Optimal Security for Keyed Hash Functions: Avoiding Time-Space Tradeoffs for Finding Collisions[*]

Cody Freitag[†]   Ashrujit Ghoshal[‡]   Ilan Komargodski[§]

Abstract

Cryptographic hash functions map data of arbitrary size to a fixed size digest, and are one of the most commonly used cryptographic objects. As it is infeasible to design an individual hash function for every input size, variable-input length hash functions are built by designing and bootstrapping a single fixed-input length function that looks sufficiently random. To prevent trivial preprocessing attacks, applications often require not just a single hash function but rather a family of keyed hash functions.

The most well-known methods for designing variable-input length hash function families from a fixed idealized function are the Merkle-Damgård and Sponge designs. The former underlies the SHA-1 and SHA-2 constructions and the latter underlies SHA-3. Unfortunately, recent works (Coretti et al. EUROCRYPT 2018, Coretti et al. CRYPTO 2018) show non-trivial time-space tradeoff attacks for finding collisions for both. Thus, this forces a parameter blowup (i.e., efficiency loss) for reaching a certain desired level of security. We ask whether it is possible to build families of keyed hash functions which are *provably* resistant to any non-trivial time-space tradeoff attacks for finding collisions, without incurring significant efficiency costs.

We present several new constructions of keyed hash functions that are provably resistant to any non-trivial time-space tradeoff attacks for finding collisions. Our constructions provide various tradeoffs between their efficiency and the range of parameters where they achieve optimal security for collision resistance. Our main technical contribution is proving optimal security bounds for converting a hash function with a fixed-sized input to a keyed hash function with (potentially larger) fixed-size input. We then use this keyed function as the underlying primitive inside the standard Merkle-Damgård and Merkle tree constructions. We strongly believe that this paradigm of using a keyed inner hash function in these constructions is the right one, for which non-uniform security has not been analyzed prior to this work.

[*]A preliminary version of this paper appears in the proceedings of EUROCRYPT 2023. This is the full version.

[†]Cornell Tech, New York, USA. Email: `cfreitag@cs.cornell.edu`.

[‡]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, Washington, USA. Email: `ashrujit@cs.washington.edu`.

[§]School of Computer Science and Engineering, Hebrew University of Jerusalem, 91904 Jerusalem, Israel and NTT Research. Email: `ilank@cs.huji.ac.il`.

Contents

1 Introduction

A cryptographic hash function is a (deterministic) algorithm that takes arbitrary length input data and outputs a fixed length digest. It is one of the most fundamental tools in modern applications of cryptography, underlying numerous widely used applications. For example, it facilitates the hash-and-sign paradigm, proofs-of-work for blockchains, and more. While it is empirically believed that concrete cryptographic hash functions satisfy various useful security properties, formalizing this seems to be currently out of reach. Thus, in the context of provable security, cryptographic hash functions are usually modeled as random oracles, i.e., completely random functions [BR93]. This allows us to analyze specific properties and argue about the concrete security of systems that use them. In this work, we focus on the property of a hash function being *collision resistant*, i.e., the idea that although collisions exist in abundance in a compressing function, it should be computationally hard to find them.

The task of finding collisions in a given compressing function is only interesting if the adversary is uniform. That is, the adversary is "fixed" before the hash function. Indeed, otherwise, a non-uniform attacker can simply have collisions hardwired. However, the uniform model of security does not capture many real-world adversaries, and therefore it is common to model adversaries as non-uniform in theoretical cryptography. Specifically, non-uniform security captures adversaries that have been designed to attack specific instances, adversaries that have gone through an expensive preprocessing stage, or even protect against (currently unknown) future attacks. Non-uniform security is also necessary for composition within larger systems [GK96]. For all of these reasons, it is widely believed by the theoretical community that modeling attackers as non-uniform is the right thing to do, despite potentially being overly conservative and including unrealistic attackers.

Dealing with non-uniform attackers in the context of hashing and collision finding makes it necessary to consider a family of keyed hash functions, rather than a single hash function. Collision finding is then defined via the following two-stage game. First, a (keyed) family H of hash functions is fixed, and the attacker can depend arbitrarily on H. Second, a random key key is sampled, and the adversary needs to find a collision in H relative to key. Intuitively, in order to attack the hash function (e.g., find a collision), a non-uniform attacker must either (a) have some hard-coded information about key, or (b) can essentially be treated as uniform.

For applications, we typically want each member of H to operate on unbounded input lengths. That is, $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ should be viewed as a two-input function, operating on (key, m), where $\mathsf{key} \in \{0,1\}^\kappa$ is the key and $m \in \{0,1\}^*$ is an arbitrary length input. Since it is practically infeasible to design a different hash function for every input length, what happens is that a single basic compressing function $h\colon \{0,1\}^a \to \{0,1\}^n$ for some $a > n$ is designed, and then it is iterated in some way to get a hash function that compresses arbitrarily. For instance, the well-known Merkle-Damgård design [Mer89, Dam89] iterates such a basic compressing function in order to get a variable-input-length hash function that can operate on arbitrary sized data up to some maximum length (e.g., 2^{64} bits).

AI-ROM. Since we consider non-uniform security in the random oracle model, we model attackers using the *auxiliary-input random oracle model* (AI-ROM), formally defined by Unruh [Unr07] although implicitly used earlier, for example, by Hellman [Hel80], Yao [Yao90], and Fiat and Naor [FN99]. In this model, we assume a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ with $a > n$ modeled as a completely random one, i.e., a random oracle [BR93]. The AI-ROM models preprocessing adversaries as two-stage algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ parameterized by S (for "space") and T (for "time"). We refer to such an attacker as an (S,T)-attacker. The first part $\mathcal{A}_1$ (i.e., the offline phase) has unbounded access to h, and its goal is to compute an S-bit "advice" σ for $\mathcal{A}_2$. The second part

$\mathcal{A}_2$ (i.e., the online phase) gets the advice σ, can make at most T queries to h, and attempts to accomplish some task involving h. In our case, $\mathcal{A}_2$ gets a random key $\mathsf{key} \leftarrow\!\!\$\ \{0,1\}^\kappa$ as a challenge and its goal is to come up with a collision in $H(\mathsf{key}, \cdot)$. Aside from the restrictions that $|\sigma| \le S$ and that $\mathcal{A}_2$ can make at most T queries to h, both $\mathcal{A}_1$ and $\mathcal{A}_2$ are allowed to be computationally unbounded.

Building a keyed hash from a single hash function. Observe that for every keyed hash construction, there is an (S,T)-attacker that finds a collision relative to a random key with probability[1] $\Omega(S/2^\kappa + T^2/2^n)$ via the following attack. First, the preprocessing adversary outputs $\Omega(S)$ collisions with respect to arbitrary distinct keys. The online adversary receives a random key. If key is in the remembered list from the preprocessing phase, it outputs the corresponding collision. Otherwise, it performs a T-query birthday-style attack. The adversary wins if either the challenge key appears in one of its preprocessed collisions (giving the $S/2^\kappa$ term) or if the birthday attack succeeds (giving the $T^2/2^n$ term). We refer to this attack as the *naive attack*, and say that a construction is *optimally secure* if there is provably no better attack. This brings us to the main question we consider in this work.

Can we build a keyed hash function (i.e., $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$) from non-keyed one (i.e., $h\colon \{0,1\}^a \to \{0,1\}^n$) with optimal non-uniform security?

If we could design an $h_a\colon \{0,1\}^a \to \{0,1\}^n$ for every $a \in \mathbb{N}$, then the above task is easy. We can simply parse the input to the appropriate h_a into two parts, one for the key and the other for the input to $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. That is, define $H(\mathsf{key}, m) = h_{\kappa+|m|}(\mathsf{key}\|m)$, where $\|$ stands for string concatenation and $|\cdot|$ stands for bit length. For this construction, Dodis, Guo and Katz [DGK17] showed that the best attack achieves advantage $O(S/2^\kappa + T^2/2^n)$, matching the advantage of the naive attack.

Unfortunately, it is infeasible to design a different hash function for every input length as discussed above. The design of a new h is a delicate and lengthy process that could take many years to test and standardize. Having a single hash function is therefore more robust security-wise. Thus, the standard procedure is to design a hash function h with fixed input size and then iterate it in some way to get a hash function that supports arbitrary input lengths.

It may seem that standard domain extension techniques for hash functions (like Merkle-Damgård, Sponge, or Merkle trees) provide a solution for this problem. Indeed, their goal is to take a hash function on a small domain and turn it into a hash function with arbitrary-size domain. But, as we point out next, the standard constructions suffer from a significant security loss. A priori, it is not even clear that this security loss is avoidable.

The security of existing constructions. First, consider (a keyed variant of) the Merkle-Damgård (MD) construction [Mer89, Dam89], perhaps the most widely popular design for getting a hash function on long inputs from one on fixed input sizes. This design is not only extremely fundamental in cryptographic theory, but it also underlies popular hash functions used in practice, most notably MD5, SHA-1, and SHA-2. The $\mathsf{MD}\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ construction iterates the basic hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ by feeding in input blocks of size $s = a - \max\{\kappa, n\}$ one by one. It first pads the message appropriately such that it is a multiple of s bits. For key $\mathsf{key} \in \{0,1\}^\kappa$ and input $m \in \{0,1\}^s$, define $\mathsf{MD}(\mathsf{key}, m) = h(\mathsf{key}\|m)$. Then, for a longer input $m \in (\{0,1\}^s)^\ell$, viewed as ℓ blocks $m_1, \ldots, m_\ell$ each from $\{0,1\}^s$, recursively define $\mathsf{MD}(\mathsf{key}, (m_1, \ldots, m_\ell)) = h(\mathsf{MD}(\mathsf{key}, (m_1, \ldots, m_{\ell-1})), m_\ell)$. We note that in the standard MD

[1]To simplify notation throughout the introduction, we suppress poly factors in n in the asymptotic $O(\cdot)$ and $\Omega(\cdot)$ notation.

construction (studied, for example, in [CDGS18, ACDW20, GK22, AGL22]), key is only explicitly included once when processing the first message block.

Collision resistance of MD in the AI-ROM was first studied by Coretti, Dodis, Guo, and Steinberger [CDGS18] and more recently by [ACDW20, GK22, AGL22]. It is known that there is an attack, loosely based on the idea of rainbow tables [Hel80, Oec03], which succeeds in finding a collision with probability $\Omega(S/2^\kappa + ST^2/2^n)$. In typical settings of parameters, the $ST^2/2^n$ term dominates the above expression and in this case it is evident that MD suffers from a significant security loss.

Concretely, in the SHA-1 construction, $a = 678$ and $\kappa = n = 160$. If we model the underlying primitive $h\colon \{0,1\}^{678} \to \{0,1\}^{160}$ as a perfectly random function, an (S, T)-attacker with $S = 2^{53}$ and $T = 2^{50}$ will find a collision with probability $\approx 2^{-7}$ (essentially completely breaking the scheme).[2] On the other hand, the best one could hope is a construction with maximal advantage $O(S/2^\kappa + T^2/2^n) \approx 2^{-60}$ (obtained by the naive attack).

Another construction we mention is the Sponge [BDPVA07, BDPA08] construction, an alternative to the Merkle-Damgård design that underlies the modern SHA-3 hashing standard. As opposed to MD, the Sponge construction relies on a random *permutation* $\Pi\colon \{0,1\}^n \to \{0,1\}^n$. Sponge iterates Π by feeding in blocks of size $r < n$ from the input one at a time in a certain way. It results with a keyed hash function $\mathsf{Sp}\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^r$ with $\kappa + r = n$. Coretti et al. [CDGS18] (see also [FGK22]) showed that there is a collision finding (S, T)-attack with advantage $\Omega(ST^2/2^\kappa + T^2/2^r)$ against Sp relative to a random key. Again, we see that there is a non-trivial security loss in this construction.

It is important to note that for every choice of S and T the above attacks on Merkle-Damgård and Sponge beat the naive attack. In particular, there is no non-trivial choice of parameters where MD or Sp achieve the optimal security bound.

Lastly, we mention two other popular (variable-input-length) hash function designs: Merkle trees [Mer87] and the BLAKE family [AHMP08, AMPH14]. The former (Merkle trees) is a popular design that has important features like local opening and can be easily parallelized. Although it is extremely popular both in theory and in practice, we are not aware of a keyed variant that has been studied in the non-uniform setting. The latter (BLAKE) is a runner-up in NIST's competition to create a new hashing standard (where Sponge ended up as the winner). This design is based on the MD design, but they allow the inner hash function h to be keyed at every invocation. We are not aware of a formal study of its security in the non-uniform setting. Looking ahead, two of our main contributions are a proposal and analysis of the non-uniform security of Merkle tree and the MD/ BLAKE design, where the inner hash function h is keyed in every invocation. Concretely, we believe that this is the right notion to consider moving forward, in terms of non-uniform security.

A different perspective. Above, we considered the scenario where $h\colon \{0,1\}^a \to \{0,1\}^n$ is given, and we want to build an $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ which is as secure as possible for every (S, T)-attacker. A different perspective, slightly more target oriented, is to first fix a desired security level (say 2^{-50}) and the power of adversaries (say $S = T = 2^{60}$) and then understand which h is needed in order to get the desired H. If we use MD (for concreteness) for H, we will need $n \geq 230$, but if we had an optimally secure construction of H, we would need only $n \geq 170$. The latter could potentially be easier to design and argue about.

[2]These parameters roughly correspond to an attacker with $\approx$ 1000 terabytes of memory that uses optimized hardware that can compute 3 billion hashes per second for a long weekend.

1.1 Our Results

We provide several constructions of keyed hash functions from non-keyed ones that do not suffer from any security loss (i.e., the naive attack that has advantage $\Theta(S/2^\kappa + T^2/2^n)$ is provably optimal). Our constructions provide various tradeoffs between their efficiency and the range of parameters (S and T) where they achieve optimal security.

Merkle-Damgård and Merkle trees with a keyed inner hash. All of our constructions can be viewed within a framework that builds on the Merkle-Damgård and Merkle tree constructions.

We start by discussing the MD-based approach. We consider an iterative hashing design where a compression phase is performed in every step using an "inner hash" function. The input for the compression phase is the current state and the next input. At the end, the compression phase outputs the next state. Of course, the inner hash function in the compression phase can use h as a subroutine. Abstractly, the compression phase for the MD-based construction is

$$y := \mathsf{compress}(\mathsf{key}, y, m),$$

where importantly the compress function takes key as input. See Figure 1 for an illustration. With this notation, the compression function of the standard MD function (at least as studied in numerous recent works including [CDGS18, ACDW20, GK22, AGL22]) is simply $\mathsf{compress}(\mathsf{key}, y, m) = h(y, m)$, and for the first step, y is initialized to key. Notably key is not included in every compression phases.

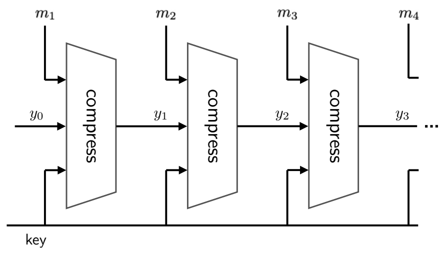

Figure 1: Our framework for building keyed hash functions based on the Merkle-Damgård construction with a keyed inner compression function.

We next consider a parallelizable hashing design that generalizes the Merkle tree hash function. Here, each input is fed into a "leaf" of the Merkle tree, along with the key value. The compression function is then used to recursively combine outputs in previous levels until a final output is generated. Crucially, we always include key in the compression function. See Figure 2 for an illustration. This framework provides an alternative to the generalized MD approach described above. It requires at most a factor of two more calls to compress, but it is extremely parallelizable. Further, it provides a local opening property, where someone can prove that an individual message block m_i was included in the hash, without providing the full message.

Our constructions are obtained by different implementations of compress, namely viewing compress as an inner keyed hash function used in the MD and Merkle tree designs. Quantitatively, the MD and Merkle tree approaches give similar results, so we focus our attention on instantiating compress in the case of the MD-based framework. However, all of our main results extend to the setting of the generalized Merkle tree framework, which we provide in Appendix A for completeness.

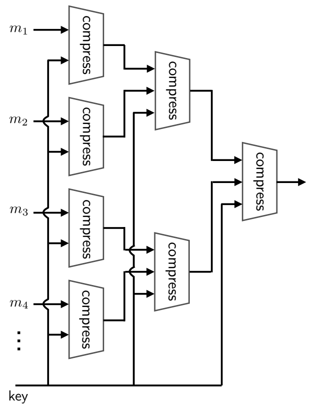

Figure 2: Our framework for building keyed hash functions based on the Merkle tree construction.

For simplicity of presentation of our results, we slightly simplify notation and assume that κ (the key length) is equal to n (the output size of the hash function).[3] In our formal theorem statements in the technical sections, κ and n are treated independently when relevant.

Efficiency: We measure efficiency of a given construction by the number of calls to h needed to evaluate H at a single point. For example, in the standard MD construction with an underlying hash that maps $\{0,1\}^a$ to $\{0,1\}^n$, to hash a b-bit input, the query complexity is $b/(a-n)$ (ignoring rounding[4]). Indeed, every application of h takes as input the previous output (n bits) and so it can process $a-n$ bits from the input each time.

Assuming a large inner hash. Our first result shows that optimal security loss is achievable. That is, we show that there is a way to take a random oracle that operates on a fixed input length and get a keyed hash H that operates on arbitrary-length inputs with the following security guarantee: for any S, T, any (S,T)-attacker has minimal possible advantage in finding a collision in H relative to a random key. In words, the new construction is a variant of MD where we also feed key as input in every block. We refer to this construction as the MD construction with a keyed inner hash, in contrast to the standard MD construction where key is only fed in the first block. At a high level, feeding the key into every invocation of h allows us to reduce the probability of finding a long collision in H to that of finding a collision in h, which achieves optimal security $O(S/2^n + T^2/2^n)$ [DGK17]. Refer to Figure 3 for an illustration of how the construction works.[5]

Theorem 1.1 (Informal; see Theorem 4.3)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > 2n$. Then, there is an $H_1\colon \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ and any (S,T)-attacker, their advantage in finding a collision in $H_1(\mathsf{key}, \cdot)$ relative to a random $\mathsf{key} \leftarrow\!\!\$\ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*

[3] We note that there are constructions that use $\kappa \neq n$ by design (e.g., BLAKE hash [AHMP08, AMPH14] uses $\kappa = n/2$).

[4] To be more precise, MD requires $\lceil (b+n+1)/(a-n) \rceil$ calls to h after padding the input with its length followed by a 1 and a sequence of 0s to fill the remaining current block. However, for ease of presentation, we ignore rounding in the introduction. In the formal theorem statements, we give exact efficiency bounds.

[5] Essentially the same construction appears in Goldwasser-Bellare's lecture notes [GB08, §8.5] where it is shown that this construction is collision resistant in the uniform setting. Our result shows that this holds in the non-uniform (AI-ROM) setting as well.

2. *One evaluation of H_1 on a given key and a b-bit message requires $b/(a-2n)$ queries to h.*

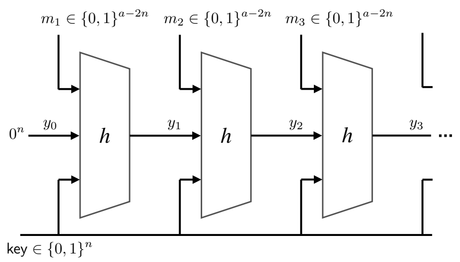

Figure 3: The construction H_1 underlying Theorem 1.1 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 2n$.

The above result is optimal in terms of security and is almost as efficient as standard MD if $a \geq 2n + \Omega(n)$. For example, if $a = 3n$, processing a b-bit input of H_1 requires querying h as many as $b/(a-2n) = b/n$ times. In the standard MD construction, only $b/(a-n) = b/(2n)$ queries are required, so our construction is less efficient than MD by a small constant factor at most 2 when $a = 3n$.

However, H_1 is significantly less efficient than MD if a is roughly $2n$, i.e. h compresses by a factor of 2. For example, if $a = 2n+1$, then processing a b-bit input of H_1 requires invoking h as many as b times. However, MD requires only $b/(n+1)$ queries. This is a significant difference. We emphasize that having an efficient construction even when $a \approx 2n$ is not only a technicality but is rather important: concretely, assuming that the basic compressing function shrinks by a factor 2 is extremely common, both in theory and in practice. Thus, our next results are focused on closing this gap.

Instantiating the keyed inner hash with standard MD. To this end, we start by considering a construction H_2 that works for *any* $a > n$ and only incurs a factor of 2 overhead in terms of efficiency relative to MD. While this may seem too good to be true, we pay in terms of the assumptions we need to make to claim optimal security for collision resistance. Namely, the scheme has "optimal security," meaning any (S,T)-attacker can find a collision with probability at most $O(S/2^n + T^2/2^n)$, only whenever $S \leq T$ and $ST^2 \leq 2^n$.

This main idea behind the construction H_2 is to instantiate the compress function in the MD-based framework of Figure 1 with a standard MD hash function. We use key as the key for MD, and we treat $y_{i-1} \| m_i$ as the message. If we use a message block size $|m_i| = n$, this results in only a factor of two overhead relative to MD (essentially, half of the invocations of h incorporate bits of the message m_i, and half of the invocations incorporate bits of the previous output y_{i-1}). This construction is depicted in Figure 4 and gives the following result.

Theorem 1.2 (Informal; see Theorem 5.2)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > n$. Then, there is an $H_2\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S,T \in \mathbb{N}$ such that $S \leq T$, $ST^2 \leq 2^n$, and any (S,T)-attacker, their advantage in finding a collision in $H_2(\mathsf{key},\cdot)$ relative to a random $\mathsf{key} \leftarrow_\$ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*

2. *One evaluation of H_2 on a given key and a b-bit input requires $2 \cdot b/(a-n)$ queries to h.*

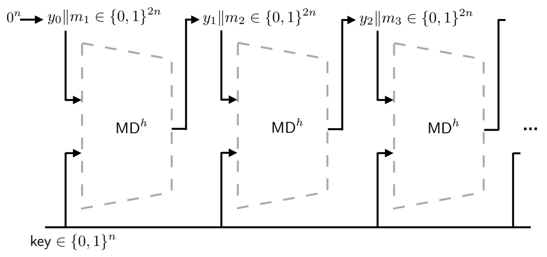
0^n
$y_0\|m_1 \in \{0,1\}^{2n}$
$y_1\|m_2 \in \{0,1\}^{2n}$
$y_2\|m_3 \in \{0,1\}^{2n}$
MD^h
MD^h
MD^h
...
$\mathsf{key} \in \{0,1\}^n$

Figure 4: The construction H_2 underlying Theorem 1.2 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > n$. The gray dotted boxes represent the $\mathsf{compress}$ function, instantiated with the Merkle-Damgård construction, that uses $\mathsf{key} \in \{0,1\}^n$ as the key and $y_{i-1}\|m_i \in \{0,1\}^{2n}$ as the message.

We note that the assumption that $ST^2 \leq 2^n$ in the construction above comes from the fact that best currently known time-space tradeoffs for the collision resistance of standard MD (culminating in [GK22, AGL22] following the works of [CDGS18, ACDW20, CGLQ20]) require this assumption to get optimal bounds when analyzing the ℓ-block MD construction when $\ell \in \omega(1)$. In the special case where we only use a 2-block variant of MD as the underlying $\mathsf{compress}$ function, [ACDW20] give tight bounds that do not require that $ST^2 \leq 2^n$ (and furthermore, [GK22] gave tight bounds for all constants ℓ). This motivates our next construction, H_3, which uses the 2-block MD construction for $\mathsf{compress}$, but requires that the input size to h satisfies $a > 3n/2$. See Figure 5 for an illustration, and the corresponding result is given in the following theorem.

Theorem 1.3 (Informal; see Corollary 5.3)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > 3n/2$. Then, there is an $H_3\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ such that $S \leq T$ and any (S,T)-attacker, their advantage in finding a collision in $H_3(\mathsf{key}, \cdot)$ relative to a random $\mathsf{key} \leftarrow\$ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*
2. *One evaluation of H_3 on a given key and a b-bit input requires $2 \cdot b/(2a - 3n)$ queries to h.*

Instantiating the keyed inner hash with a 2-level Merkle tree. For our final construction, we seek to build a hash function H_4 that is both efficient and optimally secure whenever $a \approx 2n$ (h is only compressing by a factor of 2), without assuming that $S \leq T$. In particular, $S \gg T$ makes sense in many practical scenarios: the pre-processing attacker may have much more than time T to generate its advice string of size S, the online attacker may have easy random access to a structured advice string, or the online time T may be small for applications that enforce a timeout with fixed-time communication session (see [ASS+16] as an example of an attack on a TLS session that requires relatively heavy computation in an offline phase). Lastly, we mention that the bounds we obtain on H_2 and H_3 are tight—there is a non-trivial attack whenever $S > T$ that scales with advantage $\Omega(ST\ell/2^n)$ for standard ℓ-block MD [CDGS18, ACDW20].

For H_4, we instantiate the $\mathsf{compress}$ function from the framework of Figure 1 using a keyed variant of a Merkle tree construction (this has never been formally defined or analyzed to the best of our knowledge). In our variant, we feed key into all leaves of the Merkle tree. Concretely, in our construction, we use a 2-level Merkle tree and feed key to both of them, corresponding to two distinct invocations of h, and we split the "message" $y_{i-1}\|m_i$ into the remaining input bits for the leaves. The second level of the Merkle tree combines the two outputs from the first level, to produce

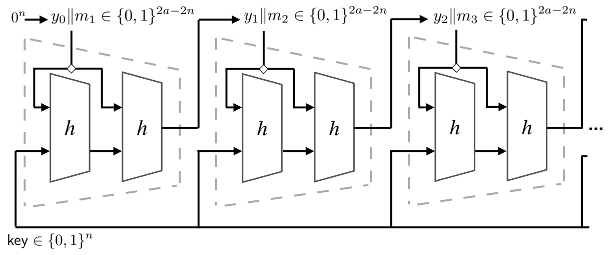
0^n
$y_0\|m_1 \in \{0,1\}^{2a-2n}$
$y_1\|m_2 \in \{0,1\}^{2a-2n}$
$y_2\|m_3 \in \{0,1\}^{2a-2n}$
h
...
key $\in \{0,1\}^n$

Figure 5: The construction H_3 underlying Theorem 1.3 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 3n/2$. The gray dotted boxes represent the compress function, instantiated with a two-block Merkle-Damgård construction, that uses key $\in \{0,1\}^n$ as the key and $y_{i-1}\|m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a - 2n$ bit message is split evenly into the first and second call to h, indicated in the figure by a diamond.

a n bit output for compress. The full construction of H_4 is illustrated in Figure 6. We conjecture that this Merkle tree-based construction is optimally secure for collision resistance (see Remark 1.5 for more details), but analyzing its security turns out to be highly non-trivial. In particular, our current analysis is only optimally secure when $ST^2 \leq 2^n$, as stated in the following theorem.

Theorem 1.4. *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a \geq 2n$. Then, there is an $H_4\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ such that $ST^2 \leq 2^n$ and any (S,T)-attacker, their advantage in finding a collision in $H_4(\mathsf{key}, \cdot)$ relative to a random key $\leftarrow_\$ \{0,1\}^\kappa$ is $O(S/2^n + T^2/2^n)$.*
2. *One evaluation of H_4 on a given key and a b-bit input requires $3 \cdot b/(2a - 3n)$ queries to h.*

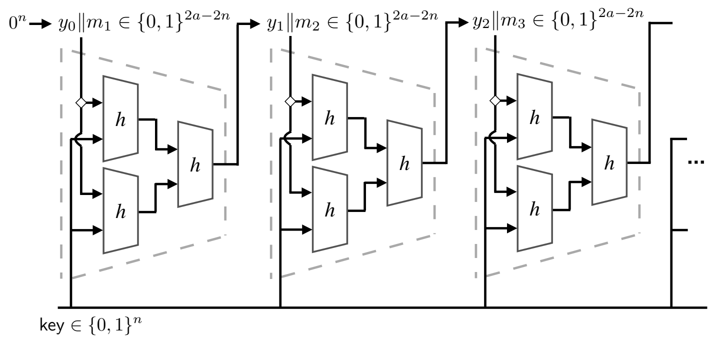
0^n
$y_0\|m_1 \in \{0,1\}^{2a-2n}$
$y_1\|m_2 \in \{0,1\}^{2a-2n}$
$y_2\|m_3 \in \{0,1\}^{2a-2n}$
h
...
key $\in \{0,1\}^n$

Figure 6: The construction H_4 underlying Theorem 1.4 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geq 2n$. The gray dotted boxes represent the compress function, instantiated with a two-level Merkle tree, that uses key $\in \{0,1\}^n$ in each leaf and $y_{i-1}\|m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a-2n$ bit message is split evenly between the two leaves of the Merkle tree, indicated by a diamond in the figure. We require $a \geq 2n$ so that both outputs from the leaves can be fed into the next layer of the Merkle tree.

	Advantage	Efficiency	Input Size	Assumptions
MD	$\Theta(ST^2/2^n)$	$b/(a-n)$	$a > n$	None
H_1 (Thm. 1.1)	$\Theta(S/2^n + T^2/2^n)$	$b/(a-2n)$	$a > 2n$	None
H_2 (Thm. 1.2)	$\Theta(S/2^n + T^2/2^n)$	$2 \cdot b/(a-n)$	$a > n$	$S \leq T, ST^2 \leq 2^n$
H_3 (Thm. 1.3)	$\Theta(S/2^n + T^2/2^n)$	$2 \cdot b/(2a-3n)$	$a > 1.5n$	$S \leq T$
H_4 (Thm. 1.4)	$\Theta(S/2^n + T^2/2^n)$	$3 \cdot b/(2a-3n)$	$a \geq 2n$	$ST^2 \leq 2^n$

Table 1: A summary of our results as well as the standard MD construction for reference. The advantage of a construction is given in terms of the probability of (S,T)-attackers to find collisions relative to a random key. The efficiency is measured in terms of the number of calls to h that maps a-bit inputs to n when processing a b-bit input. The assumptions column specifies conditions on various parameters.

We summarize our results in Table 1.

Remark 1.5 (A conjecture on the security of keyed Merkle trees)**.** *The main building block in our construction of Theorem 1.4 is a Merkle tree where the key value* key *is included only at the leaves. Concretely, we include a key* key *in each of the leaves of a Merkle tree and then fill in the rest of the leaves with bits of some message $m \in \{0,1\}^*$, and then run the Merkle tree construction (with an unkeyed hash) as normal to get an n-bit output. In this work, we analyze the simplest case where the Merkle tree has depth 2 with only two leaves. However, this naturally generalizes to any number of ℓ leaves resulting in a tree of depth $O(\log \ell)$.*

We conjecture that this approach, where only the leaves are keyed, is as secure as the Merkle tree approach of Figure 2 where the inner hash function is keyed at every invocation, including interior nodes. This latter approach requires a larger, more complicated, inner hash function, so we would like to avoid this if at all possible.

First, for the simple case of a depth two tree with keyed leaves, we conjecture that the bound we show in this work is not tight (see Theorem A.3 for the exact bound we show). Namely, we believe that we should not need to assume $ST^2 \leq 2^n$ (or make any assumptions on S, T) in order to get optimal security. Second, we believe that this intuition should extend to the arbitrary depth Merkle trees that are keyed at the leaves, and we conjecture that it should also achieve optimal (S,T) security without any assumptions on S, T. However, even getting a bound in this case that is optimal in the setting $ST^2 \leq 2^n$ we believe would be very interesting. Additionally, handling deeper Merkle trees could potentially allow for constructions that do not necessarily have even a two-to-one structure, meaning that we could build a keyed hash function based on Merkle trees without assuming the hash function h has input size $a \geq 2n$.

1.2 Related Work

The motivation for this work comes from a recent line of results on the non-uniform security loss of various hashing mechanisms.

For Merkle-Damgård's construction [Mer89, Dam89], this was first studied by Coretti et al. [CDGS18] who showed how to find collisions with probability $\Omega(S/2^\kappa + ST^2/2^n)$. The idea is reminiscent of the rainbow tables attack due to Oechslin [Oec03] (in turn building on Hellman [Hel80]). The collisions they get are rather long (of length proportional to T). Akshima et al. [ACDW20] generalized the attack to get an ℓ-block collision with probability $\Omega(S/2^\kappa + ST\ell/2^n)$ and showed that this attack is optimal for $\ell = 2$. Ghoshal and Komargodski [GK22] showed that this attack is optimal for all constant values of ℓ and Akshima, Guo, and Liu [AGL22] almost proved the tightness of the bound for all ℓs by showing that the best possible attack has advantage $O(ST\ell/2^n \cdot (1 + ST^2/2^n) + T^2/2^n)$. For a single-block Merkle-Damgård (i.e., just a compressing random oracle), Dodis, Guo, and Katz [DGK17] showed that including a random key (optimally) defeats preprocessing attacks.

For Sponge [BDPVA07, BDPA08], Coretti, Dodis, and Guo [CDG18] stated a related attack with advantage $\Omega(ST^2/2^\kappa + T^2/2^r)$ (with r being a "rate" parameter of the scheme). Again, this attack resulted in very long collisions. The attack was formalized and extended to ℓ-block collisions with advantage $\Omega(ST\ell/2^\kappa + T^2/2^r)$ by Freitag, Ghoshal and Komargodski [FGK22]. Freitag et al. also proved several upper bounds on the advantage of any attacker, but their bounds are not known to be tight.

Indifferentiability. Our work focuses on collision resistance, but there are other security properties of interest (such as inversion, second preimage resistance, pseudo-randomness, and unpredictability). In the uniform security setting there is a well-known framework called *indifferentiability* (due to Maurer, Renner, and Holenstein [MRH04]) that is used to show that a (wide) class of security goals are simultaneously met. This allows to modularly transition to a (simpler) hybrid world where a complicated hash function construction is replaced with a monolithic random oracle (see, for example, [CDMP05]). Such transitions are known to work for all single-stage games but not for multi-stage games [RSS11]. Our non-uniform security model is fundamentally a two-stage model and therefore the indifferentiability framework (as is) does not apply. It is an interesting open problem to find an analogue in the non-uniform setting.

2 Technical Overview

In this section, we give a high level overview of our main techniques. Recall, our goal is to construction a variable-input length, keyed, hash function $H^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ from an idealized, fixed-input length hash function $h\colon \{0,1\}^a \to \{0,1\}^n$.

Non-uniform security in the AI-ROM. We consider non-uniform (S,T)-attackers $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the auxiliary-input random oracle model (AI-ROM) [Unr07] with the following structure. First, h is randomly sampled from the space of all a-bit to n-bit functions. Then, in the preprocessing phase, $\mathcal{A}_1$ has unbounded access to h and outputs an advice string σ such that $|\sigma| \leq S$. The online phase $\mathcal{A}_2$ receives auxiliary input σ and a random key $\mathsf{key} \leftarrow \{0,1\}^\kappa$ as input, and then has to find two distinct messages $\mathsf{msg}, \mathsf{msg}'$ such that $H^h(\mathsf{key}, \mathsf{msg}) = H^h(\mathsf{key}, \mathsf{msg}')$ while making at most T queries to h. Our goal is to give constructions H^h such that no (S,T)-attacker as above can find a collision with better than $O(S/2^\kappa + T^2/2^n)$ probability. This is "optimal" in the sense that this matches a naive attack against a purely random H: the preprocessing attacker stores collisions for $\Omega(S)$ keys, and the online attacker either gets "lucky" and receives one of those keys as input or performs a standard birthday-style attack.

Merkle-Damgård framework with a keyed inner hash. We consider a general framework based on the Merkle-Damgård (MD) transformation where we instantiate the inner hash function with a keyed one. Let $s \in \mathbb{N}$ be the desired message block size. Then, given a function $g\colon \{0,1\}^{\kappa+n+s} \to \{0,1\}^n$, we can build a function H^g where any attack on the collision resistance of H^g implies an attack on g. The idea behind H^g is as follows. We first break our message up into blocks $m_1, \ldots, m_\ell$ of size $a - \kappa - n$. We initialize the value $y_0 = 0^n$, and for $i = 1, 2, \ldots, \ell$ we compute $y_i = g(\mathsf{key}\|y_{i-1}\|m_i)$. Finally, we output y_ℓ. It is known (e.g. see Section 8.5 of [GB08]) that if you can find a collision in the MD construction H^g for a keyed g in the uniform setting, then this implies you can find a collision in g. Indeed, this reasoning extends to the non-uniform setting with (S, T)-attackers in the AI-ROM. Hence, this shows that H^g is as secure as g. So, our new goal is to construct such a g with "optimal security" given an idealized hash function $h\colon \{0,1\}^a \to \{0,1\}^n$.

Our first observation is that if $a \geq \kappa + n + s$, then we can simply use $g = h$. Furthermore, h has optimal security $O(S/2^\kappa + T^2/2^n)$ (first formalized by [DGK17] in the AI-ROM), so we are done! So our next goal is to try to use an h from minimal assumptions. Namely, can we get a keyed hash function with arbitrary length input from any $h\colon \{0,1\}^a \to \{0,1\}^n$ where a is much smaller, i.e. even $a = n + 1$? This will allow us to focus on building as simple a primitive as possible which we can bootstrap to a full variable-input length hash function with optimal security.

Next, we note that for any $a > \max(\kappa, n)$, we can always do the standard Merkle-Damgård transformation using h to construct g, where key is not fed into every invocation of h. For standard MD, it makes sense to set $\kappa = n$ since we use the key key as the initialization vector. Recall, the MD^h construction sets $y_0 = \mathsf{key}$, computes $y_i = h(y_{i-1}\|m_i)$ for $i = 1, \ldots, \ell$, and outputs y_ℓ. This approach has a major downside in that instantiating MD without inserting key into each invocation of h suffers non-trivial time-space tradeoffs. In general, there is an attack on the general MD^h construction with advantage $\Omega(ST^2/2^n)$. This is strictly worse than the optimal bound of $O(S/2^n + T^2/2^n)$ for any setting of parameters with $S, T \gg 1$. However, this attack finds very large—roughly length T—collisions. In our setting, we only care about using MD^h to get a function g with inputs of size $2n + s$. Thus, we leverage a recent line of work (see [CDGS18, ACDW20, CGLQ20, GK22, AGL22]) that shows that if you only use MD^h on ℓ-block messages, then the best known attack has advantage at most $O(ST\ell/2^n + T^2/2^n)$ (and further this is provably tight for constant ℓ [ACDW20, GK22] and, when $ST^2 \leq 2^n$, is provably tight for all ℓ [AGL22]).

So, if we instantiate g in our framework with a fixed-length MD^h construction where we set κ and s to be equal to the output length n, we get a construction for H^h where any (S, T)-attacker has advantage at most $O(ST/2^n + T^2/2^n)$ (up to $\mathrm{poly}(n)$ factors). This is only "optimal," however, under the (strong) assumption that $S \leq T$.

Our main technical contribution is instantiating g in the framework above using a new keyed Merkle tree approach, which does not require the assumption that $S \leq T$.

2.1 Keyed Merkle Tree Analysis

For the rest of this technical overview, we focus on our analysis of the keyed Merkle tree construction.

Construction. We start by defining the 2-level construction. We want a keyed function $g\colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ from a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geq 2n$. To do so, we split the $(n + s)$-bit input into two parts, call them m_L, m_R, of size at most $a - \kappa$ (hence we require here that $a - \kappa \geq (n + s)/2$, or $a \geq \kappa + n/2 + s/2$). We concatenate each part with the key key and compute $y_L \leftarrow h(\mathsf{key}\|m_L)$ and $y_R \leftarrow h(\mathsf{key}\|m_R)$. We then concatenate y_L with y_R and feed the resulting string into h to get the output $z \leftarrow h(y_L\|y_R)$ (we require here that $a \geq 2n$). For technical purposes, we "domain separate" each call to h, so $y_L = h_1(\mathsf{key}\|m_L)$, $y_R = h_2(\mathsf{key}\|m_R)$, and $z = h_3(y_L\|y_R)$.

Analysis. We want to bound the probability that any (S, T)-attacker can find a collision in this keyed Merkle tree construction of g. To simplify the analysis here, we consider the case where κ and s are equal to n. Hence, each call to h takes two n-bit inputs and has one n-bit output. We also assume from the start that $ST^2 \leq 2^n$.

We start with the following observation. The probability that any (S,T)-attacker finds a collision at the leaves, corresponding to the calls to h_1, h_2, is at most $O(S/2^n + T^2/2^n)$. Because these calls include the key key, if such an event happened, we could reduce to finding a collision in h directly. So, the challenge is to reason about the advantage of an (S,T)-attacker finding a collision at the second level of the Merkle tree, which is only implicitly related to the key key in the Merkle tree construction. We have to somehow characterize all possible ways that an (S,T)-attack can encode information about h in its advice string from the preprocessing phase.

One of the first tools one often turns to in such analysis is to use the *presampling* technique from [Unr07, CDGS18]. We note that if we were to use the presampling technique, we would obtain a term of the form $ST/2^n$ in out bound, which is optimally secure only in the range $S \leq T$. Our main technical contribution is getting an optimally secure protocol for the range $S \gg T$, which therefore requires techniques other than presampling.

The AGL [AGL22] framework: reducing to multi-instance games. To make our lives significantly easier, we use the multi-instance framework of [AGL22], previously used in somewhat different forms in [IK10, ACDW20, CGLQ20, GK22]. At a very high level, this framework gives a way to reason about (S,T)-attackers using an *average-case* advice string rather than a worst-case one. In more detail, they show how to bound the advantage that any (S,T)-attacker $\mathcal{A}$ finds a collision in g^h by the advantage of a (uniform) attacker $\mathcal{B}$ in the following game. First, a random function h is sampled. Then, $\mathcal{B}$ has to win the following game for all $i = 1, \ldots, S$ sequentially, where it is allowed to maintain arbitrary state (that it generates) between each successive game. In each game i, the attacker $\mathcal{B}$ receives a random key key_i, its state from the previous games, and has to come up with a pair of messages $\mathsf{msg}_i, \mathsf{msg}'_i$ such that $g^h(\mathsf{key}_i, \mathsf{msg}_i) = g^h(\mathsf{key}_i, \mathsf{msg}'_i)$ using at most T queries to h. [CGLQ20, AGL22] show that if the advantage of $\mathcal{B}$ is at most δ^S, then the advantage of the (S,T)-attacker $\mathcal{A}$ is at most 2δ. The magic of this framework is that we can analyze the advantage of $\mathcal{B}$ in each game i only given its state from the previous games, instead of having to reason about arbitrary advice strings as in the case of (S,T)-attackers. Namely, we can lazily sample h on any point that $\mathcal{B}$ has not queried, in a way that is independent of $\mathcal{B}$'s current state.

Note that it suffices to show that the advantage of $\mathcal{B}$ "in game i" is at most δ *given* it has won all previous games. Let W_i be the event that $\mathcal{B}$ wins game i and $W_{<i}$ be the event that $\mathcal{B}$ wins all games before i. This follows since $\Pr[W_1 \wedge \ldots \wedge W_S] = \prod_{i=1}^{S} \Pr[W_i | W_{<i}] \leq \delta^S$ if $\Pr[W_i | W_{<i}] \leq \delta$ for all $i \in [S]$. Hence, our goal is to show that $\Pr[W_i | W_{<i}] \leq O(S/2^n + T^2/2^n)$, up to $\mathrm{poly}(n)$ factors.

Knowledge gaining event: bounding "hitting" queries. Now, to even further simplify the analysis of $\Pr[W_i | W_{<i}]$, we define a key "knowledge gaining event" (based on the techniques of [AGL22]) representing the kind of information that $\mathcal{B}$ may have encoded into its state based on the queries it made to h before game i has started. At a high level, this is an event that we show happens with very small probability (technically at most $2^{-2i \cdot n}$) for an average-case advice string for $\mathcal{B}$ at the start of game i. Then, assuming this event does not occur, we can more easily characterize the strategies of $\mathcal{B}$.

To define this event, we introduce some notation to characterize $\mathcal{B}$'s queries. We refer to all $(i-1) \cdot T$ queries $\mathcal{B}$ makes before the start of game i as "offline" queries, and we refer to the T queries made in game i as "online" queries. An offline query is said to be "hitting" if its output is equal to an output or either of the two inputs to some prior query, i.e. it "hits" a previous query. We are now ready to state our key knowledge gaining event.

- We say that $\mathsf{E}^i_{\mathsf{hit}}$ holds if there are more than $i \cdot \mathrm{poly}(n)$ hitting queries among the $(i-1) \cdot T$ offline queries.

Briefly, we justify why $\mathsf{E}_{\mathsf{hit}}^{i}$ holds with very small probability. The output for each query is uniformly sampled, and there are at most $3 \cdot (i-1) \cdot T$ values to hit across inputs/outputs in the previous $(i-1) \cdot T$ offline queries. So the probability each offline query is a hitting query is at most $3iT/2^n$, meaning we expect at most $3i^2T^2/2^n$ hitting queries accounting for all $(i-1) \cdot T$ offline queries. Furthermore, we show using a Chernoff bound that there will not be more than $i \cdot \mathrm{poly}(n) \cdot \max(1, iT^2/2^n) = O(i)$ hitting queries (assuming $ST^2 \leq 2^n$) with high probability (recall that we ignore $\mathrm{poly}(n)$ terms).

A case analysis based on collision queries. Now, assuming there are at most $i \cdot \mathrm{poly}(n)$ hitting queries, we are ready to show that $\Pr[W_i | W_{<i}] \leq O(S/2^n + T^2/2^n)$. To do so, we look at the following "collision" queries corresponding to the valid collision $\mathsf{msg}_i = (m_L, m_R) \neq \mathsf{msg}'_i = (m'_L, m'_R)$ that $\mathcal{B}$ outputs in game i (we assume that $\mathcal{B}$ makes all of these queries at some point during or before game i).

- Q_1, Q_2, Q_3 are the queries $y_L \leftarrow h_1(\mathsf{key}_i \| m_L)$, $y_R \leftarrow h_2(\mathsf{key}_i \| m_R)$, $z \leftarrow h_3(y_L \| y_R)$, respectively.
- Q'_1, Q'_2, Q'_3 are the queries $y'_L \leftarrow h_1(\mathsf{key}_i \| m'_L)$, $y'_R \leftarrow h_2(\mathsf{key}_i \| m'_R)$, $z \leftarrow h_3(y'_L \| y'_R)$, respectively.

Recall that we assumed the collision occurs among Q_3, Q'_3 (not at the leaves), so it must be the case that $(y_L, y_R) \neq (y'_L, y'_R)$ and queries Q_3, Q'_3 are distinct.

If all of these collision queries are online (were first made during game i), then clearly $\Pr[W_i | W_{<i}] \leq O(T^2/2^n)$. Specifically, as Q_3 and Q'_3 are online (and distinct by assumption) and form a collision, this follows by a birthday bound on at most T online—and hence lazily sampled—queries that $\mathcal{B}$ makes during game i. The challenge comes when analyzing the cases where $\mathcal{B}$ may have made some of these queries *before* game i, so it could have encoded information about these queries in its state. To do so, we consider the following remaining cases, which cover all possible strategies that $\mathcal{B}$ may employ.

As we already considered when both Q_3, Q'_3 are online queries, it must be the case that one of Q_3, Q'_3 must be an offline query. Assume without loss of generality that Q_3 is offline. Then either **(A)** both Q_1, Q_2 are online, **(B)** exactly one of Q_1, Q_2 are online, or Q_1, Q_2 are both offline. The latter case implies that Q_1, Q_2, Q_3 are all offline. Then either is the case that **(C)** Q'_3 is online, or Q'_3 is also offline. If Q'_3 is also offline, then either we reduce to case (A) or (B) above by symmetry, or it holds that **(D)** all $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are offline. So, it suffices to show in cases (A-D) that $\Pr[W_i | W_{<i}] \leq \delta \leq O(S/2^n + T^2/2^n)$, at least assuming $ST^2 \leq 2^n$. We proceed to give the main ideas behind each of these cases.

(A) Q_3 is offline but Q_1, Q_2 are online.

There are at most $(i-1) \cdot T$ options for the offline query Q_3. Both online queries Q_1 and Q_2 have to hit such a query, which happens with at most $(i-1) \cdot T \cdot (T/2^n)^2 = O(iT^3/2^{2n}) \leq O(T/2^n)$ when $ST^2 \leq 2^n$ since $i \leq S$.

(B) Q_1, Q_3 are offline but Q_2 is online (symmetrically for Q_1 online and Q_2 offline).

In this case, we claim we can "associate" the key key_i in game i to the query Q_3 since both Q_1 and Q_3 are offline queries. If we can associate key_i to at most k possible Q_3 queries, then this implies the probability the output of some online query hits an input of such an associated query is at most $k \cdot T/2^n$. But how many Q_3 queries can we associate to a key key_i?

In the worst case, key_i may be associated to $(i-1) \cdot T$ many Q_3 queries, but this implies a suboptimal bound of $O(iT^2/2^n)$. But this cannot be true for too many values of key

simultaneously. In particular, if a Q_3 query is associated with more than k possible values of key, this means there are k hitting queries, so each Q_3 query can be associated with at most $O(i)$ keys. This implies there are at most $O(i^2T)$ pairs of associated key values with potential Q_3 queries, meaning a random key_i value will be associated with at most $O(i^2T/2^n)$ potential Q_3 values on average. Plugging this average-case bound into k above, this implies a bound of $O(i^2T^2/2^{2n}) \leq O(S/2^n)$ assuming $ST^2 \leq 2^n$ given $i \leq S$.

(C) Q_1, Q_2, Q_3 are offline but Q'_3 is online.

In this case, Q'_3 is a distinct query from Q_3 by assumption, but again we can "associate" Q_3 with key_i as above. Then, since Q'_3 must share an output with Q_3, the same argument as above gives a bound of $O(S/2^n)$ in this case.

(D) All collision queries $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are offline.

In this case, we show that every full collision structure with respect to some key among the offline queries leads to a hitting query. Furthermore, two collision structures cannot share the same hitting query. So if there are at most $O(i)$ hitting queries, the probability $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all offline for a random key_i is at most $O(i/2^n) \leq O(S/2^n)$.

Thus, we showed that $\Pr[W_i|W_{<i}] \leq O(S/2^n + T^2/2^n)$ no matter when $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ were queried before or during game i. Further, recall the last case where there is a collision at one of the leaves (corresponding to Q_1, Q'_1 or Q_2, Q'_2), which can happen with at most $O(S/2^n + T^2/2^n)$ probability since such a collision directly involves key_i. Thus, in all possible cases, we have shown $\Pr[W_i|W_{<i}] \leq \delta \leq O(S/2^n + T^2/2^n)$, at least assuming $ST^2 \leq 2^n$. Finally, by the framework of [CGLQ20, AGL22], this implies the same bound (up to a multiplicative factor of 2) on the advantage of finding a collision for any (S, T)-attacker.

3 Preliminaries

We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the natural numbers. The set of all functions with domain D and range R is denoted by $\mathsf{Fcs}(D, R)$. We let $*$ denote a wildcard element. For example $(*, z) \in L$ is true if there is an ordered pair in L where z is the second element (the type of the wildcard element shall be clear from the context). For a random variable X we use $\mathsf{E}[X]$ to denote its expected value. We use $x \leftarrow\$ D$ to denote sampling x uniformly sampling from the elements of D. All logarithms in this paper are for base 2 unless otherwise specified.

For a bit-string s, we use $|s|$ to denote the number of bits in s. For two strings s_1, s_2, we use $s_1 \| s_2$ to denote the concatenation of two strings. We use standard regular expression notation where s^* denotes 0 or more copies of s, s^+ denotes one or more copies of s, and s^k denotes k copies of s. Similarly, for a set S, we use S^*, S^+, and S^k to represent 0 or more, 1 or more, of k elements takes from a set S. In particular, we use $\{0,1\}^*$ to represent any arbitrary string of bits. We use the notation $\{0,1\}^{\leq k}$ to represent a string of length at most k.

Chernoff bound. We state a Chernoff bound which we use in the technical part of the paper.

Proposition 3.1. Let $n \in \mathbb{N}$. Let $X_1, X_2, \ldots, X_n$ be independent 0-1 random variables. Let $X = \sum_{i=1}^{n} X_i$. Let μ' be such that $\mathsf{E}[X] \leq \mu'$. Then we have that

$$\Pr\left[X \geq (1+\delta)\mu'\right] \leq e^{-\delta\mu'/3} .$$

```
Game $G^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$
    1. $h \leftarrow_{\$} \mathsf{Fcs}(\{0,1\}^a, \{0,1\}^n)$
    2. $\sigma \leftarrow_{\$} \mathcal{A}_1(h)$
    3. $\mathsf{key} \leftarrow_{\$} \{0,1\}^\kappa$
    4. $(\mathsf{msg}, \mathsf{msg}') \leftarrow_{\$} \mathcal{A}_2^h(\sigma, \mathsf{key})$
    5. Return true if:
        (a) $\mathsf{msg} \neq \mathsf{msg}'$, and
        (b) $g^h(\mathsf{key}, \mathsf{msg}) = g^h(\mathsf{key}, \mathsf{msg}')$
    6. Else, return false
```

Figure 7: The collision resistance game $G^{\mathsf{ai\text{-}cr}}_{g^h}$ in AI-ROM for a function g^h based on a random oracle $h : \{0,1\}^a \to \{0,1\}^n$. The construction g^h has a parameter κ associated with it, where κ is the bit length of key used in g.

Notice that this version is somewhat non-standard as it even works when we know only an upper bound on the expectation (usually, in standard formulations of Chernoff bound, we need to know the expectation exactly). For the sake of completeness we include a proof of this Chernoff bound in Appendix B.

Auxiliary-input Random Oracle Model (AI-ROM). The auxiliary-input random oracle model, introduced by Unruh [Unr07], captures the power of non-uniform adversaries against random oracles. An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in this model is formalized as a two stage adversary. In its first stage, which is referred to as the preprocessing phase, $\mathcal{A}_1$ has unbounded access to the random oracle h, and outputs any arbitrary S-bit advice string or auxiliary input σ. In the second stage, referred to as the online phase, gets σ as input, $\mathcal{A}_2$ can make at most T queries to its oracle h. Its aim is to accomplish some task involving h, e.g. find a collision in a construction based on h. We refer to such an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as an (S,T)-attacker.

Collision resistance of g^h in AI-ROM. We next formalize the keyed-collision resistance of an iterated hash function construction g relative to a hash function $h : \{0,1\}^a \to \{0,1\}^n$ in the AI-ROM. The construction g has a parameter κ associated with it, where κ is the bit length of key used in g. It first samples a random function $h : \{0,1\}^a \to \{0,1\}^n$. The adversary $\mathcal{A}_1$ gets unbounded access to h, and it outputs an advice string σ. At this time, $\mathcal{A}_2$ is given the auxiliary input σ, a randomly sampled key from $\{0,1\}^\kappa$, as well as oracle access to h, and it needs to find $\mathsf{msg} \neq \mathsf{msg}'$ such that $g^h(\mathsf{key}, \mathsf{msg}) = g^h(\mathsf{key}, \mathsf{msg}')$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}$ is formally defined in Fig. 7.

Definition 3.2 (AI-CR Advantage)**.** The advantage of an adversary $\mathcal{A}$ against the collision resistance of g^h in the AI-ROM is

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A}) = \Pr\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A}) = \mathsf{true}\right].$$

For parameters $S, T \in \mathbb{N}$, we overload notation and denote

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T) = \max_{\mathcal{A}} \left\{\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A})\right\},$$

where the maximum is over all (S, T)-attackers.

Throughout the paper, for any (S, T)-attacker $\mathcal{A}$ that outputs messages $\mathsf{msg}, \mathsf{msg}'$ that causes $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A})$ to output $\mathtt{true}$ on a key key, we assume that $\mathcal{A}$ has fully queried $g^h(\mathsf{key}, \mathsf{msg})$ and $g^h(\mathsf{key}, \mathsf{msg}')$. This is true without loss of generality (up to constant factors in the advantage) as if there exists any (S, T)-attacker $\mathcal{A}$ that does not, you can construct an $(S, T + 2\ell)$-attacker $\mathcal{B}$ that does, where g^h requires at most ℓ invocations of h to compute either $g^h(\mathsf{key}, \mathsf{msg})$ or $g^h(\mathsf{key}, \mathsf{msg}')$. As $\ell \leq T$, the resulting attacker will have comparable advantage up to constant factors in T. We note that this is a standard assumption in existing related works in the AI-ROM.

On padding. The variable-input length hash functions we consider of this work all act on messages which have been parsed into many fixed size blocks of some specified size s. We therefore need a padding function that takes arbitrary length inputs and converts them to a sequence of fixed-size blocks. We need to ensure that this padding function maintains certain properties like injectivity in order to guarantee that if an adversary finds a collision on the padded versions of messages, then it implies a collision with respect to the underlying messages as well. For the purpose of this paper, we define the following padding function, which is a slightly simplified version of the padding function used by the SHA family of hash functions (see [GB08, Section 8.5] for more discussion on MD-compliant padding functions). The function pad we use takes in a message $\mathsf{msg} \in \{0,1\}^*$, an integer $s \in \mathbb{N}$ representing the size of each block, and an integer n that stipulates that $|\mathsf{msg}| \leq 2^n$. The construction is formally defined as follows.

$\underline{\mathsf{pad}(\mathsf{msg}, s, n)}$:

1. Let $k = s - ((|\mathsf{msg}| + n) \bmod s + 1)$.
2. Interpret $|\mathsf{msg}| \in [2^n]$ as an n-bit string.
3. Output $(m_1, \ldots, m_\ell) \in (\{0,1\}^s)^\ell$ where $m_1 \| \ldots \| m_\ell = \mathsf{msg} \| |\mathsf{msg}| \| 1 \| 0^k$.

We formalize the guarantees we use for this padding function in the following theorem.

Theorem 3.3 (Padding). *Let $s, n \in \mathbb{N}$. The function $\mathsf{pad}(\mathsf{msg}, s, n)$ on messages $\mathsf{msg} \in \{0,1\}^{2^n}$ satisfies the following properties:*

1. *$|\mathsf{pad}(\mathsf{msg}, s, n)| \in (\{0,1\}^s)^\ell$ for $\ell = \lceil (|\mathsf{msg}| + n + 1)/s \rceil$.*
2. *There is a unique decoding procedure that outputs msg given $\mathsf{pad}(\mathsf{msg}, s, n)$, and outputs $\perp$ on invalid padded messages.*
3. *If $\mathsf{pad}(\mathsf{msg}, s, n) = \mathsf{pad}(\mathsf{msg}', s, n)$, then $\mathsf{msg} = \mathsf{msg}'$.*
4. *If $|\mathsf{msg}| < |\mathsf{msg}'|$, then $\mathsf{pad}(\mathsf{msg}, s, n)$ is not a suffix of $\mathsf{pad}(\mathsf{msg}', s, n)$.*

Proof. The length of a padded message follows by construction.

For the decoding process, note that to recover msg from $\mathsf{pad}(\mathsf{msg}, s, n)$, you simply need to remove the trailing 0s and preceding 1. Parse the remaining final n bits as an integer $|\mathsf{msg}|$ and check that there are exactly $|\mathsf{msg}|$ bits leftover. If such a recovery process fails, we output $\perp$ and say that the padded message is invalid (as it corresponds to no message msg).

To see injectivity, suppose two messages $\mathsf{msg}, \mathsf{msg}'$ are padded to the same value. This means they both end in $1 \| 0^k$ for $k = (|\mathsf{msg}| + n + 1) \bmod s$ and $|\mathsf{msg}| = |\mathsf{msg}'|$. Thus, if both are valid padded messages, they only consist of $|\mathsf{msg}|$ remaining bits, which are the same, implying that $\mathsf{msg} = \mathsf{msg}'$, as required.

Construction $\mathsf{KMD}^h(\mathsf{key}, \mathsf{msg})$:

1. $(m_1, m_2, \ldots, m_\ell) \leftarrow \mathrm{pad}(\mathsf{msg}, s, n)$ where $\ell = \lceil (|\mathsf{msg}| + n + 1)/s \rceil$ given by Theorem 3.3.
2. Initialize $y_0 = 0^n$, and compute $y_i \leftarrow h(\mathsf{key}, y_{i-1} \| m_i)$ for $i = 1, \ldots, \ell$.
3. Output y_ℓ.

Figure 8: The keyed Merkle-Damgård construction $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ given any underlying function $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, where κ is the key length, n is the output length, and s is the message block size.

Now suppose that $|\mathsf{msg}| < |\mathsf{msg}'|$ but $\mathsf{pad}(\mathsf{msg}, s, n)$ is a suffix of $\mathsf{pad}(\mathsf{msg}', s, n)$. This implies that both strings end in $1 \| 0^k$ for $k = (|\mathsf{msg}| + n + 1) \bmod s$ and $|\mathsf{msg}| = |\mathsf{msg}'|$. Thus, one of the encodings must be invalid as we assumed $|\mathsf{msg}| < |\mathsf{msg}'|$. ■

4 Merkle-Damgård Framework with a Keyed Inner Hash

In this section, we lay out the general framework for our main results, based on the Merkle-Damgård transform using a keyed inner hash. We note that this framework has been explicitly considered in the uniform setting in Section 8.5 of the lecture notes of Goldwasser and Bellare [GB08].[6] We extend this framework to the preprocessing setting, modeled by the AI-ROM of Unruh [Unr07], noting that the high level ideas are similar.

For a key length κ, output length n, and message block size s, we assume an underlying primitive $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$. In other words, viewing the primitive h as a function from $\{0,1\}^a$ to $\{0,1\}^n$, this implies that $a = \kappa + n + s$.

Given such a primitive h, we define the following keyed Merkle-Damgård hash function $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. On input key $\mathsf{key} \in \{0,1\}^\kappa$ and message $\mathsf{msg} \in \{0,1\}^*$ of length at most 2^n, the function KMD^h first pads the message to split it into $\ell = \lceil (b + n + 1)/s \rceil$ message blocks of size s as in Theorem 3.3. It then essentially computes the Merkle-Damgård hash function using the underlying hash function h, except that the key key is inserted into every invocation of h. This is formalized in Figure 8.

Since the key key is included in every call to the underlying primitive h, it follows that any (S, T)-attacker that finds a collision in KMD^h with respect to a key key also finds a collision in h with respect to key. This is formalized via a reduction, which gives the following theorem. Again, we note that the following theorem very closely follows the reduction given in [GB08, Section 8.5], but we give the full details in the AI-ROM for completeness.

Theorem 4.1. *Let $\kappa, n, s \in \mathbb{N}$. Let $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ be any function, and let $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. Then, for every $S, T \in \mathbb{N}$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMD}^h}(S, T) \leq \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S, T).$$

[6]The existence of this variant of the Merkle-Damgård transform has gone completely unnoticed in recent works studying non-uniform security of this transformation [CDGS18, ACDW20, GK22, AGL22].

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any (S, T)-attacker against the collision resistance of the construction KMD^h. Recall that we assume for simplicity that if $\mathcal{A}_2$ outputs a successful collision $\mathsf{msg}_{\mathcal{A}} \neq \mathsf{msg}'_{\mathcal{A}}$ such that $\mathsf{KMD}^h(\mathsf{key}, \mathsf{msg}_{\mathcal{A}}) = \mathsf{KMD}^h(\mathsf{key}, \mathsf{msg}'_{\mathcal{A}})$, then $\mathcal{A}_2$ queried all the necessary values to fully compute each of these functions. We construct an (S, T)-attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the collision resistance of h as follows.

In the preprocessing phase, $\mathcal{B}_1(h)$ simply computes $\sigma \leftarrow \mathcal{A}_1(h)$ and outputs σ where $|\sigma| \leq S$. In the online phase, $\mathcal{B}_2^h(\sigma, \mathsf{key})$ is given a random key $\mathsf{key} \leftarrow \{0,1\}^\kappa$ that it needs to find a collision for. $\mathcal{B}_2$ computes $(\mathsf{msg}_{\mathcal{A}}, \mathsf{msg}'_{\mathcal{A}}) \leftarrow \mathcal{A}_2^h(\sigma, \mathsf{key})$, where it uses its own query access to the function h to answer queries that $\mathcal{A}_2$ makes to h. $\mathcal{B}_2$ sets $(m_1, \dots, m_\ell) \leftarrow \mathrm{pad}(\mathsf{msg}_{\mathcal{A}}, s, n)$ and $(m'_1, \dots, m'_{\ell'}) \leftarrow \mathrm{pad}(\mathsf{msg}'_{\mathcal{A}}, s, n)$, where $\ell = \lceil (|\mathsf{msg}_{\mathcal{A}}| + n + 1)/s \rceil$ and $\ell' = \lceil (|\mathsf{msg}'_{\mathcal{A}}| + n + 1)/s \rceil$. It initializes $y_0 = y'_0 = 0^n$ as in the construction of KMD^h. For $i = 1, \dots, \ell$, it looks for the queries $y_i \leftarrow h(\mathsf{key}, y_{i-1} \| m_i)$, and for $j = 1, \dots, \ell'$, it looks for the queries $y'_j \leftarrow h(\mathsf{key}, y'_{j-1} \| m'_j)$ made by $\mathcal{A}_2$ (which must exist if $\mathcal{A}_2$ succeeds in outputting a collision by assumption). Let $i^\star, j^\star$ be the minimal $i, j > 0$ such that $y_i = y'_j$ but $y_{i-1} \| m_i \neq y'_{j-1} \| m'_{j-1}$. If such an $i^\star, j^\star$ exists, $\mathcal{B}_2$ outputs $(\mathsf{msg}, \mathsf{msg}')$ where $\mathsf{msg} = y_{i^\star-1} \| m_{i^\star}$ and $\mathsf{msg}' = y'_{j^\star-1} \| m'_{j^\star}$. Otherwise, $\mathcal{B}_2$ aborts and outputs $\perp$.

It remains to analyze the complexity and success probability of $\mathcal{B}$. First, as $\mathcal{B}$ simply runs $\mathcal{A}$ in the preprocessing and online phases, it follows that if $\mathcal{A}$ is an (S, T)-attacker, then so is $\mathcal{B}$.

For the success probability, we claim that whenever $\mathcal{A}$ succeeds, then $\mathcal{B}$ also succeeds. To see this, first observe that $y_\ell = y'_{\ell'}$ by assumption. If $y_{\ell-1} \| m_{\ell-1} \neq y'_{\ell'-1} \| m'_{\ell'-1}$, that implies that such an $i^\star, j^\star$ exist leading to a valid collision, and we are done. Otherwise, we have that $y_{\ell-1} = y'_{\ell'-1}$. Assume without loss of generality that $\ell' \geq \ell$ and let $\ell' = \ell + i$ for $i \geq 0$. Peeling back message blocks one at a time, $\mathcal{B}$ will either find a collision in h relative to key, or we will have two sequences of identical message blocks $(m_1, \dots, m_\ell)$ and $(m'_{i+1}, \dots, m'_{i+\ell})$. We proceed to show that if $\mathcal{B}$ does not find a collision, then $\mathcal{A}$ must have not actually succeeded in finding a collision.

First, consider the case where $\ell' = \ell$ so $i = 0$. By injectivity of pad of Theorem 3.3, this implies that $\mathsf{msg}_{\mathcal{A}} = \mathsf{msg}'_{\mathcal{A}}$, so $\mathcal{A}$ in fact did not succeed at finding a valid collision. Next, consider the case where $\ell' > \ell$ so $i > 0$. Theorem 3.3 guarantees that $\mathrm{pad}(\mathsf{msg}_{\mathcal{A}}, s, n)$ is not a suffix of $\mathrm{pad}(\mathsf{msg}'_{\mathcal{A}}, s, n)$, so one of the padded messages must be invalid.

Thus, whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ will also succeed, so it follows that

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMD}^h}(S, T) \leq \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S, T),$$

as required. ■

Next, we recall that if h is a keyed function modeled as a random oracle (in the AI-ROM of Unruh [Unr07]), Dodis, Guo, and Katz [DGK17] give the following bound on the success probability that any (S, T)-attacker can find a collision in h with respect to a random key.

Theorem 4.2 ([DGK17])**.** *Let $h \colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S, T) \leq \frac{2S + 2\kappa}{2^\kappa} + \frac{50T^2}{2^n}.$$

Combining Theorems 4.1 and 4.2, we get the following result.

Theorem 4.3. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a > \kappa + n$. Let $h \colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h \colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S, T) \leq \frac{2S + 2\kappa}{2^\kappa} + \frac{50T^2}{2^n}.$$

Construction $\mathsf{FMD}_b^h(\mathsf{key}, \mathsf{msg})$:

1. Parse msg into ℓ blocks $(m_1, m_2, \ldots, m_\ell)$ each of size s where $\ell = \lceil b/s \rceil$ and m_ℓ is padded with 0s if needed.
2. Initialize $y_0 = \mathsf{key}$, and compute $y_i \leftarrow h(y_{i-1}, m_i)$ for $i = 1, \ldots, \ell$.
3. Output y_ℓ.

Figure 9: The standard Merkle-Damgård construction with fixed input message length b $\mathsf{FMD}_b^h\colon \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ given any underlying function $h\colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$, where n is the key and output length, and s is the message block size.

2. *One evaluation of H^h on messages of length b requires $\lceil (b+n+1)/s \rceil$ queries to h, where $s = a - \kappa - n$.*

Proof. As $a > \kappa + n$, we define $s = a - \kappa - n > 0$. Then, we view h as a function from $\{0,1\}^\kappa \times \{0,1\}^{n+s}$ to $\{0,1\}^n$ and use it in the construction KMD^h of Figure 8 to get the hash function H^h required by the theorem.

The bound on the advantage immediately follows as a corollary of Theorems 4.1 and 4.2. As for efficiency, we note that padding a message $\mathsf{msg} \in \{0,1\}^b$ via Theorem 3.3 results in a message consisting of ℓ blocks each of length s, where $\ell = \lceil (b+n+1)/s \rceil$. Each block of the message requires a single invocation of h, so evaluating KMD^h on msg requires $\ell = \lceil (b+n+1)/s \rceil$ queries to h as required. ■

5 Instantiating the Inner Hash: Standard MD

We next consider instantiating the Merkle-Damgård framework of Section 4 whenever the underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ has input length a such that $n < a < \kappa + n + s$, where κ is the key length, n is the output length, and s is the desired message block size. Specifically, our goal is to use such a primitive $h\colon \{0,1\}^a \times \{0,1\}^n$ to build a larger (fixed-length) hash function $g^h\colon \{0,1\}^\kappa \times \{0,1\}^{n+s}$ that *can* be plugged into the construction KMD^{g^h}.

The first approach we consider is by simply building g from h using the standard Merkle-Damgård construction where h is not keyed in every invocation. We emphasize that in the standard version of MD, a random initialization vector/ key is still included in the first invocation of h. However, it is not included in the subsequent invocations of h, allowing h to take in smaller inputs overall.

Given an underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a > \max(\kappa, n)$, we define the standard Merkle-Damgård hash function with fixed input message length b $\mathsf{FMD}_b^h\colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ as follows. For sake of simplicity and due to the nature of the MD construction, we will assume that the key length κ is equal to the output length n. Let $s = a - n$ be the message block size we will include in each invocation of the underlying h. On input a key $\mathsf{key} \in \{0,1\}^n$ and a message $\mathsf{msg} \in \{0,1\}^b$, the function FMD_b^h splits the message msg into $\ell = \lceil b/s \rceil$ message blocks of size s (adding 0s to the last block if needed). It initializes $y_0 = \mathsf{key}$, and for $i = 1, \ldots, \ell$, computes y_i as the hash of y_{i-1} concatenated with m_i using h. The output of FMD_b^h is then y_ℓ. This is formalized in Figure 9.

If we instantiate $g^h := \mathsf{FMD}_b^h$ only on input messages of size at most $b = n + s$ as required to instantiate it inside KMD^{g^h}, then we only need to worry about ℓ-block collisions for $\ell = \lceil (n + s)/(a - n) \rceil$. Akshima, Guo, and Liu [AGL22] currently show the best known upper bound on the advantage of finding general ℓ-block collisions in MD^h, given by the following theorem.

Theorem 5.1 ([AGL22])**.** *Let $b, n, s \in \mathbb{N}$ such that $b > s$, and set $\ell = \lceil b/s \rceil$. Let $h\colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{FMD}_b^h}(S,T) \le \begin{cases} \frac{200n\cdot(ST+T^2)}{2^n} & \text{if } \ell = 2, \text{ and} \\ \frac{34n\cdot ST\ell}{2^n} \cdot \max\left(1, \frac{ST^2}{2^n}\right) + \frac{2\cdot T^2}{2^n} & \text{if } \ell > 2. \end{cases}$$

We note that Ghoshal and Komargodski [GK22] give a bound of $O(ST/2^n + T^2/2^n)$ whenever ℓ is a constant, which doesn't require the assumption that $ST^2 \le 2^n$. However, their bound does not extend to super constant ℓ.

Combined with Theorem 4.1, we get the following result.

Theorem 5.2. *Let $a, n, s \in \mathbb{N}$ be such that $a > n$. Let $\ell = \lceil (n+s)/(a-n) \rceil$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \le \begin{cases} \frac{200n\cdot(ST+T^2)}{2^n} & \text{if } \ell = 2, \text{ and} \\ \frac{34n\cdot ST\ell}{2^n} \cdot \max\left(1, \frac{ST^2}{2^n}\right) + \frac{2\cdot T^2}{2^n} & \text{if } \ell > 2. \end{cases}$$

2. *One evaluation of H^h on messages of length b requires $\ell \cdot \lceil (b+n+1)/s \rceil$ queries to h.*

Proof. As $a > n$, we parse $h\colon \{0,1\}^n \times \{0,1\}^{a-n} \to \{0,1\}^n$ and use it to construct FMD_{n+s}^h as defined in Figure 9. We then use FMD_{n+s}^h as the primitive underlying our MD-based hash function of Figure 8. So, we set

$$H^h := \mathsf{KMD}^{\mathsf{FMD}_{n+s}^h}.$$

The bound on the advantage follows as a corollary to Theorems 4.1 and 5.1, where the message length required for the FMD_{n+s}^h construction is only $n+s$. Furthermore, this implies that FMD_{n+s}^h requires $\ell = \lceil (n+s)/(a-n) \rceil$ invocations of h per invocation of FMD_{n+s}^h, and $\mathsf{KMD}^{\mathsf{FMD}_{n+s}^h}$ requires $\lceil (b+n+1)/s \rceil$ invocations of FMD_{n+s}^h, giving the resulting efficiency bound. ■

We emphasize that because our reduction in Theorem 4.1 is generic, any improvement on the bound of [AGL22, GK22] for finding an ℓ-block collision in MD will immediately imply an improved bound in Theorem 5.2.

We next state a corollary of Theorem 5.2 where we restrict to using $\ell = 2$ invocations of h in the underlying FMD_{n+s}^h construction. For this setting, the bound of [AGL22] above is optimal (up to poly(n) factors), and we observe that the resulting bound matches our desired bound of $O(S/2^n + T^2/2^n)$ whenever $S \le T$.

Corollary 5.3. *Let $a, n, s \in \mathbb{N}$ be such that $a \ge 3n/2 + s/2$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

```
Game $G^{\mathsf{mi\text{-}cr}}_{g,S}(\mathcal{B})$
    1. $h \leftarrow\!\!\$\ \mathsf{Fcs}(\{0,1\}^a, \{0,1\}^n)$
    2. $\mathsf{key}_1, \ldots, \mathsf{key}_S \leftarrow\!\!\$\ \{0,1\}^\kappa$
    3. $\mathsf{st} \leftarrow \perp$
    4. For $i = 1$ to $S$
        (a) $(\mathsf{msg}, \mathsf{msg}', \mathsf{st}) \leftarrow\!\!\$\ \mathcal{B}^h(\mathsf{st}, \mathsf{key}_i)$
        (b) Output false if:
            i. $\mathsf{msg} = \mathsf{msg}'$, or,
            ii. $g^h(\mathsf{key}_i, \mathsf{msg}) \neq g^h(\mathsf{key}_i, \mathsf{msg}')$
    5. Otherwise, output true
```

Figure 10: The multi-instance game $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$, where $\mathcal{B}$ is a uniform adversary with oracle access to the function h.

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leq \frac{200n \cdot (ST + T^2)}{2^n}$$

2. *One evaluation of H^h on messages of length b requires $2 \cdot \lceil (b + n + 1)/s \rceil$ queries to h.*

Proof. Restricting to $\ell = 2$ in Theorem 5.2, we require that $\ell = \lceil (n+s)/(a-n) \rceil \leq 2$. This holds as long as $a \geq 3n/2 + s/2$, as required. ∎

6 Instantiating the Inner Hash: Two-Level Merkle Tree

In this section, we instantiate the Merkle-Damgård framework of Section 4, that uses a keyed inner hash, in the setting where the underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ satisfies $a \geq \max(2n+2, \kappa + \lceil n/2 \rceil + 3)$.

The compression function in this instantiation of the MD-based framework, is a Merkle tree with two leaves, where we additionally input the key into each leaf. We describe next the framework introduced in [AGL22] that we use to analyze the collision-resistance of this construction in AI-ROM.

6.1 The [AGL22] Framework

In this section, we briefly introduce the framework given by [AGL22] which is useful in analyzing non-uniform security. An earlier version of this framework was introduced by [CGLQ20, ACDW20] inspired by techniques used in proving constructive Chernoff bounds in [IK10] and later refined by [ACDW20, GK22, AGL22] to upper bound $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T)$. This framework involves upper bounding the advantage of an (S,T)-attacker using the advantage of a uniform adversary for a multi-instance game that has to find collisions for S randomly chosen values of key.

We define the "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{g^h,S}(\mathcal{B})$ in Fig. 10. We refer an adversary playing $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$ and making at most T queries for each key as a (S,T)-MI adversary. For any (S,T)-MI adversary, define

$$\mathsf{Adv}^{\text{mi-cr}}_{g^h}(\mathcal{B}) = \Pr\left[\mathsf{G}^{\text{mi-cr}}_{g^h,S}(\mathcal{B}) = \mathsf{true}\right] .$$

Further,

$$\mathsf{Adv}^{\text{mi-cr}}_{g^h}(S,T) = \max_{\mathcal{B}} \mathsf{Adv}^{\text{mi-cr}}_{g^h}(\mathcal{B}) ,$$

where the maximum is taken over all (S,T)-MI adversaries. The following key lemma relates $\mathsf{Adv}^{\text{ai-cr}}_{g^h}(S,T)$ to $\mathsf{Adv}^{\text{mi-cr}}_{g^h}(S,T)$, which is proven in [AGL22].

Lemma 6.1 ([AGL22])**.** *Fix $S,T \in \mathbb{N}$ and $0 \le \delta \le 1$, if $\mathsf{Adv}^{\text{mi-cr}}_{g^h}(S,T) \le \delta^S$, then $\mathsf{Adv}^{\text{ai-cr}}_{g^h}(S,T) \le 2\delta$.*

Offline and Online queries. Since the adversary in the multi-instance game is stateful, we can assume without loss of generality that it does not repeat queries since they can simply remember the answers. Additionally, [AGL22] formalized the notion of "offline" and "online" queries during a particular instance of the game. When running the adversary on key_i, the queries that were made while the adversary was run on $\mathsf{key}_1, \ldots, \mathsf{key}_{i-1}$ are collectively known as the "offline" queries, and the queries made while running on key_i are "online queries".

6.2 Two-Level Merkle Tree

In this section, we present our construction of a keyed Merkle tree and analyze its collision resistance in the AI-ROM using the framework in the previous section. Specifically, given an underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \ge \max(2n+2, \kappa + \lceil n/2 \rceil + 3)$, we define a keyed, 2-level Merkle tree $\mathsf{2MT}^h_b$ for message length $b \le 2a - 2\kappa - 4$.

Before we define $\mathsf{2MT}^h_b$, we introduce notation that allows use to "domain-separate" h into three separate functions. Given a fixed hash function $h\colon \{0,1\}^a \to \{0,1\}^n$, we define three domain-separated functions $h_1, h_2, h_3\colon \{0,1\}^{a-2} \to \{0,1\}^n$, where $h_i(x)$ outputs $h(\hat{i}\|x)$ where $\hat{i} \in \{0,1\}^2$ is the 2-bit binary representation of i. Moreover, we refer to a query $h(\hat{i}\|*)$ as a query to the function h_i (which is also clearly a query to h).

To construct $\mathsf{2MT}^h_b\colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$, we use h_1 and h_2 above to process the two leaves of the depth-2 Merkle tree, where we include $\mathsf{key} \in \{0,1\}^\kappa$ in each leaf. We then feed those outputs as input to h_3 to get the output of $\mathsf{2MT}^h_b$. This construction is formalized in Figure 11.

Our main result of this section is the following theorem, which bounds the probability that any (S,T)-attacker finds a collision in $\mathsf{2MT}^h_b$.

Theorem 6.2. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \ge \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for $s = 2a - 2\kappa - n - 4$, the construction $\mathsf{2MT}^h_{n+s}\colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ of Figure 11 satisfies the following.*

- *For any $S,T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\text{ai-cr}}_{g^h}(S,T) \le \left(\frac{S}{2^\kappa} \cdot \left(14n + 42n\gamma + 42n\gamma^2\right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2}\right)\right) ,$$

  where $\gamma = ST^2/2^n$.

We prove this theorem using the framework described in Section 6.1. So, by Lemma 6.1, it suffices to prove the following lemma which bounds the advantage of an (S,T)-MI adversary.

```
$\underline{\mathsf{2MT}^h_b(\mathsf{key}, \mathsf{msg})}$
    1. Parse $\mathsf{msg} \in \{0,1\}^{2a-2\kappa-4}$ as $\mathsf{msg}_L \| \mathsf{msg}_R$ where $\mathsf{msg}_L, \mathsf{msg}_R \in \{0,1\}^{a-\kappa-2}$.
    2. Compute $y_L \leftarrow h_1(\mathsf{key} \| \mathsf{msg}_L)$.
    3. Compute $y_R \leftarrow h_2(\mathsf{key} \| \mathsf{msg}_R)$.
    4. Output $z \leftarrow h_3(y_L \| y_R)$.
```

Figure 11: The two-level, keyed Merkle tree construction $\mathsf{2MT}^h_b \colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ with fixed input message length b given any underlying function $h\colon \{0,1\}^a \to \{0,1\}^n$, where $a \geq \max(2n+2, \kappa+3)$ and $b \leq 2a - 2\kappa - 4$. h_1, h_2, h_3 are all domain-separated using the first two bits of h to encode $1, 2, 3$, respectively.

Lemma 6.3. *Let $S, T \in \mathbb{N}$. Then*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) \leq \left(\frac{S}{2^\kappa} \cdot \left(7n + 21n \cdot \gamma + 21n \cdot \gamma^2\right) + \frac{T^2}{2^n} \cdot \left(1 + \frac{\gamma}{T} + \frac{1}{T^2}\right)\right)^S ,$$

where $\gamma = ST^2/2^n$.

Proof. Following the techniques of [AGL22], we reduce the task of bounding $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T)$ to that of bounding any T-query adversaries advantage of succeeding in iteration i given that it has succeeded in all previous iterations. Fix any (S,T)-MI attacker $\mathcal{A}$. Let W_i be the indicator random variable that $\mathcal{A}$ wins on key_i in $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$. Define the random variable $W_{<i} := W_1 \wedge \ldots \wedge W_{i-1}$. We have that

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{A}) = \Pr\left[W_1 \wedge W_2 \wedge \ldots \wedge W_S\right] = \prod_{i=1}^{S} \Pr\left[W_i | W_{<i}\right] .$$

We will prove that for every $\mathcal{A}$ and each $i \in [S]$, $\Pr\left[W_{<i+1}\right] \leq (\delta_S)^i$ where

$$\delta_S = \frac{T^2}{2^n} + 7n \cdot \frac{S}{2^\kappa} + 21n \cdot \frac{S^2T^2}{2^{n+\kappa}} + \frac{ST^3}{2^{2n}} + 21n \cdot \frac{S^3T^4}{2^{2n+\kappa}} + \frac{1}{2^n}.$$

Note that δ_S is equal to the term in the lemma statement raised to the S power. Also, we observe that if $\Pr\left[W_{<i}\right] \leq (\delta_S)^i$, then we are done since $\Pr\left[W_{<i+1}\right] \leq \Pr\left[W_{<i}\right]$. Therefore, we assume from here on that $\Pr\left[W_{<i}\right] > (\delta_S)^i$.

To simplify the analysis of $\Pr[W_i|W_{<i}]$, we define the "knowledge-gaining event" $\mathsf{E}^i_{\mathsf{hit}}$ as follows. We say that an offline query to h (via h_1, h_2, or h_3) is a "hitting" query if its output is equal to the output or the left/ right inputs of a prior h query (after removing the first two bits if necessary). We define $\mathsf{E}^i_{\mathsf{hit}}$ to be the event that there are more than $7n \cdot \max(i, 3i^2T^2/2^n)$ hitting queries among the first $(i-1) \cdot T$ offline queries.

Conditioning on this event, we upper bound $\Pr[W_i|W_{<i}]$ for any $i \in [S]$ as follows.

$$\begin{aligned}\Pr\left[W_i|W_{<i}\right] &= \Pr\left[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}\right] \cdot \Pr\left[\neg\mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\ &\quad + \Pr\left[W_i|W_{<i} \wedge \mathsf{E}^i_{\mathsf{hit}}\right] \cdot \Pr\left[\mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\ &\leq \Pr\left[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}\right] + \Pr\left[\mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\ &\leq \Pr\left[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}\right] + \frac{\Pr\left[\mathsf{E}^i_{\mathsf{hit}}\right]}{\Pr\left[W_{<i}\right]} \\ &\leq \Pr\left[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}\right] + \Pr\left[\mathsf{E}^i_{\mathsf{hit}}\right] \cdot (2^{n\cdot i}),\end{aligned}$$

where the last line follows by the assumption that $\Pr[W_{<i}] > (\delta_S)^i \geq (1/2^n)^i$. Hence, it suffices to separately bound $\Pr[\mathsf{E}^i_{\mathsf{hit}}]$ and $\Pr[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}]$, which we do in Claims 6.4 and 6.5, respectively, below.

Specifically, we show in Claim 6.4 that $\Pr\left[\mathsf{E}^i_{\mathsf{hit}}\right] \leq (2^{-n\cdot 2i})$, so $\Pr\left[\mathsf{E}^i_{\mathsf{hit}}\right] \cdot (2^{n\cdot i}) \leq 1/2^n$. Then in Claim 6.5, we show that $\Pr[W_i|W_{<i} \wedge \neg\mathsf{E}^i_{\mathsf{hit}}] \leq \delta_S - 1/2^n$.

It follows that for any (S,T)-MI attacker $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{A}) = \Pr\left[W_{<S+1}\right] = \prod_{i=1}^{S} \Pr\left[W_i|W_{<i}\right] \leq (\delta_S)^S.$$

As this holds for any such $\mathcal{A}$, it follows that $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) \leq (\delta_S)^S$, as required by the lemma statement. It remains to prove the claims.

Claim 6.4. $\Pr\left[\mathsf{E}^i_{\mathsf{hit}}\right] \leq (2^n)^{-2i}$.

Proof. Let the random variable Z_j be an indicator random variable for the j-th query among the first $(i-1)\cdot T$ queries being a hitting query. Define the random variable $Z := \sum_{j=1}^{(i-1)\cdot T} Z_j$ to denote the total number of hitting queries. Thus, it suffices to show that $\Pr\left[Z \geq 7n \cdot \max(i, 3i^2T^2/2^n)\right] \leq 2^{-2n\cdot i}$.

First, note that output of each query can hit either the output of a previous query, or one of the left/right inputs. As each new output is lazily sampled and has to match n bits of some previous value, this implies that

$$\mathsf{E}\left[Z_j\right] = \Pr[Z_j = 1] \leq 3iT/2^n = 3iT/2^n.$$

Using linearity of expectation over at most iT possible offline queries, we have that

$$\mathsf{E}\left[Z\right] \leq 3i^2T^2/2^n.$$

We next give a general tail bound on Z for any upper bound $\mu_{\max}$ on $\mathsf{E}\left[Z\right]$. Let $\delta := 7n - 1$, which is at least 1 as $n \geq 1$. So, we can apply the Chernoff bound of Proposition 3.1 to get the following tail bound on Z.

$$\begin{aligned}\Pr\left[Z \geq 7n \cdot \mu_{\max}\right] &\leq \Pr\left[Z \geq (1+\delta)\cdot\mu_{\max}\right] \\ &\leq \exp\left(-\mu_{\max}\cdot\delta/3\right) \\ &\leq 2^{-\mu_{\max}\cdot(7n/3-1/3)\cdot\log e} \\ &\leq 2^{-2n\cdot\mu_{\max}}.\end{aligned}$$

We consider two cases: (a) $i \geq 3i^2T^2/2^n$, or (b) $i < 3i^2T^2/2^n$. In case (a), this implies that $\mathsf{E}\left[Z\right] \leq i = \mu_{\max}$, so by the above tail bound,

$$\Pr\left[Z \geq 7n \cdot i \mid i \geq 3i^2T^2/2^n\right] \leq 2^{-2n\cdot i}.$$

In case (b), we use $\mu_{\max} = 3i^2T^2/2^n$ and the fact that $\mu_{\max} \leq i$ in the tail bound above. Furthermore, $i < \mu_{\max}$, so it follows that

$$\Pr\left[Z \geq 7n \cdot (3i^2T^2/2^n) \mid i < 3i^2T^2/2^n\right] \leq 2^{-2n \cdot i}.$$

Combining the two cases, it holds that

$$\Pr\left[Z \geq 7n \cdot \max(i, 3i^2T^2/2^n)\right] \leq 2^{-2n \cdot i},$$

as required.

■

Claim 6.5. $\Pr\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] \leq \delta_S - 1/2^n$.

Proof. Suppose that the adversary $\mathcal{A}$ wins in game i on key key_i. We can assume without loss of generality that the adversary makes all the queries that are required to compute a collision. We define the queries $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ corresponding to a collision on key_i as follows:

1. $Q_1 = (1, (\mathsf{key}_i \| x_1), y_1)$, where $y_1 = h_1(\mathsf{key}_i \| x_1)$ was queried.
2. $Q_2 = (2, (\mathsf{key}_i \| x_2), y_2)$, where $y_2 = h_1(\mathsf{key}_i \| x_2)$ was queried.
3. $Q_3 = (3, (y_1 \| y_2), z)$, where $z = h_3(y_1 \| y_2)$ was queried.
4. $Q_1' = (1, (\mathsf{key}_i \| x_1'), y_1')$ where $y_1' = h_1(\mathsf{key}_i \| x_1')$ was queried.
5. $Q_2' = (2, (\mathsf{key}_i \| x_2'), y_2')$, where $y_2' = h_1(\mathsf{key}_i, x_2')$ was queried.
6. $Q_3' = (3, (y_1' \| y_2'), z)$, where $z = h_3(y_1' \| y_2')$ was queried.

If there are multiple such candidates of queries $(Q_1, Q_2, Q_3, Q_1', Q_2', Q_3')$, we fix the candidate that is the lexicographically smallest.

We categorize the possibilities for the queries of key_i into one of the following cases and sub-cases for the following analysis. Recall that an offline query is one made before game i started, and an online query is one made during game i.

1. The queries Q_3, Q_3' are identical queries. Assume without loss of generality that the queries Q_1, Q_1' are different (the other symmetric possibility is that the queries Q_2, Q_2' are different, since otherwise if Q_1, Q_1' are same, Q_2, Q_2' are same, Q_3, Q_3' are same then by definition they cannot lead to a collision). We further have the following sub-cases.
   (a) Q_1, Q_1' are both online queries.
   (b) One of Q_1, Q_1' is offline, and the other is online.
   (c) Q_1, Q_1' are both offline.
2. Q_3, Q_3' are different queries. We have the following sub-cases.
   (a) Q_3, Q_3' are both online queries.
   (b) Q_3 is offline, but Q_1, Q_2 are online.
   (c) Q_3 is offline, and one of Q_1, Q_2 is online, and the other is offline.
   (d) Q_1, Q_2, Q_3 are offline but Q_3' is online.

Case	Probability
1a	$\frac{T^2}{2^n}$
1b	$\frac{iT^2}{2^{n+\kappa}}$
1c	$7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}}$
2a	$\frac{T^2}{2^n}$
2b	$\frac{iT^3}{2^{2n}}$
2c	$7n \cdot \frac{i^2T^2}{2^{n+\kappa}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}$
2d	$7n \cdot \frac{i^2T^2}{2^{n+\kappa}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}$
2e	$7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}}$

Table 2: Upper bounds on probability of the cases

(e) $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all offline.

First, we argue that these cases are exhaustive. If the queries Q_3, Q'_3 are identical, we can assume without loss of generality that Q_1, Q'_1 are different queries that have the same answer. So the three possibilities are both are online (sub-case 1a), exactly one of them is online (sub-case 1b), and both are offline (sub-case 1c). When Q_3, Q'_3 are different, the possibility that they are both online is captured by sub-case 2a. If one of them is offline–say Q_3 without loss of generality– then either one of Q_1, Q_2 is online (captured by sub-case 2b), or one of Q_1, Q_2 is offline and the other online (captured by sub-case 2c), or they are both offline (this is captured by sub-case 2d since we know Q'_3 is online). Finally when Q_3, Q'_3 are both offline – if any of Q_1, Q_2, Q'_1, Q'_2 is online, this is captured by one of the sub-cases 2b and 2c. The final possibility is $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all old which is captured by sub-case 2e.

We will upper bound $\Pr[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}]$ by upper bounding the probability of each of these cases happening. We will prove the upper bound on the probability of the cases as shown in Table 2. As $\Pr\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right]$ is at most the maximum of the probabilities in Table 2, it follows that

$$\Pr\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] \leq \frac{T^2}{2^n} + 7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}} + \frac{iT^3}{2^{2n}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}.$$

We proceed to prove the upper bound on the probability of each case happening one by one.

Case 1a: ($Q_3 = Q'_3$, and Q_1, Q'_1 are both online queries) We can simply upper bound the probability of this case happening by the probability of two h_1 online queries producing a collision, which is at most $T^2/2^n$ by a birthday bound.

Case 1b: ($Q_3 = Q'_3$, and exactly one of Q_1, Q'_1 are online) Assume without loss of generality that Q_1 is the offline query and Q'_1 is online. Q_1 must correspond to some offline h_1 query that contains key_i as a prefix. If there are k such offline queries, then one of the online queries has to hit the same output as one of those k queries to be the Q'_1 query. This implies a bound of $k \cdot T/2^n$ probability. But, the expected number of offline h_1 queries that had key_i as a prefix is at most $(i-1) \cdot T/2^\kappa$.

Thus, this case happens with probability at most

$$\sum_{k=0}^{(i-1)\cdot T} \Pr[\text{there are } k \text{ offline } h_1 \text{ queries prefixed by } \mathsf{key}_i] \cdot k \cdot T/2^n$$
$$= \mathsf{E}\left[\text{number of offline } h_1 \text{ queries prefixed by } \mathsf{key}_i\right] \cdot T/2^n$$
$$\leq iT^2/2^{n+\kappa}.$$

Case 1c: ($Q_3 = Q'_3$, and both of Q_1, Q'_1 are offline) If this case happens, there are two different offline h_1 queries on $(\mathsf{key}\|*)$ such that they both produce the same answer, i.e., in particular there is hitting query of the form $h_1(\mathsf{key}\|*)$. If this happens for k different keys, there are at least k hitting queries. Since we know that there are at most $7n \cdot \max(i, 3i^2T^2/2^n)$ hitting queries, and key is sampled uniformly at random this case happens with probability at most $7n \cdot i/2^\kappa + 7n \cdot (3i^2T^2/2^n)/2^\kappa = 7n \cdot i/2^\kappa + 21n \cdot i^2T^2/2^{\kappa+n}$.

Case 2a: ($Q_3 \neq Q'_3$, and both of Q_3, Q'_3 are online) We can simply upper bound the probability of this case happening by the probability of two h_3 online queries producing a collision, which is at most $T^2/2^n$ by a birthday bound.

Case 2b: ($Q_3 \neq Q'_3$, Q_3 is offline, but Q_1, Q_2 are online) For this case to occur, the adversary needs to make distinct online queries Q_1, Q_2 whose outputs hit the input of some offline query Q_3. Since there are $(i-1)\cdot T$ offline queries, and T online queries, this happens with probability at most $(i-1)\cdot T \cdot T^2/2^{2n} \leq iT^3/2^{2n}$.

Case 2c: ($Q_3 \neq Q'_3$, Q_3 is offline, and exactly one of Q_1, Q_2 is online) Assume without loss of generality that Q_2, Q_3 are offline and Q_1 is online. We say that an offline h_3 query on $(x\|y)$ is "associated" with key_i if either there exists an offline query $h_1(\mathsf{key}_i\|*) = x$ or there exists an offline query $h_2(\mathsf{key}_i\|*) = y$. Given that the event $\mathsf{E}^i_{\mathsf{hit}}$ does not hold, there cannot exist a set of more than $k = 7n \cdot \max(i, 3i^2T^2n/2^n)$ distinct h_1 or h_2 queries that all have the same output, since otherwise there will be k hitting queries. Hence, an h_3 query can be associated with at most $7n\cdot\max(i, 3i^2T^2n/2^n)$ distinct key values. Given that there are at most $(i-1)\cdot T$ offline h_3 queries, it follows that there are at most $7n \cdot iT \cdot \max(i, 3i^2T^2n/2^n)$ different pairs of associated h_3 queries and key values. Therefore, for any randomly sampled key, the expected number of h_3 queries it is associated with it is at most $7n \cdot iT \cdot \max(i, 3i^2T^2n/2^n)/2^\kappa$.

For the specific key_i value sampled, if there are k h_3 queries associated with key_i, then there are at most k possible values of the output of some potential online Q_1 query. Hence, the probability of this case happening is at most

$$\sum_{k=0}^{(i-1)T} \Pr\left[\text{there are } k \text{ } h_3 \text{ queries associated with } \mathsf{key}_i\right] \cdot k \cdot (T/2^n)$$
$$= T/2^n \cdot \mathsf{E}\left[\text{number of } h_3 \text{ queries associated with } \mathsf{key}_i\right]$$
$$\leq 7n \cdot i^2T^2/2^{n+\kappa} + 21n \cdot i^3T^4/2^{2n+\kappa}$$

Case 2d: ($Q_3 \neq Q'_3$, Q_1, Q_2, Q_3 are offline but Q'_3 is online) Suppose for the value of key_i sampled there are k h_3 queries associated with key_i as above. Then, there are at most k possible values that some potential online Q'_3 query has to hit to cause this case to happen. Thus, the analysis follows identically to the previous case, giving a bound of $7n \cdot i^2T^2/2^{n+\kappa} + 21n \cdot i^3T^4/2^{2n+\kappa}$.

Case 2e: ($Q_3 \neq Q'_3$ and $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all offline) We refer to a collision structure for key as a set of offline queries of the form $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ corresponding to key. We claim that if there are k values of key with a corresponding collision structures queried, then there must be at least k different hitting queries.

First it is easy to see that for every key that gets classified into this case, at least one out of Q_3, Q_3' is a hitting query because they have the same answer. It remains to show that a set of k collision structures implies k distinct hitting queries.

To do so, we build a set of hitting queries as follows assuming there are k values of key with queried collision structures. For every such key value in lexicographic order, do the following.

- If one of Q_1, Q_2, Q_1', Q_2' is a hitting query, add it to this set,
- Otherwise, Q_3 or Q_3' must be a hitting query, so add it to the set.

We now claim this set has size at least k. Note that if a Q_1, Q_2, Q_1', Q_2' is added to the set, it is unique since it is an h_1 or h_2 query that contains key, so it increases the size of the set by one. Otherwise, a Q_3 or Q_3' query is added. Assume without loss of generality this Q_3 is added. Either this increases the size of the set by one, or Q_3 was already in the set. In the latter case, there must exist some previous key $\widetilde{\mathsf{key}}$ for which some h_3 query corresponding to Q_3 was added. We use $\tilde{Q}_1, \tilde{Q}_2, \tilde{Q}_3, \tilde{Q}_1', \tilde{Q}_2', \tilde{Q}_3'$ to denote the corresponding queries in the collision structure for $\widetilde{\mathsf{key}}$. By assumption $Q_3 = \tilde{Q}_3$ and neither of the two collision structures have a h_1 or h_2-type hitting query. But this is impossible since there must exist a pair of h_1 or h_2 queries, containing key and $\widetilde{\mathsf{key}}$, with output contained in the input of $Q_3 = \tilde{Q}_3$, so there must be a h_1 or h_2-type hitting query for either key or $\widetilde{\mathsf{key}}$. Therefore, for k collision structures, there will be at least k distinct hitting queries.

Given the above claim regarding collision structures, and assuming $\mathsf{E}^i_{\mathsf{hit}}$ doesn't hold, this implies there are at most $7n \cdot \max(i, 3i^2T^2/2^n)$ distinct key values with collision structures queried. But this implies that a randomly sampled key_i gets classified into this case is at most

$$7n \cdot \max(i, 3i^2T^2/2^n)/2^\kappa = 7n \cdot i/2^\kappa + 21n \cdot i^2T^2/2^{n+\kappa}.$$

■

This completes the proof of Lemma 6.3. ■

6.3 Variable-Input Length Hash from Two-Level Merkle Trees

We combine the construction of Section 6.2 with the framework of Section 4 to get a variable-input length hash function. This construction is optimally secure as long as $ST^2 \leq 2^n$ and requires an underlying function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geq \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$. This results in the following theorem.

We note if we modify values of κ, n by additive constant factors, we can get the same result as below with only $O(1)$ multiplicative loss in security. In this sense, we can achieve the theorem below assuming a function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$, i.e. $a = 2n$ so only compressing by a factor exactly two.

Theorem 6.6. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \geq \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leq \left(\frac{S}{2^\kappa} \cdot \left(14n + 42n\gamma + 42n\gamma^2\right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2}\right)\right),$$

where $\gamma = ST^2/2^n$.

2. *One evaluation of H^h on a message b bits long requires $3 \cdot \lceil (b+n+1)/s \rceil$ queries to h where $s = 2a - 2\kappa - n - 4$.*

Proof. We define $H := \mathsf{KMD}^{\mathsf{2MT}^h_{n+s}}$ where $\mathsf{2MT}^h_{n+s}$ is defined in Figure 11. From Theorem 4.1, we have that $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T)$ is upper bounded by $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{2MT}^h_{n+s}}(S,T)$. Therefore, the bound on the advantage of any (S,T)-attacker on H^h follows from Theorem 6.2.

We have that $\mathsf{2MT}^h_{n+s}\{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, for $s = 2a - 2\kappa - n - 4$. A b bit message, after padding, will result in $\lceil (b+n+1)/s \rceil$ message blocks that are fed into $\mathsf{2MT}^h_{n+s}$. For each call of $\mathsf{2MT}^h_{n+s}$, we need 3 calls to h, which implies the bound on the efficiency of H^h.

■

Acknowledgements

Ashrujit Ghoshal's work was partially supported by NSF grants CNS-2026774, CNS-2154174, a JP Morgan Faculty Award, a CISCO Faculty Award, and a gift from Microsoft. Part of Ashrujit Ghoshal's work was done during an internship at NTT Research. Cody Freitag is supported in part by the National Science Foundation Graduate Research Fellowship under Grant No. DGE–2139899 and DARPA Award HR00110C0086. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Defense Advanced Research Projects Agency (DARPA). Ilan Komargodski is the incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University, supported in part by an Alon Young Faculty Fellowship, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643).

References

[ACDW20] Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgård hash functions. In *Advances in Cryptology - CRYPTO*, pages 157–186, 2020. 3, 4, 7, 10, 11, 12, 17, 21

[AGL22] Akshima, Siyao Guo, and Qipeng Liu. Time-space lower bounds for finding collisions in Merkle-Damgård hash functions. CRYPTO, 2022. 2, 3, 4, 7, 10, 11, 12, 14, 17, 20, 21, 22, 23

[AHMP08] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. SHA-3 proposal BLAKE. *Submission to NIST*, 92, 2008. 3, 5

[AMPH14] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. 2014. 3, 5

[ASS$^+$16] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: breaking TLS using sslv2. In *USENIX*, pages 689–706, 2016. 7

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology - EUROCRYPT*, pages 181–197, 2008. 3, 10

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007. 3, 10

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993. 1

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *Advances in Cryptology - CRYPTO*, pages 693–721, 2018. 10

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *Advances in Cryptology - EUROCRYPT*, pages 227–258, 2018. 3, 4, 7, 9, 11, 12, 17

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO*, pages 430–448, 2005. 10

[CGLQ20] Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *FOCS*, pages 673–684, 2020. 7, 11, 12, 14, 21

[Dam89] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO*, pages 416–427, 1989. 1, 2, 9

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *Advances in Cryptology - EUROCRYPT*, pages 473–495, 2017. 2, 5, 10, 11, 18

[FGK22] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Time-space tradeoffs for sponge hashing: Attacks and limitations for short collisions. In *Advances in Cryptology - CRYPTO*, 2022. 3, 10

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. 1

[GB08] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography, 2008. 5, 11, 16, 17

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996. 1

[GK22] Ashrujit Ghoshal and Ilan Komargodski. On time-space tradeoffs for bounded-length collisions in Merkle-Damgård hashing. In *Advances in Cryptology - CRYPTO*, 2022. 3, 4, 7, 10, 11, 12, 17, 20, 21

[Har15] Nick Harvey. CPSC 536N: Randomized Algorithms Lecture Notes, University of British Columbia, 2015. 35

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. 1, 3, 9

[IK10] Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *APPROX*, pages 617–631. 2010. 12, 21

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO*, pages 369–378, 1987. 3

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO*, pages 218–238, 1989. 1, 2, 9

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, 2004. 10

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO*, pages 617–630, 2003. 3, 9

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *EUROCRYPT*, pages 487–506, 2011. 10

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *Advances in Cryptology - CRYPTO*, pages 205–223, 2007. 1, 10, 12, 15, 17, 18

[Yao90] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *STOC*, pages 84–94, 1990. 1

A Merkle Tree Framework with a Keyed Inner Hash

In this section, we give another framework similar to the MD-based framework in Section 5. This framework is a keyed version of the Merkle tree construction where the key is used in every invocation of the inner hash function, not just at the leaves.

For a key length κ, output length n, and message block size s (where $s \geq 2n$), we assume an underlying primitive $h\colon \{0,1\}^\kappa \times \{0,1\}^{s+1} \to \{0,1\}^n$. Given h, we define the keyed Merkle tree hash function $\mathsf{KMT}^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. On input key $\mathsf{key} \in \{0,1\}^\kappa$, and message msg. The function KMT^h first pads the message to split it into $\ell = \lceil (b+1)/s \rceil$ message blocks of size s using padding pad' which is defined as follows.
$\underline{\mathsf{pad}'(\mathsf{msg}, s)}$:

1. Let $k = s - ((|\mathsf{msg}| \bmod s) + 1)$.
2. Output $(m_1, \ldots, m_\ell) \in (\{0,1\}^s)^\ell$ where $m_1 \| \ldots \| m_\ell = \mathsf{msg} \| 1 \| 0^k$.

Similar to Theorem 3.3 we can prove the following theorem for pad'.

Theorem A.1. *Let $s \in \mathbb{N}$. The function $\mathsf{pad}'(\mathsf{msg}, s)$ on messages $\mathsf{msg} \in \{0,1\}^*$ satisfies the following properties:*

1. $|\mathsf{pad}'(\mathsf{msg}, s)| \in (\{0,1\}^s)^\ell$ *for* $\ell = \lceil (|\mathsf{msg}| + 1)/s \rceil$.
2. *There is a unique decoding procedure that outputs* msg *given* $\mathsf{pad}'(\mathsf{msg}, s)$*, and outputs* $\perp$ *on invalid padded messages.*

Note that the second point implies that pad' is injective. We omit the proof of this theorem because it follows from the proof of Theorem 3.3.

After padding the message msg to get message blocks $(m_1, \ldots, m_\ell)$, the construction essentially computes a standard Merkle tree with message blocks at the leaves, using the hash function $h(\mathsf{key}, \cdot)$ as the compression function. For technical reasons, we additionally use a single bit appended to key to indicate whether a block in the Merkle tree corresponds to a leaf (specified by a 0) or an interior node (specified by a 1). This is done to avoid trivial collisions that can be obtained by just outputting a subtree of a given tree (say the root and its two children). To deal with trees with ℓ leaves where ℓ is not a power of 2, we simply "push" a value up the levels of the tree until it needs to be merged. This is formalized in Figure 11. We illustrate an example of how the Merkle tree is built for $\ell = 7$ in Fig. 12. Since the key key is included in every call to the underlying primitive h, it follows that any (S,T)-attacker that finds a collision in KMT^h with respect to a key key also finds a collision in h with respect to key. This is formalized via a reduction, which gives the following theorem.

Theorem A.2. *Let $\kappa, n, s \in \mathbb{N}$ such that $s \geq 2n$. Let $h\colon \{0,1\}^\kappa \times \{0,1\}^{s+1} \to \{0,1\}^n$ be any function, and let $\mathsf{KMT}^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ as shown in Figure 13. Then, for every $S, T \in \mathbb{N}$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMT}^h}(S,T) \leq \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T).$$

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any (S,T)-attacker against the collision resistance of the construction KMT^h. Recall that we assume for simplicity that if $\mathcal{A}_2$ outputs a successful collision $\mathsf{msg}_{\mathcal{A}} \neq \mathsf{msg}'_{\mathcal{A}}$ such that $\mathsf{KMD}^h(\mathsf{key}, \mathsf{msg}_{\mathcal{A}}) = \mathsf{KMD}^h(\mathsf{key}, \mathsf{msg}'_{\mathcal{A}})$, then $\mathcal{A}_2$ queried all the necessary values to fully compute each of these functions. We construct an (S,T)-attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the collision resistance of h as follows.

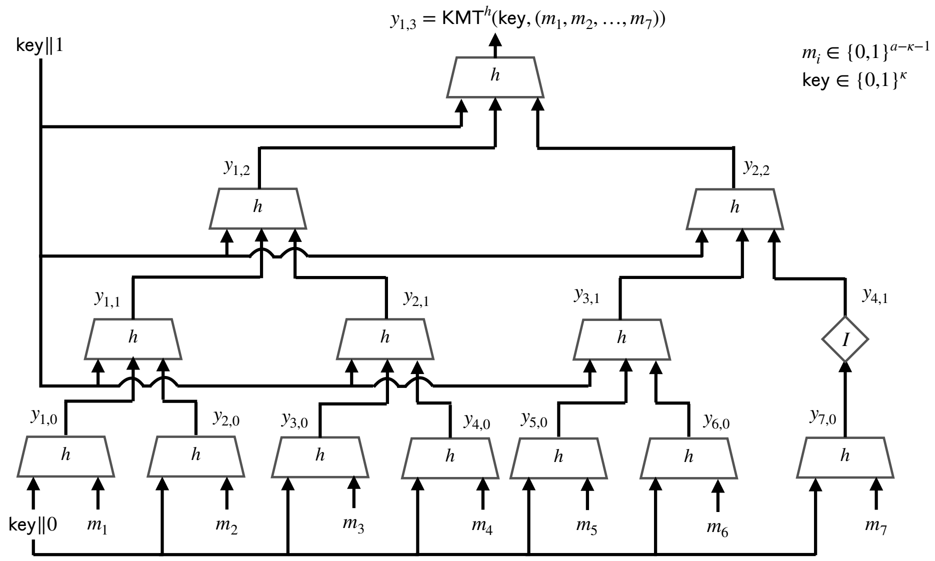

Figure 12: The construction KMT^h given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geq \max(\kappa + 2, 2n+1)$. Let $s = a - \kappa - 1$. The inputs to h are padded with appropriate number of zeros if the size of the input is less than $\kappa + s + 1$ bits. I is the identity function that is used to "push" a value up the tree until it needs to be merged.

In the preprocessing phase, $\mathcal{B}_1(h)$ simply computes $\sigma \leftarrow \mathcal{A}_1(h)$ and outputs σ where $|\sigma| \leq S$. In the online phase, $\mathcal{B}_2^h(\sigma, \mathsf{key})$ is given a random key $\mathsf{key} \leftarrow \{0,1\}^\kappa$ that it needs to find a collision for. $\mathcal{B}_2$ computes $(\mathsf{msg}_\mathcal{A}, \mathsf{msg}'_\mathcal{A}) \leftarrow \mathcal{A}_2^h(\sigma, \mathsf{key})$, where it uses its own query access to the function h to answer queries that $\mathcal{A}_2$ makes to h. $\mathcal{B}_2$ sets $(m_1, \ldots, m_\ell) \leftarrow \mathsf{pad}'(\mathsf{msg}_\mathcal{A}, s)$ and $(m'_1, \ldots, m'_{\ell'}) \leftarrow \mathsf{pad}'(\mathsf{msg}'_\mathcal{A}, s)$, where $\ell = \lceil (|\mathsf{msg}_\mathcal{A}| + 1)/s \rceil$ and $\ell' = \lceil (|\mathsf{msg}'_\mathcal{A}| + 1)/s \rceil$. Assume without loss of generality that $\ell' \geq \ell$.

$\mathcal{B}$ looks at the queries made by $\mathcal{A}_2$ and recovers the corresponding Merkle trees with leaves $(m_1, \ldots, m_\ell)$ and $(m'_1, \ldots, m'_{\ell'})$ ($\mathcal{A}_2$ must have made the queries which are required to compute these trees since $\mathcal{A}_2$ succeeds in outputting a collision by assumption). It looks for any collisions in h among these queries in the Merkle tree. If it finds one, it outputs the collision, otherwise aborts and outputs $\perp$.

It remains to analyze the complexity and success probability of $\mathcal{B}$. First, as $\mathcal{B}$ simply runs $\mathcal{A}$ in the preprocessing and online phases and makes no additional queries to h, it follows that if $\mathcal{A}$ is an (S,T)-attacker, then so is $\mathcal{B}$.

To analyze the success probability of $\mathcal{B}$, we claim that whenever $\mathcal{A}$ succeeds, then $\mathcal{B}$ also succeeds. To help in our analysis, we define a recursive procedure CollisionSearch (in Figure 14) that takes as input two different nodes in a Merkle tree with the same value (i.e., same inputs and output), treated as the roots, and outputs true if there is a collision and false otherwise.

First, we claim that CollisionSearch outputs true only if it finds a collision. This is because it is always called on two roots which have the same value and returns true only if the two children of the roots have different values (in which case it has found a collision because the roots have the same value), or if one root is a leaf and the other is not (in this case it has to be the collision because the leaves have input prefixed by $\mathsf{key}\|0$ while other nodes have inputs prefixed by $\mathsf{key}\|1$).

As a result, it remains to argue that whenever CollisionSearch returns false, $\mathcal{A}$ fails to find a collision. If $\ell = \ell'$, then the two Merkle trees will have the same structure, so CollisionSearch will return false only if all the leaves of the two trees are same. This means that $(m_1, \ldots, m_\ell) =$

```
Construction $\mathsf{KMT}^h(\mathsf{key}, \mathsf{msg})$:

1. $(m_1, m_2, \ldots, m_\ell) \leftarrow \mathsf{pad}'(\mathsf{msg}, s)$ where $\ell = \lceil (|\mathsf{msg}| + 1)/s \rceil$.
2. Initialize $y_{i,0} \leftarrow h(\mathsf{key} \| 0, m_i)$ for $i = 1, \ldots, \ell$.
3. $\ell' \leftarrow \ell$.
4. For $j = 1$ to $\lceil \log \ell \rceil$.
    (a) Compute $y_{i,j} = h(\mathsf{key} \| 1, y_{2i-1,j-1} \| y_{2i,j-1} \| 0^{s-2n})$ for $i = 1, \ldots, \lfloor \ell'/2 \rfloor$.
    (b) If $\ell' \bmod 2 = 1$ then $y_{(\ell'+1)/2,j} = y_{\ell',j-1}$, $\ell' \leftarrow (\ell'+1)/2$.
    (c) Else $\ell' \leftarrow \ell'/2$.
5. Output $y_{1,\lceil \log \ell \rceil}$.
```

Figure 13: The Merkle tree construction with a keyed inner hash.

$(m'_1, \ldots, m'_\ell)$, so $\mathsf{msg} = \mathsf{msg}'$ by injectivity of pad'. Therefore, $\mathcal{A}$ did not find a valid collision, as required. If one of ℓ or ℓ' is 1 and $\mathsf{CollisionSearch}$ returns false, then it must be that $\ell = \ell' = 1$, meaning $\mathcal{A}$ has failed to find a collision as above. Finally, consider the case where $\ell' > \ell \geq 2$. Let ℓ_L, and ℓ_R be the number of leaves in the left and right subtrees of the tree with ℓ leaves, i.e., $\ell_L + \ell_R = \ell$. Similarly define ℓ'_L, ℓ'_R so that $\ell'_L + \ell'_R = \ell'$. Then it must be the case that either $\ell_L \neq \ell'_L$ or $\ell_R \neq \ell'_R$ at every step. It follows inductively that at every step of the recursion, either the left or the right subtree will have a different number of leaves, so eventually, $\mathsf{CollisionSearch}$ will arrive at a case when one subtree has one leaf and the other has more than one, returning true, in contradiction.

Thus, whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ will also succeed. Therefore, it follows that

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMT}^h}(S, T) \leq \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S, T),$$

as required. ■

We combine the construction of Section 6.2 with the framework presented above to get a variable-input length hash function. This construction is optimally secure as long as $ST^2 \leq 2^n$ and requires an underlying function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geq \max(\kappa + \lceil n/2 \rceil + 3, 2n + 2, \kappa + 3\lceil n/2 \rceil - 1)$. This results in the following theorem.

We note if we modify values of κ, n by additive constant factors, we can get the same result as below with only $O(1)$ multiplicative loss in security. In this sense, we can achieve the theorem below assuming a function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$, i.e. $a = 2n$ so only compressing by a factor exactly two.

Theorem A.3. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \geq \max(\kappa + 3\lceil n/2 \rceil + 2, 2n + 2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$, letting $\gamma = ST^2/2^n$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S, T) \leq \left(\frac{S}{2^\kappa} \left(14n + 42n\gamma + 42n\gamma^2 \right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2} \right) \right).$$

2. *One evaluation of H^h on a message b bits long requires at most $6 \cdot \lceil (b+1)/s \rceil - 3$ queries to h where $s = 2a - 2\kappa - n - 4$.*

```
Procedure CollisionSearch(root_1, root_2):
    1. Let the number of leaves of the tree rooted at root_1 be $\ell$, and the number of leaves of the tree rooted at root_2 be $\ell'$.
    2. If $\ell = \ell' = 1$, then return false.
    3. If $\ell = 1, \ell' > 1$ or $\ell' = 1, \ell > 1$, then return true.
    4. Find the left and the right child of root_1, (going down the $I$ nodes if required), call these root_{1,L} and root_{1,R}. Find the left and the right child of root_2, (going down the $I$ nodes if required), call these root_{2,L} and root_{2,R}.
    5. If the value at root_{1,L} and root_{2,L} are different or if the value at root_{1,R} and root_{2,R} are different return true.
    6. Make a recursive call to CollisionSearch(root_{1,L}, root_{2,L}), and return true if it returns true.
    7. Make a recursive call to CollisionSearch(root_{1,R}, root_{2,R}), and return true if it returns true.
    8. Otherwise return false.
```

Figure 14: Recursive collision search procedure used in the proof of Theorem A.2

Proof. Let $s = 2a - 2\kappa - n - 4$. We define $H := \mathsf{KMT}^{\mathsf{2MT}^h_{n+s}}$ where $\mathsf{2MT}^h_{n+s}$ is defined in Figure 11. Since we have that $a \geq \kappa + 3\lceil n/2 \rceil + 2$, it implies $s = 2a - 2\kappa - n - 4 \geq 2n$. Therefore, from Theorem A.3, we have that $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T)$ is upper bounded by $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{2MT}^h_{n+s}}(S,T)$. Therefore, the bound on the advantage of any (S,T)-attacker on H^h follows from Theorem 6.2 since $a \geq \max(\kappa + 3\lceil n/2 \rceil + 2, 2n+2) \geq \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$.

We have that $\mathsf{2MT}^h_{n+s}\{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, for $s = 2a - 2\kappa - n - 4$. A b bit message, after padding, will result in $\lceil (b+1)/s \rceil$ message blocks that are fed into $\mathsf{2MT}^h_{n+s}$. To build a Merkle tree on $\lceil (b+1)/s \rceil$ blocks we need $2\lceil (b+1)/s \rceil$ calls to $\mathsf{2MT}^h_{n+s}$, and for each call of $\mathsf{2MT}^h_{n+s}$, we need 3 calls to h, which implies the bound on the efficiency of H^h.

■

B Proof of Proposition 3.1

For the sake of completeness we include a proof from [Har15] of the Chernoff bound in Proposition 3.1.

Proof. [Proposition 3.1]

We state the two following claims that we use in our proof.

Claim B.1. *For all $t \geq 0, 0 \leq x \leq 1$*

$$e^{tx} \leq 1 + (e^t - 1)x$$

Claim B.2. *Let $\delta \geq 1$. We have that*

$$\frac{e^\delta}{(1+\delta)^{1+\delta}} \leq e^{-\delta/3}\ .$$

Let $t = \ln(1+\delta)$, where ln denotes the natural logarithm. We have that

$$\begin{aligned}\Pr\left[X \geq (1+\delta)\mu'\right] &\leq \Pr\left[e^{tX} \geq e^{t(1+\delta)\mu'}\right] \\ &\leq \frac{\mathsf{E}\left[e^{tX}\right]}{e^{t(1+\delta)\mu'}} = \frac{\mathsf{E}\left[\prod_{i=1}^{n} e^{tX_i}\right]}{e^{t(1+\delta)\mu'}} = \frac{\prod_{i=1}^{n} \mathsf{E}\left[e^{tX_i}\right]}{e^{t(1+\delta)\mu'}} \ .\end{aligned}$$

The first inequality above follows since e^x is an increasing function and $t \geq 0$. The second inequality follows from Markov's inequality. The last step follows since X_is are independent. Notice that using the fact that $0 \leq \mathsf{E}[X_i] \leq 1$ and Claim B.1 we have that

$$\begin{aligned}\mathsf{E}\left[e^{tX_i}\right] &\leq \mathsf{E}\left[1 + (e^t - 1)X_i\right] \\ &= 1 + (e^t-1)\mathsf{E}[X_i] \leq e^{(e^t-1)\mathsf{E}[X_i]} \ .\end{aligned}$$

The second step above follows from linearity of expectation, and the last step follows since $1+x \leq e^x$. Plugging in this upper bound for $\mathsf{E}\left[e^{tX_i}\right]$, we get that

$$\begin{aligned}\Pr\left[X \geq (1+\delta)\mu'\right] &\leq \frac{\prod_{i=1}^{n} e^{(e^t-1)\mathsf{E}[X_i]}}{e^{t(1+\delta)\mu'}} = e^{(e^t-1)\sum_{i=1}^{n}\mathsf{E}[X_i] - t(1+\delta)\mu'} \\ &\leq e^{(e^t-1)\mu' - t(1+\delta)\mu'} = e^{\delta\mu' - (1+\delta)\ln(1+\delta)\mu'} = \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu'}\end{aligned}$$

Above, we plugged in the value of t. Finally, using Claim B.2 we have that

$$\Pr\left[X \geq (1+\delta)\mu'\right] \leq \left(e^{-\delta/3}\right)^{\mu'} = e^{-\frac{\delta\mu'}{3}} \ .$$

■

We finally prove the two claims that we used above.

Proof. [Claim B.1] Let $f(x) = 1 + (e^t-1)x - e^{tx}$. We have that $f(0) = 0 = f(1)$. The derivative of $f(x)$ with respect to x is $f'(x) = e^t - 1 - te^{tx}$. It is clear that the derivative of $f(x)$ changes sign only at one point between 0 and 1, and it is positive at $x = 0$ since $e^t \geq 1 + t$. Because $f(x)$ is continous and differentiable, its derivative is positive at $x = 0$, its derivative changes sign only once between 0 and 1, and $f(0) = f(1) = 0$, we have that $f(x) \geq 0$ for all $0 \leq x \leq 1$. Therefore for all $0 \leq x \leq 1$,

$$e^{tx} \leq 1 + (e^t - 1)x$$

■

Proof. [Claim B.2] Consider the functions $f(x) = (1+x)\ln(1+x) - x$ (where ln denotes the natural logarithm), $g(x) = x/3$. We have that $f(1) = 2\ln 2 - 1 > 1/3 = g(1)$. The derivative of $f(x)$ with respect to x is $f'(x) = \ln(1+x)$, which is at least $\ln 2 > 1/3$ for $x \geq 1$ because ln is an increasing function. The derivative of $g(x)$ with respect to x is $g'(x) = 1/3$.

Therefore $f(x) \geq g(x)$ for all $x \geq 1$. Since e^x is an increasing function and $\delta \geq 1$, we have that

$$e^{-f(\delta)} \leq e^{-g(\delta)} \ .$$

It follows that

$$\frac{e^\delta}{(1+\delta)^{1+\delta}} \leq e^{-\delta/3} \ .$$

■