework that extends the standard random oracle model (ROM) to account for aborts. Within this model, we derive bounds for various security notions and establish generic indifferentiability results demonstrating how to construct aborting random oracles from standard ones. Consequently, the derived bounds ultimately hold in the standard ROM. In this way, the aROM and its associated bounds provide a convenient and easy-to-use framework for analyzing cryptographic constructions that rely on potentially aborting hash functions.

To illustrate the utility of our framework, we apply our techniques to two settings: (1) the analysis of SNARK-friendly incomparable hypercube encodings, a core primitive in hash-based signature schemes, and (2) the analysis of grinding in Fiat–Shamir-based non-interactive arguments. Through our generic indifferentiability results, we can easily translate these analyses into concrete security bounds in the standard (non-aborting) random oracle model.

Acknowledgments. Tessaro's research was partially supported by NSF grants CNS-2154174 and CNS-2426905.

©IACR 2026. This is the full version of an article that will be published in the proceedings of CRYPTO 2026.

Contents

1 Introduction **3**
- 1.1 Starting Point: Hash-Based Signatures and Encodings . . . 3
- 1.2 Using Poseidon to Hash into Hypercubes . . . 4
- 1.3 Our Contribution: Aborting Hash Functions and Random Oracles . . . 5

2 Overview of Our Work **6**
- 2.1 Aborting Hash Functions, Random Oracles and Indifferentiability . . . 6
- 2.2 Application: Encodings in Hash-Based Signatures . . . 8
- 2.3 Application: Grinding in Fiat–Shamir . . . 9
- 2.4 Retrospective and Other Applications . . . 10

3 Related Work **10**

4 Aborting Hash Functions and Random Oracles **11**
- 4.1 Notation and Preliminaries . . . 11
- 4.2 Aborting Hash Functions and Random Oracles . . . 12
- 4.3 Security Notions for Aborting Hash Functions . . . 13

5 Indifferentiability Results **15**
- 5.1 Indifferentiability . . . 15
- 5.2 A Generic Proof for Indifferentiability . . . 16
- 5.3 Iterative Generalization of the Result . . . 19
- 5.4 From Aborting Random Oracles to Random Oracles . . . 21

6 Application: Hashing into Structured Domains **22**
- 6.1 Hashing into Hypercubes . . . 23
- 6.2 Hashing into Permutations . . . 24
- 6.3 Hashing into Hypercube Layers . . . 26

7 Application: Hypercube Encodings **28**
- 7.1 Incomparable Encodings . . . 29
- 7.2 Generalized Target Sum Encoding . . . 30
- 7.3 Encoding Instantiations . . . 31

8 Application: Grinding in Fiat-Shamir **32**
- 8.1 Interactive Proofs and Round-By-Round Soundness . . . 32
- 8.2 Compilation to Non-Interactive Arguments Using Grinding . . . 33

A Quantum Indifferentiability **41**

B Toolbox **44**
- B.1 Helpful Mathematical Lemma . . . 44
- B.2 Existing (Q)ROM Tools . . . 45
- B.3 New (Q)ROM Tools . . . 46

C Proof of Lemma 13 **47**

1 Introduction

Hash functions are ubiquitous in modern cryptography. They underpin fundamental primitives such as digital signatures [Mer79, Sch90, BLS01, GPV08], encryption schemes [ABR98, FO99], and more advanced tools like succinct argument systems [BCS16, BBHR18].

Consider a setting where we have a hash function whose outputs lie in one domain (e.g., bit strings of sufficient length), but we wish to construct a new hash function with outputs in a different domain. If this domain consists of bounded integers, a well-known folklore approach is to apply modular reduction; however, this method introduces bias into the resulting hash function that needs to be analyzed carefully. Alternatively, one can employ rejection sampling, which yields a uniform distribution but incurs aborts with non-negligible probability. Consequently, the resulting hash function deviates significantly from the behavior expected of a random oracle [BR93], and so setting parameters based on bounds in the random oracle model is inappropriate. Another option is to repeat rejection sampling until successful to make the abort probability negligible, but this in turn increases the computational cost by requiring multiple calls to the initial hash function. This approach has been used for example for hashing into elliptic curves [BCI+10, FFS+13] and to sample from (discrete) Gaussians in lattice-based cryptography [LW22].

This work starts from the observation that hash functions aborting with non-negligible probability remain useful in many contexts, and in particular, when constructing hash functions with structured outputs, where they offer better overall performance than alternatives guaranteed to nearly always produce an output. To better understand these use cases, this work initiates the study of *aborting hash functions* – hash functions that may fail on certain inputs. We introduce an idealized model for such functions and relate it to the standard random oracle model via indifferentiability. Our main motivation stems from recent hash-based signature designs, though we show broader applications of our framework beyond this domain.

1.1 Starting Point: Hash-Based Signatures and Encodings

Our starting point and main motivation is the recent hash-based signature proposal by the Ethereum Foundation [DKKW25c]. According to current plans, this scheme is intended to replace BLS signatures at the consensus layer, thereby protecting a blockchain ecosystem worth billions of dollars against the emerging quantum threat. Their proposal introduces a general framework for variants of XMSS signatures [BDH11], which was later further optimized and concretely specified [KKW25, DKKW25b].

Aggregation via Circuit-Friendliness. The reason not to use XMSS without any change is that these signatures later need to be *aggregated*, similar to how BLS signatures are aggregated today in Ethereum. As hash-based signatures lack the algebraic structure enabling native aggregation, the proposed approach relies on SNARKs: aggregation is achieved by succinctly proving knowledge of a list of *valid* signatures. This, in turn, requires the signature verification algorithm to be represented as a circuit, and the efficiency of the aggregation procedure depends crucially on the circuit-friendliness of this verification. To this end, the line of work recommends Poseidon2 [GKS23] as a circuit-efficient hash function and employs combinatorial optimizations to reduce the number of verification hashes while maintaining high security and small signature size.

Hypercube Encodings. At the heart of the XMSS framework lies an *encoding*, a function that maps the message and a random seed to a hypercube[1] vertex $x \in \mathbb{Z}_{\mathsf{w}}^{v}$. This mapping may fail for some seeds, and the signature scheme uses it as follows:

- The *signer* samples seeds ρ until it finds one for which the encoding succeeds and produces x. It then computes the signature based on x and includes ρ in the signature.
- The *verifier* checks that the encoding succeeds for the supplied ρ and subsequently verifies the signature using x.

For security, the set of encoding outputs must satisfy an incomparability condition, which can be guaranteed by making sure the output of the encoding is restricted to a subset $\mathcal{C} \subseteq \mathbb{Z}_{\mathsf{w}}^{v}$. Moreover, the

[1] Following [KKW25], we refer to $\mathbb{Z}_{\mathsf{w}}^{v}$ as a hypercube, although its hypercube structure is immaterial for this discussion.

encoding must exhibit a form of collision resistance. These properties are achieved by first hashing the message and seed using Poseidon, and then mapping the resulting hash into $\mathcal{C}$. The last step is non-trivial both due to a mismatch between the output of Poseidon and $\mathbb{Z}_{\mathsf{w}}^v$ and due to the need to restrict to a subset. For simplicity of this introduction, let us focus only on the mismatch with $\mathbb{Z}_{\mathsf{w}}^v$.

Circuit-Friendly Hypercube Encodings. As noted above, signature verification must be circuit-friendly to enable efficient aggregation. A closer look at the verification algorithm reveals two sources of complexity:

1. *Code.* The structure of the set of outputs x should allow verification to require only few hash computations. Prior work [DKKW25c, KKW25] has already achieved significant improvements here.

2. *Encoding algorithm.* The verifier must recompute the encoding on the message and seed ρ to obtain x, so the encoding algorithm itself must also be circuit-friendly.

Unfortunately, existing encoding constructions rely on big-integer arithmetic to map the Poseidon hash into the hypercube $\mathbb{Z}_{\mathsf{w}}^v$, which renders the encoding algorithm circuit-unfriendly.

Security and Setting Parameters. The security of the XMSS framework [DKKW25c] is proven in the standard model by assuming properties of Poseidon. Concrete parameters are then chosen by deriving bounds for these properties in the random oracle model. Thus, parameter selection relies on the heuristic of modelling Poseidon as a random oracle. Any improvement to the scheme must preserve this heuristic. For example, when constructing new hash functions from Poseidon, one should avoid treating them as independent random oracles unless an indifferentiability result is established.

1.2 Using Poseidon to Hash into Hypercubes

As discussed above, a key component of a circuit-friendly encoding is a secure hash function mapping into (a subset of) the hypercube $\mathbb{Z}_{\mathsf{w}}^v$. For circuit-friendliness, we aim to construct such a hash function from a function that outputs vectors in $\mathbb{Z}_p^\ell$, such as Poseidon [GKS23]. Concretely, Poseidon over small prime-order fields can provide an arbitrarily long vector for a 31-bit prime p. The main challenge is therefore:

How can we transform the field elements output by Poseidon into a hypercube element while preserving security and circuit-friendliness?

We next review existing approaches and their limitations, then discuss new directions.

Existing Approaches. Two main approaches have been proposed in [DKKW25c, KKW25, DKKW25b]:

- *Injective Embedding.* Obtain ℓ field elements from Poseidon and injectively map them into a slightly larger hypercube. Namely, interpret them as one large integer, and express this integer in base w to obtain a vertex in $\mathbb{Z}_{\mathsf{w}}^v$. This is secure if ℓ is large enough [DKKW25c].

- *Modular Reduction.* Turn the ℓ field elements into a large integer, reduce it modulo the desired hypercube (subset) size, and map it bijectively (via precomputed layer tables) to the target subset [KKW25, DKKW25b]. With sufficiently large ℓ, the bias introduced by the modulo operation is negligible.

Both methods rely on *big integer arithmetic*, which is not circuit-friendly. Our goal is therefore to design a Poseidon-based hash function avoiding such operations. Ideally, our function should be *indifferentiable* from a random oracle, assuming Poseidon is modeled as one. In this way, we could reuse bounds from [DKKW25c] to set parameters while ultimately relying on the *same* heuristics.

Towards an Element-Wise Approach. To eliminate big integer arithmetic, we need to treat Poseidon outputs individually. Consider the following straw man approach with $\mathsf{w} = 16$ and $v = 6\ell$ for ℓ as above, and p being a 31-bit prime of form $p = 2^{24}d + 1$ for some integer d (such primes exist, e.g., $p = 2^{31} - 2^{24} + 1$):

1. Obtain ℓ field elements $A_1, \ldots, A_\ell \in \mathbb{Z}_p$ from Poseidon.
2. From each A_i, take 24 bits and interpret them as 6 elements in $\mathbb{Z}_{\mathsf{w}}$.

Although simple and efficient, this approach fails because Poseidon outputs are uniform over $\mathbb{Z}_p$, not as 31-bit strings. Due to the form of p, the resulting bias from truncation is $1/p$ for each element, so it is about ℓ/p in total, which is non-negligible as p is small. Hence, the resulting distribution is far from uniform and unsuitable as a random oracle instantiation.

Rejecting Field Elements. A natural fix is to use *rejection sampling*. Write $p = 2^k\mathsf{w}^z + \alpha$ for integers $z, k, \alpha \geq 0$ and process Poseidon outputs $A_1, \ldots, A_\ell$ as follows:

- Mark A_i as *bad* if $A_i \geq 2^k \cdot \mathsf{w}^z$. Else, mark it as *good*.
- For each good A_i, decompose $A_i = 2^k(a_{i,z-1}\mathsf{w}^{z-1} + \cdots + a_{i,1}\mathsf{w} + a_{i,0}) + c$ for $c < 2^k$, and add $a_{i,0}, \ldots, a_{i,z-1} \in \mathbb{Z}_{\mathsf{w}}$ to the output.

This ensures uniformity in $\mathbb{Z}_{\mathsf{w}}$, as trimming k bits (by removing c) is a regular mapping[2] on the accepted range. To make the method efficient, we want z large (many outputs per field element) and α small (few rejections). For large enough ℓ, we can successfully generate enough (i.e., v) values except with negligible probability. Consequently, the process fails only with negligible probability, and indifferentiability from a random oracle may be provable.

Eliminating Control Flow. While effective, rejection sampling introduces branching: each A_i must be checked for being bad, introducing a significant amount of conditional jumps in the verifier code. This harms circuit-friendliness. To avoid this, we can exploit the *asymmetry* between signer and verifier:

- The *signer* searches for a *pure run*, i.e., a sequence $A_1, \ldots, A_\ell$ where *all elements are good*, by varying the seed or counter.
- The *verifier* simply asserts that all received elements are good, in addition to its remaining checks.

This removes conditional jumps from the verifier while leaving the search effort to the signer.

A Problem with Indifferentiability. Unfortunately, our existing tools for indifferentiability are not sufficient to analyze this approach. To see this, observe that the hash function could be modeled in two ways:

- *Option 1: Retries within the hash.* The hash function itself includes retries, allowing an indifferentiability argument. But defining the hash function in this way, would mean forcing the verifier to re-run the retries[3], defeating the purpose of the pure-run optimization.
- *Option 2: Retries as part of the encoding.* The signer provides the seed or counter, and the verifier evaluates a fixed run. This, however, means that we have a function that aborts with non-negligible probability, which is clearly not indifferentiable from a random oracle.

We therefore conclude that new idealizations are needed to model *aborting hash functions*.

1.3 Our Contribution: Aborting Hash Functions and Random Oracles

We initiate the formal study of *aborting hash functions* – hash functions that may abort on a non-negligible fraction of inputs. Our work provides the first unified framework for reasoning about them.

[2]For a regular mapping, each element in the range has the same number of inputs mapping to it.

[3]This is because the verifier runs the encoding, which runs the full hash function, not just part of it. E.g., the verifier would check in this way that the provided counter is indeed the *first* counter for which a pure run is observed.

Aborting Random Oracles. We introduce the *aborting random oracle model*, an idealized abstraction that captures the behavior of hash functions that occasionally abort. We analyze its relationship to the classical random oracle model. For instance, we show – via *generic indifferentiability results* – under which conditions an aborting hash function obtained from a random oracle behaves like an aborting random oracle. This establishes that the aborting random oracle model is *no stronger* as a heuristic than the standard model. We further give *bounds for several fundamental security properties* of aborting hash functions when modeled as aborting random oracles.

Applications. We describe several applications showing the usefulness of the aborting random oracle framework, ranging from encodings in hash-based signatures to grinding in the Fiat-Shamir transform. All of these have in common that the aborting random oracle framework significantly simplifies their analysis. Also, in the case of encodings, it allows for more efficient constructions than what we would obtain by going through indifferentiability to an ordinary random oracle.

2 Overview of Our Work

We give an intuitive overview of our new framework and its applications.

2.1 Aborting Hash Functions, Random Oracles and Indifferentiability

Motivated by the discussion above, we introduce and study a novel variant of the random oracle model, which we call the aborting random oracle model.

Aborting Hash Functions and Random Oracles. Our aborting random oracle model provides an idealized framework for analyzing aborting hash functions, that is, functions of the form $\mathsf{H}_\perp : \mathcal{D} \to \mathcal{H} \cup \{\perp\}$, where $\perp \notin \mathcal{H}$ denotes a distinguished abort symbol. In contrast to standard hash functions, such functions may output $\perp$ on a non-negligible or even substantial fraction of their inputs. Although one can formalize desirable properties of aborting hash functions directly in the standard model (see below), obtaining concrete and interpretable security bounds often calls for a heuristic abstraction such as the random oracle model. However, treating an aborting hash function as an ordinary (i.e., non-aborting) random oracle is inappropriate when the abort probability is non-negligible, e.g., when the function aborts on nearly half of all inputs.

This gap calls for an extended idealization capable of capturing the behavior of such functions. We therefore introduce the *aborting random oracle model*, which augments the classical random oracle with explicit aborts:

Definition 1 (Informal version of Definition 6)**.** A *θ-aborting random oracle* $\mathsf{H}_\perp$ is a function $\mathcal{D} \to \mathcal{H} \cup \{\perp\}$ defined so that for each input $x \in \mathcal{D}$, the value $\mathsf{H}_\perp(x)$ is sampled independently as follows: with probability θ return $\perp$, and with probability $1 - \theta$ return a value $h \xleftarrow{\$} \mathcal{H}$ chosen uniformly at random.

From Aborting to Non-Aborting: The Counter-Approach. A θ-aborting random oracle coincides with a standard random oracle when $\theta = 0$, and if θ is negligible, it is indistinguishable from a standard random oracle under any polynomial number of queries. Hence our focus is on the non-trivial regime where θ is non-negligible.

In this setting, it is still possible to simulate an ordinary random oracle using a θ-aborting one. A simple *counter-based construction* repeatedly queries the aborting oracle on input x with an increasing counter until a non-$\perp$ value is obtained; letting R be a bound on the number of attempts, and assuming negligible θ^{R}, the counter-based construction is indifferentiable from a true random oracle. We detail this in Section 5.4. Note that this view conceptualizes earlier constructions of random oracles using a counter-based approach. On the other hand, we stress that we cover this counter-based transformation just for completeness, and it is not the intended mode of operation for aborting hash functions in our applications.

Generic Indifferentiability. With the aborting random oracle model in place, we now ask under which conditions an aborting hash function can legitimately be idealized in this way. To answer this, we give a class of *generic indifferentiability* results (Section 5). These results apply in the following natural setting.

Suppose we start from a well-understood, non-aborting hash function H that we are comfortable modeling as a standard random oracle. Now consider an aborting hash function $\mathsf{H}_\perp$ obtained by applying a postprocessing step to the output of H, where the postprocessing may occasionally abort. Our results provide conditions on this postprocessing ensuring that $\mathsf{H}_\perp$ is indifferentiable from a θ-aborting random oracle, and they yield an explicit expression for θ. Concretely, let H output values in a set $\mathcal{X}$, partitioned into a *good* subset $\mathcal{X}^\top$ and a *bad* subset $\mathcal{X}^\perp$. In the basic case, we consider postprocessing of the form $\mathsf{H}_\perp = f \circ \mathsf{H}$ with $f\colon \mathcal{X}^\top \to \mathcal{H}$, where the composition outputs $\perp$ whenever H lands in $\mathcal{X}^\perp$. Our indifferentiability theorem applies whenever f is *admissible*:

Definition 2 (Informal version of Definition 13)**.** An efficiently computable function $f\colon \mathcal{X}^\top \to \mathcal{H}$ is *admissible* if it is regular (all preimage sets $f^{-1}(h)$ have equal size) and if, for any image, a uniformly random preimage can be efficiently sampled.

Theorem 1 (Informal version of Theorem 3). *If f is admissible and H is modeled as a random oracle, then $\mathsf{H}_\perp = f \circ \mathsf{H}$ is indifferentiable from a θ-aborting random oracle, where $\theta = |\mathcal{X}^\perp|/|\mathcal{X}|$.*

Modelling $\mathsf{H}_\perp$ as a θ-aborting random oracle is therefore no stronger an heuristic than modelling H as a standard random oracle. Moreover, we can now easily translate security bounds obtained in the aborting random oracle model directly into the familiar (non-aborting) random oracle model.

We further extend this result to iterative multi-stage postprocessing, which is especially convenient for our later applications. Suppose the postprocessing is a *sequence* of functions $f_i\colon \mathcal{X}_i^\top \to \mathcal{X}_{i+1}$, where $\mathcal{X}_i = \mathcal{X}_i^\top \cup \mathcal{X}_i^\perp$, and the evaluation aborts whenever an intermediate value falls outside its corresponding domain $\mathcal{X}_i^\top$. The resulting aborting hash function is denoted as $(f_\ell, \ldots, f_1) \circ \mathsf{H}$.

Theorem 2 (Informal version of Theorem 4). *If every f_i is admissible and H is modeled as a random oracle, then $(f_\ell, \ldots, f_1) \circ \mathsf{H}$ is indifferentiable from a θ-aborting random oracle, where $\theta = 1 - \prod_i |\mathcal{X}_i^\top|/|\mathcal{X}_i|$.*

Security Notions and Bounds. We now return to security properties of aborting hash functions, which we discuss in detail in Section 4.3. Naturally, one may want to adapt standard model notions, such as preimage resistance and (multi-)collision resistance, to the aborting setting. For this overview, we focus on collision resistance as an example. Other notions are discussed in Section 4.3.

A key subtlety is that any nontrivial aborting hash function is *not* collision resistant in the classical sense: if the function outputs $\perp$ on, say, half of its inputs, then any two such inputs trivially form a collision. However, these abort collisions should not be relevant when the function is used as intended, e.g., when verifiers check that only non-aborting values are used. This motivates the following relaxed notion, which focuses only on non-aborting outputs:

Definition 3 (Informal version of Definition 7)**.** An aborting hash function is *non-aborting collision resistant* if it is hard to find distinct inputs $x \neq x'$ such that $\mathsf{H}_\perp(x) = \mathsf{H}_\perp(x') \neq \perp$.

The natural next step is to analyze this notion in the θ-aborting random oracle model[4]. Interestingly, we can provide two proofs, illustrating different techniques and yielding different bounds:

- *Reduction-Based Proof.* Under some rationality assumptions on θ, we reduce to standard collision resistance of a larger random oracle $\mathsf{H}\colon \mathcal{D} \to \mathcal{N}$, where $|\mathcal{N}| = |\mathcal{H}|/(1-\theta)$. One may view $\mathcal{N}$ as containing $\mathcal{H}$ together with many distinguishable copies of $\perp$. This yields a security bound of $q^2/|\mathcal{N}| = (1-\theta)q^2/|\mathcal{H}|$, where q is the number of oracle queries.
- *Direct Proof.* Instead, we can just use a simple union bound over all $\binom{q}{2} \approx q^2$ pairs of queries, and show that a fixed pair forms a non-aborting collision with probability $(1-\theta)^2/|\mathcal{H}|$. This gives the total bound $(1-\theta)^2 q^2/|\mathcal{H}|$.

The first proof showcases a general technique that may be useful for deriving bounds by reusing existing bounds for standard random oracles. On the other hand, the second bound is strictly tighter, especially when θ is large, and illustrates how working directly in the aborting random oracle model can yield qualitatively sharper results.

[4]Via indifferentiability, the resulting bounds transfer to the ordinary random oracle model.

The Quantum Setting. Although our primary focus is not the quantum setting, it is natural to consider a quantum analogue of the aborting random oracle model, in the spirit of the quantum random oracle model [BDF+11]. Our central results extend to this setting: our indifferentiability theorems can be adapted to the quantum setting by replacing lazy sampling with k-wise independent functions in a slightly non-trivial way (cf. Remark 8), and the reduction-based collision resistance proof above also carries over by invoking collision resistance of a quantum random oracle. In contrast, obtaining a quantum analogue of the direct (and tighter) proof remains an interesting open problem for future work.

2.2 Application: Encodings in Hash-Based Signatures

Recall from our motivation that hypercube encodings are a key component of hash-based signatures, and that aborting hash functions allow for more circuit-friendly constructions. In Section 7 we outline two new encodings and show how they can be analyzed cleanly within the aborting random oracle framework. We do not claim to fully resolve the encoding problem, but to provide encodings that are strictly more circuit-friendly than existing ones and, more importantly, to illustrate how our framework offers reusable tools for analyzing a broad class of future encodings.

Encoding from Hashing. We begin with the target-sum encoding from prior work [HKRY23, ZCY23, DKKW25c, DKKW25b]. On input a message and a random seed, the encoding hashes both to obtain a hypercube vertex $x \in \mathbb{Z}_{\mathsf{w}}^{v}$. It outputs x only if it lies in a designated *layer* of the hypercube, i.e., if $\sum_i x_i = T$ for a fixed target[5] T. Otherwise, it aborts. The signer repeats the encoding with fresh randomness until it succeeds, while the verifier evaluates it once using the seed included in the signature.

Our observation, already hinted at in the introduction, is that the encoding can be improved by using an aborting hash function $\mathsf{H}_\perp$: we simply abort whenever $\mathsf{H}_\perp$ aborts. Although this may slightly increase signing time (depending on the abort probability of $\mathsf{H}_\perp$), it can substantially reduce circuit complexity. Crucially, we avoid converting $\mathsf{H}_\perp$ into a non-aborting hash via the counter-based construction; doing so would also force the verifier to evaluate $\mathsf{H}_\perp$ multiple times. By working directly with aborting hash functions – and idealizing them as aborting random oracles – we can avoid this.

Security of the Hash. The first part of the security analysis largely parallels that of [DKKW25c, DKKW25b]. In particular, the encoding is secure assuming that the (now aborting) hash function $\mathsf{H}_\perp$ satisfies a suitable adaptation of *multi-target collision resistance with random sampling* (rTCR). We then establish rTCR bounds for $\mathsf{H}_\perp$ when modeled as a (classical or quantum) aborting random oracle, following the structure of the proof in the ordinary random oracle model from [DKKW25c]. As a side remark, we extend parts of the adaptive reprogramming framework of [GHHM21, DKKW25a] to the aborting setting.

Translation using Indifferentiability. The central technical task is now to design circuit-friendly aborting hash functions that can plausibly be treated as aborting random oracles. Suppose we start from a base hash function H mapping into $\mathbb{Z}_p^\ell$ that we are comfortable modeling as a standard random oracle[6]. To apply indifferentiability, we ensure that the constructed aborting hash function can be expressed as $\mathsf{H}_\perp = (f_\ell, \ldots, f_1) \circ \mathsf{H}$ for a sequence of admissible functions f_i. Once this decomposition is in place, our indifferentiability theorem immediately transfers the rTCR bound from the aborting random oracle model to the ordinary random oracle model. This yields the desired security guarantee for the encoding when H is modeled as a standard random oracle.

Instantiation 1. Full Hypercube. This is the instantiation described in the introduction. We write $p = Q\mathsf{w}^z + \alpha$ and hash the input (message and seed) using H. If any resulting field element A_i, $i \in [\ell]$, satisfies $A_i \geq Q\mathsf{w}^z$, we classify it as bad and let the hash abort. Otherwise, we decompose each A_i into z elements of $\mathbb{Z}_{\mathsf{w}}$ as explained in the introduction, and take the first v of the resulting ℓz elements as our output.

[5] Restricting to a single layer enforces the incomparability property discussed earlier, though we do not focus on this here.
[6] For instance, think of H as being Poseidon.

This construction naturally fits our framework by letting $\mathcal{G} \subset \mathbb{Z}_p$ denote the set of good elements. The function can be written as $(f_2, f_1) \circ \mathsf{H}$, where $f_1 \colon \mathcal{G}^\ell \to \mathbb{Z}_{\mathsf{w}}^{\ell z}$ performs the decomposition, and $f_2 \colon \mathbb{Z}_{\mathsf{w}}^{\ell z} \to \mathbb{Z}_{\mathsf{w}}^{v}$ truncates the result. Both functions are admissible (see Section 6.1), allowing us to apply our indifferentiability results.

Interlude: Hashing into Permutations. Before presenting our second instantiation, we take a detour to illustrate how our framework provides a clean method for hashing into the set of all permutations over $[n]$. Without aborts, achieving uniformity here is notoriously subtle. A naive approach – sampling independent elements from $[n]$ and discarding repeats until obtaining n distinct values – requires $\Omega(n \ln n)$ samples by a coupon collector bound. The classical *shuffle* method [Dur64] uses only $O(n)$ steps, but each step requires a uniform sample from a shrinking domain; in our setting, however, we only have access to uniform samples from $\mathbb{Z}_p$. Avoiding bias without increasing the number of steps is highly non-trivial in the non-aborting setting.

By allowing aborts, our framework yields a simple solution. Generalizing the ideas from hashing into $\mathbb{Z}_{\mathsf{w}}^{v}$, we filter out bad field elements and then decompose each remaining element using a *mixed-radix* representation. With this, we can produce a uniformly random element $(a_2, \ldots, a_n) \in \mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_n$, which can serve directly as the randomness required by the Fisher–Yates shuffle [Dur64]. The resulting permutation is uniform, and the construction remains circuit-friendly. For full details, see Section 6.2.

Instantiation 2. Layer of the Hypercube. Our second aborting hash function is slightly more involved than the first, but it allows us to hash *directly* into a fixed layer of the hypercube; that is, it either aborts or outputs an element $x \in \mathbb{Z}_{\mathsf{w}}^{v}$ satisfying $\sum_i x_i = T$. In particular, the abort in the target sum encoding itself will never occur.

The construction proceeds by first mapping uniformly into the corresponding *simplex*, i.e., the set of vectors $\hat{x} \in \mathbb{Z}^v$ with $\sum_i \hat{x}_i = T$. These vectors lie in $\mathbb{Z}^v$ but may fall outside $\mathbb{Z}_{\mathsf{w}}^{v}$; the final step of the hash function simply checks this and aborts if $\hat{x} \notin \mathbb{Z}_{\mathsf{w}}^{v}$. To map into the simplex, we take inspiration from a classical problem in statistics: sampling uniformly v real numbers in $[0, 1]$ such that their sum is exactly 1. One way to do this is to sample $v - 1$ points uniformly in $(0, 1)$, sort them, append 0 and 1, and take the v consecutive differences.

Translating this idea to the integer setting introduces a crucial subtlety: sorting is admissible only when the input contains no duplicates. We therefore need a way to obtain a sequence of $v-1$ *distinct* points in $n = T + v - 1$. Fortunately, our earlier construction for hashing into permutations provides exactly the needed techniques. We give the complete construction and admissibility analysis in Section 6.3.

2.3 Application: Grinding in Fiat–Shamir

Another application of our framework is the formal analysis of *grinding* in non-interactive arguments such as SNARKs and STARKs. Grinding is a common practical technique for boosting concrete soundness [Sta21], yet it is often described informally and typically treated as an ad-hoc modification of protocols. Our framework provides a clean and fully formal way to analyze it. We give the full treatment in Section 8.

Grinding. Consider a multi-round public-coin interactive proof where verifier messages (challenges) lie in a space $\mathcal{C}$. Recall that under the Fiat–Shamir transformation [FS87], each challenge is replaced by hashing the current transcript. Suppose that in some round a malicious prover can, with non-negligible probability, find a *lucky* challenge that lets it violate soundness.[7] Concretely, assume that about 2^κ hash queries suffice to find such a challenge. Grinding increases this soundness level from κ to $\kappa + \tau$. To do so, we let the hash output include both the challenge and τ extra bits, and we let the prover supply an additional nonce as hash input. A hash output is considered valid only if these τ bits are all zero; the verifier checks this condition. An honest prover must therefore try about 2^τ nonces, introducing a modest overhead. A malicious prover, however, must now find an input that *simultaneously* yields a lucky challenge *and* zeros in the extra τ bits. This requires about $2^{\kappa+\tau}$ attempts, which means that intuitively, the effective soundness increases by τ bits.

[7] We formalize this using the standard notion of round-by-round soundness.

Analysis via Aborting Random Oracles. Our framework models this cleanly. Firstly, we view the Fiat–Shamir transformation as using a θ-aborting random oracle $\mathsf{H}_\perp$ to generate challenges, where $\theta = 1 - 2^{-\tau}$. The soundness of the resulting non-interactive argument can be shown directly from the round-by-round soundness of the interactive protocol in the aborting random oracle model, giving a bound of $\kappa + \tau$ bits. Secondly, we implement $\mathsf{H}_\perp$ using a standard random oracle H mapping into $\mathcal{C} \times \{0,1\}^\tau$, and an admissible post-processing function f:

$$\mathsf{H}_\perp = f \circ \mathsf{H}, \qquad f \colon \mathcal{C} \times \{0^\tau\} \to \mathcal{C},\ (\alpha, 0^\tau) \mapsto \alpha.$$

By our generic indifferentiability theorem, the security analysis in the aborting model carries over to the standard random-oracle model.

2.4 Retrospective and Other Applications

Looking back at our applications, the key advantage of aborting hash functions is their inherent *asymmetry*: it suffices for one party to perform a grinding search to obtain a non-aborting input, while all other parties can verify this with a single evaluation. We leverage this property for signatures, where the signer searches for a non-aborting seed with modest effort, while the verifier simply checks that the supplied seed does not abort. We use the same principle in non-interactive arguments, where the prover grinds to obtain a suitable nonce and the verifier only needs to confirm that this nonce is non-aborting.

The aborting random oracle framework provides a clean way to analyze the security of such constructions. Combined with our indifferentiability results, this allows us to transfer the resulting bounds directly to the standard random oracle model. We believe this asymmetry is broadly useful, and we encourage readers to explore further applications. For example, in the context of BLS signatures and hashing to elliptic curves [BLS01].

3 Related Work

Here, we summarize relevant prior work and discuss how our work relates to it.

Indifferentiability. The notion of indifferentiability was introduced by Maurer, Renner, Holenstein [MRH04] and adapted to the specific case of hash functions by Coron et al. [CDMP05]. Ristenpart, Shacham, and Shrimpton [RSS11] in particular clarified that an indifferentiable hash function securely instantiates a random oracle in single-stage security game. Indiferrentiability has established itself as a desirable security requirement for hash functions–for instance, an indifferentiability proof is at the core of the design rationale behind Sponges/SHA-3 [BDPV08]. The notion was further extended to block cipher constructions (cf. [CHK+16, ABD+13, DSSL16]).

The notion of indifferentiability was extended to the quantum setting as well in a number of works (e.g., [CETU18, Zha19]).

Hashing into Structured Sets and Distributions. Outside of cryptography, there exist a vast literature on how to convert uniform samples from one domain to another (cf. Knuth-Yao [KY76] and the references therein), but these works do not study the security analysis of the resulting construction formally.

A related, but technically different, problem is that of hashing into elliptic curves, where one transforms a hash function into bit strings (modeled as a random oracle) into one hashing into a structured range. Indifferentiable constructions are known (e.g., [BCI+10, FFS+13]). Crucially, these constructions never abort. One could obtain such a construction using the generic approach presented in Section 5.4, but the constructions from [BCI+10, FFS+13] achieve the same goal by invoking the underlying hash function *twice* and leveraging very specific algebraic properties of elliptic curves.

Recently, Lu and Waters [LW22] generalized this approach by introducing a notion of an "explainable" sampler, i.e., one for which given a sample one can produce a random string producing such sample with the correct conditional distribution. Such sampler can be combined with a random oracle to obtain a near-perfect indifferentiable hash function producing outputs following a particular distribution described by the sampler. Note that, unlike this paper, the resulting hash function is not meant to abort (except, possibly, with negligible probability). Nonetheless, Lu and Waters's explainability notion allows for a

tunable closeness parameter and show that imperfect explainability has meaningful applications while falling short of giving indifferentiability. They also show that every sampler for a small-range distribution can be explained, and additionally show explainable samplers for a wide range of distributions, including discrete Gaussians. The notion of "Universal Samplers" by Hofheinz et al. [HJK+16] also gives a form of explainable sampler for *every* efficiently samplable distribution, but their approach relies on indistinguishability obfuscation.

These works are very related in spirit, but they differ fundamentally from the approach of this paper by aiming to achieve perfect hash functions that do not abort. In contrast, we show that allowing for abort gives us much simpler constructions while minimally impacting applications.

Other Random Oracle Variants. We note that a number of variants of the random oracle models have been considered in the literature [CT10, Zha22, JLLW23, CCG+23, MMZ25], but these are unrelated to the goals of this work. For instance, this includes the pseudorandom oracle model [JLLW23] used to construct obfuscation, the arithmetized random oracle model [CCG+23] used to construct proof-carrying data. All of these models represent a stronger heuristic than the ordinary random oracle model. In contrast, we show that our aborting random oracles can be constructed from standard random oracles and vice versa, and thus we view our work as a framework that gives tools to simplify analyses, instead of introducing a new heuristic.

Hash-Based Signatures. One of our primary motivations, and a key application of our results, is the study of hash-based signature schemes, for which there exists an extensive body of work [BDH11, BHRvV21, HK22, HBD+22, ZCY23, HKRY23]. Most relevant to our setting is a recent line of research [DKKW25c, KKW25, DKKW25b] that develops proving-friendly variants of XMSS [BDH11] relying on hypercube encodings, instantiated using the Poseidon2 hash function [GKS23]. Our work introduces new encodings that further improve proving-friendliness. Our aborting random-oracle framework allows to give a clean analysis of these novel encodings.

Fiat-Shamir with Aborts. At first glance, our aborting random oracle framework may appear related to the Fiat–Shamir with Aborts (FSwA) paradigm [Lyu09, DLL+17, KLS18, BBD+23]. This resemblance stems from the fact that, in our applications, the signer/prover retries until the hash invocation does not abort, which seems reminiscent of how signatures are produced in FSwA-based constructions. However, a key difference lies in the nature of the abort condition: in FSwA, the abort predicate typically *depends on the secret key* and therefore cannot, in general, be checked by the verifier. In contrast, for an aborting hash function the abort condition is publicly verifiable, allowing anyone to determine whether a given hash query aborts or not. This is important, e.g., in the grinding application.

4 Aborting Hash Functions and Random Oracles

In this section, we introduce aborting hash functions, which are hash functions that may abort on some inputs. We also introduce aborting random oracles as an idealization for those. The aborting random oracle model generalizes the random oracle model. We also introduce security notions for aborting hash functions, which are analogues of classical hash function notions such as collision resistance.

4.1 Notation and Preliminaries

We use standard cryptographic notation and terminology such as PPT (probabilistic polynomial-time) and negligible functions (sometimes denoted by negl) throughout this work. For instance, λ denotes the security parameter, and all algorithms get λ implicitly as input. For a deterministic algorithm Alg, we write $y := \mathsf{Alg}(x)$ to represent the deterministic output y on input x. When Alg is randomized, we use $y \leftarrow \mathsf{Alg}(x)$ to indicate that y is sampled by running Alg with fresh uniform random coins ρ. If we wish to make the randomness ρ explicit, we write $y := \mathsf{Alg}(x;\rho)$, in which case $\mathsf{Alg}(\cdot;\rho)$ becomes deterministic. The notation $y \in \mathsf{Alg}(x)$ denotes any possible output that Alg may produce on input x with non-zero probability. Sampling an element uniformly from a finite set X is denoted by $x \xleftarrow{\$} X$, and for sampling from a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$. For a distribution $\mathcal{D}$ over a set $\mathcal{X}$, its distance to uniform is

$\sum_{x \in \mathcal{X}} |\Pr[\mathcal{D} = x] - 1/|\mathcal{X}||$. For an integer r, we define $[r] := \{1, \ldots, r\} \subseteq \mathbb{N}$. We use $\perp$ to denote a dedicated abort symbol. The notation $\mathbb{Z}_q$ for some integer q refers to the set $\{0, \ldots, q-1\}$, which can be treated as a ring or field (for q prime). We use $\mathbf{T}(\mathsf{Alg})$ to denote the running time of an algorithm Alg.

4.2 Aborting Hash Functions and Random Oracles

Let us start with the definition of aborting hash functions, which are hash function that may output a dedicated abort symbol $\perp$ on some inputs.

Definition 4 (Aborting Hash Function)**.** An *aborting hash function* is an efficiently computable function

$$\mathsf{H}_\perp : \mathcal{D} \to \mathcal{H} \cup \{\perp\}.$$

Here, $\mathcal{D}$ is the input domain, $\mathcal{H}$ is called the hash space (and assumed to be finite) and $\perp \notin \mathcal{H}$ denotes a dedicated abort symbol.

Remark 1 (Input Domains). We consider primarily two cases for $\mathcal{D}$:

- *Keyed.* Here, $\mathcal{D} = \mathcal{P} \times \mathcal{M}$ for a parameter space $\mathcal{P}$ and a message space $\mathcal{M}$. That is, our hash functions are parameterized by a (non-secret) hash key $P \in \mathcal{P}$. This is usually important when we define properties of hash functions in the standard model, as otherwise solutions (e.g., collisions) can be hard-coded.
- *Unkeyed.* Here, $\mathcal{D} = \mathcal{M}$. That is, we do not have a key. When we work in the (aborting) random oracle model, our bounds apply to such functions as well. Equivalently, we can think of this as having a key space $\mathcal{P}$ with $|\mathcal{P}| = 1$.

In Definition 17, we will require $\mathcal{D}$ to have even more structure.

We now define the aborting random oracle model, as an idealization for aborting hash functions. This model is similar to the random oracle model, but each call to the random oracle now aborts independently with some probability θ. Conditioned on not aborting, it behaves like a random oracle on this input. For the definition and throughout the paper, it will be useful to define the following distribution.

Definition 5 (Aborting Uniform Distribution)**.** For $\theta \in [0, 1]$ and a finite set $\mathcal{H}$, we define the *aborting uniform distribution* $\mathcal{D}_\perp^{\mathcal{H},\theta}$ as follows:

1. With probability θ, output $\perp$.
2. Otherwise, output a uniformly sampled element $h \xleftarrow{\$} \mathcal{H}$.

Definition 6 (Aborting Random Oracle Model)**.** Let $\mathsf{H}_\perp : \mathcal{D} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function and $\theta \in [0, 1]$. When we say that $\mathsf{H}_\perp$ is modeled as a *θ-aborting random oracle*, we mean that every algorithm in the security experiment has access an oracle O, which implements a random function with the following distribution: each $\mathsf{O}(x)$ for $x \in \mathcal{D}$ is sampled *independently* according to the aborting uniform distribution $\mathcal{D}_\perp^{\mathcal{H},\theta}$. We may then just say that parties have oracle access to $\mathsf{H}_\perp$ for simplicity.

Remark 2 (Lazy Sampling). Similar to standard random oracles, θ-aborting random oracles can be implemented via lazy sampling in the following way. The oracle internally maintains a table $\mathsf{h}[\cdot]$ and $\mathsf{def}[\cdot]$ and is defined as follows on input $x \in \mathcal{D}$:

1. If $\mathsf{def}[x] = \perp$: Sample $\mathsf{h}[x] \leftarrow \mathcal{D}_\perp^{\mathcal{H},\theta}$ and set $\mathsf{def}[x] := 1$.
2. Return $\mathsf{h}[x]$.

Remark 3. By definition, a 0-aborting random oracle is the same as a random oracle. On the other hand, a 1-aborting random oracle always returns $\perp$. The cases in which we are primarily interested in this work are θ-aborting random oracles for some θ that is non-negligibly far from 0 and 1.

4.3 Security Notions for Aborting Hash Functions

Next, we define security notions for aborting hash functions and show bounds for them in the aborting random oracle model. These are analogues of the standard security notions for hash functions, but adapted to the fact that the hash function may abort frequently. We start with collision resistance. Here, note that we do not count it as a collision if the inputs make the hash function abort. This is because we are interested in functions that abort on a non-negligible fraction of inputs, and so finding two distinct inputs that abort may be trivial. We directly generalize the definition to multi-collisions.

Definition 7 (Non-Aborting k-Collision Resistance)**.** Let $\mathsf{H}_\perp \colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function. We say that $\mathsf{H}_\perp$ satisfies *non-aborting k-collision resistance* if for any PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{na\text{-}coll\text{-}res}}_{\mathsf{H}_\perp,k}(\mathcal{A}) := \Pr\left[\begin{array}{ll} & \forall i,j \in [k] \colon i \neq j \implies M^{(i)} \neq M^{(j)} \\ \wedge & \forall i,j \in [k] \colon \mathsf{H}_\perp(P, M^{(i)}) = \mathsf{H}_\perp(P, M^{(j)}) \\ \wedge & \forall i \in [k] \colon \mathsf{H}_\perp(P, M^{(i)}) \neq \perp \end{array} \middle| \begin{array}{c} P \xleftarrow{\$} \mathcal{P}, \\ (M^{(i)})_{i=1}^k \leftarrow \mathcal{A}(P) \end{array} \right].$$

We give two security bounds for collision resistance. The first one showcases a general technique that often applies and may be used to reduce to known bounds for non-aborting (i.e., standard) random oracles. Then, we give a more direct analysis that has less restrictions on θ and gives a *tighter* bound.

Lemma 1 (Non-Aborting k-Collision Resistance – Bound Via Reduction)**.** *Let $\mathsf{H}_\perp \colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function modeled as a θ-aborting random oracle, where $\theta|\mathcal{H}|/(1-\theta)$ is assumed to be an integer. Then, for any PPT algorithm $\mathcal{A}$ that makes at most q queries to $\mathsf{H}_\perp$, we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}coll\text{-}res}}_{\mathsf{H}_\perp,k}(\mathcal{A}) \leq (1-\theta)^{k-1} \frac{\binom{q}{k}}{|\mathcal{H}|^{k-1}} \leq (1-\theta)^{k-1} \frac{q^k}{|\mathcal{H}|^{k-1}}.$$

Proof. We consider a reduction that has access to an adversary $\mathcal{A}$ against non-aborting k-collision resistance for $\mathsf{H}_\perp$, and itself plays in the standard k-collision resistance game for a random oracle

$$\mathsf{H} \colon \mathcal{P} \times \mathcal{M} \to \mathcal{N}, \text{ where } |\mathcal{N}| = |\mathcal{H}|/(1-\theta) = |\mathcal{H}| + \theta|\mathcal{H}|/(1-\theta).$$

Here, we assume that $\mathcal{N}$ is the disjoint union of $\mathcal{H}$ and another set $\mathcal{F}$ of size $\theta|\mathcal{H}|/(1-\theta)$. The reduction gets as input $P \in \mathcal{P}$ and has oracle access to H. It runs $\mathcal{A}$ on input P and with oracle access to $\mathsf{H}_\perp$, where it simulates queries to $\mathsf{H}_\perp$ as follows:

$$\mathsf{H}_\perp(P, M) := \begin{cases} \mathsf{H}(P, M), & \text{if } \mathsf{H}(P, M) \in \mathcal{H} \\ \perp, & \text{if } \mathsf{H}(P, M) \in \mathcal{F} \end{cases}.$$

Then, when $\mathcal{A}$ outputs $(M^{(i)})_{i=1}^k$, the reduction simply forwards it. Clearly, the simulation is perfect[8]. Further, we argue that if $\mathcal{A}$ breaks non-aborting k-collision resistance of $\mathsf{H}_\perp$, then the reduction breaks k-collision resistance of H. To see this, assume that $\mathcal{A}$ indeed succeeds. Then all $M^{(i)}$ are distinct. Further, we have $\mathsf{H}_\perp(P, M^{(i)}) \neq \perp$ for all i, which in particular implies that $\mathsf{H}_\perp(P, M^{(i)}) = \mathsf{H}(P, M^{(i)})$. Therefore, the multi-collision is also a multi-collision for H. The bound then follows by applying the standard multi-collision bound (e.g., [BKP18]) to H, which maps to $\mathcal{N}$, which is of size $|\mathcal{H}|/(1-\theta)$. □

Lemma 2 (Non-Aborting k-Collision Resistance – Direct Bound)**.** *Let $\mathsf{H}_\perp \colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function modeled as a θ-aborting random oracle. Then, for any PPT algorithm $\mathcal{A}$ that makes at most q queries to $\mathsf{H}_\perp$, we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}coll\text{-}res}}_{\mathsf{H}_\perp,k}(\mathcal{A}) \leq (1-\theta)^{k} \frac{\binom{q}{k}}{|\mathcal{H}|^{k-1}} \leq (1-\theta)^{k} \frac{q^k}{|\mathcal{H}|^{k-1}}.$$

[8] Note that $\mathcal{A}$ does not have access to H, and so indifferentiability is not needed here.

Remark 4. Note that the difference to Lemma 1 is (1) we have no restriction on θ, and (2) the bound is tighter, as $1-\theta$ is raised to k.

Proof of Lemma 2. Denote the ith query to $\mathsf{H}_\perp$ by[9] $h_i = \mathsf{H}_\perp(P_i, M_i)$. Without loss of generality, we can assume that each input is queried at most once. For each subset of k queries $S \subseteq [q],\ |S| = k$, we define the following event:

- Event Coll_S: This event occurs if for all distinct $i, j \in S$, we have $\mathsf{H}_\perp(P_i, M_i) = \mathsf{H}_\perp(P_j, M_j)$ and $\mathsf{H}_\perp(P_i, M_i) \neq \perp$.

Fix such a set S, and let $j^* := \min S$ be the smallest element in S. Further, set $h^* := h_{j^*}$ to be the corresponding hash. Then, we have

$$\begin{aligned}\Pr[\mathsf{Coll}_S] &= \Pr[\forall i \in S\colon h_i \neq \perp \wedge h_i = h^*] \\ &= \Pr[\forall i \in S\colon h_i \neq \perp] \cdot \Pr[\forall i \in S\colon h_i = h^* \mid \forall i \in S\colon h_i \neq \perp].\end{aligned}$$

The first term is clearly $(1-\theta)^k$ by definition of the θ-aborting random oracles. The second term is $1/|\mathcal{H}|^{k-1}$. Now, a union bound over all sets S shows the claimed bound. □

Next, we consider preimage resistance, which is defined in a natural way. Notably, the target image is sampled from the non-aborting set $\mathcal{H}$. After that, we also look at a non-aborting variant of second-preimage resistance.

Definition 8 (Non-Aborting Preimage Resistance)**.** Let $\mathsf{H}_\perp\colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function. We say that $\mathsf{H}_\perp$ satisfies *non-aborting preimage resistance* if for any PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{na\text{-}pre\text{-}res}}_{\mathsf{H}_\perp}(\mathcal{A}) := \Pr\left[\ \mathsf{H}_\perp(P, M) = h \ \middle|\ \begin{array}{c} P \xleftarrow{\$} \mathcal{P}, h \xleftarrow{\$} \mathcal{H} \\ M \leftarrow \mathcal{A}(P, h) \end{array}\ \right].$$

Lemma 3 (Non-Aborting Preimage Resistance)**.** *Let $\mathsf{H}_\perp\colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function modeled as a θ-aborting random oracle. Then, for any PPT algorithm $\mathcal{A}$ that makes at most q queries to $\mathsf{H}_\perp$, we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}pre\text{-}res}}_{\mathsf{H}_\perp}(\mathcal{A}) \leq (1-\theta)\frac{q}{|\mathcal{H}|}.$$

Proof. Denote the ith query to $\mathsf{H}_\perp$ by $\mathsf{H}_\perp(P_i, M_i)$. Without loss of generality, we can assume that each input is queried at most once. Let $P \xleftarrow{\$} \mathcal{P}, h \xleftarrow{\$} \mathcal{H}$, and define the following events:

- Event Found_i: This event occurs if $\mathsf{H}_\perp(P_i, M_i) = h$.

Fixing $i \in [q]$, note that for Found_i to occur, we must have $\mathsf{H}_\perp(P_i, M_i) \neq \perp$, which happens with probability $1-\theta$, and conditioned on that, Found_i occurs with probability $1/|\mathcal{H}|$. If the adversary breaks non-aborting preimage resistance, at least one Found_i must occur. Thus, the bound follows from a union bound. □

Definition 9 (Non-Aborting Second-Preimage Resistance)**.** Let $\mathsf{H}_\perp\colon \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function. We say that $\mathsf{H}_\perp$ satisfies *non-aborting second-preimage resistance* if for any PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{na\text{-}secpre\text{-}res}}_{\mathsf{H}_\perp}(\mathcal{A}) := \hat{\Pr}\left[\ M' \neq M \wedge \mathsf{H}_\perp(P, M') = h \ \middle|\ \begin{array}{c} (P, M) \xleftarrow{\$} \mathcal{P} \times \mathcal{M}, \\ h := \mathsf{H}_\perp(P, M), \\ M' \leftarrow \mathcal{A}(P, M, h) \end{array}\ \right],$$

where $\hat{\Pr}$ indicates that the probability is *conditioned* on the event $h \neq \perp$.

Remark 5. In the above definition, we implicitly assume that the hash function is non-degenerate in the following sense: in the standard model, we assume that the hash function does not return $\perp$ on every input. In the θ-aborting random oracle model, we assume that $\theta > 0$. This ensures that the conditioning in the definition makes sense.

[9] While the experiment considers only a single P that is of interest, it could of course be that the adversary submits other parameters P_i in the ith oracle query.

Lemma 4 (Non-Aborting Second-Preimage Resistance). *Let* $\mathsf{H}_\perp : \mathcal{P} \times \mathcal{M} \to \mathcal{H} \cup \{\perp\}$ *be an aborting hash function modeled as a* θ*-aborting random oracle for* $\theta < 1$*. Then, for any PPT algorithm* $\mathcal{A}$ *that makes at most* q *queries to* $\mathsf{H}_\perp$*, we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}secpre\text{-}res}}_{\mathsf{H}_\perp}(\mathcal{A}) \leq (1-\theta)\frac{q}{|\mathcal{H}|}.$$

Proof. Consider the experiment in the definition of non-aborting second-preimage resistance, in the aborting random oracle model. Denote it by $\mathbf{G}_0$. That is, first a θ-aborting random oracle $\mathsf{H}_\perp$ is sampled, then $(P, M) \xleftarrow{\$} \mathcal{P} \times \mathcal{M}$ are sampled and h is set as $h := \mathsf{H}_\perp(P, M)$. Then, $\mathcal{A}$ is run on input (P, M, h) and with oracle access to $\mathsf{H}_\perp$, and outputs M'. Denote by Win the event that $M' \neq M \wedge \mathsf{H}_\perp(P, M') = h$. We want to bound $\hat{\Pr}_{\mathbf{G}_0}[\mathsf{Win}]$, where $\hat{\Pr}$ denotes the conditional probability conditioned on $h \neq \perp$. We first eliminate the conditional probability as follows: we consider a hybrid experiment $\mathbf{G}_1$ in which we first sample $(P, M) \xleftarrow{\$} \mathcal{P} \times \mathcal{M}$ and $h \xleftarrow{\$} \mathcal{H}$. Then, we sample the function $\mathsf{H}_\perp$ as a θ-aborting random oracle conditioned on $h = \mathsf{H}_\perp(P, M)$. This can easily be done by programming $\mathsf{H}_\perp(P, M) := h$ and sampling the remaining images as before independently from the aborting uniform distribution $\mathcal{D}^{\mathcal{H},\theta}_\perp$. Then, we run $\mathcal{A}$ as before. One can see that now we have $\hat{\Pr}_{\mathbf{G}_0}[\mathsf{Win}] = \Pr_{\mathbf{G}_1}[\mathsf{Win}]$. Note that the probability we consider in $\mathbf{G}_1$ is not conditioned on anything. Next, we bound this probability. For that, denote the ith query to $\mathsf{H}_\perp$ by $\mathsf{H}_\perp(P_i, M_i)$. Without loss of generality, we can assume that each input is queried at most once. Define the following events:

- Event Found_i: This event occurs if $\mathsf{H}_\perp(P_i, M_i) = h$ and $M_i \neq M$.

Fix $i \in [q]$, and note that for Found_i to occur, we must have $\mathsf{H}_\perp(P_i, M_i) \neq \perp$, which happens with probability $1 - \theta$. Conditioned on that, Found_i occurs with probability $1/|\mathcal{H}|$. Finally, if the adversary breaks non-aborting second-preimage resistance, at least one Found_i must occur. The bound follows from a union bound. □

5 Indifferentiability Results

The bounds that we have presented in the aborting random oracle model in isolation may only be of limited applicability. Namely, if we want to rely on them, we need to assume that our aborting hash function behaves like an aborting random oracle for all practical purposes. This would be a highly non-standard heuristic that we should aim to avoid. Instead, we propose to follow a different approach: assume now that our aborting hash function is constructed by *postprocessing* a more-established (non-aborting) hash function H, e.g., SHA-3 [Nat15] or Poseidon [GKS23], where the postprocessing may abort. Then, we would like to show that the resulting aborting hash function has some security property when modeling H as a standard random oracle. In this section, we give the tools to follow this approach: namely, we show under which conditions on the postprocessing the resulting function is indifferentiable from an aborting random oracle.

5.1 Indifferentiability

Let us first give the definition of indifferentiability. Consider the case where we have a (non-aborting) random oracle H and use it to construct a different function G^{H}, by calling it black-box in some algorithm G. We would like to argue that this behaves exactly like some idealized primitive, e.g., another random oracle, an ideal cipher, or an aborting random oracle. If we just use indistinguishability, we do not account for the fact that parties now still have access to H. To account for that, Maurer et al. [MRH04] introduced the notion of indifferentiability. Here, we require that we can switch from the "real" world, in which parties have access to the random oracle H and our construction G^{H} to an "ideal" world in which parties have access to the ideal primitive that we want (instead of G^{H}), and a simulated random oracle, which is just simulated using black-box access to this ideal primitive. As it turns out (and as we show below) this notion is the right notion to consider if we want to argue that G^{H} is just as good as the ideal primitive we want.

In this work, we are interested in aborting random oracles that are constructed from standard random oracles. Consequently, we tailor our indifferentiability definition to this setting.

Definition 10 (Indifferentiability from aborting ROM)**.** Let H be a function modeled as a random oracle, and let G be an algorithm that takes as input an element from $\mathcal{D}$ and returns an output in $\mathcal{H} \cup \{\perp\}$, while having oracle access to H. Let $\mathsf{G}^{\mathsf{H}} \colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}$ be the resulting aborting hash function. Then, we say that G^{H} is *indifferentiable from a θ-aborting random oracle when modeling H as a random oracle*, if there is a (stateful) PPT simulator Sim with oracle access to $\mathsf{H}_\perp$, such for every PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{G},\mathsf{H},\mathsf{H}_\perp,\mathsf{Sim}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\mathsf{H},\mathsf{G}^{\mathsf{H}}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{Sim}^{\mathsf{H}_\perp},\mathsf{H}_\perp}(1^\lambda) = 1\right] \right|,$$

where $\mathsf{H}_\perp \colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}$ is a θ-aborting random oracle. Note that in the second experiment the simulator Sim answers queries of $\mathcal{A}$ to the first oracle, while having itself oracle access to $\mathsf{H}_\perp$. We call Sim the indifferentiability simulator. We note that, in line with, e.g., [BDPV08, BCI+10, AMP10, CPZ24], we allow Sim to depend explicitly on an upper bound q to the number of oracle queries.

Remark 6 (The other way around)*.* The definition can naturally be adapted to the case in which we want to build a random oracle from a θ-aborting random oracle, as we do in Section 5.4. To do so, our simulator needs to implement the θ-aborting random oracle given the random oracle. As this is not the main focus of our work, we omit a fully formal treatment.

5.2 A Generic Proof for Indifferentiability

In this section, we study a specific class of aborting hash functions, namely, functions of the form $\mathsf{H}_\perp = f \circ \mathsf{H}$, for a non-aborting hash function H and a postprocessing function f. Intuitively, $f \circ \mathsf{H}$ runs H and aborts if the output is in $\mathcal{X}^\perp$. Otherwise, it applies f to the output. We study under which conditions the resulting function is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle. In Section 5.3, we also generalize this result to multiple sequentially executed postprocessing functions.

Definition 11 (Postprocessing Hash Functions)**.** Let $\mathcal{X} := \mathcal{X}^\top \cup \mathcal{X}^\perp$ be a set, which is partitioned into two disjoint sets, i.e., $\mathcal{X}^\top \cap \mathcal{X}^\perp = \emptyset$. Let $\mathsf{H} \colon \mathcal{D} \to \mathcal{X}$ be a (non-aborting) hash function, and let $f \colon \mathcal{X}^\top \to \mathcal{H}$ be a function. Then, we define the aborting hash function

$$f \circ \mathsf{H} \colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}, \quad \text{with} \quad z \mapsto \begin{cases} f(\mathsf{H}(z)), & \text{if } \mathsf{H}(z) \in \mathcal{X}^\top, \\ \perp, & \text{if } \mathsf{H}(z) \in \mathcal{X}^\perp \end{cases}.$$

For technical reasons, we will need the following definition, which models that we can (almost perfectly) sample uniformly from a set.

Definition 12 (Near-Perfect Sampleability)**.** Let S be a finite set. We say that S is *near-perfectly sampleable* if there exists an randomized algorithm Samp, taking a parameter ε as input and outputs an element $s \in S$, such that:

- **Distribution.** For every $\varepsilon > 0$, the statistical distance between $\mathsf{Samp}(\varepsilon)$ and the uniform distribution over S is at most ε.
- **Running Time.** The running time of $\mathsf{Samp}(\varepsilon)$ is at most polylogarithmic in $1/\varepsilon$.

This generalizes to families of sets S_h, indexed by h, where the sampler additionally takes h as input, and the above properties must hold for any h. Here, we require a single runtime bound polylogarithmic in $1/\varepsilon$ to hold for *all* h.

Remark 7. A typical instance where near perfect sampleability arises is from rejection sampling: suppose we have a sampling algorithm for S that runs in time T_0 and either fails (with probability θ) or otherwise outputs a uniform element from S. Rejection sampling with at most T trials or until we do not fail turns this into a sampler with running time bounded by $T \cdot T_0$ that fails with probability θ^T and otherwise outputs a uniformly random element from S; this translates to a statistical distance bounded by θ^T.

Note that if the underlying machine model only allows flipping uniform coins, we cannot perfectly sample from $[n]$ unless n is a power of 2 [KY76]. However, $[n]$ is near-perfect sampleable for any $n \geq 0$

using appropriate rejection sampling. While this subtlety is usually ignored in the literature, we aim to be a bit more precise in this work, given that such issues arise explicitly in our applications in Section 6 and Section 7.

We stress that ε in our definition can be picked arbitrarily small without significant increase in running time. In particular, ε can be chosen smaller than any exponentially small error term. This means that the reader can think of it as being essentially zero.

Intuitively, if we assume H is a random oracle, then $f \circ \mathsf{H}$ should abort with probability $\theta = |\mathcal{X}^{\perp}|/|\mathcal{X}|$. We now define an admissibility condition on f under which we show $f \circ \mathsf{H}$ to be indifferentiable from a θ-aborting random oracle. Our definition of admissibility is very similar to the one in earlier works that study non-aborting random oracles hashing into elliptic curves [BF01, BCI+10].

Definition 13 (Admissibility)**.** A function $f\colon \mathcal{X}^{\top} \to \mathcal{H}$ is called *admissible* if the following hold:

- **Efficiently Decidable.** There is a deterministic polynomial time algorithm deciding if an input x is in $\mathcal{X}^{\top}$.
- **Efficiently Computable.** There is a deterministic polynomial time algorithm computing f.
- **Regular.** For every $h \in \mathcal{H}$, we have $|f^{-1}(h)| = |\mathcal{X}^{\top}|/|\mathcal{H}| \in \mathbb{N}$.
- **Sampleable.** The family of sets $f^{-1}(h)$ indexed by h is near-perfectly sampleable.

Theorem 3 (Generic Indifferentiability). *Let $\mathcal{X} := \mathcal{X}^{\top} \cup \mathcal{X}^{\perp}$ be a set, which is partitioned into two disjoint sets, i.e., $\mathcal{X}^{\top} \cap \mathcal{X}^{\perp} = \emptyset$. Further assume that $\mathcal{X}$ is near-perfectly sampleable. Let $\mathsf{H}\colon \mathcal{D} \to \mathcal{X}$ be a (non-aborting) hash function which is modeled as a (non-aborting) random oracle, and let $f\colon \mathcal{X}^{\top} \to \mathcal{H}$ be a function.*

Then, if f is admissible, then $\mathsf{H}_{\perp} = f \circ \mathsf{H}$ is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, for

$$\theta = |\mathcal{X}^{\perp}|/|\mathcal{X}|.$$

In particular, there is a simulator Sim with oracle access to $\mathsf{H}_{\perp}$ such for every $\varepsilon^* > 0$, and every q and for every PPT algorithm $\mathcal{A}$ making at most q oracle queries, we have

$$\mathsf{Adv}^{\mathsf{indiff}}_{f\circ\mathsf{H},\mathsf{H},\mathsf{H}_{\perp},\mathsf{Sim}}(\mathcal{A}) \leq \varepsilon^*.$$

Further, to answer any oracle query, Sim makes a single query to $\mathsf{H}_{\perp}$ and runs in time at most $O(\theta^{-1}) \cdot \mathsf{poly}(\log(q), \log(1/\varepsilon^*), \log(1/\theta))$.

Proof. To prove the theorem, we first define the simulator Sim. Recall that it has access to the θ-aborting random oracle $\mathsf{H}_{\perp}$. For the construction of our simulator, we make use of a sequence of random functions:

$$\Psi^{\perp}\colon \mathcal{D} \to \mathcal{X}^{\perp},$$
$$\Psi^{\top}_h\colon \mathcal{D} \to f^{-1}(h) \text{ for every } h \in \mathcal{H}.$$

Simulator and Analysis. We will specify how the simulator implements these functions at the end of the proof. In the first part of the proof, we assume that they are given (e.g., as additional oracles to the simulator) and show perfect indifferentiability. Here is how we define our simulator:

- **Simulator $\mathsf{Sim}^{\mathsf{H}_{\perp}}$ on input** $z \in \mathcal{D}$:
  1. Query $h := \mathsf{H}_{\perp}(z)$. We have $h \in \mathcal{H} \cup \{\perp\}$.
  2. If $h = \perp$: Return $\Psi^{\perp}(z) \in \mathcal{X}^{\perp}$.
  3. If $h \neq \perp$: Return $\Psi^{\top}_h(z) \in f^{-1}(h)$.

We first note that the simulator indeed provides a *function*, namely, on the same input it always produces the same output, and that the claim about the number of queries made by Sim is clear. Further, we observe that $f \circ \mathsf{Sim}^{\mathsf{H}_{\perp}} = \mathsf{H}_{\perp}$. This can easily be seen by considering the two cases $\mathsf{H}_{\perp}(z) = \perp$ and $\mathsf{H}_{\perp}(z) \neq \perp$ separately.

The second step of the proof is to show that no PPT algorithm can distinguish between the oracles $(\mathsf{H}, f \circ \mathsf{H})$ and the oracles $(\mathsf{Sim}^{\mathsf{H}_\perp}, \mathsf{H}_\perp)$. We will show something stronger: the two are identically distributed.

First, because $f \circ \mathsf{Sim}^{\mathsf{H}_\perp} = \mathsf{H}_\perp$ with probability 1, both the distribution of $(\mathsf{H}, f \circ \mathsf{H})$ and the distribution of $(\mathsf{Sim}^{\mathsf{H}_\perp}, \mathsf{H}_\perp)$ are supported on pairs of functions (g_1, g_2) with $g_2 = f \circ g_1$. Since in both cases, g_1 uniquely determines g_2, it suffices to look at only the distribution of g_1, which means that we need to show that H and $\mathsf{Sim}^{\mathsf{H}_\perp}$ are identically distributed. By definition, the distribution of H is uniquely defined by the property that the family (over z) of random variables $\mathsf{H}(z)$ is independent, identically distributed with $\Pr[\mathsf{H}(z) = h'] = 1/|\mathcal{X}|$ for all $z \in \mathcal{D}, h' \in \mathcal{X}$. So it suffices to show that the same property holds for $\mathsf{Sim}^{\mathsf{H}_\perp}$.

We first show independence of the family (parameterized by z) of random variables $\mathsf{Sim}^{\mathsf{H}_\perp}(z)$: this follows from the fact that Sim uses independent randomness for each input z and because the family (parameterized by z) of random variables $\mathsf{H}_\perp(z)$ is independent.

Next, we show that the individual $\mathsf{Sim}^{\mathsf{H}_\perp}(z)$ have the correct distribution: For this, let $z \in \mathcal{D}, h' \in \mathcal{X}$ and consider two cases: in the first case, we have $h' \in \mathcal{X}^\perp$. Then we can compute

$$\begin{aligned}\Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h'\right] &= \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \wedge \mathsf{H}_\perp(z) = \perp\right] + \underbrace{\Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \wedge \mathsf{H}_\perp(z) \neq \perp\right]}_{=0} \\ &= \Pr[\mathsf{H}_\perp(z) = \perp] \cdot \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \mid \mathsf{H}_\perp(z) = \perp\right] \\ &= \theta \cdot \frac{1}{|\mathcal{X}^\perp|} = \frac{|\mathcal{X}^\perp|}{|\mathcal{X}|} \cdot \frac{1}{|\mathcal{X}^\perp|} = \frac{1}{|\mathcal{X}|}\ ,\end{aligned}$$

by definition of θ and the way Sim works.

In the second case, we have $h' \in \mathcal{X}^\top$ and we can similarly compute

$$\begin{aligned}\Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h'\right] &= \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \wedge \mathsf{H}_\perp(z) = \perp\right] + \Pr[E_z \wedge \mathsf{H}_\perp(z) \neq \perp] \\ &= 0 + \Pr[\mathsf{H}_\perp(z) \neq \perp] \cdot \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \mid \mathsf{H}_\perp(z) \neq \perp\right] \\ &= (1-\theta) \cdot \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \mid \mathsf{H}_\perp(z) \neq \perp\right].\end{aligned}$$

Looking closer at the last probability, we have

$$\begin{aligned}\Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \mid \mathsf{H}_\perp(z) \neq \perp\right] = \ & \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \wedge \mathsf{H}_\perp(z) = f(h') \mid \mathsf{H}_\perp(z) \neq \perp\right] \\ & + \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \wedge \mathsf{H}_\perp(z) \neq f(h') \mid \mathsf{H}_\perp(z) \neq \perp\right].\end{aligned}$$

Note that the second term is 0, as $f \circ \mathsf{Sim}^{\mathsf{H}_\perp} = \mathsf{H}_\perp$. We can therefore continue:

$$\begin{aligned}& \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h'\right] \\ =\ & (1-\theta) \cdot \Pr[\mathsf{H}_\perp(z) = f(h') \mid \mathsf{H}_\perp(z) \neq \perp] \cdot \Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h' \mid \mathsf{H}_\perp(z) = f(h') \neq \perp\right] \\ =\ & (1-\theta) \cdot \frac{1}{|\mathcal{H}|} \frac{1}{|f^{-1}(f \circ A(z))|}.\end{aligned}$$

Using regularity and our definition of θ, this is equal to

$$\Pr\left[\mathsf{Sim}^{\mathsf{H}_\perp}(z) = h'\right] = \frac{|\mathcal{X}^\top|}{|\mathcal{X}|} \cdot \frac{1}{|\mathcal{H}|} \cdot \frac{|\mathcal{H}|}{|\mathcal{X}^\top|} = \frac{1}{|\mathcal{X}|},$$

as desired. This shows that $\mathsf{Sim}^{\mathsf{H}_\perp}(z)$ has the correct distribution. By the above considerations, this means that the indifferentiability advantage is zero, assuming we have perfect implementations of $\Psi^\perp$ and $\Psi_h^\top$.

Implementing $\Psi^\perp$ and $\Psi_h^\top$. Now, let us explain how Sim can implement the functions $\Psi^\perp$ and $\Psi_h^\top$. It does so via lazy sampling[10], and using the samplers from the near-perfect sampleability of $\mathcal{X}$ and the family $f^{-1}(h), h \in \mathcal{H}$. More concretely, to sample randomly from $f^{-1}(h)$, it uses the sampler with statistical distance ε_1 (to be determined below). To sample randomly from $\mathcal{X}^\perp$, sample for T trials from $\mathcal{X}$ and accept the sample if it is in $\mathcal{X}^\perp$. If we are not successful after T trials, we use a default sample from $\mathcal{X}^\perp$. To sample from $\mathcal{X}$, we use the near-perfect sampler for $\mathcal{X}$ with parameter ε_2 (to be determined below). Then, the statistical distance of such a $\mathcal{X}^\perp$-sample from the uniform distribution over $\mathcal{X}^\perp$ is at most $(1-\theta)^T + T\varepsilon_2$. For q queries, the total distance to the simulator above (that has access to perfect functions $\Psi^\perp$ and $\Psi_h^\top$) is then at most

$$q \cdot (\varepsilon_1 + (1-\theta)^T + T\varepsilon_2) \leq \varepsilon^*,$$

where this bound holds when we define $T, \varepsilon_1, \varepsilon_2$ as

$$\begin{aligned}
\varepsilon_1 &= \varepsilon^*/(3q),\\
\varepsilon_2 &= \varepsilon^*/(3qT),\\
T &= \log(\varepsilon^*/3q)/\log(1-\theta) \leq \ln 2 \log(\varepsilon^*/3q)/\theta,
\end{aligned}$$

where the last inequality uses Lemma 15. With these definitions, the running time of our simulator is as claimed. □

Remark 8 (Quantum Indifferentiability). Our indifferentiability results can be extended to the quantum setting, where the adversary can query its oracles in superposition. To see this intuitively, note that our simulator shows perfect indifferentiability, i.e., the produced distributions are identical, if we assume that $\Psi^\perp, \Psi_h^\top$ are perfect. The main challenge in the quantum case is now to implement the functions $\Psi^\perp, \Psi_h^\top$ without lazy sampling. We show this in detail in Appendix A. Notably, we also get an arbitrarily small ε^*. The simulator makes two queries to answer a query.

5.3 Iterative Generalization of the Result

We now generalize our result to the following setting: assume that the random oracle output is first postprocessed with a function f_1 as above. Then, an additional abort may happen, before the output is postprocessed with a function f_2, and so on. In the following definition, we formalize this iterative postprocessing. The definition generalizes Definition 11. Then, we show that if *each f_i is admissible individually*, the result is indifferentiable from an aborting random oracle, which generalizes Theorem 3. The setting is visualizated in Figure 1.

Definition 14 (Iterative Postprocessing Hash Functions)**.** Let $\ell \in \mathbb{N}$ be a number and for each $i \in [\ell]$, let $\mathcal{X}_i := \mathcal{X}_i^\top \cup \mathcal{X}_i^\perp$ be a set, which is partitioned into two disjoint sets, i.e., $\mathcal{X}_i^\top \cap \mathcal{X}_i^\perp = \emptyset$. Also, consider a final set denoted by $\mathcal{X}_{\ell+1} = \mathcal{H}$. Let $\mathsf{H}\colon \mathcal{D} \to \mathcal{X}_1$ be a (non-aborting) hash function. For each $i \in [\ell]$, let $f_i\colon \mathcal{X}_i^\top \to \mathcal{X}_{i+1}$ be a function. Then, we define the aborting hash function

$$(f_\ell, \ldots, f_1) \circ \mathsf{H}\colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}$$

as follows on input $z \in \mathcal{D}$:

1. Compute $x_1 := \mathsf{H}(z)$. Note that $x_1 \in \mathcal{X}_1$.
2. For $i = 1, \ldots, \ell$:
   (a) If $x_i \in \mathcal{X}_i^\perp$, return $\perp$. Else, observe that $x_i \in \mathcal{X}_i^\top$
   (b) Set $x_{i+1} := f_i(x_i)$. Note that $x_{i+1} \in \mathcal{X}_{i+1}$.
3. Return $h := x_{\ell+1}$.

[10] In the quantum case, this needs to be implemented differently, see Remark 8 below.

Theorem 4 (Generic Indifferentiability of Iterative Postprocessing). *Let $\ell \in \mathbb{N}$ be a constant and for each $i \in [\ell]$, let $\mathcal{X}_i := \mathcal{X}_i^\top \cup \mathcal{X}_i^\perp$ be a set, which is partitioned into two disjoint sets, i.e., $\mathcal{X}_i^\top \cap \mathcal{X}_i^\top = \emptyset$. Also, consider a final set denoted by $\mathcal{X}_{\ell+1} = \mathcal{H}$. Further assume that $\mathcal{X}_1$ is near-perfectly sampleable. Let $\mathsf{H} \colon \mathcal{D} \to \mathcal{X}_1$ be a (non-aborting) hash function which is modeled as a (non-aborting) random oracle, and for each $i \in [\ell]$, let $f_i \colon \mathcal{X}_i^\top \to \mathcal{X}_{i+1}$ be a function.*

Then, if each f_i is admissible, then $(f_\ell, \ldots, f_1) \circ \mathsf{H}$ is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, for

$$\theta = 1 - \prod_{i=1}^{\ell} \frac{|\mathcal{X}_i^\top|}{|\mathcal{X}_i|}.$$

In particular, there is a simulator Sim with oracle access to $\mathsf{H}_\perp$ such for every $\varepsilon^* > 0$, and every q and for every PPT algorithm $\mathcal{A}$ making at most q oracle queries, we have

$$\mathsf{Adv}^{\mathsf{indiff}}_{(f_\ell,\ldots,f_1)\circ\mathsf{H},\mathsf{H},\mathsf{H}_\perp,\mathsf{Sim}}(\mathcal{A}) \leq \varepsilon^*.$$

Further, to answer any oracle query, Sim makes a single query to $\mathsf{H}_\perp$ and runs in time at most $O(\theta^{-1}) \cdot \mathsf{poly}(\log(q), \log(1/\varepsilon^*), \log(1/\theta))$.

Proof. The proof of this theorem will follow directly from Lemma 5 below, in combination with Theorem 3. □

Lemma 5. *Consider the same assumptions as in Theorem 4. Then, there exists an admissible function $g \colon \mathcal{X}_1^* \to \mathcal{H}$ for some set $\mathcal{X}_1^* \subseteq \mathcal{X}_1$ of size*

$$|\mathcal{X}_1^*| = |\mathcal{X}_1^\top| \cdot \prod_{i=2}^{\ell} \frac{|\mathcal{X}_i^\top|}{|\mathcal{X}_i|}.$$

such that the following equality of functions holds with probability 1 over the choice of H:

$$(f_\ell, \ldots, f_1) \circ \mathsf{H} = g \circ \mathsf{H}.$$

Proof. The proof is inductive over ℓ. The base case $\ell = 1$ is clear by definition. Now, consider the inductive step with $\ell \geq 2$ being arbitrary. We consider $(f_\ell, \ldots, f_2, f_1) \circ \mathsf{H}$, where $f_1 \colon \mathcal{X}_1^\top \to \mathcal{X}_2$ and $f_2 \colon \mathcal{X}_2^\top \to \mathcal{X}_3$, and so on. Each f_i is assumed to be admissible. We claim that there is an admissible function $\bar{f}_{1,2} \colon \tilde{\mathcal{X}}_1^\top \to \mathcal{X}_3$ with $\tilde{\mathcal{X}}_1^\top \subseteq \mathcal{X}_1$ and

$$(f_\ell, \ldots, f_2, f_1) \circ \mathsf{H} = (f_\ell, \ldots, \bar{f}_{1,2}) \circ \mathsf{H}. \tag{1}$$

Once we have established that, we will be able to apply the induction hypothesis. We define $\bar{f}_{1,2}$ as $f_2 \circ f_1|_{\tilde{\mathcal{X}}_1^\top}$, where

$$\tilde{\mathcal{X}}_1^\top := \left\{ x \in \mathcal{X}_1^\top \mid f_1(x) \in \mathcal{X}_2^\top \right\} \subseteq \mathcal{X}_1^\top.$$

Because of regularity of f_1, each element in $\mathcal{X}_2$ has exactly $|\mathcal{X}_1^\top|/|\mathcal{X}_2|$ many preimages. This holds in particular for each element in $\mathcal{X}_2^\top \subseteq \mathcal{X}_2$. Therefore, we have

$$|\tilde{\mathcal{X}}_1^\top| = \frac{|\mathcal{X}_1^\top|}{|\mathcal{X}_2|} \cdot |\mathcal{X}_2^\top|.$$

Then, assuming $\bar{f}_{1,2}$ is admissible and (1) indeed holds, we can apply the induction hypothesis to $(f_\ell, \ldots, \bar{f}_{1,2}) \circ \mathsf{H}$. Then, we know that there is an admissible function $g \colon \mathcal{X}_1^* \to \mathcal{H}$ such that $(f_\ell, \ldots, f_2, f_1) \circ \mathsf{H} = (f_\ell, \ldots, \bar{f}_{1,2}) \circ \mathsf{H} = g \circ \mathsf{H}$, where

$$|\mathcal{X}_1^*| = |\tilde{\mathcal{X}}_1^\top| \cdot \prod_{i=3}^{\ell} \frac{|\mathcal{X}_i^\top|}{|\mathcal{X}_i|} = \frac{|\mathcal{X}_1^\top|}{|\mathcal{X}_2|} \cdot |\mathcal{X}_2^\top| \cdot \prod_{i=3}^{\ell} \frac{|\mathcal{X}_i^\top|}{|\mathcal{X}_i|} = |\mathcal{X}_1^\top| \cdot \prod_{i=2}^{\ell} \frac{|\mathcal{X}_i^\top|}{|\mathcal{X}_i|},$$

as desired.

It remains to show that $\bar{f}_{1,2}$ is admissible and (1) indeed holds. We start with arguing (1). Indeed, the two are the same functions:

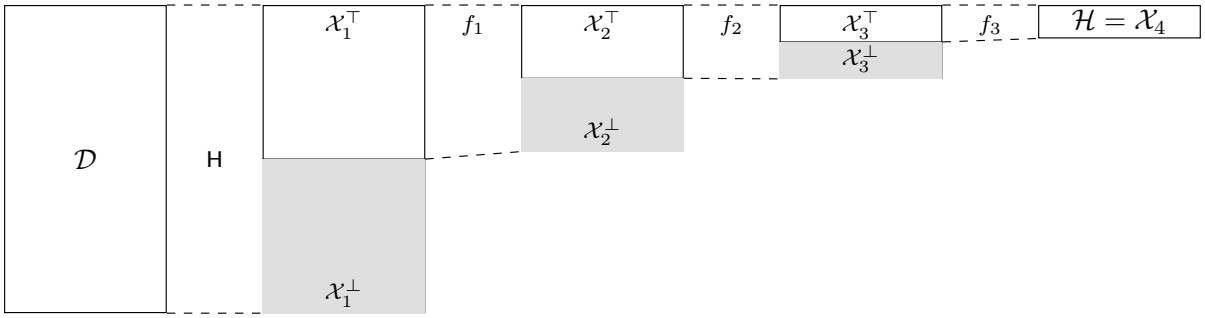

Figure 1: Visualization of the function $(f_\ell, \ldots, f_1) \circ \mathsf{H} \colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}$ in Definition 14.

- *Case 1:* $\mathsf{H}(z) \in \mathcal{X}_1^\perp$. Then, both $(f_\ell, \ldots, f_2, f_1) \circ \mathsf{H}$ and $(f_\ell, \ldots, \bar{f}_{1,2}) \circ \mathsf{H}$ return $\perp$.
- *Case 2:* $\mathsf{H}(z) \in \mathcal{X}_1^\top \setminus \tilde{\mathcal{X}}_1^\top$. Function $(f_\ell, \ldots, \bar{f}_{1,2}) \circ \mathsf{H}$ returns $\perp$, and $(f_\ell, \ldots, f_2, f_1) \circ \mathsf{H}$ also returns $\perp$ because $f_1(\mathsf{H}(z)) \notin \mathcal{X}_2^\top$.
- *Case 3:* $\mathsf{H}(z) \in \tilde{\mathcal{X}}_1^\top$. Here, both functions compute $f_1(\mathsf{H}(z)) \in \mathcal{X}_2^\top$, then apply f_2, and then continue in the same way.

Finally, we argue that $\bar{f}_{1,2}$ is admissible:

- *Efficiently Decidable.* To efficiently decide if $x \in \tilde{\mathcal{X}}_1^\top$, first check if $x \in \mathcal{X}_1^\top$ (efficient as f_1 is admissible), then compute $f_1(x)$ (efficient as f_1 is admissible), then check if $f_1(x) \in \mathcal{X}_2^\top$ (efficient as f_2 is admissible).
- *Efficiently Computable.* First compute $x' = f_1(x)$ (efficient as f_1 is admissible), then compute $f_2(x')$ (efficient as f_2 is admissible).
- *Regular.* This is because $f_1|_{\tilde{\mathcal{X}}_1^\top}$ is regular (which relies on the definition of $\tilde{\mathcal{X}}_1^\top$) and f_2 is regular.
- *Sampleable.* First, sample preimage under f_2, then sample preimage of that under $f_1|_{\tilde{\mathcal{X}}_1^\top}$ (this can be done as for f_1).

□

5.4 From Aborting Random Oracles to Random Oracles

To complete the picture, we show how to turn a θ-aborting random oracle $\mathsf{H}_\perp$ into a standard (i.e., non-aborting) random oracle H in an indifferentiable way. The idea is to run the θ-aborting random oracle multiple times, with a counter as additional input, and take the first non-aborting output. The maximum number of attempts R needs to be set such that the probability of failure θ^{R} is negligible.

Remark 9. We emphasize that the construction outlined in this section is *not* what we want to use in the applications we envision. Instead, we will use aborting hash functions directly. This enables one party (e.g., the signer) to find a counter or seed such that the hash function does not abort. The remaining parties (e.g., the verifier) then only need to check that the provided counter or seed yields a non-aborting hash call. In particular, it is not checked that the counter is the minimal non-aborting one. Our aborting random oracle framework allows us to analyze such applications. On the other hand, the transformation from aborting random oracles to random oracles that we present in this section is merely to show the *equivalence* of θ-aborting random oracles and random oracles in some sense. It also conceptualizes the classical "try-and-increment" approach for constructing hash functions mapping into groups [BLS01, Tib12].

Let us make the construction more precise: assume that $\mathsf{H}_\perp \colon \mathcal{D} \times [\mathsf{R}] \to \mathcal{H}$ is a θ-aborting random oracle. We build a random oracle $\mathsf{H} \colon \mathcal{D} \to \mathcal{H}$ as follows:

- Let $h_0 \in \mathcal{H}$ be a default element.

- For each $x \in \mathcal{D}$, define the set $S_x^* := \{j \in [\mathsf{R}] \mid \mathsf{H}_\perp(x, j) \neq \perp\}$. If $S_x^* \neq \emptyset$, let $j_x^* := \min S_x^*$.
- For each $x \in \mathcal{D}$, set $\mathsf{H}(x) := h_0$ if $S_x^* = \emptyset$, and set $\mathsf{H}(x) := \mathsf{H}_\perp(x, j_x^*)$ otherwise.

We now argue that H is indifferentiable from a random oracle when $\mathsf{H}_\perp$ is modeled as a θ-aborting random oracle and θ^{R} is negligible. We omit a formal treatment and just give an informal overview.

Lemma 6 (Informal)**.** *If $\mathsf{H}_\perp$ is modeled as a θ-aborting random oracle and θ^{R} is negligible, then H is indifferentiable from a random oracle (cf. Remark 6). For any adversary making at most q queries to its oracles, its advantage against indifferentiability is at most $q\theta^{\mathsf{R}}$.*

Proof Sketch. Our job is to construct a simulator Sim that can simulate access to $\mathsf{H}_\perp$ by making black-box use of H. More precisely, we need to show that for every adversary $\mathcal{A}$ making at most q queries, the following are indistinguishable:

- *Real World.* $\mathcal{A}$ gets oracle access to a θ-aborting random oracle $\mathsf{H}_\perp$ and to H constructed as above.
- *Simulated World.* $\mathcal{A}$ gets access to $\mathsf{Sim}^{\mathsf{H}}$ and H, where H is now a random oracle.

Our simulator will internally make use of a θ-aborting random oracle Ψ, which is not given to $\mathcal{A}$. The simulator can easily implement this via lazy sampling. Here is how our simulator provides the response to a query $(x, j) \in \mathcal{D} \times [\mathsf{R}]$:

1. If S_x^* is undefined:
   - Set $S_x^* := \emptyset$.
   - For $i = 1, \ldots, \mathsf{R}$: Insert i into S_x^* with probability $1 - \theta$.
2. Now that S_x^* is defined, proceed as follows:
   (a) If $S_x^* = \emptyset$: Return $\perp$. Else, let $j_x^* := \min S_x^*$.
   (b) If $j = j_x^*$: Return $\mathsf{H}(x)$.
   (c) If $j < j_x^*$: Return $\perp$.
   (d) If $j > j_x^*$: Return $\Psi(x, j)$.

Now, we briefly argue indistinguishability of the two worlds above: For that, note that the event Fail that $\mathcal{A}$ ever queries an x (to one of its oracles) such that $S_x^* = \emptyset$ occurs in both worlds with the same probability, which is at most $q\theta^{\mathsf{R}}$. Further, conditioned on $\neg\mathsf{Fail}$, it is easy to see that the adversary sees identically distributed oracles in both worlds. □

Remark 10. We assume that the proof sketch here can be adapted to work for quantum indifferentiability, by using Lemma 16. However, as neither the counter-based approach nor the quantum setting is the main focus of this work, we omit details on this.

6 Application: Hashing into Structured Domains

In this section, we show a few applications of our generic indifferentiability result. Namely, we define efficient aborting hash functions that map into specific domains. Such hash functions are useful for our application in Section 7. For this section, let $p > 0$ be an integer and $\ell \in \mathbb{N}$ be a positive integer. We assume that a hash function $\mathsf{H}\colon \mathcal{D} \to \mathbb{Z}_p^\ell$ that outputs vectors is given. For concreteness, the reader may think of this hash function as being the circuit-friendly Poseidon hash function[11] [GKS23] and p being a prime, although our results here apply to any hash function of that form, and any integer p. In the following, we construct new hash functions that postprocess the output of H. We will then apply our generic indifferentiability results from Section 5, noting that $\mathbb{Z}_p^\ell$ is indeed near-perfectly sampleable, and so only admissibility of the postprocessing needs to be checked. Our constructions rely on the following well-known fact, which we recall for completeness.

[11] Poseidon provides an arbitrarily long output in the sponge mode.

Lemma 7 (Mixed Radix, Well-known fact). *Let $t_1, \ldots, t_k \geq 2$, and set $T = \prod_{i=1}^{k} t_i$. For each $i \in [k]$, define the weight $w_i := \prod_{j=i+1}^{k} t_j$ with $w_k := 1$. Then, for every integer $x \in \mathbb{Z}_T$, there exist unique integers $(a_1, a_2, \ldots, a_k) \in \mathbb{Z}_{t_1} \times \cdots \times \mathbb{Z}_{t_k}$ such that $x = \sum_{i=1}^{k} a_i w_i$. In particular, the following mapping is a bijection:*

$$\mathbb{Z}_{t_1} \times \cdots \times \mathbb{Z}_{t_k} \longrightarrow \mathbb{Z}_T, \qquad (a_1, \ldots, a_k) \longmapsto \sum_{i=1}^{k} a_i w_i \,.$$

6.1 Hashing into Hypercubes

Let $\mathsf{w}, v > 0$ be integers specifying a hypercube $\mathbb{Z}_{\mathsf{w}}^{v}$. Write $p = Q\mathsf{w}^z + \alpha$ for integers $z, Q, \alpha \geq 0$. Assume that $\ell = \lceil v/z \rceil$ so that $\ell z \geq v$. We construct the following aborting hash function

$$\mathsf{H}^{\mathrm{hc}}_{\mathsf{w},v,z,Q} \colon \mathcal{D} \to \mathbb{Z}_{\mathsf{w}}^{v} \cup \{\perp\}.$$

On input $x \in \mathcal{D}$, it works as follows:

1. Run $(A_1, \ldots, A_\ell) := \mathsf{H}(x)$, where $A_i \in \mathbb{Z}_p$, $i \in [\ell]$.
2. If there exists an $i \in [\ell]$ with $A_i \geq Q\mathsf{w}^z$, return $\perp$.
3. For $i = 1, \ldots, \ell$:
   (a) Decompose A_i uniquely into $c_i < Q, d_i < \mathsf{w}^z$ such that $A_i = Qd_i + c_i$.
   (b) Decompose d_i uniquely (cf. Lemma 7) into $a_{(i-1)z+j} \in \mathbb{Z}_w$, $0 \leq j \leq z-1$ as
   $$d_i = a_{(i-1)z+z-1}\mathsf{w}^{z-1} + \cdots + a_{(i-1)z+1}\mathsf{w} + a_{(i-1)z}. \tag{2}$$
4. Return $(a_0, \ldots, a_{v-1})$.

The next lemma shows that modelling $\mathsf{H}^{\mathrm{hc}}_{\mathsf{w},v,z,Q}$ as an aborting random oracle ultimately relies on the same heuristic as just modelling Poseidon as a random oracle. The proof of the lemma relies on our generic indifferentiability result.

Lemma 8. *The aborting hash function $\mathsf{H}^{hc}_{\mathsf{w},v,z,Q}$ is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, with the same indifferentiability simulator and advantage as given by Theorem 4, and with*

$$\theta = 1 - \left(\frac{Q\mathsf{w}^z}{p}\right)^{\ell}.$$

Proof. We use our generic indifferentiability result, Theorem 4. To this end, we will show that there is are admissible functions $f_1 \colon \mathcal{X}^\top \to \mathbb{Z}_{\mathsf{w}}^{\ell z}$ and $f_2 \colon \mathbb{Z}_{\mathsf{w}}^{\ell z} \to \mathbb{Z}_{\mathsf{w}}^{v}$ with $\mathcal{X}^\top \subseteq \mathbb{Z}_p^\ell$, $|\mathcal{X}^\top| = (1-\theta)\mathbb{Z}_p^\ell$, and $\mathsf{H}^{\mathrm{hc}}_{\mathsf{w},v,z,Q} = (f_2, f_1) \circ \mathsf{H}$. Function f_2 just truncates the last $\ell z - v \geq 0$ elements. This is clearly admissible, and so we can just focus on f_1. We first define the set $\mathcal{G}$ of *good* elements, which allows us to easily specify $\mathcal{X}^\top$:

$$\mathcal{G} := \{A \in \mathbb{Z}_p \mid A_i < Q\mathsf{w}^z\} \subseteq \mathbb{Z}_p, \qquad \mathcal{X}^\top := \mathcal{G}^\ell \subseteq \mathbb{Z}_p^\ell.$$

Now, it is clear that $|\mathcal{X}^\top| = (p-\alpha)^\ell = (1-\theta)p^\ell$ by definition of θ. Next, we define the function $f_1 \colon \mathcal{X}^\top \to \mathbb{Z}_{\mathsf{w}}^{v}$. It maps $(A_1, \ldots, A_\ell)$ to $(a_0, \ldots, a_{\ell z-1})$ as in the main loop of the hash function, namely, by decomposing each A_i into d_i and c_i, and then d_i into the a_j as in (2). Note that in the way we defined f_1 and f_2, it is now clear that $\mathsf{H}^{\mathrm{hc}}_{\mathsf{w},v,z,Q} = (f_2, f_1) \circ \mathsf{H}$. It remains to argue that f_1 is admissible, where it is clear that it is efficiently decidable and efficiently computable. To argue that it is regular and sampleable, it is sufficient to focus on the mapping on individual elements, namely, the mapping

$$\mathcal{G} \ni A \longmapsto (a_0, \ldots, a_{z-1}), \quad A = Q(a_{z-1}\mathsf{w}^{z-1} + \cdots + a_1\mathsf{w} + a_0) + c, \;\; c < Q.$$

For regularity, we claim that each image $(a_0, \ldots, a_{z-1})$ has exactly Q preimages under this mapping. Namely, this is exactly the set

$$\left\{Q(a_{z-1}\mathsf{w}^{z-1} + \cdots + a_1\mathsf{w} + a_0) + c \mid c \in \{0, \ldots, Q-1\}\right\} \subseteq \mathcal{G}.$$

Here, we used the definition of $\mathcal{G}$, as elements outside of $\mathcal{G}$ cannot be *uniquely* represented in this way. To see that the mapping is sampleable, note that we can simply sample a random c in $\{0, \ldots, Q-1\}$ and compute A. □

Example 1 (Hypercube in [DKKW25b] + KoalaBear). Let $p = 2^{31} - 2^{24} + 1$ be the KoalaBear prime, which is frequently used for Poseidon instantiations. Say we want to use the hypercube given by $v = 64, \mathsf{w} = 8$, as suggested in [DKKW25b]. We can write

$$p = 127 \cdot 8^8 + 1, \text{ i.e., } Q = 127, z = 8, \alpha = 1.$$

With this, we get 8 elements from each field element, which means that $\ell = v/z = 64/8 = 8$ field elements are sufficient, i.e., one elementary[12] Poseidon call is sufficient. The abort probability θ is

$$\theta = 1 - ((p-1)/p)^8 \approx 3.75 \cdot 10^{-9}.$$

As we can see, this is non-negligible cryptographically, so indifferentiability from a random oracle is not doable. But at the same time, this is small enough so that the additional retries from hash function aborts do not impact the honest signer in practice, when we will use this in the hash-based signature context, see Section 7.

6.2 Hashing into Permutations

In this section we show an algorithm to hash into the set of all permutations over $[n]$, such that it is indifferentiable from a θ-aborting random oracle when modeling the underlying base hash function H as a random oracle. Throughout the algorithm we will need a sample from $\mathbb{Z}_n$, then from $\mathbb{Z}_{n-1}$, from $\mathbb{Z}_{n-2}$, and so on. In order to get those samples, we first partition the set $\{2, \ldots, n\}$ into ℓ disjoint sets $\mathcal{F}_j$, $j \in [\ell]$ with $\bigcup_{j\in[\ell]} \mathcal{F}_j = \{2, \ldots, n\}$. This yields a way of writing $n!$ as a product of ℓ terms s_j, as follows:

$$n! = \prod_{i\in[n]\setminus\{1\}} i = \prod_{j\in[\ell]} \underbrace{\left(\prod_{i\in\mathcal{F}_j} i\right)}_{=:s_j}.$$

We find ℓ representations of p, which, to recall, is the modulus of our base hash function $\mathsf{H}\colon \mathcal{D} \to \mathbb{Z}_p^\ell$:

$$p = Q_j s_j + r_j, \quad j \in [\ell]. \tag{3}$$

Our goal is to minimize ℓ and to minimize $\sum_j r_j$. The former will minimize the amount of randomness we need to get from the base hash function, and the latter will minimize the abort probability of the resulting aborting hash function (see Lemma 9 and Remark 12). Now we construct an aborting hash function

$$\mathsf{H}_n^{\mathrm{perm}}\colon \mathcal{D} \to \mathcal{S}_n \cup \{\perp\}.$$

On input $x \in \mathcal{D}$, it works as follows:

1. Run $(A_1, \ldots, A_\ell) := \mathsf{H}(x)$, where $A_i \in \mathbb{Z}_p$, $i \in [\ell]$.
2. If there exists an $j \in [\ell]$ with $A_j \geq Q_j s_j$, return $\perp$.
3. Obtain $(a_2, \ldots, a_n) \in \mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_n$ as follows: For $j \in [\ell]$:
   (a) Decompose A_j uniquely into $c_j < Q_j$, $d_j < s_j$ with $A_j = Q_j d_j + c_j$.

[12] An elementary Poseidon call can output 16 or 24 field elements, depending on which permutation width is used.

(b) Decompose d_j uniquely (cf. Lemma 7) into $\{a_i\}_{i\in\mathcal{F}_j}$ where $a_i \in \mathbb{Z}_i$ as

$$d_j = a_{i_1} + i_1 a_{i_2} + i_1 i_2 a_{i_3} + \cdots + a_{i_{k_j}} i_1 i_2 \cdots i_{k_j-1}, \tag{4}$$

where $\mathcal{F}_j = \{i_1, i_2, \ldots, i_{k_j}\}$ with $i_1 < i_2 < \cdots < i_{k_j}$.

This is possible because $A_j < Q_j \cdot s_j = Q_j \cdot i_1 \cdot i_2 \cdots i_{k_j}$.

4. Run the Fisher-Yates shuffle [Dur64] using $(a_2, \ldots, a_n)$ as randomness to obtain a permutation $\pi \in \mathcal{S}_n$. Concretely:
   (a) Initialize an array $B = [1, \ldots, n]$. The array is 0-indexed.
   (b) For i from $n-1$ down to 1: Swap $B[i]$ and $B[a_{i+1}]$.
   (c) Use B as the permutation.

Remark 11 (Generating Non-Repeating Sequences). Assume we want to generate an ordered sequence of $k < n$ distinct elements from $[n]$. Then, we can easily generalize the construction here without generating a full permutation. To this end, we simply partition the set $\{n-k+1, \ldots, n\}$ into sets $\mathcal{F}_j$, and run the same steps to get randomness $(a_{n-k+1}, \ldots, a_n)$, with which we run the first k steps of the Fisher-Yates shuffle. We then return the k elements in the back of the array. The same analysis as above directly generalizes to this construction.

The next lemma shows that modelling $\mathsf{H}_n^{\text{perm}}$ as an aborting random oracle ultimately relies on the same heuristic as just modelling Poseidon as a random oracle. The proof of the lemma relies on our generic indifferentiability result.

Lemma 9. *The aborting hash function H_n^{perm} is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, with the same indifferentiability simulator and advantage as given by Theorem 4, and with*

$$\theta = 1 - \prod_{i\in[\ell]} \left(1 - \frac{r_i}{p}\right).$$

Remark 12. For practical purpose it is convenient to have the following approximation for θ when $\sum_{i\in[\ell]} r_i$ is small compared to p:

$$\theta \approx \frac{\sum_{i\in[\ell]} r_i}{p}.$$

Indeed, for $r_i \ll p$ we have

$$1 - \prod_{i\in[\ell]} \left(1 - \frac{r_i}{p}\right) \approx 1 - \prod_{i\in[\ell]} e^{-\frac{r_i}{p}} = 1 - e^{-\frac{\sum_{i\in[\ell]} r_i}{p}} \approx \frac{\sum_{i\in[\ell]} r_i}{p}.$$

Proof of Lemma 9. We use our generic indifferentiability result, Theorem 4. To this end, we will show that there are two admissible functions $f_1 \colon \mathcal{X}^\top \to \mathcal{W}$ and $f_2 \colon \mathcal{W} \to \mathcal{S}_n$, with with $\mathcal{X}^\top \subseteq \mathbb{Z}_p^\ell$, $|\mathcal{X}^\top| = (1-\theta)\mathbb{Z}_p^\ell$, and $\mathsf{H}_n^{\text{perm}} = (f_2, f_1) \circ \mathsf{H}$. Here $\mathcal{W} = \mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_n$ and f_2 is the Fisher-Yates shuffle that maps $(a_2, \ldots, a_n)$ to the permutation $\mathcal{S}_n$. Note that the Fisher-Yates shuffle is a bijection from $[2] \times [3] \times \cdots \times [n]$ to $\mathcal{S}_n$, which is efficiently computable and invertible, so it is also admissible. Hence, we can focus only on f_1. The analysis of f_1 is very similar to the proof of Lemma 8. We first define sets $\mathcal{G}_j$ of *good* elements, for each $j \in [\ell]$. This also allows us to specify $\mathcal{X}^\top$:

$$\forall j \in [\ell] : \quad \mathcal{G}_j := \{A \in \mathbb{Z}_p \mid A_j < Q_j s_j\} \subseteq \mathbb{Z}_p.$$
$$\mathcal{X}^\top := \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_\ell \subseteq \mathbb{Z}_p^\ell.$$

Clearly, we have that $|\mathcal{X}^\top| = \prod_{i\in[\ell]}(p - r_i) = (1-\theta)p^\ell$ by our definition of θ. With that, the function $f_1 \colon \mathcal{X}^\top \to \mathbb{Z}_{\mathsf{w}}^v$ maps the elements $(A_1, \ldots, A_\ell)$ to $(a_2, \ldots, a_n)$ as in the algorithm, namely, by decomposing each A_i as in (4). By definition, it is now clear that $\mathsf{H}_n^{\text{perm}} = (f_2, f_1) \circ \mathsf{H}$. It remains to argue that f_1 is

admissible, where it is clear that it is efficiently decidable and efficiently computable. Thus, it suffices to argue that the mapping from $\mathcal{X}^\top$ to $(a_2, \ldots, a_n)$ is admissible:

$$\begin{aligned}\mathcal{G}_j \ni A_j &\longmapsto (a_{i_1}, \ldots, a_{i_{k_j}}),\\ A_j &= Q_j \left(a_{i_1} + i_1 a_{i_2} + i_1 i_2 a_{i_3} + \cdots + a_{i_{k_j}} i_1 i_2 \cdots i_{k_j - 1}\right) + c_j,\ c_j < Q_j.\end{aligned}$$

For regularity, we claim that each image $(a_{i_1}, \ldots, a_{i_{k_j}})$ has exactly Q_j preimages under this mapping. This is exactly the set

$$\left\{ Q_j \left(a_{i_1} + i_1 a_{i_2} + i_1 i_2 a_{i_3} + \cdots + a_{i_{k_j}} i_1 i_2 \cdots i_{k_j - 1}\right) + c_i \mid c_j \in \mathbb{Z}_{Q_j} \right\} \subseteq \mathcal{G}_j.$$

The mapping is sampleable, as we can simply sample a random c_j in $\mathbb{Z}_{Q_j}$ and compute A_j. □

Example 2. Let $p = 257$, $n = 10$. Then we use the following decomposition of $[10]$:

$$\{2, \ldots, 10\} = \underbrace{\{5, 10\}}_{\mathcal{F}_1} \cup \underbrace{\{2, 3, 6, 7\}}_{\mathcal{F}_2} \cup \underbrace{\{4, 9\}}_{\mathcal{F}_3} \cup \underbrace{\{8\}}_{\mathcal{F}_4}$$

so that

$$10! = \underbrace{5 \cdot 10}_{s_1} \cdot \underbrace{2 \cdot 3 \cdot 6 \cdot 7}_{s_2} \cdot \underbrace{4 \cdot 9}_{s_3} \cdot \underbrace{8}_{s_4}$$

This partitioning yields the following representations of p:

$$\begin{aligned}257 &= 5 \cdot (5 \cdot 10) + 7,\\ 257 &= 1 \cdot (2 \cdot 3 \cdot 6 \cdot 7) + 5,\\ 257 &= 7 \cdot (4 \cdot 9) + 5,\\ 257 &= 32 \cdot 8 + 1.\end{aligned}$$

Because we partitioned our set into $\ell = 4$ subsets, we assume that the base hash function maps into $\mathbb{Z}_p^\ell = \mathbb{Z}_{257}^4$. By Remark 12, the probability of an abort is approximately $(7 + 5 + 5 + 1)/257 \approx 0.07$. To further illustrate the construction, take an arbitrary base hash in $\mathbb{Z}_{257}^4$, e.g., $(38, 202, 250, 8)$. Then, we compute a_j as follows:

$$\begin{aligned}38 &= 5 \cdot (\underbrace{2}_{a_5} + 5 \cdot \underbrace{1}_{a_{10}}) + 3\\ 202 &= 1 \cdot (\underbrace{0}_{a_2} + 2 \cdot \underbrace{1}_{a_3} + 6 \cdot \underbrace{3}_{a_6} + 36 \cdot \underbrace{5}_{a_7}) + 0\\ 250 &= 7 \cdot (\underbrace{3}_{a_4} + 4 \cdot \underbrace{8}_{a_9}) + 5\\ 8 &= 32 \cdot (\underbrace{0}_{a_8}) + 8\end{aligned}$$

6.3 Hashing into Hypercube Layers

Let $v, \mathsf{w}, T \in \mathbb{N}$ be integers such that $0 \le T \le v(\mathsf{w} - 1)$. These specify the hypercube $\mathbb{Z}_{\mathsf{w}}^v$ and its layer

$$\mathcal{C} := \left\{ (x_1, \ldots, x_v) \in \mathbb{Z}_{\mathsf{w}}^v \;\middle|\; \sum_{i=1}^{v} x_i = T \right\} \subseteq \mathbb{Z}_{\mathsf{w}}^v.$$

Also, consider the following *integer* version of $\mathcal{C}$:

$$\mathcal{C}_{\mathbb{Z}} := \left\{ (x_1, \ldots, x_v) \in \mathbb{Z}^v \;\middle|\; \sum_{i=1}^{v} x_i = T \right\} \subseteq \mathbb{Z}^v,\ \ \mathcal{C} = \mathcal{C}_{\mathbb{Z}} \cup \mathbb{Z}_{\mathsf{w}}^v.$$

We make the same assumptions about decomposing p as in Section 6.2, for getting permutations over $[n]$, where $n = T + v - 1$. We construct the following aborting hash function

$$\mathsf{H}^{\mathsf{lay}}_{\mathsf{w},v,T} \colon \mathcal{D} \to \mathcal{C} \cup \{\perp\}.$$

On input $x \in \mathcal{D}$, it works as follows, with $n = T + v - 1$.

1. Sample an ordered sequence of $v-1$ *distinct* elements in $[n]$ as follows:
   (a) Get a permutation $\pi \in \mathcal{S}_n$ from x as in Section 6.2. This may return $\perp$.
   (b) Let the first $v-1$ elements of π be $(y_1, \ldots, y_{v-1}) \in [n]^{v-1}$.
   (c) Alternatively, directly apply the technique from Remark 11.
2. Sort $(y_1, \ldots, y_{v-1})$ to get the sequence $(y_{(1)}, \ldots, y_{(v-1)})$.
3. Set $y_{(0)} := 0$ and $y_{(v)} := T + v = n + 1$.
4. Let $(x_1, \ldots, x_v) := (y_{(1)} - y_{(0)} - 1, \ldots, y_{(v)} - y_{(v-1)} - 1)$. Note: $(x_1, \ldots, x_v) \in \mathcal{C}_{\mathbb{Z}}$.
5. If $(x_1, \ldots, x_v) \notin \mathcal{C}$, return $\perp$.
6. Return $(x_1, \ldots, x_v)$

Note that the first step involves representations of p by parameters s_i, r_i, Q_i as in Section 6.2.

Lemma 10. *The aborting hash function $\mathsf{H}^{\mathsf{lay}}_{\mathsf{w},v,T}$ is indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, with the same indifferentiability simulator and advantage as given by Theorem 4 and with*

$$\theta = 1 - \frac{|\mathcal{C}|}{|\mathcal{C}_{\mathbb{Z}}|} \prod_{i \in [\ell]} \left(1 - \frac{r_i}{p}\right) = 1 - \frac{\tilde{\ell}_T}{\binom{T+v-1}{v-1}} \prod_{i \in [\ell]} \left(1 - \frac{r_i}{p}\right),$$

where $\tilde{\ell}_T$ is the coefficient of X^T in the polynomial $(1 + X + \cdots + X^{\mathsf{w}-1})^v$.

Proof. We keep the proof at a high level. As for the previous indifferentiability results (Lemmata 8 and 9), it follows by defining appropriate admissible functions f_i and applying our generic indifferentiability result, Theorem 4. Precisely, the algorithm can be written as applying the following functions in order to the output of the base hash:

1. The maps f_1, f_2 from the proof of Lemma 9, followed by truncation to get $(y_1, \ldots, y_{v-1}) \in [n]^{v-1}$. It has been shown in Lemma 9 that f_1, f_2 are admissible and the non-abort probability for these is $\prod_i \left(1 - \frac{r_i}{p}\right)$. Truncation is clearly admissible and does not introduce aborts.
2. Sorting $(y_1, \ldots, y_{v-1})$ to get the sequence $(y_{(1)}, \ldots, y_{(v-1)})$. Formally, this maps from the subset of vectors in $[n]^{v-1}$ without duplicates to the set $\binom{[n]}{v-1}$. As the input contains no duplicates, sorting is regular. This is also sampleable by applying a random permutation. The step introduces no additional aborts.
3. Taking the differences of the $y_{(i)}$ is an efficiently invertible bijection and therefore admissible. Again, there are no additional aborts introduced here. The result of this step is an output in $\mathcal{C}_{\mathbb{Z}}$. As this is a bijection, we also know that this set has size $\binom{n}{v-1} = \binom{T+v-1}{v-1}$.
4. The final function is the identity on $\mathcal{C}$, which is just formally introduced to model aborts from being in $\mathcal{C}_{\mathbb{Z}} \setminus \mathcal{C}$. The ratio of $\mathcal{C}$ and $\mathcal{C}_{\mathbb{Z}}$ is exactly $\tilde{\ell}_T / \binom{T+v-1}{v-1}$.

□

Example 3 (Hashing-Optimized Hypercube in [DKKW25b] + KoalaBear). Consider the same hypercube and base hash function as in Example 1, i.e., $p = 2^{31} - 2^{24} + 1 = 2130706433$, $v = 64, \mathsf{w} = 8$. Also, consider the target sum $T = 77$ defining the hypercube layer $\mathcal{C}$ and $\mathcal{C}_{\mathbb{Z}}$. The first step of our algorithm generates $v - 1$ distinct elements out of $n = T + v - 1 = 140$ elements. Assume that the technique outlined in Remark 11 is used for that. That is, we need to partition the set $\{n - (v-1) + 1, \ldots, n\}$ into disjoint sets and then p accordingly. Here $n - (v-1) + 1$, and we can use the following decompositions of p, which we have found by a naive algorithm:

$$
\begin{aligned}
2130706433 &= 12 \cdot (140 \cdot 124 \cdot 108 \cdot 92) + 60838913 \\
2130706433 &= 12 \cdot (139 \cdot 123 \cdot 107 \cdot 91) + 133024565 \\
2130706433 &= 13 \cdot (138 \cdot 122 \cdot 106 \cdot 90) + 42705713 \\
2130706433 &= 13 \cdot (137 \cdot 121 \cdot 105 \cdot 89) + 116849588 \\
2130706433 &= 14 \cdot (136 \cdot 120 \cdot 104 \cdot 88) + 39657473 \\
2130706433 &= 14 \cdot (135 \cdot 119 \cdot 103 \cdot 87) + 115287923 \\
2130706433 &= 15 \cdot (134 \cdot 118 \cdot 102 \cdot 86) + 50163473 \\
2130706433 &= 15 \cdot (133 \cdot 117 \cdot 101 \cdot 85) + 126838658 \\
2130706433 &= 16 \cdot (132 \cdot 116 \cdot 100 \cdot 84) + 72773633 \\
2130706433 &= 17 \cdot (131 \cdot 115 \cdot 99 \cdot 83) + 26291648 \\
2130706433 &= 17 \cdot (130 \cdot 114 \cdot 98 \cdot 82) + 106116593 \\
2130706433 &= 18 \cdot (129 \cdot 113 \cdot 97 \cdot 81) + 69139631 \\
2130706433 &= 19 \cdot (128 \cdot 112 \cdot 96 \cdot 80) + 38797313 \\
2130706433 &= 20 \cdot (127 \cdot 111 \cdot 95 \cdot 79) + 14746733 \\
2130706433 &= 20 \cdot (126 \cdot 110 \cdot 94 \cdot 78) + 98276033 \\
2130706433 &= 1681 \cdot (125 \cdot 109 \cdot 93) + 669308
\end{aligned}
$$

The success probability of generating the $v - 1$ distinct elements is $\prod_i (1 - r_i/p) \approx 0.5864$, and the ratio of $\mathcal{C}$ and $\mathcal{C}_{\mathbb{Z}}$ is $|\mathcal{C}|/|\mathcal{C}_{\mathbb{Z}}| \approx 0.6099$, resulting in a total abort probability of $\theta \approx 0.6423$. This means that just 2 to 3 attempts are sufficient in expectation.

Example 4 (Size-Optimized Hypercube in [DKKW25b] + KoalaBear). Now, consider the hypercube with $v = 32, \mathsf{w} = 26$, $T = 239$ so we need $n = T + v - 1 = 270$. Then the following decomposition works:

$$
\begin{aligned}
2130706433 &= 122 \cdot (270 \cdot 259 \cdot 248) + 14904353 \\
2130706433 &= 124 \cdot (269 \cdot 258 \cdot 247) + 5061977 \\
2130706433 &= 125 \cdot (268 \cdot 257 \cdot 246) + 12769433 \\
2130706433 &= 127 \cdot (267 \cdot 256 \cdot 245) + 3933953 \\
2130706433 &= 128 \cdot (266 \cdot 255 \cdot 244) + 12239873 \\
2130706433 &= 130 \cdot (265 \cdot 254 \cdot 243) + 4383533 \\
2130706433 &= 131 \cdot (264 \cdot 253 \cdot 242) + 13266449 \\
2130706433 &= 133 \cdot (263 \cdot 252 \cdot 241) + 6361805 \\
2130706433 &= 135 \cdot (262 \cdot 251 \cdot 240) + 17633 \\
2130706433 &= 32654 \cdot (261 \cdot 250) + 32933 \\
2130706433 &= 32911 \cdot (260 \cdot 249) + 48293
\end{aligned}
$$

The success probability of generating the $v - 1$ distinct elements is $\prod_i (1 - r_i/p) \approx 0.9662$. The ratio of $\mathcal{C}$ and $\mathcal{C}_{\mathbb{Z}}$ is $|\mathcal{C}|/|\mathcal{C}_{\mathbb{Z}}| \approx 0.2253$. We get $\theta \approx 0.7822$, and in expectation 4 to 5 attempts suffice.

7 Application: Hypercube Encodings

We now show an application of our framework to hypercube encodings. To recall, these are a crucial part of recent hash-based signature designs [DKKW25c]. As already explained in the introduction, the core

of these encodings is a hash function mapping into (parts) of a hypercube, while being implemented from a hash function mapping into a sequence of field elements. The aborting random oracle model in combination with our indifferentiability results enables us to get security bounds for such encodings, while ultimately only relying on the heuristic that the internal hash function is modeled as a random oracle.

7.1 Incomparable Encodings

Let us first recall the definition of incomparable encodings and their security notion. We follow the definitions in the recent proposal for Ethereum [DKKW25c], so that our results can directly be plugged into their framework.

Definition 15 (Incomparable Encoding Scheme)**.** An *incomparable encoding* (IE) with message length ℓ_{msg}, public parameter space $\mathcal{P}$, randomness space $\mathcal{R}$, lifetime L, base w, dimension v, and code $\mathcal{C} \subseteq \mathbb{Z}_{\mathsf{w}}^{v}$ is an efficiently computable function

$$\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{\ell_{\mathsf{msg}}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\bot\},$$

such that for every *distinct* codewords $x = (x_1, \ldots, x_v) \in \mathcal{C}$ and $x' = (x'_1, \ldots, x'_v) \in \mathcal{C}$, we have

$$(\exists i \in [v] : x_i < x'_i) \wedge (\exists i' \in [v] : x'_{i'} < x_{i'})\,.$$

Definition 16 (Target Collision Resistance for IE)**.** Let $\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{\ell_{\mathsf{msg}}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\bot\}$ be an incomparable encoding scheme with code $\mathcal{C} \subseteq \mathbb{Z}_{\mathsf{w}}^{v}$. Let $K \in \mathbb{N}$ be an integer. We say that IncEnc satisfies *target collision resistance*, if for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{t\text{-}coll\text{-}res}}_{\mathsf{IncEnc},K}(\mathcal{A}) := \Pr\left[\begin{array}{l} x^* \neq \bot \\ \wedge \quad \exists (\mathsf{m},\rho) \neq (\mathsf{m}^*,\rho^*) \text{ s.t. } (\mathsf{m},\rho,\mathsf{ep}^*,x^*) \in \mathcal{L} \end{array} \middle| \begin{array}{c} P \xleftarrow{\$} \mathcal{P}, \\ (\mathsf{m}^*,\rho^*,\mathsf{ep}^*) \leftarrow \mathcal{A}^{\mathsf{O}}(P), \\ x^* := \mathsf{IncEnc}(P,\mathsf{m}^*,\rho^*,\mathsf{ep}^*) \end{array} \right],$$

where oracle O maintains an initially empty list $\mathcal{L}$ and works as follows on input $\mathsf{m} \in \{0,1\}^{\ell_{\mathsf{msg}}}$, $\mathsf{ep} \in [L]$:

1. If there exists an entry $(\mathsf{m}', \rho, \mathsf{ep}, x) \in \mathcal{L}$ (with the same ep), then return $\bot$.
2. Set $\mathsf{ctr} := 0$ and $x := \bot$. While $\mathsf{ctr} < K$ and $x = \bot$:
   (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$.
   (b) Set $x := \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep})$.
   (c) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
3. If $x = \bot$, insert $(\mathsf{m}, \bot, \mathsf{ep}, \bot)$ into $\mathcal{L}$ and return $\bot$.
4. Else (note: $x \in \mathcal{C}$), insert $(\mathsf{m}, \rho, \mathsf{ep}, x)$ into $\mathcal{L}$ and return (x, ρ).

Lemma 11 (Informal, see [DKKW25c])**.** *If* IncEnc *is an incomparable encoding that satisfies target collision resistance, then the signature construction from [DKKW25c] yields a secure (synchronized) signature scheme. In particular, to get k bits of security, it is sufficient to determine v, w and the remaining parameters of* IncEnc *such that for any algorithm $\mathcal{A}$ running in time $\mathbf{T}(\mathcal{A}) \leq 2^k$, we have*

$$\mathsf{Adv}^{\mathsf{t\text{-}coll\text{-}res}}_{\mathsf{IncEnc},K}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5)}.$$

Then, the remaining parameters of the signature scheme can be determined from v, w as in [DKKW25c, DKKW25b].

Remark 13 (Number of Repetitions and Correctness). The maximum number of repetitions K is a parameter of the signature scheme and has an impact on the bound for target collision resistance of the encoding. Therefore, to set parameters, one first needs to know K. Intuitively, if the encoding aborts rarely, then a small K is sufficient, whereas otherwise K needs to be chosen larger. The concrete value of K can be determined in practice via experiments to achieve a reasonable correctness error.

7.2 Generalized Target Sum Encoding

We now generalize the target sum encoding from [DKKW25c, DKKW25b].

Encoding. Let $v, \mathsf{w}, T \in \mathbb{N}$ be integers such that $0 \leq T \leq v(\mathsf{w}-1)$. Let $\mathsf{H}_\perp \colon \mathcal{P} \times \mathcal{T} \times (\{0,1\}^{\ell_{\mathsf{msg}}} \times \mathcal{R}) \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function, where $\mathcal{H} \subseteq \mathbb{Z}_{\mathsf{w}}^v$. Define the code

$$\mathcal{C} := \left\{ (x_1, \ldots, x_v) \in \mathbb{Z}_{\mathsf{w}}^v \;\middle|\; \sum_{i=1}^{v} x_i = T \right\} \subseteq \mathbb{Z}_{\mathsf{w}}^v.$$

We define the encoding function

$$\mathsf{IncEnc}^{\mathsf{ts}}[\mathsf{H}_\perp, T] \colon \mathcal{P} \times \{0,1\}^{\ell_{\mathsf{msg}}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\perp\}.$$

It works as follows on input $P \in \mathcal{P}, \mathsf{m} \in \{0,1\}^{\ell_{\mathsf{msg}}}, \rho \in \mathcal{R}, \mathsf{ep} \in [L]$:

1. Set $h := \mathsf{H}_\perp(P, \mathsf{tweakm}(\mathsf{ep}), (\mathsf{m}, \rho))$.
2. If $h = \perp$ or $h \notin \mathcal{C}$, return $\perp$. Else, return h.

Here, we assume $\mathsf{tweakm} \colon [L] \to \mathcal{T}$ is a fixed publicly known injective mapping.

Analysis. Note that compared to previous encodings, the main difference here is that we use an aborting hash function. This enables efficient instantiations, and the analysis requires to use our new framework. To use it, we follow the following steps:

1. We show security of the encoding is tightly implied by a property of $\mathsf{H}_\perp$ (Definition 17 and Lemma 12). For this, we closely follow [DKKW25a], but with additionally checking that outputs do not abort.
2. We give a security bound for this property when $\mathsf{H}_\perp$ is modeled as an aborting random oracle (Lemma 13).
3. When we give instantiations of $\mathsf{H}_\perp$ from Poseidon, we use our generic indifferentiability results from Section 5 to translate this to security bounds that hold if Poseidon is modeled as a random oracle.

Note that one can use these bounds in combination with Lemma 11 to set parameters.

Definition 17 (Multi-Target Collision Resistance with Random Sampling)**.** Let $K \in \mathbb{N}$ be an integer. Let $\mathsf{H}_\perp \colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{H} \cup \{\perp\}$ be an aborting hash function. Let $\mathsf{Prop} \colon \mathcal{H} \to \{0,1\}$ be a function that represents some property on the output space. We say that $\mathsf{H}_\perp$ satisfies *(non-aborting) multi-target collision resistance with random sampling*, if for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}_\perp,\mathsf{Prop},K}(\mathcal{A}) := \Pr\left[\begin{array}{l} \exists (M,T,\rho) \in Q \text{ s.t. } \rho \neq \perp \wedge (M^*,\rho^*) \neq (M,\rho) \\ \quad \wedge\ \mathsf{H}_\perp(P,T,(M^*,\rho^*)) = \mathsf{H}_\perp(P,T,(M,\rho)) \neq \perp \end{array} \;\middle|\; \begin{array}{c} P \xleftarrow{\$} \mathcal{P}, \\ (M^*,\rho^*) \leftarrow \mathcal{A}^{\mathsf{O}}(P) \end{array} \right].$$

where the oracle O maintains an initially empty list Q and works as follows on input $(T, M) \in \mathcal{T} \times \mathcal{M}$:

1. If there is a tuple $(T, M', \rho') \in Q$, for some M', ρ', then return $\perp$.
2. Otherwise, set $\mathsf{ctr} := 0$ and $x := \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
   (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$.
   (b) Set $x := \mathsf{H}_\perp(P, T, (M, \rho))$.
   (c) If $x \neq \perp$ and $\mathsf{Prop}(x) = 1$: Insert (T, M, ρ) into Q.
   (d) Else: Set $x := \perp$, $\rho := \perp$.
   (e) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
3. If $x = \perp$: Insert $(T, M, \perp)$ into Q.

4. Output (x, ρ).

Lemma 12 (Security of Encoding). *Consider the function* $\mathsf{IncEnc}^{ts}[\mathsf{H}_\perp, T]$ *as defined above, and any* $K \in \mathbb{N}$. *Let* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *be the predicate that outputs* 1 *if and only if its input is in* $\mathcal{C}$. *Then, for every algorithm* $\mathcal{A}$, *there is an algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *making exactly as many queries as* $\mathcal{A}$, *and such that*

$$\mathsf{Adv}^{\mathsf{t\text{-}coll\text{-}res}}_{\mathsf{IncEnc}^{ts}[\mathsf{H}_\perp,T],K}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}_\perp,\mathsf{Prop},K}(\mathcal{B}).$$

Proof. This is an easy proof by reduction, proven as in Lemma 8 in [DKKW25a]. □

Lemma 13 (Bound in the Aborting ROM). *Let* $K \in \mathbb{N}$ *be an integer. Let* $\mathsf{H}_\perp\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{H} \cup \{\perp\}$ *be an aborting hash function modeled as a* θ*-aborting random oracle, where* θ *is rational. Let* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *be the predicate that outputs* 1 *if and only if its input is in* $\mathcal{C}$. *Then, for any algorithm* $\mathcal{A}$ *that makes at most* q *classical queries to* $\mathsf{H}_\perp$ *and at most* p *classical queries to* O, *we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}_\perp,\mathsf{Prop},K}(\mathcal{A}) \le \frac{pK \cdot q'}{|\mathcal{R}|} + (1-\theta)\frac{q'+1}{|\mathcal{H}|},$$

where $q' := q + pK$. *On the other hand, if* $\mathcal{A}$ *makes at most* q *quantum queries to* $\mathsf{H}_\perp$ *and at most* p *classical queries to* O, *we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}_\perp,\mathsf{Prop},K}(\mathcal{A}) \le 2pK \cdot \sqrt{\frac{q'}{|\mathcal{R}|}} + 8(1-\theta)\frac{(q'+1)^2}{|\mathcal{H}|}.$$

Proof. The proof is postponed to Appendix C. □

Remark 14. Note that the bound in Lemma 13 matches the known bound in the standard non-aborting random oracle model [DKKW25c], except that we have the additional $(1-\theta)$ factor. In other words, we get the same bound as if we were to hash into a larger space of size $|\mathcal{H}|/(1-\theta)$.

7.3 Encoding Instantiations

We now give instantiations of the hash function $\mathsf{H}_\perp$ used in the encoding $\mathsf{IncEnc}^{\mathsf{ts}}[\mathsf{H}_\perp, T]$, by making black-box use of a hash function $\mathsf{H}\colon \mathcal{D} \to \mathbb{Z}_p^\ell$, which can be thought as being Poseidon [GKS23] for concreteness. Then, we apply indifferentiability result to turn the bound from Lemma 13 into a bound that holds when Poseidon is modeled as a random oracle. Thus, the parameters for our encodings can be set using the same heuristic as in prior work [DKKW25c, DKKW25b]. To reiterate: the advantage of our encodings is that they avoid big integer arithmetic, making them more proving-friendly.

Mapping into the Full Hypercube. This instantiation uses the function $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}$ from Section 6.1. It is indifferentiable from a θ-aborting random oracle by Lemma 8, and so we get the following bound.

Corollary 1 (Bound in the ROM). *Fix* $K, Q, v, \mathsf{w}, z, \alpha \in \mathbb{N}$ *such that* $p = Q\mathsf{w}^z + \alpha$. *Assume that* $\mathsf{H}\colon \mathcal{D} \to \mathbb{Z}_p^\ell$ *is modeled as a random oracle, for* $\mathcal{D} = \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R})$. *Let* $\mathsf{H}^{hc}_{\mathsf{w},v,z,Q}\colon \mathcal{D} \to \mathbb{Z}_{\mathsf{w}}^v \cup \{\perp\}$ *be the function obtained from* H *as in Section 6.1. Let* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *be the predicate that outputs* 1 *if and only if its input is in* $\mathcal{C}$. *Then, for any algorithm* $\mathcal{A}$ *that makes at most classical* q *queries to* H *and at most* p *classical queries to* O, *we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}^{hc}_{\mathsf{w},v,z,Q},\mathsf{Prop},K}(\mathcal{A}) \le \varepsilon^* + \frac{pK \cdot q'}{|\mathcal{R}|} + \left(\frac{Q\mathsf{w}^z}{p}\right)^\ell \frac{q'+1}{\mathsf{w}^v},$$

where $q' := q + pK$ *and* $\varepsilon^* > 0$ *can be made arbitrarily small. On the other hand, if* $\mathcal{A}$ *makes at most* q *quantum queries to* H *and at most* p *classical queries to* O, *we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}^{hc}_{\mathsf{w},v,z,Q},\mathsf{Prop},K}(\mathcal{A}) \le \varepsilon^* + 2pK \cdot \sqrt{\frac{q'}{|\mathcal{R}|}} + 8\left(\frac{Q\mathsf{w}^z}{p}\right)^\ell \frac{(q'+1)^2}{\mathsf{w}^v},$$

where $q' := 2q + pK$ *and* $\varepsilon^* > 0$ *can be made arbitrarily small.*

Proof. This is a standard application of indifferentiability, as shown by Lemma 8, to translate the bound from Lemma 13 into a bound in the random oracle model. For concreteness, we explain how that works: We start in the game that we want to analyze, i.e., the multi-target collision resistance with random sampling game from Definition 17 for the hash function $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}\colon \mathcal{D} \to \mathbb{Z}_{\mathsf{w}}^{v} \cup \{\perp\}$, which is constructed from the random oracle H. That is, in this game, H is a random oracle to which $\mathcal{A}$ has access and that is used to implement $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}$. We now consider an alternative game, in which $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}$ is a θ-aborting random oracle, and the oracle that models H is implemented by $\mathsf{Sim}^{\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}}$, where Sim is the indifferentiability simulator (cf. Definition 10) that exists because of Lemma 8. In particular, each query that $\mathcal{A}$ makes to H is answered by Sim, which makes exactly one query to $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}$ to answer the query. Whenever the game wants to evaluate $\mathsf{H}^{\mathsf{hc}}_{\mathsf{w},v,z,Q}$, it evaluates it directly by calling the oracle instead of evaluating it via H. It can be seen by an easy reduction to indifferentiability that the probability that the adversary wins is ε^*-close in the two games. Finally, we can bound the probability that the adversary wins our alternative game using Lemma 13. To this end we build a reduction that plays in the game in Lemma 13, which runs $\mathcal{A}$, and provides H-oracle access to $\mathcal{A}$ via Sim. Note that the reduction makes exactly as many queries as $\mathcal{A}$ does. □

Mapping into A Layer Of the Hypercube. Using the construction from Section 6.3, we can also directly map into the desired layer of the hypercube. We get the following corollary.

Corollary 2 (Bound in the ROM)**.** *Fix parameters s_j, r_j, Q_j as in Section 6.3 Assume that $\mathsf{H}\colon \mathcal{D} \to \mathbb{Z}_p^{\ell}$ is modeled as a random oracle, for $\mathcal{D} = \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R})$. Let $\mathsf{H}^{lay}_{\mathsf{w},v,T}\colon \mathcal{D} \to \mathcal{C} \cup \{\perp\}$. be the function obtained from H as in Section 6.3. Let $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ be the predicate that outputs 1 if and only if its input is in $\mathcal{C}$. Then, for any algorithm $\mathcal{A}$ that makes at most q classical queries to $\mathsf{H}_{\perp}$ and at most p classical queries to O, we have*

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}^{lay}_{\mathsf{w},v,T},\mathsf{Prop},K}(\mathcal{A}) \leq \varepsilon^* + \frac{pK \cdot q'}{|\mathcal{R}|} + \underbrace{\prod_{i\in[\ell]}\left(1 - \frac{r_i}{p}\right)}_{\leq 1} \frac{q'+1}{\binom{T+v-1}{v-1}},$$

where $q' := q + pK$ and $\varepsilon^* > 0$ can be made arbitrarily small. On the other hand, if $\mathcal{A}$ makes at most q quantum queries to H and at most p classical queries to O, we have

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}^{lay}_{\mathsf{w},v,T},\mathsf{Prop},K}(\mathcal{A}) \leq \varepsilon^* + 2pK \cdot \sqrt{\frac{q'}{|\mathcal{R}|}} + 8 \cdot \underbrace{\prod_{i\in[\ell]}\left(1 - \frac{r_i}{p}\right)}_{\leq 1} \frac{(q'+1)^2}{\binom{T+v-1}{v-1}},$$

where $q' := 2q + pK$ and $\varepsilon^* > 0$ can be made arbitrarily small.

Proof. Again, this follows easily via indifferentiability (Lemma 10), and the bound in the aborting random oracle model (Lemma 13). We refer to the proof of Corollary 1 for a more detailed explanation. □

8 Application: Grinding in Fiat-Shamir

Grinding is a powerful technique that is used in practice [Sta21] to boost the concrete security level of succinct argument systems obtained via the Fiat-Shamir transform [FS87], while only making minimal sacrifices in prover efficiency. Despite being a central component of these practical systems, it is often treated informally as an extension of the protocol. In this section, we show how our aborting random oracle model can be used to formally reason about grinding, while ultimately obtaining security bounds in the standard random oracle model using our indifferentiability results. We think that this offers a new way of looking at grinding that is conceptually simple and easy to analyze.

8.1 Interactive Proofs and Round-By-Round Soundness

Elaborating on what we have said above, grinding is used as a security boosting trick when compiling (public coin) interactive proofs that satisfy the notion of round-by-round soundness [CCH+19, Hol19] into non-interactive arguments. We define interactive proofs with round-by-round soundness next.

Definition 18 (**NP** Relation)**.** A relation $\mathcal{R}$, implicitly parameterized by the security parameter λ, that contains pairs $(\mathsf{stmt}, \mathsf{witn})$ of statements and witnesses is called an **NP** *relation*, if

- There exists a polynomial p_1 such that $|\mathsf{stmt}| \leq p_1(\lambda)$ for all $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$.
- There exists a polynomial p_2 such that $|\mathsf{witn}| \leq p_2(|\mathsf{stmt}|)$ for all $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$.
- The relation can be decided deterministically in polynomial time.

We define $\mathcal{L}_{\mathcal{R}}$ to be the set of all stmt such that there is a witn with $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$.

Definition 19 (Interactive Proofs)**.** Let $\mathcal{R}$ be an **NP** relation. An *interactive proof for* $\mathcal{R}$ with ℓ rounds and (finite) challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ is a pair of algorithms PPT $\mathsf{IP} = (\mathsf{P}, \mathsf{V})$, where V is deterministic, such that the following completeness property holds: For every $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$, we have

$$\Pr\left[\mathsf{V}(\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell) = 1 \;\middle|\; \begin{array}{c} \pi_1 \leftarrow \mathsf{P}(\mathsf{stmt}, \mathsf{witn}),\ \alpha_1 \xleftarrow{\$} \mathcal{C}_1, \\ \pi_2 \leftarrow \mathsf{P}(\mathsf{stmt}, \mathsf{witn}, \pi_1, \alpha_1),\ \alpha_2 \xleftarrow{\$} \mathcal{C}_2, \\ \vdots \\ \pi_\ell \leftarrow \mathsf{P}(\mathsf{stmt}, \mathsf{witn}, \pi_1, \alpha_1, \ldots, \pi_{\ell-1}, \alpha_{\ell-1}),\ \alpha_\ell \xleftarrow{\$} \mathcal{C}_\ell, \end{array}\right] = 1.$$

Definition 20 (Round-by-Round Soundness)**.** Let $\mathcal{R}$ be an **NP** relation and $\mathsf{IP} = (\mathsf{P}, \mathsf{V})$ be an interactive proof for $\mathcal{R}$ with ℓ rounds and challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$. We say that IP is $(\epsilon_1, \ldots, \epsilon_\ell)$-round-by-round sound if there is a (not necessarily efficient) function state such that:

- If $\mathsf{stmt} \notin \mathcal{L}_{\mathcal{R}}$, then $\mathsf{state}(\mathsf{stmt}) = 0$.
- For every prover-turn transcript $\mathsf{tr} = (\mathsf{stmt}, \ldots, \alpha_i)$ and every prover message π_{i+1}, we have

$$\mathsf{state}(\mathsf{tr} \parallel \pi_{i+1}) = \mathsf{state}(\mathsf{tr}).$$

- For every round-i verifier-turn transcript $\mathsf{tr} = (\mathsf{stmt}, \ldots, \pi_i)$ with $\mathsf{state}(\mathsf{tr}) = 0$, we have

$$\Pr_{\alpha_i \xleftarrow{\$} \mathcal{C}_i}[\mathsf{state}(\mathsf{tr} \parallel \alpha_i) = 1] \leq \epsilon_i.$$

- If $\mathsf{tr} = (\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell)$ is a full transcript and $\mathsf{state}(\mathsf{tr}) = 0$, then

$$\mathsf{V}(\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell) = 0.$$

8.2 Compilation to Non-Interactive Arguments Using Grinding

One can compile an interactive proof into a non-interactive argument in the random oracle model by relying on the well-known Fiat-Shamir paradigm. To present this formally, let us first define non-interactive arguments and their soundness. We define them with respect to an oracle, which will later be a random oracle or an aborting random oracle, or even a collection of such oracles.

Definition 21 (Non-Interactive Argument)**.** Let $\mathcal{R}$ be an **NP** relation and O be an oracle. A *non-interactive argument for* $\mathcal{R}$ is a pair of PPT algorithms $\mathsf{NIA} = (\mathsf{Prove}, \mathsf{Verify})$, where Verify is deterministic, which both have oracle access to O, and satisfy the following properties:

- **Completeness.** For every $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$, we have

$$\Pr\left[\mathsf{Verify}^{\mathrm{O}}(\mathsf{stmt}, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}^{\mathrm{O}}(\mathsf{stmt}, \mathsf{witn})\right] \geq 1 - \mathsf{negl}(\lambda),$$

  where the probability is taken over O and the coins of Prove.

- **Soundness.** For every PPT algorithm $\mathcal{A}$, the following advantage is negligible

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIA}}(\mathcal{A}) := \Pr\left[\mathsf{Verify}^{\mathrm{O}}(\mathsf{stmt}, \pi) = 1 \wedge \mathsf{stmt} \notin \mathcal{L}_{\mathcal{R}} \mid (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{\mathrm{O}}(1^\lambda)\right].$$

Fiat-Shamir. The Fiat-Shamir transform now works as follows, for a given interactive proof: each verifier message is replaced by a random oracle call that takes as input the current partial transcript. More formally, let $\mathsf{IP} = (\mathsf{P}, \mathsf{V})$ be an interactive proof for some **NP** relation $\mathcal{R}$ with ℓ rounds and challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$. Let $\mathsf{H}_1, \ldots \mathsf{H}_\ell$ be random oracles, where $\mathsf{H}_i \colon \{0,1\}^* \to \mathcal{C}_i$ can take as input partial transcripts[13] Construct the following non-interactive proof system $\mathsf{FS}[\mathsf{IP}, (\mathsf{H}_i)_i]$:

- $\mathsf{FS}[\mathsf{IP}, (\mathsf{H}_i)_i].\mathsf{Prove}^{\mathsf{H}_1,\ldots\mathsf{H}_\ell}(\mathsf{stmt}, \mathsf{witn})$:
  1. For $i = 1, \ldots, \ell$:
     (a) Compute $\pi_i \leftarrow \mathsf{P}(\mathsf{stmt}, \mathsf{witn}, (\pi_j)_{j<i})$.
     (b) Set $\alpha_i := \mathsf{H}_i(\mathsf{stmt}, (\alpha_j)_{j<i}, (\pi_j)_{j\le i})$.
  2. Return $\pi := (\pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell)$.
- $\mathsf{FS}[\mathsf{IP}, (\mathsf{H}_i)_i].\mathsf{Verify}^{\mathsf{H}_1,\ldots\mathsf{H}_\ell}(\mathsf{stmt}, \pi)$:
  1. Parse $\pi = (\pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell)$.
  2. For $i = 1, \ldots, \ell$: If $\alpha_i \neq \mathsf{H}_i(\mathsf{stmt}, (\alpha_j)_{j<i}, (\pi_j)_{j\le i})$, return 0.
  3. Return $\mathsf{V}(\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell)$.

Remark 15 (Other Fiat-Shamir Variants). Our presentation uses the variant of Fiat-Shamir in which the full transcript is included in the hash H_i. In particular, we include the verifier messages $(\alpha_j)_{j<i}$ explicitly. A variant more commonly used in practice does not include these explicitly, but instead they are derived iteratively from the prover messages during verification. We opted for this variant to simplify presentation of the proof (it makes the definition of event $\mathsf{Bad}_{i,j}$ more convenient), and our results translate to the variant without the explicit $(\alpha_j)_{j<i}$ as well. We leave working out the details for future work.

Using Round-By-Round Soundness. Round-by-round soundness is exactly the right notion to analyze this: for each random oracle query, which specifies a partial transcript for round i, the probability that this transcripts moves from state 0 to state 1 when being extended with the verifier message is at most the round-by-round soundness error ϵ_i. Thus, the probability that the adversary ever moves from state 0 to state 1 is at most $\sum_i q_i \epsilon_i$, where q_i is the number of random oracle queries made for round i.

Grinding. In practice, it may often be that the ϵ_i do not yield the desired security level, e.g., they just yield 110 bits ($\epsilon_i \le 2^{-110}$), but 128 bits are desired. One prominent way of boosting the security level is by relying on grinding: each random oracle call for the round that should be boosted would now take as input not only the transcript but also a salt, and the output would be the verifier message and *at the same time* a string $w \in \{0,1\}^\tau$. The non-interactive verifier would now additionally check that $w = 0^\tau$. Intuitively, this means that an adversary needs to do 2^τ *times* as much work as before. On the other hand, the honest prover can just try 2^τ different salts. In other words, we have added τ bits of security, while only increasing the running time by an additive 2^τ grinding step.

Using Aborting ROs. We observe that we can formally analyze grinding by just replacing the random oracle H in the Fiat-Shamir transform with a θ-aborting random oracle $\mathsf{H}_\perp$, for $\theta = 1 - 2^{-\tau}$. Later, we can use our indifferentiability result from Section 5 to get a bound in the random oracle model. To do this, we rewrite the Fiat-Shamir transform so that it also works with aborting random oracles and incorporates the retries. We use $\mathcal{S}$ to denote the sufficiently large space of salts, and denote this modified Fiat-Shamir transform by $\mathsf{FS}^\star[\mathsf{IP}, (\mathsf{H}_{\perp i})_i]$. It makes use of θ_i-aborting random oracles $\mathsf{H}_{\perp i}$, and we highlight the differences to the previous version:

- $\mathsf{FS}^\star[\mathsf{IP}, (\mathsf{H}_{\perp 1})_i].\mathsf{Prove}^{\mathsf{H}_{\perp 1},\ldots\mathsf{H}_{\perp \ell}}(\mathsf{stmt}, \mathsf{witn})$:
  1. For $i = 1, \ldots, \ell$:

[13] For simplicity, we just write $\{0,1\}^*$ for the input space. Formally, the input space could be the finite set of all transcripts, assuming an upper bound on the prover message length is given.

(a) Compute $\pi_i \leftarrow \mathsf{P}(\mathsf{stmt}, \mathsf{witn}, (\pi_j)_{j<i})$.
(b) While $\alpha_i = \bot$: Sample $s_i \xleftarrow{\$} \mathcal{S}$ and set $\alpha_i := \mathsf{H}_{\bot i}(\mathsf{stmt}, (\alpha_j)_{j<i}, (\pi_j, s_j)_{j\le i})$.

2. Return $\pi := (\pi_1, \alpha_1 \ldots, \pi_\ell, \alpha_\ell, s_1, \ldots, s_\ell)$.

- $\mathsf{FS}^\star[\mathsf{IP}, (\mathsf{H}_{\bot 1})_i].\mathsf{Verify}^{\mathsf{H}_{\bot 1}, \ldots \mathsf{H}_{\bot \ell}}(\mathsf{stmt}, \pi)$:

1. Parse $\pi = (\pi_1, \alpha_1 \ldots, \pi_\ell, \alpha_\ell, s_1, \ldots, s_\ell)$.
2. For $i = 1, \ldots, \ell$: If $\alpha_i \neq \mathsf{H}_{\bot i}(\mathsf{stmt}, (\alpha_j)_{j<i}, (\pi_j, s_j)_{j\le i})$ or $\alpha_i = \bot$, return 0.
3. Return $\mathsf{V}(\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \pi_\ell, \alpha_\ell)$.

Remark 16. Strictly speaking, this does not satisfy our definition of an argument system, as the prover is only expected polynomial time. But we can simply stop it at some point and get overwhelming completeness. We ignore this detail for now.

Now, we show soundness of this transform in the aborting random oracle model, and then lift the result to the standard random oracle model using our indifferentiability results from Section 5.

Lemma 14 (Soundness in the Aborting ROM)**.** *Let $\mathsf{H}_{\bot i}$ be θ_i-aborting random oracles for $i \in [\ell]$, and assume that IP is $(\epsilon_1, \ldots, \epsilon_\ell)$-round-by-round sound for negligible ϵ_i. Then, $\mathsf{FS}^\star[\mathsf{IP}, (\mathsf{H}_i)_i]$ is sound. More concretely, for every algorithm $\mathcal{A}$ that makes at most q_i queries to $\mathsf{H}_{\bot i}$, we have*

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{FS}^\star[\mathsf{IP}, (\mathsf{H}_i)_i]}(\mathcal{A}) \le \sum_{i=1}^{\ell} (q_i + 1) \cdot (1 - \theta_i)\epsilon_i.$$

Proof. Let state denote the function that exists according to the round-by-round soundness of IP. Consider the soundness experiment in which $\mathcal{A}$ has access to θ_i-aborting random oracles $\mathsf{H}_{\bot i}$. Without loss of generality, assume that all oracle queries are distinct. Define the following bad events:

- Event $\mathsf{Bad}_{i,j}$: The j-th query to $\mathsf{H}_{\bot i}$ is of the form $\alpha_i = \mathsf{H}_{\bot i}(\mathsf{stmt}, (\alpha_j)_{j<i}, (\pi_j, s_j)_{j\le i})$ with (1) $\alpha_i \neq \bot$, (2) $\mathsf{state}(\mathsf{tr}) = 0$, but (3) $\mathsf{state}(\mathsf{tr} \parallel \alpha_i) = 1$, where $\mathsf{tr} = (\mathsf{stmt}, \pi_1, \alpha_1, \ldots, \alpha_{i-1}, \pi_i)$.

First, one can see that if we condition on the event that $\mathsf{Bad}_{i,j}$ does not occur for any i, j, then the probability of $\mathcal{A}$ breaking the soundness game is 0. This is because if the verifier accepts, then the full transcript that it inputs into V must be in state 1. But as no $\mathsf{Bad}_{i,j}$ occurs and the verifier asserts that no α_i is $\bot$, we never move from state 0 to state 1 when we extend this transcript starting from (stmt) one component after another. This means that the initial transcript (stmt) that just contains the statement must be in state 1. Therefore, $\mathsf{stmt} \in \mathcal{L}_{\mathcal{R}}$, so $\mathcal{A}$ does not win.

Now, let us analyze the probability of $\mathsf{Bad}_{i,j}$ for a fixed i, j. First of all, the probability that $\alpha_i \neq \bot$ is $1 - \theta_i$. Conditioned on that, α_i is sampled at uniformly random from $\mathcal{C}_i$ and independently of tr. Therefore, we can apply round-by-round soundness to argue that (2) and (3) occur with probability at most ϵ_i. With a union bound over all queries (there are q_i made by $\mathcal{A}$, and one by the game to check the winning condition) and oracles, we get the result. □

We can now derive a corollary that shows that grinding as implemented in practice and discussed above is secure in the random oracle model. For that, we simply instantiate the aborting random oracles in a specific way.

Corollary 3 (Soundness in the ROM)**.** *For each $i \in [\ell]$, assume $\tau_i \in \mathbb{N}$ is fixed. Consider the following random oracles H_i and functions f_i:*

$$\mathsf{H}_i \colon \{0,1\}^* \to \mathcal{C}_i \times \{0,1\}^{\tau_i},$$
$$f_i \colon \mathcal{C}_i \times \{0^{\tau_i}\} \to \mathcal{C}_i,\ (\alpha_i, 0^{\tau_i}) \mapsto \alpha_i,$$

Assume that IP is $(\epsilon_1, \ldots, \epsilon_\ell)$-round-by-round sound for negligible ϵ_i. Then, $\mathsf{FS}^\star[\mathsf{IP}, (f_i \circ \mathsf{H}_i)_i]$ is sound. More concretely, for every algorithm $\mathcal{A}$ that makes at most q_i queries to $\mathsf{H}_{\bot i}$, we have

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{FS}^\star[\mathsf{IP}, (f_i \circ \mathsf{H}_i)_i]}(\mathcal{A}) \le \varepsilon^* + \sum_{i=1}^{\ell} (q_i + 1) \cdot 2^{-\tau_i}\epsilon_i,$$

where $\varepsilon^* > 0$ can be made arbitrarily small.

Proof. This follows because f_i is an admissible function for each i. Hence, using Theorem 3 it is indifferentiable from a θ_i-aborting random oracle with $\theta_i = 1 - 2^{-\tau_i}$. Applying Lemma 14 yields the result. Let us more precise: We want to bound the advantage of an algorithm $\mathcal{A}$ in the soundness game, when the H_i are modeled as a random oracle. To this end, we first switch to a hybrid game in which the functions $f_i \circ \mathsf{H}_i$ are replaced by θ_i-aborting random oracles, and the H_i are simulated using the indifferentiability simulator, which has black-box access to the aborting random oracles. This switch is possible with only a ε^*-change in advantage due to our generic indifferentiability result (Theorem 3), which can be applied as the f_i are admissible. At that point, we can build a reduction that itself plays the role of the adversary in Lemma 14, and simulates the hybrid game for $\mathcal{A}$. □

Remark 17 (On the concrete ε^*). Looking into the proof of Theorem 3, we can make ε^* more precise. Namely, we apply it for each $i \in [\ell]$, and get a bound of the form $(q_i+1) \cdot (\varepsilon_{1,i} + (1-\theta_i)^{T_i} + T_i\varepsilon_{2,i})$, where T_i is a parameter that the reduction can freely pick, trading running time for success probability. For this specific admissible function f_i, it is easy to see that $\varepsilon_{1,i} = 0$ (sampling a preimage just corresponds to concatenation with 0^{τ_i}) and $\varepsilon_{2,i} = 0$ (assuming that one can efficiently sample uniformly from $\mathcal{C}_i \times \{0,1\}^{\tau_i}$). Also, as $\theta_i = 1 - 2^{-\tau_i}$, the bound becomes (over all rounds)

$$\varepsilon^* = \sum_{i=1}^{\ell} (q_i + 1) 2^{-\tau_i T_i}.$$

This is essentially 0 when setting T_i large enough.

References

[ABD+13] Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the indifferentiability of key-alternating ciphers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 531–550. Springer, Berlin, Heidelberg, August 2013. (Cited on Page 10.)

[ABR98] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, September 1998. (Cited on Page 3.)

[AMP10] Elena Andreeva, Bart Mennink, and Bart Preneel. On the indifferentiability of the Grøstl hash function. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 88–105. Springer, Berlin, Heidelberg, September 2010. (Cited on Page 16.)

[BBD+23] Manuel Barbosa, Gilles Barthe, Christian Doczkal, Jelle Don, Serge Fehr, Benjamin Grégoire, Yu-Hsuan Huang, Andreas Hülsing, Yi Lee, and Xiaodi Wu. Fixing and mechanizing the security proof of fiat-shamir with aborts and Dilithium. Cryptology ePrint Archive, Report 2023/246, 2023. (Cited on Page 11.)

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. (Cited on Page 3.)

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Berlin, Heidelberg, August 2010. (Cited on Page 3, 10, 16, 17.)

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016. (Cited on Page 3.)

[BDF+10] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. Cryptology ePrint Archive, Report 2010/428, 2010. (Cited on Page 44, 45.)

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Berlin, Heidelberg, December 2011. (Cited on Page 8.)

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129. Springer, Berlin, Heidelberg, November / December 2011. (Cited on Page 3, 11.)

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the Sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Berlin, Heidelberg, April 2008. (Cited on Page 10, 16.)

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Berlin, Heidelberg, August 2001. (Cited on Page 17.)

[BHRvV21] Joppe W. Bos, Andreas Hülsing, Joost Renes, and Christine van Vredendaal. Rapidly verifiable XMSS signatures. *IACR TCHES*, 2021(1):137–168, 2021. (Cited on Page 11, 45.)

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 671–684. ACM Press, June 2018. (Cited on Page 13.)

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001. (Cited on Page 3, 10, 21.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. (Cited on Page 3.)

[CCG+23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 379–404. Springer, Cham, April 2023. (Cited on Page 11.)

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. (Cited on Page 32.)

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Berlin, Heidelberg, August 2005. (Cited on Page 10.)

[CETU18] Tore Vincent Carstens, Ehsan Ebrahimi, Gelo Noel Tabia, and Dominique Unruh. On quantum indifferentiability. Cryptology ePrint Archive, Report 2018/257, 2018. (Cited on Page 10.)

[CHK+16] Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 29(1):61–114, January 2016. (Cited on Page 10.)

[CPZ24] Joseph Carolan, Alexander Poremba, and Mark Zhandry. (Quantum) indifferentiability and pre-computation. Cryptology ePrint Archive, Report 2024/1727, 2024. (Cited on Page 16.)

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010. (Cited on Page 11.)

[DKKW25a] Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. Hash-based multi-signatures for post-quantum ethereum. Cryptology ePrint Archive, Paper 2025/055, 2025. (Cited on Page 8, 30, 31, 45.)

[DKKW25b] Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. Technical note: LeanSig for post-quantum ethereum. Cryptology ePrint Archive, Paper 2025/1332, 2025. (Cited on Page 3, 4, 8, 11, 24, 28, 29, 30, 31.)

[DKKW25c] Justin Drake, Dmitry Khovratovich, Mikhail A. Kudinov, and Benedikt Wagner. Hash-based multi-signatures for post-quantum ethereum. *CiC*, 2(1):13, 2025. (Cited on Page 3, 4, 8, 11, 28, 29, 30, 31.)

[DLL+17] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. (Cited on Page 11.)

[DSSL16] Yevgeniy Dodis, Martijn Stam, John P. Steinberger, and Tianren Liu. Indifferentiability of confusion-diffusion networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 679–704. Springer, Berlin, Heidelberg, May 2016. (Cited on Page 10.)

[Dur64] Richard Durstenfeld. Algorithm 235: random permutation. Communications of the ACM, vol.7, no.7, p.420, 1964. (Cited on Page 9, 25.)

[FFS+13] Reza R. Farashahi, Pierre Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and J. Felipe Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Mathematics of Computation*, 82(281):491–512, 2013. (Cited on Page 3, 10.)

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999. (Cited on Page 3.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. (Cited on Page 9, 32.)

[GHHM21] Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 637–667. Springer, Cham, December 2021. (Cited on Page 8, 45.)

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203. Springer, Cham, July 2023. (Cited on Page 3, 4, 11, 15, 22, 31.)

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. (Cited on Page 3.)

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. (Cited on Page 11.)

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Berlin, Heidelberg, December 2016. (Cited on Page 11.)

[HK22] Andreas Hülsing and Mikhail A. Kudinov. Recovering the tight security proof of SPHINCS+. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 3–33. Springer, Cham, December 2022. (Cited on Page 11.)

[HKRY23] Andreas Hülsing, Mikhail A. Kudinov, Eyal Ronen, and Eylon Yogev. SPHINCS+C: Compressing SPHINCS+ with (almost) no cost. In *2023 IEEE Symposium on Security and Privacy*, pages 1435–1453. IEEE Computer Society Press, May 2023. (Cited on Page 8, 11.)

[Hol19] Justin Holmgren. On round-by-round soundness and state restoration attacks. Cryptology ePrint Archive, Report 2019/1261, 2019. (Cited on Page 32.)

[HRS16] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 387–416. Springer, Berlin, Heidelberg, March 2016. (Cited on Page 45.)

[JLLW23] Aayush Jain, Huijia Lin, Ji Luo, and Daniel Wichs. The pseudorandom oracle model and ideal obfuscation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 233–262. Springer, Cham, August 2023. (Cited on Page 11.)

[KKW25] Dmitry Khovratovich, Mikhail A. Kudinov, and Benedikt Wagner. At the top of the hypercube - better size-time tradeoffs for hash-based signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 93–123. Springer, Cham, August 2025. (Cited on Page 3, 4, 11.)

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Cham, April / May 2018. (Cited on Page 11.)

[KY76] Donald E. Knuth and Andrew C. Yao. The complexity of nonuniform random number generation. 1976. (Cited on Page 10, 16.)

[LW22] George Lu and Brent Waters. How to sample a discrete gaussian (and more) from a random oracle. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 653–682. Springer, Cham, November 2022. (Cited on Page 3, 10.)

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009. (Cited on Page 11.)

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979. (Cited on Page 3.)

[MMZ25] Mary Maller, Nicolas Mohnblatt, and Aranxta Zapico. Ivc in the open-and-sign random oracle model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 333–365. Springer, 2025. (Cited on Page 11.)

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Berlin, Heidelberg, February 2004. (Cited on Page 10, 15.)

[Nat15] National Institute of Standards and Technology. SHA-3 Standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standards Publication (FIPS)*, 2015. (Cited on Page 15.)

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Berlin, Heidelberg, May 2011. (Cited on Page 10.)

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990. (Cited on Page 3.)

[Sho06] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006. (Cited on Page 42.)

[Sta21] StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Report 2021/582, 2021. (Cited on Page 9, 32.)

[Tib12] Mehdi Tibouchi. A note on hashing to bn curves. *SCIS. IEICE*, 40, 2012. (Cited on Page 21.)

[ZCY23] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Revisiting the constant-sum winternitz one-time signature with applications to SPHINCS$^+$ and XMSS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 455–483. Springer, Cham, August 2023. (Cited on Page 8, 11.)

[Zha12] Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 758–775. Springer, Berlin, Heidelberg, August 2012. (Cited on Page 43.)

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 239–268. Springer, Cham, August 2019. (Cited on Page 10.)

[Zha22] Mark Zhandry. Augmented random oracles. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 35–65. Springer, Cham, August 2022. (Cited on Page 11.)

Appendix

A Quantum Indifferentiability

In this section, we discuss how the indifferentiability Theorem 3 generalizes to the quantum setting. In the quantum setting, we are interested in aborting *quantum* random oracles, to which (quantum) algorithms can make superposition queries. The result of such a query then may be a superposition of $\perp$ and non-$\perp$ results. As in the classical case, we are interested in aborting quantum random oracles constructed from standard quantum random oracles and the notion of indifferentiability we are interested in is given below in Definition 22, matching Definition 10.

Definition 22 (Indifferentiability from aborting QROM)**.** Let H be a function modeled as a (quantum) random oracle and let $\mathsf{G}\colon \mathcal{D} \to \mathcal{H} \cup \{\perp\}$ be a (classical) algorithm having oracle access to H. Then, we say that G^{H} is *quantum indifferentiable from a θ-aborting quantum random oracle when modeling H as a quantum random oracle*, if there is a (stateful) quantum polynomial time simulator Sim with quantum oracle access to $\mathsf{H}_\perp$, such for every quantum polynomial time algorithm $\mathcal{A}$ (taking only classical bits as input and output) that has quantum access to *both* its oracles, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{G},\mathsf{H},\mathsf{H}_\perp,\mathsf{Sim}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{\mathsf{H},\mathsf{G}^{\mathsf{H}}}(1^\lambda)=1\right] - \Pr\left[\mathcal{A}^{\mathsf{Sim}^{\mathsf{H}_\perp},\mathsf{H}_\perp}(1^\lambda)=1\right]\right|,$$

where $\mathsf{H}_\perp\colon \mathcal{D} \to \mathcal{H}\cup\{\perp\}$ is a θ-aborting quantum random oracle. Note that in the second experiment the simulator Sim answers (quantum) queries of $\mathcal{A}$ to the first oracle, while having itself quantum oracle access to $\mathsf{H}_\perp$. We call Sim the indifferentiability simulator and, as in Definition 10, we allow it to depend on an upper bound on the number of oracle queries of $\mathcal{A}$.

With this definition, we have a quantum analogue of Theorem 3 as follows.

Theorem 5 (Generic Indifferentiability – Quantum Case). *Let $\mathcal{X} := \mathcal{X}^\top \cup \mathcal{X}^\perp$ be a set, which is partitioned into two disjoint sets, i.e., $\mathcal{X}^\top \cap \mathcal{X}^\perp = \emptyset$. Further assume that $\mathcal{X}$ is near-perfectly sampleable. Let $\mathsf{H}\colon \mathcal{D} \to \mathcal{X}$ be a (non-aborting) hash function which is modeled as a (non-aborting) quantum random oracle, and let $f\colon \mathcal{X}^\top \to \mathcal{H}$ be a function.*

Then, if f is admissible, then $\mathsf{H}_\perp = f \circ \mathsf{H}$ is quantum indifferentiable from a θ-aborting random oracle when modeling H as a random oracle, for

$$\theta = |\mathcal{X}^\perp|/|\mathcal{X}|.$$

In particular, there is a simulator Sim with quantum oracle access to $\mathsf{H}_\perp$ such for every $\varepsilon^* > 0$, and every q and for every quantum polynomial time algorithm $\mathcal{A}$ making at most q quantum oracle queries, we have

$$\mathsf{Adv}^{\mathsf{indiff}}_{f\circ\mathsf{H},\mathsf{H},\mathsf{H}_\perp,\mathsf{Sim}}(\mathcal{A}) \le \varepsilon^*.$$

Further, to answer any (quantum) oracle query, Sim makes at most two (quantum) queries to $\mathsf{H}_\perp$ and runs in time at most $q\theta^{-1}\cdot \mathsf{poly}(\log(q), \log(1/\varepsilon^*), \log(|\mathcal{H}|), \log(1/\theta))$.

We note that the samplers that we require for the near-perfect sampleability of $\mathcal{X}$ and for the admissibility of f in the conditions of Theorem 5 are required to be classical algorithms. Also, the dependency on $\log(|\mathcal{H}|)$ in the running time of our simulator is an artifact of our proof. See Remark 18 below.

Proof. For the proof, we may assume that the domain $\mathcal{D}$ is finite and contained in $\mathcal{D} \subseteq \{0,1\}^n$ for some $n > 0$. We will discuss the case of infinite $\mathcal{D}$ afterwards in Remark 19. The construction of the required Sim is very similar to the construction the classical counterpart Theorem 3. Indeed, let us first assume that we have quantum oracle access to a sequence of random functions

$$\Psi^\perp\colon \mathcal{D} \to \mathcal{X}^\perp,$$
$$\Psi^\top_h\colon \mathcal{D} \to f^{-1}(h) \text{ for every } h \in \mathcal{H}.$$

Recall that we defined the simulator in the classical version as

- **Simulator** $\mathsf{Sim}^{\mathsf{H}_\perp}_{\text{classical}}$ **on input** $z \in \mathcal{D}$:

1. Query $h \coloneqq \mathsf{H}_\perp(z)$. We have $h \in \mathcal{H} \cup \{\perp\}$.
2. If $h = \perp$: Return $\Psi^\perp(z) \in \mathcal{X}^\perp$.
3. If $h \neq \perp$: Return $\Psi_h^\top(z) \in f^{-1}(h)$.

and we showed that this simulator achieves (classical) perfect indifferentiability, i.e., with an adversarial advantage of 0, if we had a *perfect* implementation of $\Psi^\perp$ and $\Psi_h^\top$. For the quantum case, we define our (quantum) simulator $\mathsf{Sim}_{\mathsf{quantum}}$ as the quantum analogue of this classical simulator, relative to $\Psi^\perp$ and $\Psi^\top$. For the sake of concreteness:

- **Simulator** $\mathsf{Sim}_{\mathsf{quantum}}^{\mathsf{H}_\perp}$ **on input** $z \in \mathcal{D}$:
  1. Query $h \coloneqq \mathsf{H}_\perp(z)$, stored in some ancillary registers.
  2. Query $x_1 \coloneqq \Psi^\perp(z) \in \mathcal{X}^\perp$ stored in some ancillary registers.
  3. Query $x_2 \coloneqq \Psi_h^\top(z) \in \mathcal{X}^\perp$ stored in some ancillary registers.
  4. Set the result of the computation to x_1 if $h = \perp$, and to x_2 otherwise.
  5. Uncompute x_2, x_1, h to reset the ancillary registers, by calling $\Psi_h^\top(z), \Psi^\perp(z), \mathsf{H}_\perp(z)$ again (in that order).

Then the analysis of the classical case implies that this simulator achieves perfect quantum indifferentiability if we had a perfect (quantum) implementation of $\Psi^\perp$ and $\Psi_h^\top$. To prove the theorem, will first show to implement $\Psi^\perp$ and $\Psi_h^\top$ in the quantum case. As in the classical case, these implementations will have some deviation from a perfectly uniform function $\mathcal{D} \to \mathcal{X}^\perp$ resp. $\mathcal{D} \to f^{-1}(h)$. We will then argue how these deviations translate to an advantage in the definition of quantum indifferentiability and see that the advantage can be made smaller than any $\varepsilon^* > 0$ while keeping the running time bounded by $q\theta^{-1} \cdot \mathsf{poly}(\log(1/\varepsilon^*), \log(q), \log(|\mathcal{H}|))$ as required.

Our construction will depend on some parameters $T, \varepsilon_{\mathcal{H}}^*, \varepsilon_2^*, \varepsilon_0^*$ that our error bounds and running times will subsequently depend on. We will show how to set these parameters at the end of the proof.

Since we need to provide quantum access to $\Psi^\perp$ resp. $\Psi_h^\top$, the lazy sampling strategy we used in the classical case no longer works. Instead, we use a $4q$-wise independent hash function to derive the randomness of a sampler into the desired target range. Let us start with the $\Psi^\perp$ case: Since $\mathcal{X}$ is near-sampleable, there exists a randomized algorithm $\mathsf{Samp}_{\mathcal{X}}$ taking arbitrary $\varepsilon_0^* > 0$ as input and outputting random element from $\mathcal{X}$ with a distribution ε_0^*-close to uniform in time $\mathrm{polylog}(1/\varepsilon_0^*)$. Since $\mathcal{X}^\top$ is efficiently decidable, we can construct a sampler $\mathsf{Samp}\colon \mathcal{D} \to \mathcal{X}^\perp$ via rejection sampling as follows: sample using $\mathsf{Samp}_{\mathcal{X}}(\varepsilon_0^*)$ and retry until the result is not in $\mathcal{X}^\top$, aborting after T unsuccessful trials. In the latter case, we just output some default output from $\mathcal{X}^\perp$. Let ε_1^* be the statistical distance between the output distribution of Samp and the uniform distribution on $\mathcal{X}^\perp$. By construction,

$$\varepsilon_1^* \leq (1-\theta)^T + T\varepsilon_0^*$$

Let $n_1 = T \cdot \mathrm{polylog}(1/\varepsilon_0^*)$ be the number of random bits used by this sampler. By derandomizing, we thus obtain a deterministic algorithm

$$\mathsf{Samp}_1\colon \{0,1\}^{n_1} \to \mathcal{X}^\perp$$

for some $n_1 = T \cdot \mathrm{polylog}(1/\varepsilon_0^*)$ running in time bounded by $T \cdot \mathrm{polylog}(1/\varepsilon_0^*)$.

Next, let $\mathcal{H}^{(n,n_1)}$ be a family of $4q$-wise independent hash functions $K : \{0,1\}^n \to \{0,1\}^{n_1}$. Concretely, for $n \geq n_1$, we use the family of functions given by identifying $\{0,1\}^n$ with a finite field with 2^n elements and taking the set of univariate polynomials of degree $\leq 4q-1$, followed by truncating the result to n_1 bits. For $n_1 > n$, we additionally identify the domain $\{0,1\}^n$ with a subset $\{0,1\}^n \subseteq \{0,1\}^{n_1}$ and also choose the family of univariate polynomial of degree $\leq 4q-1$ without the need for truncation. Note that we only evaluate the polynomials in points from $\{0,1\}^{n_1}$.

Now, we let Sim sample a function K_1 from $\mathcal{H}^{(n,n_1)}$ and define $\Psi^\perp$ as

$$\Psi^\perp(z) \coloneqq \mathsf{Samp}_1(K_1(z)) \ .$$

Here, sampling K_1 also involves choosing an identification $\{0,1\}^{n_1}$ with a finite field with 2^{n_1} elements, which can be done by choosing an irreducible polynomial over $\mathbb{F}_2$ of degree n_1. See [Sho06] for an

algorithm for doing so. This may fail (due to rejection sampling) with arbitrarily small error probability $\varepsilon^*_{\mathcal{H}} > 0$ and runs in time $\mathsf{poly}(n_1, \log(1/\varepsilon^*_{\mathcal{H}}))$. In case of such a failure, our simulator just aborts.

We use the same strategy for $\Psi^\top$: since f is admissible, there exists a randomized algorithm Samp' taking arbitrary ε^*_2 and h as input, and outputting random elements from $f^{-1}(h)$ with a distribution ε^*_2-close to uniform over $f^{-1}(h)$, in time $\mathsf{polylog}(1/\varepsilon^*_2)$. Let $n_2 = \mathsf{polylog}(1/\varepsilon^*_2)$ be the number of random bits used by this sampler. By derandomizing, this gives a deterministic algorithm

$$\mathsf{Samp}_2 \colon \mathcal{H} \times \{0,1\}^{n_2} \to \mathcal{X}$$

for some $n_2 = \mathsf{polylog}(1/\varepsilon^*_2)$ running in time bounded by $\mathsf{polylog}(1/\varepsilon^*_2)$.

Now, we let Sim sample a function K_2 from $\mathcal{H}^{(n,n_2)}$ and define $\Psi^\top$ as

$$\Psi^\top_h(z) \coloneqq \mathsf{Samp}_2(h, K_2(z)) \ .$$

Here, again, sampling K_2 incurs a global abort probability of $\varepsilon^*_{\mathcal{H}}$.

Let us analyse this construction now by a hybrid argument: let $\mathbf{G}_{\text{start}}$ be the game where the adversary $\mathcal{A}$ interacts with $\mathsf{Sim}^{\mathsf{H}_\perp}$ and $\mathsf{H}_\perp$ and $\mathbf{G}_{\text{end}}$ be the game where the adversary interacts with H and $f \circ \mathsf{H}$. Recall that we limit the number of queries of $\mathcal{A}$ to q, but not its running time, so our proof is information-theoretical.

Hybrid Game $\mathbf{G}_0$. First, we consider $\mathbf{G}_0$, defined as $\mathbf{G}_{\text{start}}$, but where the sampling of K_1 and K_2 never fails. By construction, the distinguishing advantage between $\mathbf{G}_0$ and $\mathbf{G}_{\text{start}}$ is at most $2\varepsilon^*_{\mathcal{H}}$.

Hybrid Game $\mathbf{G}_1$. Next, we consider $\mathbf{G}_1$, which differs from $\mathbf{G}_0$ by replacing K_1 with a uniformly chosen function $\widetilde{K}_1 \colon \{0,1\}^n \to \{0,1\}^{n_1}$. By [Zha12], a $4q$-wise independent hash function is perfectly indistinguishable from a uniformly random function by quantum algorithms that make at most $2q$ (quantum) queries to it. By construction, Sim makes at most $2q$ queries to $\Psi^\perp$, hence at most $2q$ queries to K_1 resp. $\widetilde{K}_1$, so the distinguishing advantage between $\mathbf{G}_0$ and $\mathbf{G}_1$ is 0.

Hybrid Game $\mathbf{G}_2$. $\mathbf{G}_2$ is obtained from $\mathbf{G}_1$ by changing from $\Psi^\perp$ to a random function $\widetilde{\Psi}^\perp \colon \mathcal{D} \to \mathcal{X}^\perp$ that samples each $\widetilde{\Psi}^\perp(z)$ iid. according to the output distribution of Samp. Since we use $\widetilde{K}_1$ to provide uniformly random coins to Samp_1 and Samp_1 is the derandomization of Samp, $\mathbf{G}_1$ is equivalent to $\mathbf{G}_2$, so the distinguishing advantage between $\mathbf{G}_1$ and $\mathbf{G}_2$ is 0.

Hybrid Game $\mathbf{G}_3$. Similarly to the transition from $\mathbf{G}_0$ to $\mathbf{G}_2$, we consider $\mathbf{G}_3$, which differs from $\mathbf{G}_2$ by replacing $\Psi^\top$ by a random function $\widetilde{\Psi}^\top$, where the $\widetilde{\Psi}^\top_h(z)$ are independent for each $h \in \mathcal{H}$ and $z \in \mathcal{D}$, and distributed according to the output distribution of $\mathsf{Samp}'(h, \varepsilon^*_2)$. Again, by [Zha12], the distinguishing advantage between $\mathbf{G}_2$ and $\mathbf{G}_3$ is 0.

Hybrid Game $\mathbf{G}_4$. Now, consider $\mathbf{G}_4$, which differs from $\mathbf{G}_3$ by changing $\widetilde{\Psi}^\perp$ to a perfectly uniformly random function, rather than one with output distribution determined by Samp. Since the statistical distance between the output distribution of Samp and uniform is at most ε^*_1, the distinguishing advantage between $\mathbf{G}_3$ and $\mathbf{G}_4$ is at most $4 \cdot (2q)^2 \sqrt{\varepsilon^*_1} = 16q^2\sqrt{\varepsilon^*_1}$ by Lemma 16.

Hybrid Game $\mathbf{G}_5$. Finally, we consider $\mathbf{G}_5$, which differs from $\mathbf{G}_4$ by changing $\widetilde{\Psi}^\top$ to a random function $\bar{\Psi}^\top$ where each $\bar{\Psi}^\top_h(z)$ is independent for $h \in \mathcal{H}, z \in \mathcal{D}$ and distributed uniformly in $f^{-1}(h)$. By applying Lemma 16 once for each $h \in \mathcal{H}$ (to change the oracles' distribution for one single h), the distinguishing advantage between $\mathbf{G}_4$ and $\mathbf{G}_5$ is at most $16|\mathcal{H}|q^2\sqrt{\varepsilon^*_2}$.

Total Advantage, Parameters, Running Time. By the argument we made in the proof of the classical version of this theorem, $\mathbf{G}_5$ and $\mathbf{G}_{\text{final}}$ are perfectly indistinguishable, so the total distinguishing advantage is

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{indiff}}_{f\circ\mathsf{H},\mathsf{H},\mathsf{H}_\perp,\mathsf{Sim}}(\mathcal{A}) &\leq 2\varepsilon^*_{\mathcal{H}} + 16q^2\sqrt{\varepsilon^*_1} + 16|\mathcal{H}|q^2\sqrt{\varepsilon^*_2} \\
&\leq 2\varepsilon^*_{\mathcal{H}} + 16q^2\sqrt{(1-\theta)^T + T\varepsilon^*_0} + 16|\mathcal{H}|q^2\sqrt{\varepsilon^*_2} \\
&\leq 2\varepsilon^*_{\mathcal{H}} + 16q^2(1-\theta)^{T/2} + 16q^2\sqrt{T\varepsilon^*_0} + 16|\mathcal{H}|q^2\sqrt{\varepsilon^*_2}.
\end{aligned}$$

We now set parameters as

$$\begin{aligned}
\varepsilon^*_{\mathcal{H}} &\coloneqq \frac{1}{8}\varepsilon^* \\
T &\coloneqq 2\frac{\log(64q^2/\varepsilon^*)}{\log(1/(1-\theta))} \leq 2\ln 2\frac{\log(64q^2/\varepsilon^*)}{\theta} \\
\varepsilon^*_0 &\coloneqq \frac{{\varepsilon^*}^2}{4192q^4T} \\
\varepsilon^*_2 &\coloneqq \frac{{\varepsilon^*}^2}{4192q^2|\mathcal{H}|^2}
\end{aligned}$$

to make each of the four summands in the bound for the total distinguishing advantage at most $\varepsilon^*/4$. For the bound on T, we used Lemma 15.

The running time of each query of the simulator is easily seen to be bounded by $O(q\theta^{-1}) \cdot \mathsf{poly}(\log(1/\varepsilon^*), \log(q), \log(|\mathcal{H}|), \log(1/\theta))$. Note that the $O(q)$-factor comes from the need to evaluate the hash functions K_1 and K_2, which involves polynomials of degree $O(q)$. The $O(\theta^{-1})$-factor comes from the T-fold repetition inside the evaluation of Samp_1. We remark that, because evaluating K_1 and K_2 incurs an $O(q)$ cost anyway, the initialization of K_1 and K_2 (which involves sampling $O(q)$ coefficients of the polynomial) is not dominating the running time; we may assume that this initialization is done at the first call to Sim and do not treat it separately.

□

Remark 18 (Dependency on $|\mathcal{H}|$). In our proof of Theorem 5, the advantage of the adversary had a dependency on $|\mathcal{H}|$. This dependency comes from the fact that we used Lemma 16 once for each $h \in \mathcal{H}$, i.e., $|\mathcal{H}|$ many times. However, a more careful analysis (which involves modifying the proof from [BDF+10]) shows that an appropriate variant of this lemma also holds true for families of oracles parameterized by h. This can be used to eliminate the dependency on $|\mathcal{H}|$ from the advantage and subsequently the dependency on $\log(|\mathcal{H}|)$ from the running time. However, since the analysis of quantum algorithms is not the focus of this work, we are content with the less tight statement of Theorem 5 and will not prove this.

Remark 19 (Infinite $\mathcal{D}$). In our proof above, we assumed that the domain $\mathcal{D}$ was finite and contained in some $\mathcal{D} \subseteq \{0,1\}^n$. For the more general case (still assuming finite codomain), we note that our construction in fact gives a uniform family of (quantum) circuits, parameterized by n, which can be viewed as the appropriate generalization to infinite domains. However, this depends on the underlying model of (quantum) computation and communication with the oracles, which we do not want to specify in this work. Instead, we assume that for any reasonable model of quantum computation and communication with oracles, whenever we query some oracle $O(z)$ with a superposition of values, the support of this superposition is contained in some $\{0,1\}^n$ and n is known to the oracle as *classical* input. This is because n is bounded by the number of qbits sent to O. Consequently, a generalization of our simulator from Theorem 5 to infinite $\mathcal{D}$ may just construct and dispatch to the appropriate version of our construction with fixed, hardwired n.

B Toolbox

B.1 Helpful Mathematical Lemma

Lemma 15. *Let $0 \leq x < 1$ be a real number and let $\ln$ denote the natural logarithm. Then*

$$\ln(1/(1-x)) \geq x.$$

Proof. Let $g(x) \coloneqq \ln(1/(1-x)) - x$. We have $g(0) = 0$ and the derivative $\frac{\mathrm{d}}{\mathrm{d}x}g(x) = \frac{x}{1-x}$ is non-negative for $0 \leq x < 1$. It follows that $g(x) \geq 0$ for $0 \leq x < 1$. □

B.2 Existing (Q)ROM Tools

Lemma 16 (Lemma 3 in [BDF+10]). *Let $\mathcal{A}$ be an algorithm, making at most q quantum queries to an oracle O that outputs values in a set $\mathcal{X}$. The oracle O follows one of two distributions. The first is the random oracle distribution over $\mathcal{X}$, denoted by O_0. The second is the distribution of oracles where the value of the oracle at each input is identically and independently distributed by some distribution $\mathcal{D}$ over $\mathcal{X}$, denoted by O_1. Then, we have*

$$\left|\Pr\left[\mathcal{A}^{O_0}(1^\lambda)=1\right]-\Pr\left[\mathcal{A}^{O_1}(1^\lambda)=1\right]\right|\leq 4q^2\sqrt{\epsilon},$$

where ϵ is the distance to uniform:

$$\epsilon:=\sum_{x\in\mathcal{X}}\left|\Pr\left[\mathcal{D}=x\right]-\frac{1}{|\mathcal{X}|}\right|.$$

Lemma 17 (Adaptive reprogramming [GHHM21, DKKW25a]). *Let X_1, X_2 and Y be finite sets, and let $\mathcal{A}$ be an algorithm. We define the following advantage:*

$$\mathsf{Adv}^{\mathsf{repro}}(\mathcal{A})=\left|\Pr\left[\mathcal{A}^{O_0,\mathsf{Reprogram}}(1^\lambda)=1\right]-\Pr\left[\mathcal{A}^{O_1,\mathsf{Reprogram}}(1^\lambda)=1\right]\right|,$$

where the oracles are defined as follows:

1. *$O_0\xleftarrow{\$}Y^{X_1\times X_2}$, i.e., $O_0\colon X_1\times X_2\to Y$ is a random function.*
2. *$O_1:=O_0$ is a copy of O_0.*
3. *The oracle $\mathsf{Reprogram}(x_2)$ is defined the following way:*
   (a) *Sample $(x_1,y)\xleftarrow{\$}X_1\times Y$.*
   (b) *$O_1:=O_1^{(x_1,x_2)\mapsto y}$, i.e., O_1 is reprogrammed such that $O_1(x_1,x_2)=y$.*
   (c) *Return x_1.*

   Notably, $\mathsf{Reprogram}$ does not change O_0.

With this definition, for any algorithm $\mathcal{A}$ that makes at most R many classical calls to $\mathsf{Reprogram}$ and q many (classical or quantum) queries to O_b, we have

$$\mathsf{Adv}^{\mathsf{repro}}(\mathcal{A})\leq\begin{cases}R\cdot q/|X_1|, & \text{if } \mathcal{A} \text{ is a classical algorithm}\\ 2R\cdot\sqrt{q/|X_1|}, & \text{if } \mathcal{A} \text{ is a quantum algorithm}\end{cases},$$

where classical $\mathcal{A}$ has classical access to $O.$, and quantum $\mathcal{A}$ has quantum access to $O.$.

Lemma 18 (HRS-Framework [HRS16, BHRvV21, DKKW25a]). *Let $\mathcal{S}$ be a set, and let $\mathcal{F}=\{f\colon\mathcal{S}\to\{0,1\}\}$ be the collection of all functions that map elements of $\mathcal{S}$ to $\{0,1\}$. Let $\eta\in[0,1]$. Define a family of distributions D_η on $\mathcal{F}$ such that a function $f\leftarrow D_\eta$ drawn from D_η satisfies*

$$f\colon x\mapsto\begin{cases}1 & \text{with probability } \eta,\\ 0 & \text{with probability } 1-\eta\end{cases}\quad\text{for any } x\in\mathcal{S},$$

where all choices are made independently. For any algorithm $\mathcal{A}$, we define

$$\mathsf{Adv}^{\eta,\mathsf{avg\text{-}search}}(\mathcal{A}):=\Pr\left[f(x)=1\mid f\leftarrow D_\eta,\ x\leftarrow\mathcal{A}^f(1^\lambda)\right].$$

With this definition, for any algorithm $\mathcal{A}$ that makes at most q (classical or quantum) queries to f, it holds that

$$\mathsf{Adv}^{\eta,\mathsf{avg\text{-}search}}(\mathcal{A})\leq\begin{cases}\eta(q+1), & \text{if } \mathcal{A} \text{ is a classical algorithm}\\ 8\eta(q+1)^2, & \text{if } \mathcal{A} \text{ is a quantum algorithm}\end{cases},$$

where classical $\mathcal{A}$ has classical access to f, and quantum $\mathcal{A}$ has quantum access to f.

B.3 New (Q)ROM Tools

Lemma 19 (Aborting Adaptive Reprogramming). *Let $\theta \in [0,1]$ be a rational number. Let X_1, X_2 and Y be finite sets, and let $\mathcal{A}$ be an algorithm. We define the following advantage:*

$$\mathsf{Adv}^{\mathsf{ab\text{-}repro}}_{\theta}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{O_{\perp,0},\mathsf{Reprogram}_{\perp}}(1^{\lambda})=1\right] - \Pr\left[\mathcal{A}^{O_{\perp,1},\mathsf{Reprogram}_{\perp}}(1^{\lambda})=1\right]\right|,$$

where the oracles are defined as follows:

1. *$O_{\perp,0}$ is a θ-aborting random oracle.*
2. *$O_{\perp,1} := O_{\perp,0}$ is a copy of the fully evaluated $O_{\perp,0}$.*
3. *The oracle $\mathsf{Reprogram}_{\perp}(x_2)$ is defined the following way:*
   (a) *Sample $x_1 \xleftarrow{\$} X_1$ and $y \leftarrow \mathcal{D}_{\perp}^{Y,\theta}$.*
   (b) *$O_{\perp,1} := O_{\perp,1}^{(x_1,x_2)\mapsto y}$, i.e., $O_{\perp,1}$ is reprogrammed such that $O_{\perp,1}(x_1,x_2)=y$.*
   (c) *Return x_1.*

   Notably, $\mathsf{Reprogram}_{\perp}$ does not change $O_{\perp,0}$.

With this definition, for any algorithm $\mathcal{A}$ that makes at most R many classical calls to $\mathsf{Reprogram}$ and q many queries to $O_{\perp,b}$, we have

$$\mathsf{Adv}^{\mathsf{ab\text{-}repro}}_{\theta}(\mathcal{A}) \leq \begin{cases} R\cdot q/|X_1|, & \text{if } \mathcal{A} \text{ is a classical algorithm} \\ 2R\cdot\sqrt{q/|X_1|}, & \text{if } \mathcal{A} \text{ is a quantum algorithm} \end{cases},$$

where classical $\mathcal{A}$ has classical access to $O_{\perp,\cdot}$, and quantum $\mathcal{A}$ has quantum access to $O_{\perp,\cdot}$.

Proof. Let $\mathrm{O}_{\perp,0}, \mathrm{O}_{\perp,1}, \mathsf{Reprogram}_{\perp}$ be oracles exactly as in the statement. Assume that $\theta = k/\ell$ for integers $k \geq 0, \ell > 0$. We prove the statement via a sequence of hybrids:

Hybrid Game $\mathbf{G}_b$. This is the game from the lemma in which $\mathcal{A}$ has oracle access to $\mathrm{O}_{\perp,b}, \mathsf{Reprogram}_{\perp}$. We denote the probability that $\mathcal{A}$ outputs 1 in such a hybrid by ε_b. Our goal is to bound $|\varepsilon_0 - \varepsilon_1|$.

Hybrid Game $\mathbf{G}'_b$. In this game, the adversary has oracle access to $\mathrm{O}'_{\perp,b}, \mathsf{Reprogram}'_{\perp}$, where

- Let $\mathrm{O}_0, \mathrm{O}_1$ and $\mathsf{Reprogram}$ be oracles as in Lemma 17, where
$$\mathrm{O}_0, \mathrm{O}_1 \colon X_1 \times X_2 \to [\ell] \times Y.$$
- We define
$$\mathrm{O}'_{\perp,\beta}(x_1,x_2) = \begin{cases} \perp & \text{if } j \leq k \\ y & \text{else} \end{cases} \quad \text{for } (j,y) = \mathrm{O}_{\beta}(x_1,x_2).$$
- Oracle $\mathsf{Reprogram}'_{\perp}(x_2)$ just calls $\mathsf{Reprogram}(x_2)$.

The change from $\mathbf{G}_b$ to $\mathbf{G}'_b$ is purely conceptual, as $\mathrm{O}'_{\perp,0}, \mathrm{O}'_{\perp,1}, \mathsf{Reprogram}'_{\perp}$ and $\mathrm{O}_{\perp,0}, \mathrm{O}_{\perp,1}, \mathsf{Reprogram}_{\perp}$ are identically distributed. We denote the probability that $\mathcal{A}$ outputs 1 in such a hybrid by ε'_b. Because the change is purely conceptual, we have $\varepsilon_b = \varepsilon'_b$ for both $b \in \{0,1\}$.

Reduction Argument. One can easily argue that $\mathbf{G}'_0$ and $\mathbf{G}'_1$ are indistinguishable by applying Lemma 17 to $\mathrm{O}_0, \mathrm{O}_1, \mathsf{Reprogram}$. We get

$$|\varepsilon_0 - \varepsilon_1| = |\varepsilon'_0 - \varepsilon'_1| \leq \begin{cases} R\cdot q/|X_1|, & \text{(classical)} \\ 2R\cdot\sqrt{q/|X_1|}, & \text{(quantum)} \end{cases}.$$

□

C Proof of Lemma 13

The proof uses the tools from Appendix B.

Proof of Lemma 13. We consider the game from Definition 17, which is defined (in the θ-aborting random oracle model) as

$$P \xleftarrow{\$} \mathcal{P}, \quad (M^*, \rho^*) \leftarrow \mathcal{A}^{\mathrm{O},\mathsf{H}_\perp}(P),$$

and we want to bound the probability of the adversary "winning", i.e., of the following event:

- Event Win: This event occurs, if there is $(M, T, \rho) \in Q$ such that

$$\rho \neq \perp \ \wedge \ (M^*, \rho^*) \neq (M, \rho) \ \wedge \ \mathsf{H}_\perp(P, T, (M^*, \rho^*)) = \mathsf{H}_\perp(P, T, (M, \rho)) \neq \perp.$$

We do so via a sequence of hybrid games, where we primarily change how O and $\mathsf{H}_\perp$ are implemented.

Hybrid Game $\mathbf{G}_0$. This is the original game from Definition 17. To recall, oracle O works as follows on input $(T, M) \in \mathcal{T} \times \mathcal{M}$:

1. If there is a tuple $(T, M', \rho') \in Q$, for some M', ρ', then return $\perp$.
2. Otherwise, set $\mathsf{ctr} := 0$ and $x := \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
   (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$.
   (b) Set $x := \mathsf{H}_\perp(P, T, (M, \rho))$.
   (c) If $\mathsf{Prop}(x) = 1$ and $x \neq \perp$: Insert (T, M, ρ) into Q.
   (d) Else: Set $x := \perp$, $\rho := \perp$.
   (e) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
3. If $x = \perp$: Insert $(T, M, \perp)$ into Q.
4. Output (x, ρ).

Further, $\mathsf{H}_\perp$ is a θ-aborting random oracle. By definition, we have

$$\mathsf{Adv}^{\mathsf{na\text{-}r\text{-}tcr}}_{\mathsf{H}_\perp,\mathsf{Prop},K}(\mathcal{A}) = \Pr_{\mathbf{G}_0}[\mathsf{Win}].$$

Hybrid Game $\mathbf{G}_1$. We now reprogram the *internal non-aborting* random oracle H during calls to O. More concretely, we change how O works on input $(T, M) \in \mathcal{T} \times \mathcal{M}$, highlighting the changes:

1. If there is a tuple $(T, M', \rho') \in Q$, for some M', ρ', then return $\perp$.
2. Otherwise, set $\mathsf{ctr} := 0$ and $x := \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
   (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$ and $y \leftarrow \mathcal{D}_\perp^{\mathcal{H},\theta}$.
   (b) Program $\mathsf{H}_\perp(P, T, (M, \rho)) := y$.
   (c) Set $x := \mathsf{H}_\perp(P, T, (M, \rho))$.
   (d) If $\mathsf{Prop}(x) = 1$ and $x \neq \perp$: Insert (T, M, ρ) into Q.
   (e) Else: Set $x := \perp$, $\rho := \perp$.
   (f) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
3. If $x = \perp$: Insert $(T, M, \perp)$ into Q.
4. Output (x, ρ).

Indistinguishability follows via a reduction that runs in the aborting adaptive reprogramming game (Lemma 19) and makes at most pK reprogramming queries and at most $q' = q + pK$ oracle queries. We get

$$\left|\Pr_{\mathbf{G}_0}[\mathsf{Win}] - \Pr_{\mathbf{G}_1}[\mathsf{Win}]\right| \leq \begin{cases} pK \cdot q'/|\mathcal{R}|, & \text{for a classical adversary} \\ 2pK \cdot \sqrt{q'/|\mathcal{R}|}, & \text{for a quantum adversary} \end{cases}$$

Hybrid Game $\mathbf{G}_2$. In this game, we specify a *preparation phase*, which happens before the adversary is run, and additionally make another change to oracle O. We start with explaining the preparation phase:

1. Pick a parameter $P \xleftarrow{\$} \mathcal{P}$.
2. We pick a θ-aborting random oracle
3. For each $t \in \mathcal{T}$, we define an ordered list S_t and a dedicated pair (ρ_t^*, x_t^*) (which may be $(\bot, \bot)$) as follows:
   (a) Set $\rho_t^* = \bot$ and $x_t^* = \bot$.
   (b) Set $\mathsf{ctr} := 0$ and $x := \bot$. While $\mathsf{ctr} < K$ and $x = \bot$:
      i. Sample $\rho \xleftarrow{\$} \mathcal{R}$ and $y \leftarrow \mathcal{D}_\bot^{\mathcal{H},\theta}$.
      ii. If $\mathsf{Prop}(y) = 1$ and $y \neq \bot$: Append (ρ, y) to S_t, set $x := y$ and $(\rho_t^*, x_t^*) := (\rho, y)$.
      iii. Else: Append (ρ, y) to S_t and set $x := \bot$.
      iv. Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
4. Define the θ-aborting random oracle to be used in the game as

$$\mathsf{H}_\bot(p, t, (m, \rho)) := \begin{cases} g(t, m, \rho), & \text{if } p = P \\ \mathsf{H}_\bot{}'(p, t, (m, \rho)), & \text{if } p \neq P \end{cases},$$

   where $g\colon \mathcal{T} \times \mathcal{M} \times \mathcal{R} \to \mathcal{H} \cup \{\bot\}$ and $\mathsf{H}_\bot{}'\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{H} \cup \{\bot\}$ are θ-aborting random oracles that are used internally by the game, but not given directly to the adversary.

After this phase, the actual game starts, where $\mathcal{A}$ gets access to P and $\mathsf{H}_\bot$ as above, and O implemented as follows:

1. If there is a tuple $(T, M', \rho') \in Q$, for some M', ρ', then return $\bot$.
2. Otherwise, for each $(\rho, y) \in S_T$ (in order): Program $\mathsf{H}_\bot(P, T, (M, \rho)) := y$
3. Insert (T, M, ρ_T^*) into Q.
4. Output (x_T^*, ρ_T^*).

In other words, what we have done is (1) defined $\mathsf{H}_\bot$ indirectly by splitting it into g and $\mathsf{H}_\bot{}'$ according to the first parameter, and (2) precomputed the entire sequence of pairs (ρ, y) that appear in oracle calls to O. This change is only conceptual. Namely, for (1), observe that $\mathsf{H}_\bot$ follows exactly the same distribution as a θ-aborting random oracle. For (2), note that the distribution of the values (ρ, y) used in O is exactly as in the previous game, we just computed them earlier. We get

$$\Pr_{\mathbf{G}_1}[\mathsf{Win}] = \Pr_{\mathbf{G}_2}[\mathsf{Win}].$$

Hybrid Game $\mathbf{G}_3$. We change how the function $g\colon \mathcal{T} \times \mathcal{M} \times \mathcal{R} \to \mathcal{H} \cup \{\bot\}$ is defined, embedding a boolean function, so that we can later apply the HRS framework. To this end, let $f \leftarrow D_\eta$ with $f\colon \mathcal{T} \times \mathcal{M} \times \mathcal{R} \to \{0,1\}$ be a boolean function following distribution D_η as in Lemma 18, where

$$\eta = (1 - \theta) \cdot \frac{1}{\mathcal{H}} = (1 - \theta)/|\mathcal{H}|.$$

For each $t \in T$, set $z_t^* := x_t^*$ if $x_t^* \neq \bot$. Otherwise, sample z_t^* independently and uniformly at random from $\mathcal{H}$. We define g as follows:

$$g(t, m, \rho) = \begin{cases} z_t^*, & \text{if } f(t, m, \rho) = 1 \\ g_t(m, \rho) & \text{if } f(t, m, \rho) = 0 \end{cases},$$

where for each $t \in \mathcal{T}$, $g_t \colon \mathcal{M} \times \mathcal{R} \to (\mathcal{H} \setminus \{z_t^*\} \cup \{\bot\})$ is a $\tilde{\theta}$-aborting random oracle, with

$$\tilde{\theta} = \Pr_{h \leftarrow \mathcal{D}_\bot^{\mathcal{H},\theta}}[h = \bot \mid h \neq z_t^*] = \frac{\theta}{\theta + (1-\theta)(1 - 1/|\mathcal{H}|)}.$$

One can see that g is distributed in exactly the same way as a θ-aborting random oracle, i.e., exactly as in the previous game. We get

$$\Pr_{\mathbf{G}_2}[\mathsf{Win}] = \Pr_{\mathbf{G}_3}[\mathsf{Win}].$$

Final Reduction. Our final reduction simulates $\mathbf{G}_3$ for $\mathcal{A}$ and wins in the search game defined in Lemma 18. More precisely, the reduction gets oracle access to f (distributed as above) and simulates the game for $\mathcal{A}$, providing $\mathsf{H}_\bot$ and O as above. Eventually, the adversary outputs (M^*, ρ^*), and event Win occurs if there is $(M, T, \rho) \in Q$ such that $\rho \neq \bot$, $(M^*, \rho^*) \neq (M, \rho)$, and $\mathsf{H}_\bot(P, T, (M^*, \rho^*)) = \mathsf{H}_\bot(P, T, (M, \rho)) \neq \bot$. At that point, the reduction outputs (T, M^*, ρ^*). It is clear that the game is perfectly simulated. What remains is to argue that if Win occurs, then $f(T, M^*, \rho^*) = 1$. This is argued in two steps:

- We first claim that $\mathsf{H}_\bot(P, T, (M^*, \rho^*))$ has not been reprogrammed in O:
  - If $M \neq M^*$, then clearly it has not been reprogrammed.
  - Otherwise, we can assume that $\rho^* \neq \rho$, and we know that $\bot \neq \rho = \rho_T^*$ by construction. We will derive a contradiction.
  - Therefore, it can only be that it has been reprogrammed if $\rho^* = \rho'$ for some $(\rho', y) \in S_T$ with $\rho' \neq \rho$.
  - For such y, we have $y = \bot$ or $\mathsf{Prop}(y) = 0$. Call such y's *bad*.
  - But then, we would have $\mathsf{H}_\bot(P, T, (M^*, \rho^*)) = y$ is bad, and Win implies that it is not bad. Contradiction.
- We must have $x_T^* \neq \bot$ and therefore $z_T^* = x_T^*$. This is because if $x_T^* = \bot$, then $\rho_T^* = \bot$ by construction, but $\rho = \rho_T^*$ by construction, which contradicts Win.
- From $\mathsf{H}_\bot(P, T, (M^*, \rho^*)) = \mathsf{H}_\bot(P, T, (M, \rho))$ and $\bot \neq \rho = \rho_T^*$, it follows that $\mathsf{H}_\bot(P, T, (M^*, \rho^*)) = x_T^* = z_T^*$.
- Therefore, it must be that $f(T, M^*, \rho^*) = 1$ by construction of g and the g_t.

Note that the reduction makes at most $q' = q + pK$ queries to f. Using Lemma 18, we get

$$\Pr_{\mathbf{G}_3}[\mathsf{Win}] \leq \begin{cases} (1-\theta)(q'+1)/|\mathcal{H}|, & \text{for a classical adversary} \\ 8(1-\theta)(q'+1)^2/|\mathcal{H}|, & \text{for a quantum adversary} \end{cases}.$$

□