On Time-Space Tradeoffs for Bounded-Length Collisions in Merkle-Damgård Hashing

Ashrujit Ghoshal[1] and Ilan Komargodski[2]

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

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

Abstract. We study the power of preprocessing adversaries in finding bounded-length collisions in the widely used Merkle-Damgård (MD) hashing in the random oracle model. Specifically, we consider adversaries with arbitrary S-bit advice about the random oracle and can make at most T queries to it. Our goal is to characterize the advantage of such adversaries in finding a B-block collision in an MD hash function constructed using the random oracle with range size N as the compression function (given a random salt).

The answer to this question is completely understood for very large values of B (essentially $\Omega(T)$) as well as for $B = 1, 2$. For $B \approx T$, Coretti et al. (EUROCRYPT '18) gave matching upper and lower bounds of $\tilde{\Theta}(ST^2/N)$. Akshima et al. (CRYPTO '20) observed that the attack of Coretti et al. could be adapted to work for any value of $B > 1$, giving an attack with advantage $\tilde{\Omega}(STB/N + T^2/N)$. Unfortunately, they could only prove that this attack is optimal for $B = 2$. Their proof involves a compression argument with exhaustive case analysis and, as they claim, a naive attempt to generalize their bound to larger values of B (even for $B = 3$) would lead to an explosion in the number of cases needed to be analyzed, making it unmanageable. With the lack of a more general upper bound, they formulated the *STB conjecture*, stating that the best-possible advantage is $\tilde{O}(STB/N + T^2/N)$ for any $B > 1$.

In this work, we confirm the STB conjecture in many new parameter settings. For instance, in one result, we show that the conjecture holds for all constant values of B. Further, using combinatorial properties of graphs, we are able to confirm the conjecture even for super constant values of B, as long as some restriction is made on S. For instance, we confirm the conjecture for all $B \leqslant T^{1/4}$ as long as $S \leqslant T^{1/8}$. Technically, we develop structural characterizations for bounded-length collisions in MD hashing that allow us to give a compression argument in which the number of cases needed to be handled does not explode.

1 Introduction

Starting from the seminal work of Hellman [21], there have been significant efforts to understand the power of preprocessing attacks in various applications and constructions (e.g., [31,16,28,5,30,13,15,1,9,8,3,11,6]). Preprocessing attacks, i.e., ones that utilize a bounded amount of auxiliary information, capture the standard modeling of attackers as *non-uniform*, allowing them to obtain some arbitrary (but bounded length) "advice" before attacking the system. In this work, we continue the recent line of works studying the power of preprocessing adversaries in the context of finding collisions in the widely used Merkle-Damgård (MD) design.

Collision resistance of salted MD. The Merkle-Damgård hash function construction [24,25,26,12] is a popular design for building an arbitrary-size-input compression function from a fixed-size-input compression function. 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 MD construction is defined relative to a compressing function $h\colon [N] \times [M] \to [N]$[3], modeled as a random oracle, as follows. First, for $a \in [N]$ and $\alpha \in [M]$, let $\mathsf{MD}_h(a, \alpha) = h(a, \alpha)$. Then, define recursively

$$\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_B)) = h(\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_{B-1})), \alpha_B)$$

for $a \in [N]$ and $\alpha_1, \ldots, \alpha_B \in [M]$. The a is referred to as *salt* (sometimes also called IV) and each of the following B elements are referred to as *blocks*.

Due to the ubiquitous influence of this hashing paradigm, both in theory and practice, characterizing the complexity of finding collisions in MD_h (on a random salt) is a fundamental problem. The well-known birthday attack gives a T-query attacker with $\Theta(T^2/N)$ advantage. However, this attack is very generic: it neither takes advantage of the structure of MD_h nor does it utilize the fact that the attacker may have access to some limited amount of "advice" about h due to a long preprocessing phase. But, there is a good reason that security against non-uniform attackers has become the standard notion of security in the cryptographic literature: it captures the natural idea that an adversary may have been designed to attack specific instances, guaranteeing security against an expensive preprocessing stage, or even unknown future attacks. On the whole, it is widely believed by the theoretical community that non-uniformity is the right cryptographic modeling of attackers, despite being overly conservative and including potentially unrealistic attackers. Therefore, understanding the complexity of finding collisions in MD, allowing preprocessing, is a fundamental problem.

The auxiliary-input random oracle model. The concrete hash functions h used in real-life do not have solid theoretical foundations from the perspective of provable security. Therefore, when analyzing the security of the MD construction, the function h is typically modeled as a completely random one, i.e., a random oracle. We follow the standard approach and model preprocessing adversaries using the influential extension of the random oracle model termed *auxiliary-input random oracle model* (AI-ROM). This model was (implicitly) used, for example, in the classical works of Yao [31] and Fiat and Naor [16],

[3] We use the notation $[N]$ to denote the set $\{1, 2, \ldots, N\}$ for a natural number N.

and formally defined in the influential work of Unruh [30] which was recently revisited by Dodis, Guo, and Katz [15] and Coretti et al. [9].

The AI-ROM models preprocessing adversaries as two-stage algorithms ($\mathcal{A}_1$, $\mathcal{A}_2$) parametrized by S (for "space") and T (for "time"). The first part $\mathcal{A}_1$ has unbounded access to the random oracle h, and its goal is to compute an S-bit "advice" σ for $\mathcal{A}_2$. The second part $\mathcal{A}_2$ gets the advice σ, can make at most T queries to the random oracle, and attempts to accomplish some task involving h. In our case, $\mathcal{A}_2$ gets a random salt a as a challenge and its goal is to come up with a collision in $\mathsf{MD}_h(a, \cdot)$. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ are unbounded in running time.

Known results. Collision resistance of salted MD hash functions in the AI-ROM was first studied by Coretti, Dodis, Guo, and Steinberger [9]. Among other results, they showed an attack, loosely based on the idea of rainbow tables [21,28], with advantage $\tilde{\Omega}(ST^2/N)$.[4] [5] They further showed that this attack is optimal. Namely, no attack can have an advantage better than $\tilde{O}(ST^2/N)$. (Notice that this attack beats the naive birthday attack mentioned above for typical values of S.) In a more recent work of Akshima, Cash, Drucker, and Wee [3] observed that the attack of Coretti et al. [9] results in very long collisions, of the order of T blocks, which may limit their practical usefulness. While formally, a length T collision does violate collision resistance, it is hard to imagine a natural application where it is useful. Indeed, for reasonable values of T, say $T = 2^{60}$, it is unlikely that such a collision, which is several petabytes long, could damage any widely-used application.

Akshima et al. [3] therefore raise the very natural question of what is the complexity of finding *short* collisions.

What is the complexity, as a function of S and T (the allowed space and query bounds, respectively), of finding a B-block collision in salted MD?

Although this question is very natural and clean, as mentioned, a complete answer is known only in the extreme cases, either when $B = 2$ or when $B = \tilde{\Omega}(T)$. Indeed, when B is very close to T, the result above of Coretti et al. [9] implies that the advantage is $\tilde{\Theta}(ST^2/N)$. The case of $B = 2$ was resolved by Akshima et al. [3] who showed that the advantage is $\tilde{\Theta}(ST/N + T^2/N)$. Even for $B = 3$, a complete answer is not known: The analysis of Akshima et al. [3] consists of an elaborate case analysis tailored to the $B = 2$ case, and they claim that even for $B = 3$ the proof of their lower bound "*... would be too long and complex to write down*".[6]

The STB conjecture. In terms of upper bounds, Akshima et al. [3] showed that a variant of Coretti et al.'s (rainbow tables inspired) attack could be generalized to get a B-block collision with advantage $\tilde{\Omega}(STB/N)$. This attack generalized the attack of Coretti et al. [9] which gives an $O(T)$-block collision with probability $\tilde{\Omega}(ST^2/N)$. With the lack of better bounds on the best possible attack for a

[4] Throughout the paper, the $\tilde{}$ notation suppresses poly-logarithmic terms in N.

[5] By "advantage" we mean the probability of finding a collision.

[6] For $B = 1$ a tight bound of $\Theta(S/N + T^2/N)$ is known [15].

wide range of B's (anywhere between $B = 3$ and $B \ll T$), Akshima et al. [3] put forward the "*STB conjecture*" which posits that the optimal attack for finding length B collisions has advantage $\tilde{\Theta}(STB/N + T^2/N)$ (i.e., the better between their attack and the generic birthday attack).

We believe that our current understanding of the exact security that MD-style constructions could ideally achieve is insufficient. Therefore, given how widespread MD-based hash functions are, progress towards resolving the conjecture is highly important.

1.1 Our Results

Our main result confirms the STB conjecture in many new parameter settings. Specifically, we prove two new upper bounds on the advantage of the best attack for finding short collisions in salted MD hash functions in the AI-ROM. The first bound confirms the STB conjecture for all constant values of B. The second result confirms the STB conjecture even for super constant values of B but only for moderately large values of S compared to T.

STB conjecture is true for all constant B. We show that for any $B \in O(1)$, the advantage of any S-space T-query attacker in finding a length B collision is bounded by $\tilde{O}(ST/N + T^2/N)$, matching the known attack up to poly-logarithmic factors.

Theorem 1.1 (Informal; See Theorem 5.1). *For every constant B, the STB conjecture is true.*

This theorem is obtained as a special case of a more general bound on the advantage of any S-space T-query attacker in finding a length B collision of the form

$$\tilde{O}\left(\frac{STB^2(\log^2 S)^{B-2}}{N} + \frac{T^2}{N}\right).$$

Note that this bound is meaningful when B is a constant (or slightly bigger) but becomes vacuous when say, $B = \log N$.

STB conjecture is true for all $SB \ll T$. We show that as long as $S, B \ll T$, the conjecture is true again. Specifically, we show that whenever $S^4B^2 \in \tilde{O}(T)$, the maximal advantage of any S-space T-query attacker in finding a length B collision is obtained by the birthday attack, up to poly-logarithmic factors. For example, when $SB \leqslant T^{1/4}$, the maximal advantage is $O(T^2/N)$, and therefore the STB conjecture holds.

Theorem 1.2 (Informal; See Theorem 6.1). *For every $S^4B^2 \in \tilde{O}(T)$, the STB conjecture is true. For instance, the conjecture holds if either*

- $B \in \mathsf{poly}\log N$ *and* $S \in \tilde{O}(T^{1/4})$, *or*
- $B \in \tilde{O}(T^{1/4})$ *and* $S \in \tilde{O}(T^{1/8})$.

This theorem is obtained as a special case of a more general bound on the advantage of any S-space T-query attacker in finding a length B collision of the form

$$\tilde{O}\left(\frac{S^4TB^2}{N} + \frac{T^2}{N}\right).$$

A concrete comparison between the results. The two bounds are generally incomparable. While the bound from Theorem 1.1 is asymptotically tight whenever B is constant (independent of S, T), it becomes vacuous for say $B = \log N$. On the other hand, the bound from Theorem 1.2 is meaningful for all $B \in o(N^{1/2})$, as long as $S^4 \cdot B^2 \ll N$. For instance, assume that $S = N^{1/16}$ and $B \in \Theta(N^{\epsilon})$ (for $0 < \epsilon < 1/8$). In this setting, the bound from Theorem 1.1 is trivial. On the other hand, the bound from Theorem 1.2 gives that any successful attack must satisfy $T \in \tilde{\Omega}(N^{1/2})$ which is strictly better than what the generic $\tilde{O}(ST^2/N)$ bound gives (it only gives $T \in \tilde{\Omega}(N^{15/32})$).

Technical highlight. The main technical component in both of our bounds is a compression argument that uses a "too-good-to-be-true" attacker to non-trivially compress a uniformly random sequence of bits, thereby getting a contradiction. The setup is somewhat similar to the one of Akshima et al. [3] (although slightly more modular), but our compression argument deviates from theirs significantly. Their argument inherently relied on the fact that there are at most two blocks in the collision, therefore greatly simplifying the possible structures to consider. In contrast, we consider arbitrary length collisions, and thus we have to deal with all possible structures of collisions. Our proof identifies and analyzes a general structure for MD collisions and unveils a natural combinatorial problem that influences the resulting upper bound on the advantage of preprocessing adversaries. Specifically, it turns out that the "dominant extra" terms in both of our bounds ($(\log^2 S)^{B-2}$ in the first bound and S^3 in the second) emerge due to the need to encode a *reverse path* in a general (fan-out 1, but possibly large fan-in) directed graph, where the graph is the one induced by the queries that the adversary makes to the random oracle. Any improvement on this encoding would immediately imply a better upper bound, a fact that we hope will lead to better bounds in the future.

1.2 Discussion

As mentioned, the MD paradigm underlies numerous hash function constructions that are central building blocks in many applications. There are several popular variants of the MD paradigm implemented in practice. In this work, we follow previous works and focus on the cleanest variant for concreteness. One prominent variant withstands length extension attacks by padding the input message with its length. (In fact, this is the version suggested by Merkle and Damgård.) We remark that our results directly apply to this padded variant. Specifically, the "STB" attack finds a collision with the same number of blocks, so it readily extends to this padded variant. Our bounds on the best possible attacks also extend to this setting since the argument did not use any specific property on the collision blocks. It is interesting to study other practically used variants and understand if similar results can be obtained. To this end, we hope that the techniques we develop in this work will be helpful.

From a theoretical perspective, no single function can be collision-resistant (in the plain model), as a non-uniform attacker can trivially hardwire a collision. This is why collision resistance is considered with respect to a family of hash functions indexed by a key called *salt*. The salt is chosen after the attacker is fixed (and so is the non-uniform advice about the family of functions). Still, in practice, a single hash function is typically defined by fixing an IV, making it insecure against non-uniform attackers. This contrasts with how we define the collision resistance game, where the IV is chosen randomly after the preprocessing phase. Thus, it may seem that the expensive preprocessing needed for attacks in our model does not represent real-life scenarios. However, often the hash function used in a particular application (relying on collision-resistance) is *salted* by prepending a random salt value to the input. One such well-known application is *password hashing* [29]. Such salting essentially corresponds to the random-IV setting considered here, and, therefore, the attack becomes relevant again.

The primary motivation for our work is to make progress towards the STB conjecture, which we view as a fundamental problem. To this end, we focused on asymptotic bounds as a function of S, T, and B. We hope that the concrete bounds could be improved in future works, affecting design choices of real-life hash functions.

Follow up work. A recent work of Akshima, Guo, and Liu [4] proved a new bound on the maximal possible advantage in finding B-block collisions. Their bound is overall incomparable to ours: it is better than our Theorem 1.2 but worse than Theorem 1.1. Also, Freitag et al. [17] studied related problems in the context of sponge hashing, an alternative to the Merkle-Damgård paradigm that underlies (for instance) the SHA-3 standard.

2 Our Techniques

In this section, we provide a high-level overview of our techniques. Both of our results follow a similar high-level rationale, and thus throughout this overview, we mainly focus on the techniques for proving the STB conjecture whenever $B \in O(1)$ (Theorem 1.1). Towards the end, we describe the additional ideas needed to obtain the result for $SB \ll T$ (Theorem 1.2).

Before explaining the ideas, let us describe the challenge more precisely. We are given a compressing function $h : [N] \times [M] \to [N]$, modeled as a random oracle, and we want to upper bound the probability of a non-uniform attacker in finding a collision in an MD_h instance with a random salt. We model non-uniform attackers by thinking of them as two-stage adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The offline part $\mathcal{A}_1$ is unbounded in running time, and its only restriction is that it can output only S bits. This output is the non-uniform advice given to the online part $\mathcal{A}_2$ which is then allowed to make up to T queries after which it must output a B-block collision for MD_h and terminate. We assume unbounded running time for both parts $\mathcal{A}_1$ and $\mathcal{A}_2$ and only restrict the output-size for $\mathcal{A}_1$ and the number of queries for $\mathcal{A}_2$. We refer to such a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$ as a (S,T)-adversary. In the context of length-B collisions in MD_h, the game is as follows:

- $\mathcal{A}_1$ has unbounded access to h and it outputs $\sigma \in \{0,1\}^S$.
- $\mathcal{A}_2$ gets σ as input along with a random salt $a \in [N]$.
- $\mathcal{A}_2$ outputs α, α'.
- $\mathcal{A}$ wins if $\alpha \neq \alpha'$, α, α' consist of $\leqslant B$ blocks, and $\mathsf{MD}_h(a,\alpha) = \mathsf{MD}_h(a,\alpha')$.

There are essentially two main generic approaches known in the literature for proving bounds of this sort. The first is the so-called *pre-sampling* technique, originally due to Unruh [30], and the second is a compression argument. The first technique reduces the problem from considering general hash functions and adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ as above to a simpler model (and associated attacker) called the *bit-fixing* model. The advantage of the latter model is that it is typically easier to analyze and results in clean proofs. The second technique is based on a simple information-theoretic idea that *random bits cannot be compressed.*[7] Thus, an attacker that succeeds in finding collisions is used to compress some random information that is used in the game, and thereby contradiction is reached. This technique, while being extremely influential in many fields and problems in computer science (e.g., "Algorithmic Lovász Local Lemma" [27], lower bounds on cryptographic constructions [18,13,20], analyzing hardness of problems in the non-uniform setting [15,10] and time-space tradeoffs for quantum algorithms [7]), often results in technical and complex proofs.

It would have been convenient if any non-trivial bound on our problem could be obtained using the bit-fixing technique. Unfortunately, Akshima et al. [3] observed that finding short collisions is relatively easy in the bit-fixing model. Hence, the only remaining potentially helpful technique is based on compression. Indeed, Akshima et al. [3], as their primary technical contribution, managed to carry out such an argument for the particular case of $B = 2$, and already then their proof is highly non-trivial and consists of a tedious case analysis. We distill some of the main ideas underlying their general framework approach[8] next—this will be useful for us, as well.

The framework. We reduce the task of handling arbitrary (S,T)-adversaries to the problem of handling (S,T)-adversaries, where the preprocessing part $\mathcal{A}_1$ is degenerate and outputs a fixed string σ, independent of h. Specifically, we define a game, parameterized by $u \in \mathbb{N}_{>0}$, where $\mathcal{A}_2$ has an arbitrary size S string σ hard-coded, and its goal is to find a collision relative to a given salt. $\mathcal{A}_2$ wins the game if it succeeds in finding a collision when executed with *every one* of u uniformly random salts (there is no $\mathcal{A}_1$ in this game). The reduction shows that

[7] Specifically, it is impossible to save w bits of information about a random string, except with probability 2^{-w}.

[8] We note that [7] introduced an equivalent framework in independent work.

if $\mathcal{A}_2$ has advantage ϵ in the modified game, then the best advantage of an (S, T)-adversary in the original game is (roughly) $O(\epsilon^{1/u})$ for $u \approx S$. This reduction, formalized in Lemma 4.1, is adapted from Akshima et al. [3] and it uses the beautiful "constructive Chernoff bound" of Impagliazzo and Kabanets [22].[9]

The advantage of considering the new game is that there is no $\mathcal{A}_1$, so it is easier to handle. But, to obtain a meaningful result for the original (S, T) game, say an upper bound of ϵ, we need to prove a somewhat stronger upper bound for the new game, that is, roughly ϵ^u. This means, in other words, that we need to show how to compress about $\log(1/\epsilon)$ bits *per each one of the u salts.* (Actually, keep in mind that it suffices to achieve this on average!) For our target ϵ, we therefore have the following goal.

Main challenge: For every one of the u salts, we need to "save/compress" (on average) roughly the following number of bits:

$$\log \left(\min \left\{ \frac{N}{uT(\log u)^{2(B-2)}B^2}, \frac{N}{T^2} \right\} \right).$$

By impossibility of non-trivial compression, this would imply that $\mathcal{A}_2$ must succeed with probability at most

$$\epsilon \leqslant O\left(\left(\frac{uT(\log u)^{2(B-2)}B^2}{N} + \frac{T^2}{N} \right)^u \right),$$

which would give our result when plugged into the framework.

The compression argument. The random string that we shall compress consists of the set of salts denoted U, as well as the function h. We give encoding and decoding algorithms that use $\mathcal{A}_2$ first to encode the pair (U, h) and then use the result to fully decode them whenever $\mathcal{A}_2$ wins the game. If $\mathcal{A}_2$ wins with good enough probability, the output of the encoding procedure will be non-trivially short with good probability, which is a contradiction.

Remark 1. Akshima et al. [3] used the same approach for $B = 2$, but their proof does not seem to scale for larger values of B. Specifically, their proof involves an exhaustive case analysis. It seems like a naive attempt to generalize their bound to larger values of B would proliferate the number of cases needed to be analyzed, making it unmanageable. One of our key conceptual insights is a structural characterization of collisions in MD_h that prevents this explosion in the number of cases needed to be handled. While our analysis applies to any B, the number of total cases we consider is roughly the same as Akshima et al.

A first attempt and a glimpse at the challenge. Let us make a strong (and typically false) assumption: *the adversary $\mathcal{A}_2$ never repeats any query to h*

[9] The use of this reduction is the main (and perhaps only) point of similarity between our proof and [3]'s.

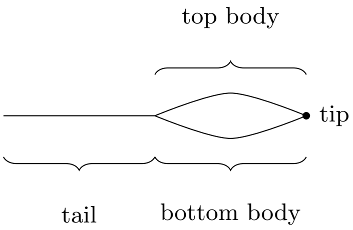


Fig. 1: A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure.

across all the u runs.[10] Since we can assume (w.l.o.g) that if $\mathcal{A}_2$ outputs (α, α') when run on salt a, it has queried h at all values needed to compute $\mathsf{MD}_h(a, \alpha)$ and $\mathsf{MD}_h(a, \alpha')$, we are guaranteed that for each of the salts $u \in U$ there are at least two *distinct* queries which have the same answer, i.e., a collision. The indices of these queries reside in $[T]$ as this is the query complexity of $\mathcal{A}_2$ when executed on the particular salt a. Thus, we could avoid encoding the answer of the second query and instead encode these two indices in T and remove the answer of the second query from evaluations of h. This saves us $\log(N/T^2)$ bits for every salt, giving us even more savings than what we are aiming for. Such compression, in turn, would imply that the birthday attack is optimal, no matter what B is (which makes sense given our assumption but is clearly false for general attackers).

A naive way to get rid of the assumption (that queries never repeat across different u runs) would be to encode the index of the other query among all uT queries made (instead of T). But, this would eventually result in another multiplicative S term in our bound. Namely, we would only be able to save $\log(N/(ST^2))$ bits per salt which is too little for us (as it leads to a trivial upper bound). This motivates us to look more closely at how MD collisions are formed, what kind of queries could be involved, and how we could leverage the fact that collisions are short to get more efficient encoding.

The mouse structure and query types. We consider the graph implicitly formed through the queries made by $\mathcal{A}_2$ when running on salts $a_1, \ldots, a_u \in U$. The nodes of the graph are the possible salts and there is a directed edge from salt a to a' with label α if $h(a, \alpha) = a'$ and this query was made by $\mathcal{A}_2$.

Suppose that $\mathcal{A}_2$, when run on salt a_i, outputs (α, α'). Since $\mathcal{A}_2$ wins on every salt in U, its output on salt a_i, denoted α, α' must satisfy: (1) $\mathsf{MD}_h(a_i, \alpha) = \mathsf{MD}_h(a_i, \alpha')$, (2) $\alpha \neq \alpha'$, and (3) α, α' are at most B blocks long. Without loss of generality we can assume that the adversary $\mathcal{A}_2$ outputs a "minimal" collision, i.e., one that does not contain a prefix which is a collision by itself.

[10] While we can assume without loss of generality that $\mathcal{A}_2$ does not repeat queries within a single execution (since it is not memory-bounded), it is not very reasonable to assume that it will never repeat queries across different executions on different salts.

For instance, say the collision is $x_1, x_2, x_3, x_4, x_5, x_6$ and $y_1, y_2, y_3, y_4, y_5, y_6$ (wrt salt a) and it happens that x_1, x_2, x_3, x_4 and y_1, y_2, y_3, y_4 already collide (wrt same salt a), we can simply ignore x_5, x_6, y_5, y_6. Considering the *core* sub-graph of the query graph that is induced by queries made by $\mathcal{A}_2$ that are *required* to evaluate $\mathsf{MD}_h(a, \alpha)$ and $\mathsf{MD}_h(a, \alpha')$, we obtain a structure that we call a **mouse structure**. Important parts of the mouse structure are the tail, top and bottom body, and the tip, as depicted in Fig. 1. Our entire approach is based on studying these mouse structures to come up with encoding strategies.

Given the concept of a mouse structure and our discussion from above about the adversary, possibly repeating queries motivates us to classify each query into one of three types. The first is called NEW and refers to queries made for the first time. The rest of the queries are called *repeated*, and they are further classified into two types, depending on whether they previously appeared in some mouse structure or not. Specifically, a repeated query is called REPEATEDMOUSE if the same query was already made by $\mathcal{A}_2$ when executed on a previous salt, and otherwise, a repeated query is called REPEATEDNONMOUSE.

Intuitively, we want to save bits when the answer of a NEW query was the input salt of a repeated query. A REPEATEDMOUSE query facilitates such savings since we can encode the answer to the query by storing its index along with the index of the previous query and the corresponding index within the mouse structure—a total of $\approx \log(uTB)$ bits instead of $\log N$—which eventually turns into an STB/N term in the upper bound, as conjectured. The problem is with encoding REPEATEDNONMOUSE queries—there seems to be no trivial way to write the index of the previously-made query with less than $\log(uT^2)$ bits, which is too much (since it will eventually turn into a ST^2/N term in the upper bound).

Some "easy" mouse structures. We observe that some cases of mouse structures readily give us a way to have efficient encoding and save sufficiently many bits. We offer three examples to convey intuition on how our analysis is done. Throughout, let us assume that every mouse structure contains at least one NEW query—otherwise, we will ignore this mouse structure altogether.[11]

As a first example, if two NEW queries form the tip of the mouse structure of salt a_j, we can simply encode the index of these queries within the queries made while handling this salt—this uses $2 \log T$ bits instead of $\log N$ bits which is sufficient. As another example, if a self-loop forms the body of the mouse structure and the self loop query is NEW or REPEATEDNONMOUSE, we can simply encode the index of the self loop query in the list of queries used to handle this salt and avoid encoding the answer of the query—this uses $\log(uT)$ bits instead of $\log N$ which is again sufficient. As the last example, suppose the answer to a NEW query is a salt that appeared in some earlier mouse structure. In this case, we can avoid encoding the answer of the NEW query and instead encode

[11] In the technical section, we refer to salts a_j in U that were not the input salt of a query when running $\mathcal{A}_2$ on a_i for $i < j$ as *fresh*. It follows that mouse structures for fresh salts will always have a NEW query. For salts that are not fresh, it is relatively straightforward to achieve some compression by avoiding storing these salts in the encoding of U. For now, the reader can imagine that all salts are fresh for simplicity.

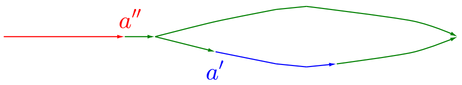


Fig. 2: An example of a "hard" mouse structure. The NEW queries are drawn in red, REPEATEDMOUSE queries are drawn in blue and the REPEATEDNONMOUSE queries are drawn in green.

the index of the query and which of the salts in the previous mouse structures is the answer—this uses $\log(2uB)$ bits (because there are at most $2B$ salts in mouse structures and at most u mouse structures) instead of $\log N$ which is also sufficient.

Some "hard" mouse structures. The aforementioned easy cases give rise to a relatively easy encoding that results in an optimal STB/N term. Next, we focus on the more complex cases, which cause our bound to have the extra $\approx B \cdot (\log S)^B$ factor.

Assume that there is a mouse structure where there are two salts a' and a'' such that a' is the input salt to a REPEATEDMOUSE query, a'' is the answer to a NEW query, and the path from a'' to a' in the mouse structure consists of only REPEATEDNONMOUSE queries (as shown in Figure 2). By definition, the distance between a' and a'' is at most B.[12] Potential savings could be achieved by not encoding a'', the answer of the NEW query, as it can essentially be extracted from already-observed queries. We can easily encode the appropriate index of the query in the mouse structure and the salt a' using roughly $\log(uB)$ bits, but how can we encode the information about the path back from a' to a''?

The non-triviality is that for each node on this path, there might be many possible ways to reach it among all the different queries that have been already made, i.e., if each node on this path has fan-in m, namely an m-multi-collision, then the natural encoding of the path back would cost at most $\log(m^B)$ bits, specifying which back edge to take for every node. Here, m could be very large, e.g., as large as S or even larger, making the whole result meaningless for most reasonable parameters settings. (We also need to encode the length of the path, which would lead to the additional multiplicative B factor, but we ignore it in the discussion here.)

We get around this problem of m potentially being very large by observing that one of the following two cases holds:

- *Many small multi-collisions:* either the fan-in of every node along the path (i.e., the number of previously-made queries whose result is a node on this path) is smaller than $\log u$, or
- *One large multi-collision:* there is (at least one) node on the path where the fan-in is at least $\log u$.

[12] By being slightly more careful, we can show that the distance is $B-2$ but we ignore this fact for the overview.

In the earlier case, we encode the path back as mentioned above, where we write the index of each back edge. This costs us $B \cdot \log \log u$ bits, which eventually translates to an extra $(\log S)^B$ term,[13] as we have in our main theorem. The question is, what do we do when the second case occurs.

The idea is to leverage the fact that there is a large multi-collision and obtain our savings from a completely different place. Specifically, we remember the index of all the queries involved in the multi-collision and the common answer. A calculation reveals that if the collision consists of $\approx \log u$ edges, we can already save enough. One subtlety is that the same multi-collision might repeat throughout many different mouse structures, and we need to make sure not to double count the savings from a single multi-collision more than what we get. The reason why we do not double count is that a single large-enough (i.e., with $\log u$ edges) multi-collision saves us enough bits for about $\log u$ different structures, and at the same time, such a multi-collision can appear in at most $\log u$ mouse structures.[14]

Let us finally remark that while the above description conveys the main idea underlying our compression strategy, it is somewhat simplified and glossed over many technicalities and the complete specification of all possible cases that our full proof covers. We refer to Section 5 for full details.

Proving the STB conjecture for $S \cdot B \ll T$. The proof of our second upper bound follows the same overall structure. The only difference is, naturally, in the way we encode reverse paths in mouse structures that have nodes corresponding to REPEATEDNONMOUSE queries between the output a'' of a NEW query and a node a' corresponding to a REPEATEDMOUSE query.

In the earlier proof, when we needed to locate the salt a'' from salt a', we encoded the number of edges in between and all the edges on the path. Here, we prove a purely graph-theoretic lemma saying that if for salt a' there are more than $\approx u^2$ salts a'' such that there are $d > 0$ edges on the shortest path back from a' to a'' in the query graph, then there must be $t \geqslant 1$ multi-collisions in the graph, such that the total number of edges involved in the t multi-collisions is at least $\approx u^2$. While the proof of this lemma is a simple inductive argument, it turns out to be extremely helpful for us. Specifically, if there are $t \geqslant 1$ different multi-collisions such that a total of at least $\approx u^2$ different queries are involved in the t multi-collisions, *we can save enough by only encoding these multi-collisions and nothing else.* To prove this fact, we consider the minimum savings we can get from encoding these t multi-collisions. We show using some elementary calculus that if there are $\approx u^2$ queries involved in t different multi-collisions, the minimum saving is more than the total amount of savings we need.

Equipped with this fact, we split our analysis into the two following scenarios.

[13] Remember that the actual term is $(\log S)^{B-2}$ and that is why the proof of Akshima et al. [3], in which it was assumed that $B = 2$, did not have an extra term that depends on S.

[14] We mention that multi-collisions in hash functions have been studied on their own right (e.g., [23,14]), but our context is totally different.

```
Game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

1. $h \leftarrow_\$ \mathsf{Fcs}([N] \times [M], [N])$
2. $a \leftarrow_\$ [N]$
3. Return $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A})$
```

```
Subroutine $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

1. $\sigma \leftarrow_\$ \mathcal{A}_1(h)$
2. $(\alpha, \alpha') \leftarrow_\$ \mathcal{A}_2^h(\sigma, a)$
3. Return true if:
    (a) $\alpha \neq \alpha'$,
    (b) $\alpha, \alpha'$ consist of $\leqslant B$ blocks from $[M]$,
    (c) $\mathsf{MD}_h(a, \alpha) = \mathsf{MD}_h(a, \alpha')$
4. Else, return false
```

Fig. 3: The bounded-length collision resistance game of salted MD hash in the AI-ROM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$.

1. The first is where for each case where we need to encode the location of a'' from a' at a distance d, there are at most $\approx u^2$ salts – here we simply encode the index of the "right" a'' using $\approx \log u^2$ bits.
2. The other scenario is when for at least one case, there are more than $\approx u^2$ salts. Here we can save enough by encoding the t multi-collisions involving at least $\approx \log u^2$ that the graph-theoretic lemma guarantees us. We get enough savings by only encoding these t multi-collisions.

The u^2 term from the first scenario above turns into an additional factor of the form S^2 in the final bound. Due to additional technicalities that we glossed over during this overview, we suffer another multiplicative factor S in our bound, which amounts to having an S^4 term. Full details appear in Section 6.

3 Preliminaries

For a positive integer $N \in \mathbb{N}_{>0}$, let $[N] = \{1, 2, \ldots, N\}$ and for $k \in \mathbb{N}$ such that $k \leqslant N$, let $\binom{[N]}{k}$ denote the set of k-sized subsets of $[N]$. For a set X, let $|X|$ be its size and X^+ denote one or more elements of X. We denote $\mathsf{Fcs}(D, R)$ the set of all functions mapping elements in D to the elements of R. We let $x \leftarrow_\$ \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. 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). If D is a set, we overload notation and let $x \leftarrow_\$ D$ denote uniformly sampling from the elements of D. For a bit-string s we use $|s|$ to denote the number of bits in s.

When referring to directed graphs in this paper, we mean directed multigraphs, i.e., these directed graphs might have parallel edges. All logarithms in this paper are for base 2 unless otherwise stated.

Auxiliary-input Random Oracle Model (AI-ROM). We use the Auxiliary-Input Random Oracle Model (AI-ROM) introduced by Unruh [30] to study non-uniform adversaries in the Random Oracle Model. This model is parameterized by two non-negative integers S and T and an adversary $\mathcal{A}$ is divided into two stages $(\mathcal{A}_1, \mathcal{A}_2)$. Adversary $\mathcal{A}_1$, referred to as the preprocessing phase of $\mathcal{A}$, has unbounded access to the random oracle h and outputs an S-bit auxiliary input σ. Adversary $\mathcal{A}_2$, referred to as the online phase, gets σ as input and can make T queries to h, attempting to accomplish some goal involving the function h. Formally, we say that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an (S,T)-AI adversary if $\mathcal{A}_1$ outputs S bits and $\mathcal{A}_2$ issues T queries to its oracle. We next formalize the salted-collision resistance of MD hash functions in AI-ROM.

Salted short collision resistance of MD in AI-ROM. We formalize the hardness of bounded-length collision resistance of salted MD hash functions in the AI-ROM. The game is parametrized by N, M, and B. The game first samples a function h uniformly at random from $\mathsf{Fcs}([N]\times[M],[N])$ and a salt a uniformly at random from $[N]$. Then, $\mathcal{A}_1$ is given unbounded access to h, and it outputs σ. At this time, $\mathcal{A}_2$ is given the auxiliary input σ, a salt a, as well as oracle access to h, and it needs to find $\alpha \neq \alpha'$ such that (1) $|\alpha|, |\alpha'| \leqslant B \cdot M$, and (2) $\mathsf{MD}_h(a,\alpha) = \mathsf{MD}_h(a,\alpha')$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$, is explicitly written in Fig. 3. In Fig. 3, we write the adversary's execution in its own subroutine only for syntactical purposes (as we shall use it later in our proof).

Definition 1 (AI-CR Advantage). *For parameters $N, M, B \in \mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of salted MD in the AI-ROM is*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(\mathcal{A}) = \Pr\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}(\mathcal{A}) = \mathsf{true}\right]$$

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

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) = \max_{\mathcal{A}}\left\{\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(\mathcal{A})\right\},$$

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

The compression lemma. Our proof uses the well-known technique of finding an "impossible compression". The main idea, formalized in the following proposition, is that it is impossible to compress a random element in set $\mathcal{X}$ to a string shorter than $\log|\mathcal{X}|$ bits long, even relative to a random string.

Proposition 1 (E.g., [13]). *Let Encode be a randomized map from $\mathcal{X}$ to $\mathcal{Y}$ and let Decode be a randomized map from $\mathcal{Y}$ to $\mathcal{X}$ such that*

$$\Pr_{x \leftarrow\$ \mathcal{X}}\left[\mathsf{Decode}(\mathsf{Encode}(x)) = x\right] \geqslant \epsilon.$$

Then, $\log|\mathcal{Y}| \geqslant \log|\mathcal{X}| - \log(1/\epsilon)$.

4 The Framework: Reducing the Problem to a Multi-instance Collision Finder

Our task here is to upper-bound the advantage of an adversary in finding a short collision in a salted MD, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$ described in Figure 3.

First, without loss of generality, in what follows, we assume that the adversary is deterministic. This follows since we can transform any probabilistic attacker into a deterministic one by hard-wiring the best randomness (see Adleman [2]).

We reduce the task of bounding the advantage of an attacker in finding a short collision in a salted MD, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$, to a "multi-instance" game where the adversary does not have a preprocessing phase but instead only has a non-uniform auxiliary input, chosen *before* the random oracle h. The latter game is easier to analyze. Although the statement and reduction below were implicit in the work of Akshima et al. [3], we make it formal and hopefully useful for future works.

We define the following "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma, \mathcal{A}_2)$, where the preprocessing part of the adversary $\mathcal{A}_1$ is degenerate and outputs the fixed string σ. More precisely, the game has the following steps:

1. $h \leftarrow_{\$} \mathsf{Fcs}([N] \times [M], [N])$
2. $U \leftarrow_{\$} \binom{[N]}{u}$
3. Define $\mathcal{A}_1$ to be the algorithm that always outputs the string σ.
4. Return true if $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)) = \mathsf{true}$ for every $a \in U$. Otherwise, return false.

For a string σ and an adversary $\mathcal{A}_2$, define

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2) = \Pr\left[\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma, \mathcal{A}_2)\right].$$

Lemma 4.1. *Fix $N, M, B, S, T, u \in \mathbb{N}_{>0}$. Then,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant 6 \cdot \left(\max_{\sigma,\mathcal{A}_2}\left\{\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2)\right\}\right)^{\frac{1}{u}} + 2^{S-u},$$

where the maximum is taken over all $\sigma \in \{0,1\}^S$ and T-query algorithms $\mathcal{A}_2$.

The proof of this lemma is similar to a proof that appears in [3]. For completeness, we provide the full details in the full version [19].

5 Proving the STB Conjecture for $B \in O(1)$

This section proves an upper bound on the advantage of any auxiliary-input adversary in the bounded-length collision resistance game of salted MD hash in the AI-ROM. The main theorem is stated next.

Theorem 5.1. *Let $C = 2^{16} \cdot 6 \cdot e^2$. For any $N, M, B, S, T \in \mathbb{N}_{>0}$ and fixing $\hat{S} := S + \log N$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant C \cdot \max\left\{\left(\frac{\hat{S}TB^2\left(\frac{3e\log\hat{S}}{\log\log\hat{S}}\right)^{2(B-2)}}{N}\right), \left(\frac{T^2}{N}\right)\right\} + \frac{1}{N}.$$

Theorem 5.1 follows as a direct corollary of Lemma 4.1 together with the following lemma and setting $u = S + \log N$.

Lemma 5.1 (Hardness for a multi-instance collision finder). *Fix N, M, B, S, T, $u \in \mathbb{N}_{>0}$ and $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2) \leqslant \left(2^{16}e^2 \cdot \max\left\{\left(\frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N}\right), \left(\frac{T^2}{N}\right)\right\}\right)^u.$$

The rest of this section is devoted to the proof of Lemma 5.1. Unlike the proof of Lemma 4.1, the proof of this lemma is novel and differs completely from that of Akshima et al. [3]. The key conceptual insight is a structural characterization of collisions in MD_h that prevents the explosion in the number of cases that [3] faced during the case analysis.

We are interested in bounding the advantage of the best strategy, i.e., a pair $(\sigma, \mathcal{A}_2)$ where $\sigma \in \{0,1\}^S$ is a fixed string and $\mathcal{A}_2$ is a T-query algorithm, of finding bounded-length collisions in a salted MD with respect to the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma, \mathcal{A}_2)$. Recall that in this game, $\mathcal{A}_2$ needs to find proper collisions for u randomly chosen salts, denoted U. The main idea in the proof is to use any such adversary $(\sigma, \mathcal{A}_2)$ to represent the function h *as well as* the set of random salts U with as few bits as possible. If the adversary is "too good to be true," we will get an impossible representation, contradicting Proposition 1.

Non-trivial range. If either

$$\frac{T^2}{N} > 1 \quad \text{or} \quad \frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N} > 1,$$

then Lemma 5.1 is trivially true. Hence, from now on we assume that both of the above left hand side terms are upper bounded by 1.

Setup. Denote

$$\zeta^* := \left(2^{16}e^2 \cdot \max\left\{\left(\frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N}\right), \left(\frac{T^2}{N}\right)\right\}\right)^u.$$

Assume the existence of an adversary $\mathcal{A} = (\sigma, \mathcal{A}_2)$, where $\sigma \in \{0,1\}^S$ is a string and $\mathcal{A}_2$ is a T-query adversary, that contradict the inequality stated in the lemma. That is, there is $\zeta > \zeta^*$ such that

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\mathcal{A}) := \zeta > \zeta^*. \tag{1}$$

Define $\mathcal{G}$ to be the set of functions-sets of salts pairs for which the attacker succeeds in winning the game for every salt in the set relative to the function, That is,

$$\mathcal{G} = \left\{(U,h) \;\middle|\; \begin{matrix} U \in \binom{[N]}{u}, \\ h \in \mathsf{Fcs}([N]\times[M],[N]), \end{matrix} \forall a \in U : \mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right\}.$$

Recall that ζ is defined to be the advantage of $\mathcal{A}$ in the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\mathcal{A})$ in which h and U are chosen uniformly, and then $\mathcal{A}$ needs to find a collision with respect to every one of the u salts in U. Therefore,

$$|\mathcal{G}| = \zeta \cdot \binom{N}{u} \cdot N^{MN}.$$

In what follows we define an encoding and a decoding procedure such that the encoding procedure gets as input U, h such that $U \in \binom{[N]}{u}$ and $h \in \mathsf{Fcs}([N] \times [M], [N])$, and it outputs an L bit string, where $L = \log\left(\zeta^* \cdot \binom{N}{u} \cdot N^{MN}\right)$. The decoding procedure takes as input the string L and outputs U^*, h^*. It will hold that $U^* = U$ and $h^* = h$ with probability ζ.[15] Using Proposition 1, this would give us that

$$\log \zeta \leqslant L - \log\left(\binom{N}{u} \cdot N^{MN}\right) \Longrightarrow \zeta \leqslant \zeta^*$$

which is a contradiction to the assumption (see (1)).

Notation and Definitions. Fix $(U, h) \in \mathcal{G}$. Let $U = \{a_1, \ldots, a_u\}$ where the a_i's are ordered lexicographically. Let $\mathsf{Qrs}(a) \in ([N] \times [M])^T$ be the list of queries that $\mathcal{A}_2$ makes to h when executed with input (σ, a). Namely, for $a \in [N]$,

$$\mathsf{Qrs}(a) = \left\{(a', \alpha') \in [N] \times [M] \mid \mathcal{A}_2(\sigma, a) \text{ queries } h \text{ on } (a', \alpha')\right\} .$$

Note that $\mathsf{Qrs}(a)$ is indeed a set as we can assume (without loss of generality) that $\mathcal{A}_2$ never repeats queries in a single execution (since $\mathcal{A}_2$ can just store all of its past queries).

We say that $a' \in \mathsf{Slts}(a)$ if there is some $\alpha' \in [M]$ such that (a', α') is an entry in $\mathsf{Qrs}(a)$. Namely, for $a, a' \in [N]$,

$$a' \in \mathsf{Slts}(a) \Longleftrightarrow \exists \alpha' \in [M] \text{ s.t. } (a', \alpha') \in \mathsf{Qrs}(a).$$

We define the set of *fresh* salts in U. A salt a_i for $i \in [u]$ is called fresh if it was never used as the salt in any query performed by $\mathcal{A}_2$ while being executed on salts a_j for $j \leqslant i-1$ which are fresh. The first salt a_1 is always fresh. A salt a_i for $i \geqslant 2$ is fresh if for any fresh a_j for $j \leqslant i-1$, $a_i \notin \mathsf{Slts}(a_j)$. Namely, denoting the set of fresh salts by U_{fresh}, we have the following inductive (on $i \in [u]$) definition:

$$a_i \in U_{\mathsf{fresh}} \Longleftrightarrow \forall j \leqslant i-1, a_j \in U_{\mathsf{fresh}} : a_i \notin \mathsf{Slts}(a_j).$$

Looking ahead, we define U_{fresh} like this because we run $\mathcal{A}_2$ on the salts in U_{fresh} in lexicographical order, and this definition ensures that each salt that $\mathcal{A}_2$ is executed on was not queried by it previously. Denote

$$F := |U_{\mathsf{fresh}}| \quad \text{and} \quad U_{\mathsf{fresh}} = \{a'_1, \ldots, a'_F\} \text{ (ordered lexicographically).}$$

[15] Essentially, we will show that for all $(U, h) \in |\mathcal{G}|$, if the encoding procedure produces output L, then the decoding procedure on input L outputs U^*, h^* such that $U^* = U$ and $h^* = h$.

Denote

$$\forall i \in [F] \colon \mathsf{Q}_i := \mathsf{Qrs}(a'_i) \quad \text{and} \quad \mathsf{Q}_{\mathsf{fresh}} := \mathsf{Q}_1 \,\|\, \ldots \,\|\, \mathsf{Q}_F,$$

where $\|$ is the concatenation operator. Let $\mathsf{Q}_{\mathsf{fresh}}[r]$ be the rth query in the list $\mathsf{Q}_{\mathsf{fresh}}$. Note that $r \in [F \cdot T]$. For every $a \in U \backslash U_{\mathsf{fresh}}$, let t_a be the minimum value such that $\mathsf{Q}_{\mathsf{fresh}}[t_a]$ is a query with salt a. Define the set of *prediction* queries as

$$\mathsf{P} := \{t_a \mid a \in U \backslash U_{\mathsf{fresh}}\}.$$

The encoding algorithm will output $U_{\mathsf{fresh}}, \mathsf{P}$, which suffices to recover the set U by running $\mathcal{A}_2$.

We let $\tilde{h}$ be the list of $h(a, \alpha)$ values when executed on distinct queries in $\mathsf{Q}_{\mathsf{fresh}}$, in the same order as they appear in $\mathsf{Q}_{\mathsf{fresh}}$, followed by the evaluation of h on the following values in lexicographical order of the inputs.

$$\{(a, \alpha) : a \in [N], \alpha \in [M]\} \backslash \mathsf{Q}_{\mathsf{fresh}} \ .$$

Therefore, $\tilde{h}$ is initialized to contain the evaluation of h at all points in its domain. Looking ahead, in the encoding procedure, we will remove elements from $\tilde{h}$ as needed to compress h.

Function and Query graphs. A notion that will be useful is that of a "function graph".

Definition 2 (Function graph). *For a function $h : [N] \times [M] \to [N]$, consider the following directed graph: it has N nodes labelled with elements of $[N]$ and each node has exactly M outgoing edges, each labelled with elements of $[M]$. There is an edge from node a_i to a_j labelled α if and only if $h(a_i, \alpha) = a_j$.*

We define the notion of query graph for an adversary as follows.

Definition 3 (Query graph). *Execution of an adversary $\mathcal{A}_2$ on salts $a'_1, \ldots, a'_F$ defines a query graph as follows. Initially the graph is empty. Whenever $\mathcal{A}_2$ queries (a, α) to h, we add a node with label a if not already present and add an edge $(a, h(a, \alpha))$ with label α if not already present.*

Fact 5.2 *The query graph is always a sub-graph of the function graph of h.*

Structure of collisions: The mouse structure. Since adversary $\mathcal{A}_2$ succeeds on all of the salts in U, it holds that for every $j \in [F]$, the output of the adversary is (α_j, α'_j) such that $\alpha_j \neq \alpha'_j$, $\mathsf{MD}_h(a_j, \alpha_j) = \mathsf{MD}_h(a_j, \alpha'_j)$ and both α_j, α'_j are at most B blocks long. We can assume without loss of generality that the colliding messages α_j and α'_j are "minimal" (because otherwise, we can trim α_j, α'_j to obtain a shorter collision). The evaluations of h in order to compute $\mathsf{MD}_h(a'_j, \alpha_j)$ and $\mathsf{MD}_h(a'_j, \alpha'_j)$ induce a structure that we call a *mouse structure* as shown in Fig. 4. More explicitly, suppose the output of the $\mathcal{A}_2$ is $(\alpha_j = (\alpha_{j,1}, \ldots, \alpha_{j,B_1}), \alpha'_j = (\alpha'_{j,1}, \ldots, \alpha'_{j,B_2}))$ for $B_1, B_2 \leqslant B$ such that $\alpha_{j,i} =$

$\alpha'_{j,i}$ for all $1 \leqslant i \leqslant k$ where $k \geqslant 0$. Define $(x_1, \ldots, x_k, y_1, \ldots, y_{k'}, z_1, \ldots, z_{k''})$ where $k' = B_1 - k, k'' = B_2 - k$ as follows.

$$x_1 = h(a'_j, \alpha_{j,1}) \ , \ x_i = h(x_{i-1}, \alpha_{j,i}) \text{ for } 1 < i \leqslant k$$

$$y_1 = \begin{cases} h(a'_j, \alpha_{j,1}) & \text{if } k = 0 \\ h(x_k, \alpha_{j,k+1}) & \text{otherwise} \end{cases} , \ y_i = h(y_{i-1}, \alpha_{j,i+k}) \text{ for } 1 < i \leqslant B_1 - k$$

$$z_1 = \begin{cases} h(a'_j, \alpha'_{j,1}) & \text{if } k = 0 \\ h(x_k, \alpha'_{j,k+1}) & \text{otherwise} \end{cases} , \ z_i = h(z_{i-1}, \alpha'_{j,i+k}) \text{ for } 1 < i \leqslant B_2 - k$$

Then $(x_1, \ldots, x_k, y_1, \ldots, y_{k'}, z_1, \ldots, z_{k''})$ form a mouse structure as specified in

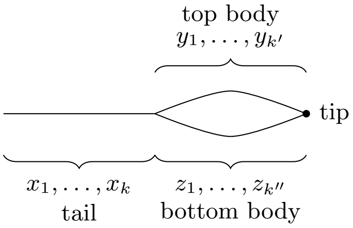


Fig. 4: A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure.

Figure 4. Without loss of generality, we can assume that the mouse structure is present in the query graph of $\mathcal{A}_2$ before it outputs the answer for salt a'_j. We refer to this structure as the mouse structure for salt a'_j.

We define $\mathsf{MouseQrs}_j$ to be the set of queries in the mouse structure. Similarly, we define $\mathsf{MouseSlts}_j$ as the set of salts that comprise the mouse structure. By definition, $1 \leqslant |\mathsf{MouseQrs}_j|, |\mathsf{MouseSlts}_j| \leqslant 2B$.

Classifying queries. We classify every one of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ into one of 3 types by scanning through them in order. Recall that $\mathsf{Q}_{\mathsf{fresh}}$ consists of F blocks, each consisting of T queries. Each block contains a mouse structure as in Figure 4. The first type of query is called NEW. A NEW query did not appear in any previous block. Non-NEW queries are called *repeated* and they are classified further into one of 2 types: REPEATEDMOUSE, and REPEATEDNONMOUSE. A query (a, α) would be a REPEATEDMOUSE query if it was made as part of a mouse structure during some earlier salt in U_{fresh}. Lastly, a REPEATEDNONMOUSE query is one that was made before but is not part of any mouse structure. That is,

1. NEW: A query with index $r \in [F \cdot T]$ is NEW if there does not exist $r' < r$ such that $\mathsf{Q}_{\mathsf{fresh}}[r'] = \mathsf{Q}_{\mathsf{fresh}}[r]$.
2. A non-NEW queries is called *repeated*. We classify the latter into two subcategories:

(a) REPEATEDMOUSE: A query in Q_j with index $s \in [T]$ such that $h(\mathsf{Q}_j[s]) = a$ is REPEATEDMOUSE if it is not NEW and $\mathsf{Q}_j[s] \in \mathsf{MouseQrs}_i$ for some $i < j$.
(b) REPEATEDNONMOUSE: A query in Q_j with index $s \in [T]$ such that $h(\mathsf{Q}_j[s]) = a$ is REPEATEDNONMOUSE if it is not NEW, $\mathsf{Q}_j[s] \notin \mathsf{MouseQrs}_i$ for all $i < j$.

Note that this classification covers all queries made during execution. The following is a simple observation.

Claim 5.3 *Every mouse structure has at least one* NEW *query.*

Proof. For every $j \in [F]$, the queries in Q_j with salt a'_j are necessarily NEW because we defined U_{fresh} to contain a'_j's that were not queried earlier by $\mathcal{A}_2$ when run on a'_i for $i < j$, and also assumed that $\mathcal{A}_2$ does not repeat queries during a single execution. ■

5.1 The Compression Argument

As mentioned, our goal is to compress (U, h), and we will achieve this by using our collision finding adversary $\mathcal{A}_2$. The encoding procedure shall output the set U_{fresh}, the set P, the list $\tilde{h}$ with some entries removed and additional lists and sets. We will be describing the details of these lists and sets below and which entries we remove from $\tilde{h}$. Our main goal is to show that we are compressing when we remove entries of $\tilde{h}$ and instead use additional lists and set. Our ways to compress will depend on the induced mouse structure in each Q_j for $j \in [F]$. To this end, we first classify the mouse structures into six broad cases. We classify the jth mouse structure for each $j \in [F]$ into the first of the following six cases it satisfies, e.g., if a mouse structure satisfies both cases 2 and 3, we categorize it into 2.

1. There is a NEW query (a, α) such that $h(a, \alpha) = a$.
2. There are two distinct NEW queries (a_1, α_1), (a_2, α_2) such that $h(a_1, \alpha_1) = h(a_2, \alpha_2)$.
3. There is a NEW query (a, α) such that $h(a, \alpha) = a'$ and $a' \in \mathsf{MouseSlts}_i$ for some $i < j$.
4. There is a REPEATEDNONMOUSE query (a, α) such that $h(a, \alpha) = a$.
5. There is at least one salt a such that $a \in \mathsf{MouseSlts}_i$ for some $i < j$ and there is a path of at most $B - 2$ edges in the mouse structure from a back to a' where a' is an answer to a NEW query.
6. There are no REPEATEDMOUSE queries in the mouse structure.

Note that these cases cover all mouse structures.

Claim 5.4 *Every mouse structure can be categorized into one of the cases 1 to 6.*

Proof. We will show that if a mouse structure does not satisfy case 6, it has to satisfy case 5, which suffices to prove our claim. Since the mouse structure is not in 6, it has a REPEATEDMOUSE query. Let the input salt of this query be a. Moreover, since the first query of the mouse structure has to be new, let the answer salt of this query be a'. Since the longest path in the mouse structure is of length B, it follows that there are at most $B-2$ edges in the mouse structure between a and a'. Hence, case 5 is satisfied. ∎

Compression budget. Recall that we need to prove that the size of the output of the encoding procedure is

$$L = \log\left(\left(2^{16}e^2\cdot\max\left\{\left(\frac{uTB^2m_0^{2(B-2)}}{N}\right),\left(\frac{T^2}{N}\right)\right\}\right)^u\cdot\binom{N}{u}\cdot N^{MN}\right)$$

bits, where $m_0 = 3e\log u/\log\log u$. In other words, we need to show that the encoding procedure saves at least

$$u\cdot\log\left(\min\left\{\frac{N}{4T^2},\frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right)-2u\log e-14u \tag{2}$$

bits overall.

Required savings in $\tilde{h}$. As mentioned earlier, the output of the encoding algorithm will consist of $U_{\mathsf{fresh}}, \mathsf{P}, \tilde{h}$, and some additional sets and lists. The lists U_{fresh} and P will suffice to recover the set U. The list $\tilde{h}$ and the additional sets and lists are used to recover h.

Denoting $|U_{\mathsf{fresh}}| = F$ and $|U| = u$, we can describe P using $\binom{FT}{u-F}$ bits. Therefore, U, which is trivially described using $\log\binom{N}{u}$ bits, can be encoded using $\log\left(\binom{FT}{u-F}\binom{N}{F}\right)$ bits.Therefore, the saving in bits in the description of U is at least

$$\begin{aligned}
\log\binom{N}{u}-\log\left(\binom{FT}{u-F}\binom{N}{F}\right) &\geqslant \log\left(\frac{\left(\frac{N}{u}\right)^u}{\left(\frac{eFT}{u-F}\right)^{u-F}\left(\frac{eN}{F}\right)^F}\right)\\
&= (u-F)\log\left(\frac{N}{FT}\right)-\log\left(e^u\left(\frac{u}{F}\right)^F\left(\frac{u}{u-F}\right)^{u-F}\right)\\
&\geqslant (u-F)\log\left(\frac{N}{uT}\right)-u\log 4e\ .
\end{aligned}\tag{3}$$

where the first inequality uses the basic bounds for binomial coefficients $(n/r)^r \leqslant \binom{n}{r} \leqslant (en/r)^r$, and the last inequality follows since $\forall x \geqslant 0\colon x \leqslant 2^x$, and $u \geqslant F$.

By subtracting (3) (how much we save in U) from (2) (how much we need to save in total), it suffices to show that we save at least

$$u \cdot \log\left(\min\left\{\frac{N}{4T^2}, \frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right) - 2u\log e$$
$$- 14u - (u-F)\log\left(\frac{N}{uT}\right) + u\log 4e$$

bits while encoding h. Since $\log(N/uT) \geqslant \log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right)$, this is at most the following number of bits.

$$F \cdot \log\left(\min\left\{\frac{N}{4T^2}, \frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right) - u\log e - 12u \tag{4}$$

To show that the compression indeed achieves the savings from (4), we will show that for every salt in U_{fresh}, we can save at least the following number of bits, except for a few cases.

$$\log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right) .$$

In the rare cases where we cannot save as much, we will incur a small penalty. We will show that the cumulative penalty we incur is at most $7u + u\log e$ bits. Additionally, we will label each of the salts in F with a few bits that describe its "type" (according to the cases described above – case 5 will have 3 subcategories, and case 6 will have 10 subcategories), and for this 5 bits will suffice. This will cost, in total, another $5u$ bits, and therefore the total size of the encoding will indeed be bounded by the term from (4).

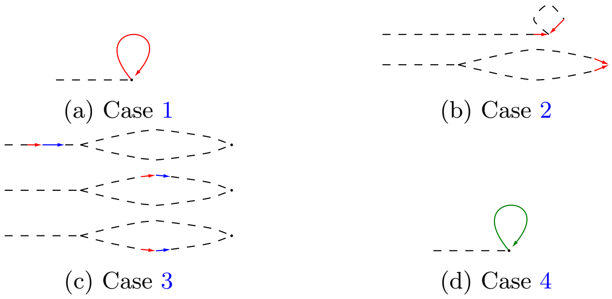
We now describe the details of how we handle each case. Assuming that the mouse structure for salt a'_j satisfies a particular case, we describe the encoding procedure, calculate the amount of compression we get, and then explain how decoding would work. In Section 5.2 we handle Cases 1 to 4, in Section 5.3, we handle case 5, and lastly in the full version [19], we handle case 6. Here we describe the details such that they are locally verifiable. We do provide the full pseudocode of encoding and decoding in the full version [19].

5.2 Handling Cases 1 to 4

In each of the four cases below, we will be saving more bits than we need, i.e., more bits than $\log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right)$.

Case 1. The jth mouse structure contains a NEW query (a, α) such that $h(a, \alpha) = a$, as depicted in Fig. 5a. The encoding procedure stores the index of the query (a, α) in Q_j in a list L_1 and removes the entry $h(a, \alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 1, then it removes the front index in the list L_1, and denote the index by i. It answers the ith h query, denoted (a, α), with a and sets $h(a, \alpha) = a$.


(a) Case 1

(b) Case 2

(c) Case 3

(d) Case 4

Fig. 5: Cases 1 to 4. The NEW queries are drawn in red, the REPEATEDMOUSE queries are drawn in blue, and the REPEATEDNONMOUSE queries are drawn in green. The black dashed lines indicate zero or more queries of any type.

Since the index of the query (a, α) in Q_j is in $[T]$, and we remove one element of $\tilde{h}$, we save $\log(N/T)$ which is more than what we need to save.

Case 2. The jth mouse structure contains two distinct NEW queries (a_1, α_1), (a_2, α_2) such that $h(a_1, \alpha_1) = h(a_2, \alpha_2)$, and (a_1, α_1) is queried before (a_2, α_2). This is depicted in Fig. 5b. The encoding procedure stores the pair indices of the queries (a_1, α_1), (a_2, α_2) in Q_j in a list L_2 and removes the entry $h(a_2, \alpha_2)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 2, then it removes the front element (i_1, i_2) in the list L_2. Suppose that the i_1th h query while running on salt a'_j is on (a_1, α_1). The decoding procedure gets the answer to this query from $\tilde{h}$. It answers the i_2th h query on (a_2, α_2) with $h(a_1, \alpha_1)$ and sets $h(a_2, \alpha_2) = h(a_1, \alpha_1)$.

Since the pair of indices of the queries $(a_1, \alpha_1), (a_2, \alpha_2)$ in Q_j are in $[T]$, and we remove one element of $\tilde{h}$, we save $\log(N/T^2)$ bits which is more than what we need to save.

Case 3. The jth mouse structure contains a NEW query (a, α) such that $h(a, \alpha) = a'$ and $a' \in \mathsf{MouseSlts}_i$ for some $i < j$. This is depicted in Fig. 5c. The encoding procedure stores the tuple consisting of i, the index of the query (a, α) in Q_j, and the lexicographical order of a' in $\mathsf{MouseSlts}_i$ in a list L_3 and removes the entry $h(a, \alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 3, it removes the front element (i_1, i_2, i_3) in the list L_3. Suppose the i_2th h query while running on salt a'_j is on (a, α). It answers the query with a' such that a' is the salt in $\mathsf{MouseSlts}_{i_1}$ whose lexicographical order is i_3. It sets $h(a, \alpha) = a'$.

Since $i \in [F]$, $F \leqslant u$, the index of (a, α) in Q_j is in $[T]$, the lexicographical index of a' in $\mathsf{MouseSlts}_i$ is in $[2B]$, and we remove one element of $\tilde{h}$, we save $\log(N/(2uTB))$ bits which is more than what we need to save.

Case 4. The jth mouse structure contains a REPEATEDNONMOUSE query (a, α) such that $h(a, \alpha) = a$. This is depicted in Fig. 5d. The encoding procedure stores the smallest index of the query (a, α) in $\mathsf{Q}_{\mathsf{fresh}}$ in a set S and removes the entry $h(a, \alpha)$ from $\tilde{h}$. Note that we never add an index corresponding to the same query multiple times to S. Indeed, if an index associated with (a, α) already appears in S, the next query on (a, α) will be a REPEATEDMOUSE query and not a REPEATEDNONMOUSE one, meaning that it cannot be added to S again.

In decoding, if the current (jth) salt is categorized as case 4, it checks for every h query on (a, α) whether S contains the index of the query in $\mathsf{Q}_{\mathsf{fresh}}$. If so it answers with a and sets $h(a, \alpha) = a$.

Since the smallest index of the query (a, α) in $\mathsf{Q}_{\mathsf{fresh}}$ is in $[FT]$, and we remove one element of $\tilde{h}$, we save at least $\log(N/(uT))$ which is more than necessary.

5.3 Handling Case 5

In this section, we describe our compression strategy in case the jth mouse structure for salt a'_j is categorized as case 5. That is, there is at least one salt d such that $d \in \mathsf{MouseSlts}_i$ for some $i < j$ and there are at most $B-2$ edges in the mouse structure between d and s where s is an answer to a NEW query. If there are several possible candidate pairs, we choose one where the number of edges is the smallest between the source and the destination salt.

Intuition. We refer to the salt s in the answer to the NEW query as the *source* salt, and the salt that appears in some earlier mouse structure as the *destination.* We are guaranteed that the path in the mouse structure from the source salt to the destination salt consists of at most $B-2$ edges that are REPEATEDNONMOUSE queries. (There is at least one intermediate edge between the source and the destination because otherwise, the answer of a new query will be the input of a REPEATEDMOUSE query, in which case this scenario would have been classified into case 3. Additionally, all the intermediate edges must be REPEATEDNONMOUSE since we consider the shortest possible path from such source to such destination.)

Let the NEW query, whose answer is s, be (a, α). Suppose the path from the s to the d in the mouse structure consists of REPEATEDNONMOUSE queries $(a_1, \alpha_1), \ldots, (a_p, \alpha_p)$ where $a_1 = s$, $p \leqslant B-2$. The main idea is to avoid encoding $s = h(a, \alpha)$ and recover it by encoding the lexicographical order of d in $\mathsf{MouseSlts}_i$ and encoding the path required to backtrack from d to s in the query graph at the time (a, α) is queried, i.e., encoding which of the past queries were $(a_p, \alpha_p), \ldots, (a_1, \alpha_1)$ (in this order). The problem is that, in general, the path back from d to s might be too expensive to encode. This depends on the number of "other" edges incident on the nodes on the real path back from t to s. If all nodes on the path have very few edges incident on them, say less than m_0 of them, we encode each back edge using $\log m_0$ bits per node, which requires with at most $\log(B(m_0)^{B-2})$ bits for the whole path back (the term B comes from encoding the length of the path). But, if some node has many adjacent edges, namely a *multi-collision* with more than m_0 edges, we will need to take advantage of this fact to obtain our savings (and not encode the path back from d to s).

Definition 4 (Large multi-collision). *We say that queries $q_1, \cdots, q_m$ form an m-way multi-collision if all the q_i's are distinct and all the $h(q_i)$ are equal. We say that the multi-collision is* large *if $m \geqslant m_0$, where $m_0 := 3e \log u / \log\log u$.*

If any of the queries in $(a_p, \alpha_p), \ldots, (a_1, \alpha_1)$ are involved in a large multi-collision of REPEATEDNONMOUSE queries in the query graph so far, we say we have "encountered a large multi-collision". In what follows, we explain how to obtain the required compression if a large multi-collision was *not* encountered.

Encoding when no large multi-collision. Suppose that the NEW query whose answer is the source salt s is (a, α), the destination salt is d and $d \in \mathsf{MouseSlts}_i$ for some $i < j$. The path back from d to s contains only nodes that have at most m_0 adjacent edges in the corresponding query graph since we have not encountered a large multi-collision. The encoding procedures constructs a tuple consisting ofthe index i, the index of query (a, α) in Q_j, the lexicographical index of d in $\mathsf{MouseSlts}_i$, and the path back from d to s in the query graph when (a, α) is queried. It stores the tuple in a list L_5. Finally, it removes the entry $h(a, \alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 5 *without a large multi-collision*, it detects the query (a, α) from its index in Q_j, then finds the salt d using the index i and the lexicographical order of d in $\mathsf{MouseSlts}_i$, and finally finds s using the path back from d to s. It answers the query with s.

Since $i \in [F]$, the index of (a, α) in Q_j is in $[T]$, the lexicographical index of a' in $\mathsf{MouseSlts}_i$ is in $[2B]$, the path back from d to s can be encoded in $\log(B(m_0)^{B-2})$ bits, and we remove one element of $\tilde{h}$, we save at least $\log(N/(2uTB^2(m_0)^{B-2}))$ bits which is more than necessary.

Encoding with large multi-collision. Suppose that the NEW query whose answer is the source salt s is (a, α), the destination salt is d and $d \in \mathsf{MouseSlts}_i$ for some $i < j$. Suppose further that the path back from d to s contains at least one node that has $m \geqslant m_0$ adjacent edges in the corresponding query graph (i.e., an m-multi-collision) such that these m edges are REPEATEDNONMOUSE queries when running $\mathcal{A}_2$ on a_j. First, observe that the multi-collision does not involve a self-loop. Indeed, if any node in the mouse structure has a REPEATEDNONMOUSE query whose answer is itself, the mouse structure would be classified into case 4, and therefore we will never reach this case.

At this point, we argue that we can record the multi-collision by encoding the indices of all queries associated with the multi-collision and the center, and remove their answers from $\tilde{h}$. To this end, we store $\log N + \log \binom{FT}{m}$ bits and remove $m \log N$ bits. We have that the saving is at least

$$m \log N - \log N - \log \binom{FT}{m} \geqslant \log \left(N^{m-1} \cdot \left(\frac{m}{eFT} \right)^m \right)$$
$$\geqslant \log \left(\left(\frac{N}{uT} \right)^{m-2} \cdot \frac{N}{T^2} \cdot \frac{m^m}{e^m u^2} \right) \geqslant \log \left(\left(\frac{N}{uT} \right)^{m-2} \cdot \frac{N}{T^2} \right) \qquad (5)$$

bits, where the first inequality follows by using the binomial inequality $\binom{FT}{m} \leqslant (eFT/m)^m$, the second inequality follows since $F \leqslant u$, and the last inequality follows since for $m \geqslant m_0 = 3e \log u / \log\log u$ it holds that $m^m \geqslant e^m u^2$. Thus,

Claim 5.5 *The number of bits saved is at least*

$$(m-1) \cdot \log\left(\min\left\{\frac{N}{T^2}, \frac{N}{uTB^2 m_0^{2(B-2)}}\right\}\right).$$

Proof. The claim follows since the minimum between the two terms is always upper bound by $N/(uT)$ and upper bounded by N/T^2. ∎

Thus, every m-multi-collision we record allows us to save the number of bits corresponding to $m-1$ mouse structures. It is left to argue that we do not over-count, namely, that we do not count the removal of the same element from $\tilde{h}$ twice. Indeed, the same multi-collision may be encountered in several mouse structures. To this end, observe that if a salt in a mouse structure is the center of a large multi-collision which we had recorded earlier, we will be in one of the following two cases:

1. There is a query in this mouse structure whose answer is the center of the multi-collision, and this query was in an earlier mouse structure.
2. There is a query in this mouse structure whose answer is the center of the multi-collision, and this query was not in any earlier mouse structure. (Note that by the structure of collisions, i.e., a mouse structure, there could be either one such query or two.)

Case 1 need not be handled. The reason is that if the condition in it holds, then the multi-collision is, in fact, outside of the (shortest) path from the source to the destination. Therefore the scenario will either be classified as a mouse structure *without* a large multi-collision or a different large multi-collision will be encountered for this mouse structure.

In case 2, first note that when we encounter a multi-collision, we add the relevant queries to the multi-collision if they were not already recorded and only then remove the corresponding entry from $\tilde{h}$. Therefore, we never remove anything twice. The last point we need to argue is that the we get enough savings. Above we showed that we have sufficient saving for $m-1$ mouse structures. Recall that we defined that we encounter a large multi-collision if at least m_0 repeatedNonMouse queries are involved in a multi-collision on some path we care about. It follows from this that a multi-collision of size m will be relevant in at most $m - m_0$ mouse structures- beyond that, it will no longer be a multi-collision because there will be less than m_0 repeatedNonMouse queries among the queries of the multi-collision. Since $m_0 \geqslant 1$, savings for $m-1$ mouse structures is sufficient for us.

Overall, as claimed earlier, case 5 has 3 different further categorizations– no multi-collisions and the two cases for multi-collisions.

The general ideas required for handling case 6 are similar to what we have already presented, but there are some subtleties. See full details in the full version [19].

6 Proving the STB Conjecture for $SB \ll T$

This section proves another upper bound on the advantage of any auxiliary-input adversary in the bounded-length collision resistance game of salted MD hash in the AI-ROM. The main theorem is stated next.

Theorem 6.1. *Let $C = 2^9 \cdot 6 \cdot e^4$. For any $N, M, B, S, T \in \mathbb{N}_{>0}$ and fixing $\hat{S} := S + \log N$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant C \cdot \max\left\{\left(\frac{T^2}{N}\right), \left(\frac{\hat{S}^4 T B^2}{N}\right)\right\} + \frac{1}{N} .$$

The proof of this theorem mostly mirrors that of Theorem 5.1, except in the way that few of the cases are handled in the compression argument. Technically, we derive the following Lemma 6.1 (an analogue of Lemma 5.1), and combine it with Lemma 4.1 to get the claimed bound in Theorem 6.1.

Lemma 6.1. *Fix $N, M, B, S, T, u \in \mathbb{N}_{>0}$, $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2) \leqslant \left(2^9 e^4 \max\left\{\left(\frac{u^4 T B^2}{N}\right), \left(\frac{T^2}{N}\right)\right\}\right)^u .$$

The proof of this lemma is in the same spirit as the proof of Lemma 5.1 with several key differences. The main difference is how we encode the path back from a given destination node to the associated source node. In Section 5, we do this in a somewhat straightforward manner by encoding the length of the path and then the index of every edge to take, where the index might be large if there is a large multi-collision associated with that node. Large-enough multi-collisions were handled separately, so we had a bound ($m_0 \approx \log u / \log\log u$) for the range of the index of each back-edge. In this section, we encode the source node by just writing its lexicographic index among all possible sources within a given distance in the query graph. Of course, there might be too many possible sources at a given distance, making it too expensive to encode. But, and this is our main technical observation in this section, if there are too many possible sources, then there must be many large multi-collision, and therefore we can save enough bits by taking advantage of it.

We proceed by setting up the graph-theoretic definitions and lemmas. In the lemma below, we show that if in the query graph there is a node v such that there are at least p nodes which have a shortest path of length d to the node v, then there must be at least $p-1$ edges involved in a multi-collision in the induced sub-graph.

Definition 5 (d-neighborhood of a vertex). *Let $G = (V, E)$ be a directed graph. We say that a node $v_1 \in V$ is in the d-neighborhood of $v_2 \in V$ if the shortest directed path from v_1 to v_2 in G consists of (exactly) d edges.*

Definition 6 (multi-collision of edges in a graph). *For an edge $e = (a, b)$, we refer to b as the target of the edge. We say that edges $e_1, \cdots, e_m$ form a m-multi-collision if all of them share a common target. We refer to the common target as the center of the multi-collision. We refer to m as the size of the multi-collision.*

Lemma 6.2. *Let $G = (V, E)$ be a directed graph. Let $d \in \mathbb{N}_{>0}$ and $p \in \mathbb{N}_{>0}$ such that $p \geqslant 2$. Suppose that there is a node $v \in V$ such that there are p distinct nodes in its d-neighborhood. Then, there are $t \geqslant 1$ distinct nodes in G, such that each of them have in-degree $\beta_i \geqslant 2$ for $i = 1, \ldots, t$, and $\sum_{i=1}^{t}(\beta_i - 1) \geqslant p - 1$.*

The proof of this lemma is via an inductive argument and appears in the full version [19].

A direct corollary is that in the query graph, if for any salt s there are at least $p+1$ salts in its d-neighborhood, then there must be p queries involved in multi-collisions on the paths from the nodes in its d-neighborhood to s. We obtain non-trivial compression for large enough p by encoding those multi-collisions.

Non-trivial encoding for multiple multi-collisions. We consider t multi-collisions of sizes $\beta_1, \ldots, \beta_t$. We encode these t multi-collisions by encoding the t centers of the multi-collision, and for each center, we encode the index of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ that form the multi-collision in a set. Recall that the indices of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ are in $[FT]$. The total number bits we need to encode is

$$\log\left(\binom{N}{t}\binom{FT}{\beta_1}\cdots\binom{FT - \sum_{i=1}^{t-1}\beta_i}{\beta_t}\right) \leqslant \log\left(\frac{N^t (uT)^{\sum_{i=1}^{t}\beta_i}}{t!\beta_1!\beta_2!\cdots\beta_t!}\right)$$

$$= \log\left(\frac{N^t (u^4T)^{\beta-2t}T^{2t}u^{8t-3\beta}}{t!\beta_1!\beta_2!\cdots\beta_t!}\right), \quad (6)$$

where the inequality follows by using $\binom{n}{r} \leqslant \frac{n^r}{r!}$ and $F \leqslant u$, and the equality follows by letting $\beta = \sum_{i=1}^{t}\beta_i$ and rearranging.

Claim 6.2 *If $\beta \geqslant e^3u^2/2$, then Eq. (6) $\leqslant \log\left(N^t(u^4T)^{\beta-2t}T^{2t}\right)$.*

We defer the proof of this claim to the full version [19]. From this claim it follows that when $\beta \geqslant e^3u^2/2$, by storing the multi-collisions as above, the amount of bits saved is at least

$$\log N^\beta - \log\left(N^t(u^4T)^{\beta-2t}T^{2t}\right) = t\log\left(\frac{N}{T^2}\right) + (\beta - 2t)\log\left(\frac{N}{u^4T}\right)$$

$$\geqslant (\beta - t)\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4T}\right\}\right) \geqslant u\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4T}\right\}\right),$$

where the second inequality follows since $t \leqslant \beta/2$, and $\beta/2 \geqslant e^3u^2/4 \geqslant u^2 \geqslant u$.

For the bound that we want to get in this section, the amount of bits we need to save *per mouse structure* is $\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4TB^2}\right\}\right)$ (see explanation below), and so *we indeed save enough from encoding one such set of multi-collisions.*

The compression argument. We encode a source node from a destination node by encoding the distance and which of the nodes at the given distance from the destination node is the source node. If the number of candidate nodes at the specified distance is larger than $e^3u^2/2$, by Lemma 6.2, we are guaranteed that there exist t multi-collisions of size $\beta_1, \beta_2, \ldots, \beta_t$ such that $\beta \geqslant \sum_{i=1}^{t}(\beta_i - 1) \geqslant e^3u^2/2$. This implies, by Claim 6.2, that we can already save enough by only encoding this set of multi-collisions. Using this argument we prove Lemma 6.1 which in turn implies Theorem 6.1, as explained above. We defer the proof of Lemma 6.1 to the full version [19].

Acknowledgements

Ilan Komargodski is supported in part by an Alon Young Faculty Fellowship, by a JPM Faculty Research Award, 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). Part of Ashrujit Ghoshal's work was done during an internship at NTT Research.

References

1. Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.: Beyond hellman's time-memory trade-offs with applications to proofs of space. In: Advances in Cryptology - ASIACRYPT. pp. 357–379 (2017) 2
2. Adleman, L.: Two theorems on random polynomial time. In: Symposium on Foundations of Computer Science, SFCS. pp. 75–83 (1978) 15
3. Akshima, Cash, D., Drucker, A., Wee, H.: Time-space tradeoffs and short collisions in merkle-damgård hash functions. In: Advances in Cryptology - CRYPTO. pp. 157–186 (2020) 2, 3, 4, 5, 7, 8, 12, 15, 16
4. Akshima, Guo, S., Liu, Q.: Time-space lower bounds for finding collisions in Merkle-Damgård hash functions. To appear in CRYPTO 2022 (2022) 6
5. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: Advances in Cryptology - CRYPTO. pp. 1–21 (2006) 2
6. Chawin, D., Haitner, I., Mazor, N.: Lower bounds on the time/memory tradeoff of function inversion. In: Theory of Cryptography - TCC. pp. 305–334 (2020) 2
7. Chung, K., Guo, S., Liu, Q., Qian, L.: Tight quantum time-space tradeoffs for function inversion. In: FOCS. pp. 673–684 (2020) 7
8. Coretti, S., Dodis, Y., Guo, S.: Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In: Advances in Cryptology - CRYPTO. pp. 693–721 (2018) 2
9. Coretti, S., Dodis, Y., Guo, S., Steinberger, J.P.: Random oracles and non-uniformity. In: Advances in Cryptology - EUROCRYPT. pp. 227–258 (2018) 2, 3
10. Corrigan-Gibbs, H., Kogan, D.: The discrete-logarithm problem with preprocessing. In: Advances in Cryptology - EUROCRYPT. pp. 415–447 (2018) 7
11. Corrigan-Gibbs, H., Kogan, D.: The function-inversion problem: Barriers and opportunities. In: Theory of Cryptography - TCC. pp. 393–421 (2019) 2

12. Damgård, I.: Collision free hash functions and public key signature schemes. In: Advances in Cryptology - EUROCRYPT. pp. 203–216 (1987) 2
13. De, A., Trevisan, L., Tulsiani, M.: Time space tradeoffs for attacks against one-way functions and prgs. In: Advances in Cryptology - CRYPTO. pp. 649–665 (2010) 2, 7, 14
14. Dinur, I.: Tight time-space lower bounds for finding multiple collision pairs and their applications. In: Advances in Cryptology - EUROCRYPT. pp. 405–434 (2020) 12
15. Dodis, Y., Guo, S., Katz, J.: Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In: Advances in Cryptology - EUROCRYPT. pp. 473–495 (2017) 2, 3, 7
16. Fiat, A., Naor, M.: Rigorous time/space trade-offs for inverting functions. SIAM J. Comput. **29**(3), 790–803 (1999) 2
17. Freitag, C., Ghoshal, A., Komargodski, I.: Time-space tradeoffs for sponge hashing: Attacks and limitations for short collisions. In: Advances in Cryptology - CRYPTO (2022) 6
18. Gennaro, R., Trevisan, L.: Lower bounds on the efficiency of generic cryptographic constructions. In: FOCS. pp. 305–313 (2000) 7
19. Ghoshal, A., Komargodski, I.: On time-space tradeoffs for bounded-length collisions in Merkle-Damgård hashing. Cryptology ePrint Archive, Paper 2022/309 (2022) 15, 22, 26, 28, 29
20. Ghoshal, A., Tessaro, S.: On the memory-tightness of hashed elgamal. In: Advances in Cryptology - EUROCRYPT. pp. 33–62 (2020) 7
21. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theory **26**(4), 401–406 (1980) 1, 3
22. Impagliazzo, R., Kabanets, V.: Constructive proofs of concentration bounds. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 617–631. Springer (2010) 8
23. Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: Advances in Cryptology - CRYPTO. pp. 306–316 (2004) 12
24. Merkle, R.C.: Secrecy, Authentication and Public Key Systems. Ph.D. thesis, UMI Research Press, Ann Arbor, Michigan (1982) 2
25. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology - CRYPTO. pp. 369–378 (1987) 2
26. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology - CRYPTO. pp. 218–238 (1989) 2
27. Moser, R.A., Tardos, G.: A constructive proof of the general lovász local lemma. J. ACM **57**(2), 11:1–11:15 (2010) 7
28. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Advances in Cryptology - CRYPTO. pp. 617–630 (2003) 2, 3
29. Sr., R.H.M., Thompson, K.: Password security - A case history. Commun. ACM **22**(11), 594–597 (1979) 6
30. Unruh, D.: Random oracles and auxiliary input. In: Advances in Cryptology - CRYPTO. pp. 205–223 (2007) 2, 3, 7, 13
31. Yao, A.C.: Coherent functions and program checkers (extended abstract). In: STOC. pp. 84–94 (1990) 2