Time-Space Tradeoffs for Sponge Hashing: Attacks and Limitations for Short Collisions *

Cody Freitag[1], Ashrujit Ghoshal[2], and Ilan Komargodski[3]

[1] Cornell Tech, New York, USA
cfreitag@cs.cornell.edu
[2] Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, Washington, USA
ashrujit@cs.washington.edu
[3]School of Computer Science and Engineering, Hebrew University of Jerusalem
91904 Jerusalem, Israel and NTT Research
ilank@cs.huji.ac.il

Abstract

Sponge hashing is a novel alternative to the popular Merkle-Damgård hashing design. The sponge construction has become increasingly popular in various applications, perhaps most notably, it underlies the SHA-3 hashing standard. Sponge hashing is parametrized by two numbers, r and c (bitrate and capacity, respectively), and by a fixed-size permutation on $r+c$ bits. In this work, we study the collision resistance of sponge hashing instantiated with a random permutation by adversaries with arbitrary S-bit auxiliary advice input about the random permutation that make T online queries. Recent work by Coretti et al. (CRYPTO '18) showed that such adversaries can find collisions (with respect to a random c-bit initialization vector) with advantage $\Theta(ST^2/2^c + T^2/2^r)$.

Although the above attack formally breaks collision resistance in some range of parameters, its practical relevance is limited since the resulting collision is very long (on the order of T blocks). Focusing on the task of finding *short* collisions, we study the complexity of finding a B-block collision for a given parameter $B \geq 1$. We give several new attacks and limitations. Most notably, we give a new attack that results in a single-block collision and has advantage

$$\Omega\left(\left(\frac{S^2T}{2^{2c}}\right)^{2/3} + \frac{T^2}{2^r}\right).$$

In certain range of parameters (e.g., $ST^2 > 2^c$), our attack outperforms the previously-known best attack. To the best of our knowledge, this is the first natural application for which sponge hashing is *provably less secure* than the corresponding instance of Merkle-Damgård hashing. Our attack relies on a novel connection between single-block collision finding in sponge hashing and the well-studied function inversion problem. We also give a general attack that works for any $B \geq 2$ and has advantage $\Omega(STB/2^c + T^2/2^{\min\{r,c\}})$, adapting an idea of Akshima et al. (CRYPTO '20).

We complement the above attacks with bounds on the best possible attacks. Specifically, we prove that there is a qualitative jump in the advantage of best possible attacks for finding unbounded-length collisions and those for finding very short collisions. Most notably, we prove (via a highly non-trivial compression argument) that the above attack is optimal for $B = 2$ in some range of parameters.

*A preliminary version of this paper appears in the proceedings of CRYPTO 2022. This is the full version.

Contents

1 Introduction **3**
- 1.1 Detour: The Case of Merkle-Damgård . . . 5
- 1.2 Our Results . . . 5
- 1.3 Future Directions . . . 7
- 1.4 Related Work . . . 8

2 Technical Overview **8**
- 2.1 Attacks . . . 9
- 2.2 Impossibility Results for Best Attacks . . . 11

3 Preliminaries **13**

4 Attacks **15**
- 4.1 Generic Attack for B-Block Collisions . . . 15
- 4.2 Preprocessing Attack for $B = 1$ . . . 18
- 4.3 Time-Space Tradeoffs for Inverting a Restricted Permutation . . . 21

5 Impossibility Results **28**
- 5.1 Advantage Upper Bound for B=1 . . . 28
- 5.2 Advantage Upper Bound for $B = 2$ . . . 29

References **38**

Appendices **40**

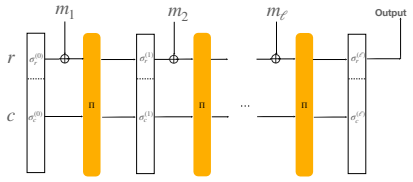
A Encoding and decoding procedure pseudocodes for Theorem 5.5. **40**

1 Introduction

Due to a series of successful attacks on widely used hash functions such as MD5, SHA-0, and SHA-1, in 2006 the National Institute of Standards and Technology (NIST) organized a competition to create a new hash standard. At that time, the existing hash functions were all based on the well-known Merkle-Damgård hash function construction [Mer82, Mer87, Mer89, Dam87]. The goal of the competition was to find an alternative, dissimilar cryptographic hashing design. It took almost a decade until the winner, a family of cryptographic functions called Keccak, become a hashing standard called SHA-3. The Keccak family is based on the *sponge construction* [BDPVA07, BDPA08] which was a novel alternative to the popular Merkle-Damgård design. By now, the sponge paradigm is used for building collision resistant hash functions, message authentication codes (MACs), pseudorandom functions (PRFs) [BDPVA11], key derivation functions [GT16], and more.

A sponge function $\mathsf{Sp}\colon \{0,1\}^* \to \{0,1\}^r$ is defined via three parameters: (1) two natural numbers r (for bitrate) and c (for capacity) so that $n = c + r$, (2) an initial state $\sigma^{(0)} = (\sigma_r^{(0)}, \sigma_c^{(0)}) \in \{0,1\}^r \times \{0,1\}^c$, and (3) a function $\Pi : \{0,1\}^n \to \{0,1\}^n$ which is usually thought of as a (public) pseudorandom permutation. The hashing operation (a.k.a. absorbing) is defined by iterating the state by computing a *round function*. Specifically, given a sequence of r-bit blocks $(m_1, m_2, \ldots, m_\ell)$, $\mathsf{Sp}(m_1, m_2, \ldots, m_\ell)$ is defined as:[1]

1. For $i = 1, \ldots, \ell$, do:
   (a) Compute the round function $\Pi((\sigma_r^{(i-1)} \oplus m_i) \parallel \sigma_c^{(i-1)})$ and let $\sigma^{(i)}$ denote the output.
   (b) Parse $\sigma^{(i)}$ as $(\sigma_r^{(i)}, \sigma_c^{(i)}) \in \{0,1\}^r \times \{0,1\}^c$.
2. Output the first r bits of $\sigma^{(\ell)}$, namely, $\sigma_r^{(\ell)}$.



Typically, $\sigma_r^{(0)}$ is initialized to 0 and $\sigma_c^{(0)}$ is a random initialization vector (IV). If one wants to be explicit, we write $\mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}$ for the sponge function. There are several common instances of r and c used in practice, for example in SHA-3-256 $c = 512$ and $r = 1088$, and in SHA-3-512 $c = 1024$ and $r = 576$. These instance are particularly useful since they were designed to be used as drop-in replacements for the corresponding SHA-2 instances, and as such they were intended to have identical (or better) security properties.

Sponge in the random permutation model. The concrete permutations Π that are used in real-life do not have solid theoretical foundations from the perspective of provable security. Therefore, when coming to analyze the security of the sponge construction, we model the permutation Π as a completely random one. That is, the permutation is randomly chosen, and all parties are given (black-box) access to it and its inverse.[2] This is called the *random permutation model* (RPM). Such bounds are used as an approximation to the best possible security level that can be achieved by the corresponding construction in the real-life implementation. This heuristic has been extensively and successfully used in the past several decades, with exceptions (i.e., examples where the real-life implementation and the ideal world construction are separated) being somewhat contrived and artificial. For "natural" applications it is widely believed that the concrete security proven in the RPM is the right bound even in the real-world, assuming the "best possible" instantiation for the idealized permutation is chosen.

As mentioned, the sponge construction was introduced by Bertoni et al. [BDPVA07] and its security was analyzed in a follow-up work [BDPA08] assuming that the underlying hash function is an invertible random permutation. The latter work showed a strong property called *indifferentiability* from a random oracle, which directly implies many other properties such as collision resistance, pseudorandomness, and more.

For instance, the following is known about Sponge's collision resistance (which is perhaps the most widely used property). For fixed c, r, the collision resistance game is defined as follows: a challenger sends a uniformly random IV to the adversary. The adversary "wins" if it is able to come up with distinct $m, m' \in \{0,1\}^*$

[1] For simplicity, we do not consider padding of the input.

[2] In typical permutation designs, including the permutations underlying the Keccak family, if you have the entire state, you can apply the inverse permutation to go backward to the previous state. This is why we also give free access to the inverse of the permutation as part of the model.

for which $\mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}(m) = \mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}(m')$. There is a well-known attack due to the original works of Bertoni et al. [BDPVA07, BDPA08]: the adversary is given an IV and it merely queries the permutation oracle on inputs of the form $(m\|\text{IV})$, where the m's are chosen uniformly at random. If a collision was observed (i.e., the adversary finds distinct m_1, m_2 such that the first r bits of $\Pi(m_1\|\text{IV})$, $\Pi(m_2\|\text{IV})$ are the same), then the adversary wins. By the well-known birthday bound, the success probability of this event is $\Omega(T^2/2^r)$. Alternatively, if two messages m_1, m_2 such that the query returned a state with the same last c bits (i.e., $\Pi(m_1\|\text{IV}) = a_1\|b$ and $\Pi(m_2\|\text{IV}) = a_2\|b$, then $m_1 \,\|\, a_1$ and $m_2 \,\|\, a_2$ form a collision. The success probability of this event is $\Omega(T^2/2^c)$. Overall, the attacker wins with probability $\Omega(T^2/2^{\min\{c,r\}})$. This is known to be the best possible attack due to the indifferentiability result of [BDPA08].

Non-uniformity / preprocessing attacks. The above discussion assumes that the adversary is uniform in the sense that it starts off with no knowledge about Π, as if it did not exist before it was invoked. However, this does not capture real-life attack scenarios where an attacker can invest a significant amount of preprocessing on the public permutation Π to speed up the actual attack whenever the IV is chosen. This is why most works (at least in theoretical cryptography) model attackers as non-uniform machines, where the attacker could obtain arbitrary but bounded-length advice, before attacking the system. The advice generation phase is called *the offline phase* and the "attack" given the advice and the challenge is called *the online phase*. The output size of the offline phase (i.e., the size of the advice) is denoted S and the number of queries allowed in the online phase is denoted T; computation is free of charge in both phases. This model, being an extension of the RPM where the online adversary may know a bounded-length hint about the permutation, is called the auxiliary-input RPM, or AI-RPM in short. This model was first explicitly put forward by Coretti, Dodis, and Guo [CDG18], naturally extending the influential auxiliary-input random oracle model (AI-ROM) from the seminal work of Unruh [Unr07] (which in turn is an explicit version of the model studied by Hellman [Hel80], Yao [Yao90], and Fiat-Naor [FN99]). Bounds on the power of "auxiliary-input" adversaries are also referred to as "time-space" trade-offs.

Although the sponge paradigm is becoming widespread, very little is known about its formal security guarantees against such attackers that may have a short preprocessed hint about the permutation computed in an offline phase. In fact, there is an attack that utilizes this extra power to achieve advantage $\Omega(ST^2/2^c + T^2/2^r)$ (notice the extra multiplicative S term).[3] The attack is based on a combination of a birthday-style attack, as above, together with a variant of an attack by Hellman [Hel80] which is nowadays referred to as rainbow tables (due to Oechslin [Oec03]). While this attack uses known techniques, we were not able to find an explicit description of it in the literature and so for completeness, we give the attack and its analysis in Section 4.1.[4] [5] Only very recently, in the beautiful work of Coretti et al. [CDG18] (henceforth CDG) it was shown that this attack is optimal; that is, no S-space T-query attackers can find a collision with probability better than $\Omega(ST^2/2^c + T^2/2^r)$.

It turns out that the above attack results in a very long collision. Specifically, for parameters S and T as above, the above attack results in a collision of length $\approx T$. While this formally breaks collision resistance, it is hard to imagine a natural application where such a collision would be helpful in an attack. Say we have a system that uses a sponge-based hash with an output of size 256 bits. Running the above attack with $S = T = 2^{60}$ would result in a collision of several petabytes long, which is likely to be practically useless for any natural attack scenario. Therefore, we ask whether there exist attacks that find shorter collisions and what is their success probability. Specifically, we introduce an additional parameter B (for blocks) and require an attacker, on a random IV, to come up with two $\leq B$-block messages that collide. The main question studied in this work is:

What is the complexity of a preprocessing attacker in finding a B block collision in a Sponge hash function, assuming the underlying permutation is modeled as random?

[3] Throughout the introduction, for easy of notation, we supress poly-logarithmic (i.e., $\mathsf{poly}(c, r)$ terms inside the big "O/Ω" notation. The formal theorems state the precise bounds.

[4] More precisely, we give a generalization of this attack which finds collisions of length $B \geq 2$, and this particular attack follows by setting $B = T$.

[5] A related bound is stated in CDG [CDG18, Table 1] but after communication with an author, they confirmed that the attack was never worked out.

1.1 Detour: The Case of Merkle-Damgård

Except being a fundamental problem with theoretical and practical importance, another motivation to study the above question comes from the recent work of Akshima et al. [ACDW20] (henceforth ACDW), who studied a similar question in the context of Merkle-Damgård hashing (henceforth MD). Recall that sponge hashing was designed to be used as a drop-in replacement for Merkle-Damgård-based hash functions, and as such, it is essential to compare their security guarantees in this natural model that allows attackers to perform preprocessing.

Recall that a Merkle-Damgård hash is defined relative to a *compression* function $h\colon [N] \times [M] \to [N]$. Hashing is performed by breaking the input message into blocks from $[M]$, and processing them one at a time with the compression function, each time combining a block of the input with the output of the previous round, where the 0th round value is the IV.[6] To obtain provable-security guarantees, the analysis models the underlying compression function h as a completely random one. Preprocessing attackers are captured by considering the AI-ROM [Hel80, Yao90, FN99, Unr07, CDG18, CDGS18] which models attackers as two-stage algorithms $(\mathcal{A}_1, \mathcal{A}_2)$. The first algorithm $\mathcal{A}_1$ is unbounded except that it generates an S-bit "advice". The second algorithm $\mathcal{A}_2$ gets the advice and makes T queries to the oracle.

Coretti et al. [CDGS18] fully characterize the collision resistance of salted-MD hashing: there exists an attack with advantage $\Omega(ST^2/N + T^2/N)$ (loosely based on the idea of rainbow tables [Hel80, Oec03]), and this is the best possible attack, as shown using the "bit-fixing" technique [Unr07]. As in the case of sponge hashing, this attack results in a very long collision, on the order of T blocks. Motivated by this observation, ACDW [ACDW20] ask whether it is strictly harder to find shorter collisions. They have two main results. The first is an extension of the above simple attack to result in B-block collisions with advantage $\Omega(STB/N + T^2/N)$. The second result is an upper bound on the advantage for $B = 2$ of $O(ST/N + T^2/N)$, showing that the above attack is tight. For $B = 1$, the problem is equivalent to finding collisions in a compressing random function, and the advantage is precisely $\Theta(S/N + T^2/N)$ [DGK17].

ACDW [ACDW20] could not prove or disprove that their $\Omega(STB/N + T^2/N)$ attack is optimal for any other value of B (except $B = 2$ and $B \in \Omega(T)$). They conjectured that it is optimal and formulated it as *the STB conjecture*. In very recent works, Akshima, Guo, and Liu [AGL22] and Ghoshal and Komargodski [GK22] proved new bounds for this problem (almost resolving the conjecture). Can we prove similar bounds for sponge hashing? Should we believe an analogous conjecture?

1.2 Our Results

We initiate the study of time-space tradeoffs for *bounded length collisions in sponge hashing*. First, the known best attack that gives a single-block collision has advantage

$$\Omega\left(\frac{S}{2^c} + \frac{T^2}{2^r}\right). \tag{1}$$

In this attack, the preprocessing is used to "remember" a collision for S different IVs. If the challenge IV is in the set of remembered IVs, then the attack succeeds (this happens with probability $S/2^c$); otherwise, we run a birthday-style attack which succeeds with probability $\Omega(T^2/2^r)$. For MD hashing, the analogous bound for $B = 1$ is known to be tight. Second, there is an attack (loosely based on rainbow tables) that has advantage $\Omega(ST^2/2^c + T^2/2^r)$ and results with a $\Omega(T)$-blocks collision [CDG18].

At this point, if one were to speculate that sponge's security guarantees are at least as good as MD's, one would guess that the above attacks should be tight, at least for $B \in \{1, 2\}$. With some luck and labor, we may even be able to prove it. This is where the situation gets interesting. We show that the above speculation is **false** for $B = 1$ and in some natural settings of parameters, **sponge is strictly less secure than MD** for this task. On the other hand, for $B = 2$ we can only prove tightness for a certain range of parameters.

[6] All of the results directly extend to the padded version, but we ignore it for simplicity.

In what follows, we elaborate on our results. We design two new attacks, one designed for any $B \geq 2$ and the other specifically for $B = 1$. We complement our attacks with "lower bounds", which are actually upper bounds on the best possible advantage. Specifically, we prove that there is a qualitative jump in the advantage of best possible attacks for finding unbounded-length collisions and those for finding very short collisions (i.e., $B \leq 2$).

Attacks

We give two new attacks, one for any $B \geq 2$ and the other is specialized for $B = 1$. The generic attack is the first to result with an arbitrary block length collision while the one specialized to $B = 1$ beats the previously known best attack, at least in some range of parameters. By the latter, to the best of our knowledge, we show the first natural application for which sponge hashing is less secure than MD.

A new attack for $B \geq 2$. The above-mentioned attack on sponge hashing that has advantage $\Omega(ST^2/2^c + T^2/2^r)$ can be modified to result with a B-block collision for $B \geq 2$ and with advantage

$$\Omega\left(\frac{STB}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}\right). \tag{2}$$

The attack follows a similar observation of ACDW [ACDW20] regarding MD hashing. Given the upper bound of CDG [CDG18] mentioned earlier, this attack is optimal for $B \in \Omega(T)$. For MD hashing, the analogous bound is known to be tight for $B = 2$ and $B \in \Omega(T)$.

A new attack for $B = 1$. We design a new attack for sponge hashing that results with a a single-block collision. Specifically, we show that if $ST^2 > 2^c$, then there is an attack with advantage

$$\Omega\left(\left(\frac{S^2T}{2^{2c}}\right)^{2/3} + \frac{T^2}{2^r}\right).$$

To see why this attack is superior to the previously known one (Eq. 1), we give a setting of parameters where it achieves a significantly higher advantage. Consider $r = c$, $S = 2^{4c/5}$, and $T = 2^{2c/5}$. Indeed, $ST^2 > 2^c$ and therefore we can apply the attack. The previously known best attack (Eq. 1) has advantage

$$\Omega\left(\frac{S + T^2}{2^c}\right) = \Omega\left(\frac{1}{2^{c/5}}\right).$$

This attack is the analog of the *provably* best attack for MD. On the other hand, our new attack has strictly better advantage

$$\Omega\left(\left(\frac{S^2T}{2^{2c}}\right)^{2/3} + \frac{T^2}{2^c}\right) = \Omega\left(\left(\frac{2^{8c/5}2^{2c/5}}{2^{2c}}\right)^{2/3} + \frac{1}{2^{c/5}}\right) = \Omega\left(1\right).$$

Thus, at least in this range of parameters, we beat the state-of-the-art attack and show that sponge is *less secure than MD*. In the example above, we chose a setting of parameters where the gap between the attacks is the largest (our attack succeeds with *constant* probability, while the previously known one succeeds with exponentially small probability). However, there are many more concrete settings where our attack is superior, although the gap could be less dramatic. We note that our bounds in this section and the technical overview are simplified for ease of parsing and refer the reader to the technical sections for the exact bounds.

Conceptual novelty: Our attack for $B = 1$ use the famous time-space tradeoffs for function inversion of Hellman [Hel80] and its extension by Fiat-Naor [FN99]. We leverage the possibility of inverse queries to the underlying permutation Π in the random-permutation model. This is in contrast to Merkle-Damgård construction which is analyzed in the random-oracle model that does not permit inverse queries. At a very high level, we use time-space tradeoffs for function inversion to "invert" the function Π^{-1} on a restricted domain. We view this conceptual connection between time-space tradeoffs for collision resistance of sponge hashing and function inversion as novel and hope that it will lead to better designs and additional attacks in the future.

	Best Attack	Advantage Upper Bound
$B = 1$	$\min\left(\frac{S^2T^2}{2^{2c}}, \left(\frac{S^2T}{2^{2c}}\right)^{2/3}\right) + \frac{S}{2^c} + \frac{T^2}{2^r}$ [Thm 4.2]	$\frac{ST}{2^c} + \frac{T^2}{2^r}$ [Thm 5.2]
$B = 2$	$\frac{ST}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}$ [Thm 4.1]	$\frac{ST}{2^c} + \frac{S^2T^4}{2^{2c}} + \frac{T^2}{2^{\min(c,r)}}$ [Thm 5.5]
$B \geq 3$	$\frac{STB}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}$ [Thm 4.1]	$\frac{ST^2}{2^c} + \frac{T^2}{2^r}$ [CDG18]

Figure 1: A summary of the attacks and advantage upper bounds for finding B-block collisions for the Sponge hash function. All bounds are given ignoring $\mathrm{poly}(c, r)$ terms. We note that the attack for $B = T$ is implicitly claimed in [CDG18] based on [CDGS18].

Lower Bounds

We complement the picture by showing "lower bounds", namely impossibility results for better attacks. (In other words, these are upper bounds on the best possible advantage of any attacker.) We prove two such lower bounds, one for the case where $B = 1$ and the other is for $B = 2$, corresponding to our attacks.

On optimal attacks for $B = 2$. We show that any attack for $B = 2$ must have advantage

$$O\left(\frac{ST}{2^c} + \frac{S^2T^4}{2^{2c}} + \frac{T^2}{2^{\min\{c,r\}}}\right).$$

We note that this bound is tight with the best known attacks for a large range of parameters, but there still may be a gap otherwise. Specifically, if $ST^3 \leq 2^c$, then the above bound simplifies to $O(ST/2^c + T^2/2^{\min\{c,r\}})$ which matches the attack from Eq. (2). Thus, any improvement on the generic attack from Eq. (2) must take advantage of the regime where $ST^3 > 2^c$.

The proof of this result *provably* cannot be obtained via the bit-fixing method. Rather, we obtain the result via a compression argument. In such arguments, an imaginary adversary that is successful too often is used to compress a uniformly random string, a task which is (information-theoretically) impossible. The compression technique has been instrumental in proving lower bounds in computer science (see the survey of Morin et al. [MMR17]). It has become useful in the context of cryptographic constructions and primitives, starting with the work of Gennaro and Trevisan [GT00]. Unfortunately, one common "feature" of such proofs is that they tend to be extremely technical and involved. Our proof is no different; in fact, it is even much more complicated than the analogous result for $B = 2$ of ACDW [ACDW20] since we work in the RPM and need to handle inverse queries.

On optimal attacks for $B = 1$. We show that any attack for $B = 1$ must have an advantage

$$O\left(\frac{ST}{2^c} + \frac{T^2}{2^r}\right).$$

The proof of this result is relatively straightforward by using an optimized version of the remarkable bit-fixing (or presampling) method [Unr07, CDGS18, CDG18]. The main point of distinction of our proof from most previous ones is that we need to apply this technique in the RPM context, so our argument needs to handle inverse queries. (This result might have been known before, but we could not find such a statement, so we give it for completeness.)

We summarize our main results as well as the known best bounds in Fig. 1.

1.3 Future Directions

Our work is the first to address the question of characterizing the complexity of a preprocessing attacker in finding a B-block collision in a Sponge hash function. Our results raise many natural open problems on

both the attacks side and lower bounds side. Regarding attacks, we have shown, somewhat surprisingly, that there is a non-trivial attack for $B = 1$ that takes advantage of inverse queries in a novel way. We hope that these ideas can be pushed forward to obtain even better attacks for $B = 1$ or beyond. Specifically, is it possible to beat the $ST/2^c$ attack for $B = 2$ in some range of parameters? In ruling out possible attacks, it would be interesting to come up with a tight upper bound on the advantage for $B = 1$ or $B = 2$. Our work suggests that ruling out attacks that use inverse queries may indeed be a complicated task. In fact, for $B = 3$ we are not aware of any upper bound on the advantage that is better than $O(ST^2/2^c + T^2/2^r)$.

1.4 Related Work

Time-space tradeoffs are fundamental to the existence of efficient algorithms. For example, look-up tables (used to avoid "online" recalculations) have been implemented since the very earliest operating systems. In cryptography (or cryptanalysis), they were first used by Hellman [Hel80] in the context of inverting random functions. Hellman's algorithm was subsequently rigorously analyzed by Fiat and Naor [FN99] where it was also extended to handle *arbitrary* (not necessarily random) functions. Limitations of such algorithms were studied by Yao [Yao90], and by De, Trevisan, and Tulsiani [DTT10] (building on works by Gennaro and Trevisan [GT00] and Wee [Wee05]). More limitations were proven by Barkan, Biham, and Shamir [BBS06] but for a restricted class of algorithms. Very recently, Corrigan-Gibbs and Kogan [CK19] showed complexity-theoretic limitations for improving the lower bound of Yao. While these techniques have mostly cryptographic origins, interesting relations were discovered to other classical problems in other fields (e.g., [AAC+17, GGH+20]). Time-space tradeoffs have been studied for other problems beyond the ones we mentioned (various cryptographic properties of random oracles, function and permutation inversion, and security of common hashing paradigms). For instance, specific modes for block ciphers (e.g., [FJM14] studied the Even-Mansour cipher), and various assumptions related to cyclic groups, such as discrete logarithms and Diffie-Hellman problems [Mih10, BL13, CGK18, CDG18].

On the salt. In the theoretical cryptography literature collision resistance is defined with respect to a family of hash functions indexed by a key. This is important to achieve the standard notion of *non-uniform* security. Indeed, no single hash function can be collision-resistant as a non-uniform attacker can just hardwire a collision. In practice, however, a single hash function is considered by fixing an IV. Thus, the relevance of our model could be questioned. However, often in applications, the hash function used is salted by prepending a random salt value to the input, for example in password hashing [ST79]. Salting essentially brings us back to the random-IV/keyed setting, where our results become relevant.

2 Technical Overview

In this section, we provide a high-level overview of our techniques. We first describe the generic attack for finding B-block collisions for $B \geq 2$. This attack is a variant of an analogous attack for MD, given by ACDW [ACDW20]. We also recall the known best attack for $B = 1$. Then, we describe our new attack for finding 1-block collisions. In particular, our attack outperforms the optimal analogous attacks for MD for specific regimes of parameter settings. Lastly, we overview the techniques used to prove limitations on the best possible attacks for finding short collisions.

Sponge notation. A sponge function is a keyed hash function that takes as input an a c-bit initialization vector IV along with an arbitrary size input and outputs an r-bit string: $\mathsf{Sp}\colon \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^r$. The second input is parsed as a sequence of r-bit blocks, denoted $(m_1, m_2, \ldots)$. On such an input $\mathsf{Sp}(\mathsf{IV}, (m_1, m_2, \ldots))$ is defined as follows. The function Sp is defined relative to a permutation $\Pi\colon \{0,1\}^{r+c} \to \{0,1\}^{r+c}$. An input to or an output of this permutation, denoted $\sigma \in \{0,1\}^{r+c}$, contains an r-bit block, denoted $\sigma[1]$, and a c-bit block, denoted $\sigma[2]$. We sometimes use $(\sigma[1], \sigma[2])$ to mean $\sigma[1] \| \sigma[2] = \sigma$.

On input $m_1, m_2, \ldots, m_\ell$ to Sp, it works as follows:

1. Initialize $\sigma^{(0)} = (\sigma^{(0)}[1], \sigma^{(0)}[2]) = (0, \mathsf{IV})$.

2. For $i = 1, \ldots, \ell$, compute $\sigma^{(i)} = \Pi((\sigma^{(i-1)}[1] \oplus m_i) \parallel \sigma^{(i-1)}[2])$.

3. Output $\sigma^{(\ell)}[1]$.

2.1 Attacks

Generic attack for finding length B collisions. In the preprocessing phase the adversary randomly samples $t \approx S$ different IVs $\mathsf{IV}_1, \ldots, \mathsf{IV}_t$ and for $i = 1, \ldots, t$ it does as follows.

1. Compute $\sigma_{i,j}$ for $j \in [B/2 - 1]$ as $\sigma_{i,j} = \Pi(0, \sigma_{i,j-1}[2])$, where $\sigma_{i,0} = (0, \mathsf{IV})$.
   The sequence $\sigma_{i,0}, \ldots, \sigma_{i,B/2-1}$ forms a "zero-walk" on IV_i.

2. Find m_i, m_i' such that $\Pi(m_i, \sigma_{i,B/2-1}[2])[1] = \Pi(m_i', \sigma_{i,B/2-1}[2])[1]$.

The preprocessing phase outputs $(\sigma_{i,B/2-1}[2], m_i, m_i')_{i=1,\ldots,t}$. In Fig. 2, we depict the preprocessing phase of the attack. In the online phase, the adversary gets a challenge IV as input. For $i = 1, \ldots, T/B$, it computes

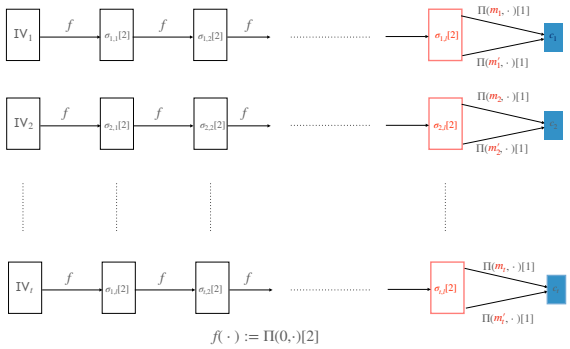


Figure 2: An illustration of the preprocessing phase of the generic attack. In red, we depict the components that are part of the output of the preprocessing phase. In blue we see the collisions that will be outputted in the online phase if some chain is hit. Notice that we denote $f(\cdot) := \Pi(0, \cdot)[2]$.

$\mathsf{IV}_i = \Pi(i, \mathsf{IV})[2]$ (for simplicity, we assume that i is in its bit representation). For each of the IV_i's, it does a zero-walk of length $B - 2$. Formally, it sets $\sigma_{i,0} \leftarrow \Pi(i, \mathsf{IV}_i)$ and then for $j = 1, \ldots, B - 1$ it does the following.

1. If there is a tuple of the form $(\sigma_{i,j-1}[2], m, m')$ in the preprocessing output, then return

$$(\sigma_{i,0}[1] \parallel \ldots \sigma_{i,j-1}[1] \parallel m), (\sigma_{i,0}[1] \parallel \ldots \sigma_{i,j-1}[1] \parallel m').$$

2. Set $\sigma_{i,j} \leftarrow \Pi(0, \sigma_{i,j-1}[2])$.

Correctness is easy to verify. We next discuss the success probability of the adversary. Suppose that the online phase of the adversary computes a $\sigma_{i,j}$ during the first half of any of the T/B zero-walks such $\sigma_{i,j}[2]$ matches the last c bits of one of the $\sigma_{i',j'}$s defined in the preprocessing phase. Then, it is guaranteed to stumble on $\sigma_{i',B/2-1}[2]$ during its zero walk. Hence, in this case, it would output a collision.

Since the adversary encounters roughly $\Omega(SB)$ distinct $\sigma_{i,j}[2]$'s in expectation during the preprocessing phase, this suffices to prove that with probability roughly $\Omega(STB/2^c)$ the online phase will win. The term $\Omega(T^2/2^c + T^2/2^r)$ appears due to birthday-style collisions. We refer the reader to Section 4.1 for details.

Attack for $B = 1$. As described in the introduction, the best attack known so far for $B = 1$ has an advantage of $O(S/2^c + T^2/2^r)$. The analogous attack for MD is provably optimal, as mentioned. However,

in contrast to the setting in MD where the ideal object is a random function, here the ideal object is a random permutation, which gives us the additional ability to make inverse queries. This is precisely the leverage that we utilize to get our improved attack. We remark that we are not aware of any prior work that takes advantage of making inverse queries in related contexts.

For $B = 1$, recall that the goal is, given a random IV, to find m, m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1]$. Our first step is a bit counter-intuitive since we actually aim to solve a *harder* task. Specifically, rather than finding an arbitrary collision, we set out to find a collision on 0, that is, find m and m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1] = 0$. This step helps us since a natural way to use inverse queries arises, as we argue next.

Main observation: Finding a collision on 0 can be obtained by finding distinct y and y' such that
$$\Pi^{-1}(0, y)[2] = \Pi^{-1}(0, y')[2] = \mathsf{IV}.$$

In other words, it suffices to find two pre-images of IV with respect to the function $f_\Pi \colon \{0,1\}^c \to \{0,1\}^c$ where $f_\Pi(x)$ outputs the last c bits of $\Pi^{-1}(0, x)$. In Fig. 3, we show the partite representations of $\Pi(\cdot)$ and $\Pi^{-1}(0, \cdot)$. Note that while $\Pi(\cdot)$ is a perfect matching, the function $f_\Pi(\cdot) = \Pi^{-1}(0, \cdot)$ has several elements in its co-domain with multiple pre-images.

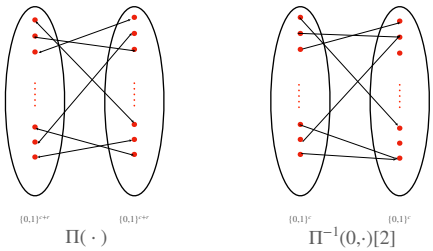


Figure 3: Partite representation of $\Pi(\cdot)$ and $\Pi^{-1}(0, \cdot)$. Notice that Π is a permutation and thus forms a perfect matching while $\Pi^{-1}(0, \cdot)$ is not a permutation and in expectation a random image will have several pre-images.

At this point, we made some progress: we reduced the problem of collision finding to a function inversion problem (for the function $f_\Pi \colon \{0,1\}^c \to \{0,1\}^c$). Indeed, preprocessing attacks for function inversion have been well studied since the 80's. Hellman [Hel80] described an algorithm that gets S bits of preprocessing on the random function $f \colon \{0,1\}^a \to \{0,1\}^b$ as input and inverts it at a point in its image making T queries to the function. It was later formally analyzed by Fiat and Naor [FN99] and shown to have advantage $\epsilon(a, b)$ at inverting $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, where

$$\epsilon(a, b) = \Omega\left(\min\left\{1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right\}\right) \tag{3}$$

We are almost done; three technical challenges remain. First, the result of Hellman applies only to random functions. On the other hand, our function is a restriction of a random permutation (which is not a random function). Fiat and Naor [FN99] showed a clever method to extend Hellman's algorithm to support any function (rather than only random ones), but this improvement is more complicated and comes with a cost in efficiency, which we would like to avoid. To this end, we re-do and adapt the analysis of Hellman to our setting by using the fact that restrictions of permutations are "close enough" to random functions. Our analysis achieves the same parameters as the original one of Hellman, up to constants.

The second problem is that we want to find a pre-image of $\mathsf{IV} \leftarrow\!\!\$\ \{0,1\}^c$ under f_Π, but IV may not even have any pre-images under f_Π, let alone two which are required for our attack. Fortunately, as f_Π is at least "close" to a random function, we can show via a balls-into-bins analysis that a constant fraction of the co-domain will have at least two distinct pre-images. Still, could it be the case that Hellman's attack

somehow fails on this fraction of the co-domain? Via a closer analysis of Hellman's attack, we show that for any function $f\colon \{0,1\}^a \to \{0,1\}^b$ and fixed element $y \in \{0,1\}^b$, the attack succeeds at finding a pre-image $x' \in f^{-1}(y)$ with probability $\epsilon(a,b)$ where

$$\epsilon(a,b) = \Omega\left(\frac{1}{b}\min\left(1, \frac{ST \cdot |f^{-1}(y)|}{2^a}, \left(\frac{S^2T \cdot |f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right)\right) \tag{4}$$

The last problem we face is that we need to find two *distinct* pre-images for IV. However, applying an inversion algorithm in a black box fashion does not guarantee that distinct inverses will be found. Thus, we also prove that Hellman's inversion algorithm finds a uniform pre-image among all possible pre-images for a given element in the co-domain.

After resolving the above technical challenges, we show that if we run Hellman's attack twice independently for the function f_Π on the image IV, if IV has at least two pre-images (which it does with constant probability), then we will find two distinct pre-images with at least $1/2$ probability times the probability that both attacks succeed. Thus, our overall success probability is roughly $\Omega(\epsilon(c,c)^2)$ where ϵ is defined in (4). We refer the reader to section 4.2 for the details.

2.2 Impossibility Results for Best Attacks

When giving new attacks for finding short collisions, the natural question is how far we can go. In other words, what are the best possible attacks? For $B = 1, 2$ in the case of MD, optimal attacks are known. Our goal here is to prove an upper bound on the advantage for the best-possible adversary that has S bits of preprocessing as input and can make T queries to Π, Π^{-1} in finding collisions of length 1 and 2 for the sponge construction.

Impossibility result for $B = 1$. We use the pre-sampling technique proposed by [Unr07] and later optimized and adapted to the AI-RPM by [CDG18] to get an advantage upper bound of roughly $O(ST/2^c + T^2/2^r)$. However, we note that this bound does not match the best $B = 1$ attacks, so it is open which side can be improved. Ideally, one could use a compression-based technique as done in [DGK17] to get a tight bound for the $B = 1$ case for MD, but it is not clear how to adapt this argument to handle inverse queries in the AI-RPM model, as we shall see below.

Impossibility result for $B = 2$. The presampling technique of [Unr07, CDG18] *provably* cannot give an advantage upper bound better than $O(ST^2/2^c + T^2/2^r)$ for $B = 2$. Since we can prove this advantage upper bound even for unbounded length collisions, it is natural to ask whether we can prove that 2-block collisions are, in fact, harder to find than collisions of arbitrary length. Aside from presampling techniques, the main technique used to rule out attacks is via a compression argument [GT00, Wee05], which we turn to for our impossibility result. As a warm up, we first give an overview for the $B = 1$ compression argument for MD from [DGK17] to highlight the key challenges in our setting.

Overview of $B = 1$ compression argument for MD. In a compression argument, the main idea is to use an adversary $\mathcal{A}$ that succeeds at some task involving a random object $\mathcal{O}$, to compress $\mathcal{O}$ beyond what is information theoretically possible. This clearly establishes a contradiction, which gives an upper bound in the success probability of $\mathcal{A}$.

Let $h\colon [N] \times [M] \to [N]$ be a hash function that is modeled as a random oracle, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an (S,T) adversary that tries to find a 1-block collision in h. $\mathcal{A}_1$ gets h as input and can output S bits of advice σ. $\mathcal{A}_2$ gets σ along with a random salt $a \in [N]$, can make T queries to h, and tries to output $m, m' \in [M]$ such that $h(a,m) = h(a,m')$ and $m \neq m'$. We show that if $\mathcal{A}$ succeeds at this task for many salts a, then we can describe h with fewer bits than possible.

To encode h, we first compute $\sigma \leftarrow \mathcal{A}_1(h)$. We let $G \subseteq [N]$ be the set of elements for which $\mathcal{A}_2$ succeeds on inputs (σ, a) for all $a \in G$. We run $\mathcal{A}_2$ on (σ, a) for all $a \in G$ in lexicographic order. The hope is that whenever $\mathcal{A}_2$ succeeds in finding a collision, we can use the corresponding queries for the collision it makes to compress the function h.

For example, if $\mathcal{A}_2(\sigma, a)$ outputs a collision (m, m'), then we can assume that $\mathcal{A}_2$ must have queried $h(a, m)$ and $h(a, m')$ at some point (we assume without loss of generality that (a, m) was queried before (a, m')). So whenever it queries $h(a, m')$, rather than encoding the output of h, we write down information to indicate that it's the same output as the query for $h(a, m)$. It is easy to see by a counting argument that at least half of the $a \in G$ cannot be queried by $\mathcal{A}_2$ more than $2T$ times. For all such a, we can refer back to the previous query $h(a, m)$ using $\log 2T$ bits, and we use another $\log 2T$ bits to identify the query $h(a, m')$. Thus, we save $\log N - 2 \log 2T$ bits per each of these a values in G, which gives a non-trivial compression of h if $T^2 < N$.

The problem with inverse queries. As we stated before, the first major roadblock we encounter when adapting this framework to the AI-RPM is the existence of inverse queries. Let's try to adapt the argument above to the setting of Sponge with 1-block collision. Now the preprocessing adversary $\mathcal{A}_1$ is given a random permutation Π and outputs some state σ with $|\sigma| \leq S$. The online adversary $\mathcal{A}_2$ receives σ and a random IV, and tries to find m, m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1]$.

Now suppose that $\mathcal{A}_2$ outputs a collision (m, m') with respect to the sponge construction. We can no longer even assume that $\mathcal{A}_2$ queries both $\Pi(m, \mathsf{IV})$ and $\Pi(m', \mathsf{IV})$! For example, it may have first queried $\Pi(m, \mathsf{IV}) = (y, u_1)$ and then queried $\Pi^{-1}(y, u_2) = (m', \mathsf{IV})$. At first glance, this doesn't seem like a problem, we can again note that part of the output of query (y, u_2) is the same as the input to the query for (m, IV). So maybe we can use the same trick as before and instead of storing the whole answer of $\Pi^{-1}(y, u')$, store information indicating that the last c bits of the answer is the same as the input of the query $\Pi(m, \mathsf{IV})$. This intuition is misleading. We can no longer do a counting argument to show that this information is short. It is not clear how to identify the query $\Pi^{-1}(y, u')$ with few bits to be able to point back to the $\Pi(m, \mathsf{IV})$ query. For example, the adversary may just query $\Pi^{-1}(y, *)$ many times and hope to hit IV twice. We hope that this example sheds light on why, at a minimum, inverse queries significantly complicate the situation and deserve extra attention.

Compression for $B = 2$ via multi-instance games. The above compression approach is not known to generalize to the case of $B \geq 2$ collisions for MD. To overcome this limitation, Akshima et al. [ACDW20] propose a beautiful framework that gives non-trivial bounds $B \geq 2$ for the case of MD. Their framework reduces the problem to a related "multi-instance" game. In a multi-instance game, the adversary has an arbitrary size string σ of S-bits hard-coded, and its goal is to find a 2-block collision for a set of $u \approx S$ uniformly random a's. The adversary $\mathcal{A}_2$ can make T queries to h when running on each of the u IVs. The key distinctions in this multi-instance game is that (1) the advice sigma that $\mathcal{A}_2$ receives is *independent* of h, and (2) we only need to analyze $\mathcal{A}_2$'s success probability for a random set of u IVs. The core of the proof is a compression argument to upper bound the advantage of this adversary. This framework unfortunately is not strong enough to deal with $B = 1$, as at best it gives the same bound as bit-fixing. However, we adapt this framework to the setting of random permutations to give a non-trivial bound for $B = 2$.

In our case, we need to build a compression argument to compress Π and a set of u random IVs, $\mathsf{IV}_1, \ldots, \mathsf{IV}_u$, (for $u \approx S$) using an adversary $\mathcal{A}_2$ which has some fixed hard-coded advice. $\mathcal{A}_2$ runs on the IVs one by one and succeeds in finding 2-block collisions for all of them. The encoding avoids storing some of the values of Π explicitly, and instead stores information about the queries of $\mathcal{A}_2$ to Π and Π^{-1} which help during the decoding procedure to recover these particular values of Π.

For $B = 2$ collisions, there are possibly 4 "crucial" queries that the adversary might make that correspond to a 2-block collision (two for each message). We possibly need to consider all combinations of ways that the queries could have been made in either the forward or the reverse direction. For the case of this overview, we zoom in on a single case where inverse queries complicate the situation, and explain how we overcome this.

Suppose on an input IV_j (where $\mathcal{A}_2$ had previously been run on inputs $\mathsf{IV}_1, \ldots, \mathsf{IV}_{j-1}$), $\mathcal{A}_2$ arrives at a collision by making the crucial queries q_1, q_2, q_3, q_4 (not necessarily in that order) such that

1. q_1 was a query to Π on (m_1, IV_j) and returned (x_1, IV'_1)
2. q_2 was a query to Π on (m_2, IV_j) and returned (x_2, IV'_2)

3. q_3 was a query to Π on $(m_3 \oplus x_1, \mathsf{IV}'_1)$ and returned (y, IV'_3)

4. q_4 was a query to Π on $(m_4 \oplus x_2, \mathsf{IV}'_2)$ and returned (y, IV'_4)

Clearly, (m_1, m_3) and (m_2, m_4) hash to the same output and hence are a collision. Now suppose queries q_1 and q_2 were first made during $\mathcal{A}_2(\mathsf{IV}_j)$,[7] while queries q_3 and q_4 were each made previously while running $\mathcal{A}_2$ on an earlier IV_i value. Now, the strategy to compress on the lines of [ACDW20] is not to include the last c bits of the answers of q_1, q_2 and the last r bits of the answer of q_4 in the encoding. Instead, we can store the index of the queries q_3, q_4 among all queries (these indices will be in $[uT]$ since there are u IVs and T queries for each of them) and store the indices of the queries q_1, q_2 among the queries made while running $\mathcal{A}_2$ on IV_j (these indices will be in $[T]$). This leads to a saving of roughly $2c + r - 2\log T - 2\log uT$ bits. For reasonable parameters of S, T, c, r, this implies a compression of at least $c - \log uT$ bits, so if $uT \approx ST < 2^c$, this gives non-trivial compression. This implies an upper bound of $ST/2^c$ on the advantage for this case.

However, with inverse queries allowed, things get more complicated. Suppose instead that queries q_3 and q_4 were made in the reverse direction, so $\mathcal{A}_2$ queries $\Pi^{-1}(y, \mathsf{IV}'_3)$ and $\Pi^{-1}(y, \mathsf{IV}'_4)$ prior to running $\mathcal{A}_2(\mathsf{IV}_j)$. In this case, we can still save the c bits from the answers of q_1, q_2. But there is no clear way to save in storing the answer to query q_4 since its answer $(m_4 \oplus x_2, \mathsf{IV}'_2)$ has seemingly no relation to either the answer or input of q_3. So, in this case, we are only able to save $2c - 2\log T - 2\log uT$ bits, which leads to non-trivial compression only if $u^2T^4 \approx S^2T^4 < 2^c$. This implies an upper bound of $S^2T^4/2^{2c}$ on the attacker's advantage for this case. Note that this is actually *better* than the $ST/2^c$ bound we got when considering only forward queries whenever $ST^3 < 2^c$. However, it is important to note that we still need to consider all possible ways in which the attacker may find a collision. We need to show even in the worst case, we can compress Π in order to get an upper bound on the advantage.

The above highlights just one of the several subtleties that inverse queries introduce in the proof. The ability of the adversary to make queries in two directions makes the encoding and decoding procedures significantly more complicated and lengthy. See Section 5 for full details.

3 Preliminaries

We let $[N] = \{1, 2, \ldots, N\}$ for $N \in \mathbb{N}$ and for $k \in \mathbb{N}$ such that $k \leq N$, let $\binom{S}{k}$ denote the set of k-sized subsets of S. We use $|X|$ to denote the size of a set X and use X^+ to denote one or more elements of X. The set of all permutations on D is denoted by $\mathsf{Perm}(D)$. 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). For a random variable X we use $\mathbb{E}[X]$ to denote its expected value.

We use $x \leftarrow\!\!\$\ \mathcal{D}$ to denote sampling x according to the distribution $\mathcal{D}$. 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.

All logarithms in this paper are for base 2 unless otherwise specified.

Sponge-based hashing. For $c, r \in \mathbb{N}$, let $\Pi : \{0,1\}^{c+r} \to \{0,1\}^{c+r}$ be a permutation. We define sponge-based hashing $\mathsf{Sp}_\Pi : \{0,1\}^c \times (\{0,1\}^r)^+ \to \{0,1\}^r$ as follows. For $s \in \{0,1\}^{r+c}$ we use $s[1]$ to denote its first r bits and $s[2]$ to denote its last c bits.

$\underline{\mathsf{Sp}_\Pi(\mathsf{IV}, m = (m_1, \ldots, m_B))}$
$s_0 \leftarrow 0^r \,\|\, \mathsf{IV}$
For $i = 1, \ldots, B$
  $s_i[1] \,\|\, s_i[2] \leftarrow \Pi\left((m_i \oplus s_{i-1}[1]) \,\|\, s_{i-1}[2]\right)$
Return $s_B[1]$

The elements of $\{0,1\}^r$ shall be referred to as *blocks* and IV refers to the *initialization vector* (also referred to as *salt* in the literature). This is the same abstraction of sponge-based hashing as the one used in [CDG18].

[7] Note that this assumption is easy to remove as otherwise we can achieve compression by not including IV_j in the encoding and recovering it from $\mathcal{A}_2$'s queries during decoding

Game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$	Subroutine $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$
1. $\Pi \leftarrow\!\!\$\ \mathsf{Perm}(\{0,1\}^{c+r})$	1. $\sigma \leftarrow\!\!\$\ \mathcal{A}_1(\Pi)$
2. $\mathsf{IV} \leftarrow\!\!\$\ \{0,1\}^c$	2. $(\alpha,\alpha') \leftarrow\!\!\$\ \mathcal{A}_2^{\Pi,\Pi^{-1}}(\sigma,\mathsf{IV})$
3. Return $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A})$	3. Return true if:
	(a) $\alpha \neq \alpha'$,
	(b) $\lvert\alpha\rvert, \lvert\alpha'\rvert$ are at most B blocks long and
	(c) $\mathsf{Sp}_\Pi(\mathsf{IV},\alpha) = \mathsf{Sp}_\Pi(\mathsf{IV},\alpha')$
	4. Else, return false

Figure 4: The bounded-length collision resistance game of salted sponge based hash in the AI-RPM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$.

Auxiliary-input Random Permutation Model (AI-RPM). We use the Auxiliary-Input Random Permutation Model (AI-RPM) introduced by Coretti, Dodis and Guo [CDG18] to study non-uniform adversaries in the Random Permutation Model (this was a natural extension of the AI-ROM model proposed by Unruh in [Unr07]). 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 permutation Π and it outputs an S-bit auxiliary input σ. Adversary $\mathcal{A}_2$, referred to as the online phase, gets σ as input and can make a total of T queries to Π,Π^{-1}, and attempts to accomplish some goal involving Π. 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 oracles. We next formalize the collision resistance of sponge-based hash functions in AI-RPM.

Short collision resistance of sponge-based hashing in AI-RPM. We formalize the hardness of bounded-length collision resistance of sponge-based hash functions in the AI-RPM. The game is parameterized by c,r. The game first samples a permutation Π uniformly at random from $\mathsf{Perm}(\{0,1\}^{c+r})$ and IV uniformly at random from $\{0,1\}^c$. Then, $\mathcal{A}_1$ is given unbounded access to Π, and it outputs σ. At this time, $\mathcal{A}_2$ gets σ and IV as input and has oracle access to Π,Π^{-1}. It needs to find $\alpha\neq\alpha'$ such that (1) $\mathsf{Sp}_\Pi(\mathsf{IV},\alpha)=\mathsf{Sp}_\Pi(\mathsf{IV},\alpha')$ and (2) α,α' consist of $\leq B$ blocks from $\{0,1\}^r$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$, is explicitly written in Fig. 4. In Fig. 4, we write the adversary's execution in its own subroutine only for syntactical purposes (as we shall use it later).

Definition 3.1 (AI-CR Advantage)**.** For parameters $c,r,B\in\mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of sponge in the AI-RPM is

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

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

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

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

The compression lemma. Our proof of the impossibility result for $B=2$ 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 3.2 (E.g., [DTT10])**.** 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

$$\Pr_{x \xleftarrow{\$} \mathcal{X}}[\mathsf{Decode}(\mathsf{Encode}(x)) = x] \geq \epsilon.$$

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

4 Attacks

In this section, we first provide the generic attack for finding B-block collisions inspired by the analogous attack for MD in [ACDW20]. We then provide our new AI-RPM attack for finding 1-block collision (Section 4.2). Additionally, in Section 4.3, we prove the key lemma for our attack. The key lemma is a preprocessing attack for inverting a function f which is a restricted random permutation. The attack is closely related to that of Hellman [Hel80], but we provide rigorous analysis for our specific application for completeness.

4.1 Generic Attack for B-Block Collisions

We give a (S,T) adversary $\mathcal{A}$ that has advantage $O(STB/2^c + T^2/2^c + T^2/2^r)$ against $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$. The main idea for this attack is similar to the zero-walk attack for finding B-block collisions in the Merkle-Damgård construction introduced in [ACDW20] which was in turn inspired by an attack in [CDGS18].

High level idea. In the preprocessing phase the adversary randomly samples $t \approx S$ different IVs $\mathsf{IV}_1, \ldots, \mathsf{IV}_t$ and for each of them computes $\sigma_{i,j}$ for $j \in [B/2-1]$ as $\sigma_{i,j} = \Pi(0, \sigma_{i,j-1}[2])$, where $\sigma_{i,0} = (0, \mathsf{IV})$. The sequence $\sigma_{i,0}, \ldots, \sigma_{i,B/2-1}$ forms a "zero-walk" on IV_i. It then finds m_i, m_i' such that $\Pi(m_i, \sigma_{i,B/2-1}[2])[1] = \Pi(m_i', \sigma_{i,B/2-1}[2])[1]$ for $i = 1, \ldots, t$. It outputs

$$(\sigma_{i,B/2-1}[2], m_i, m_i')_{i=1,\ldots,t}\ .$$

In the online phase, the adversary gets a challenge IV as input. For $i = 1, \ldots, T/B$, it computes $\mathsf{IV}_i = \Pi(i, \mathsf{IV})[2]$. For each of the IV_i's, it does a zero-walk of length $B-2$. If on any of the walks it hits an IV that the preprocessing phase output then it outputs a collision. The reason this attack achieves an advantage of $\Omega(STB/2^c)$ is because in the preprocessing phase the adversary roughly hits $\Omega(SB)$ distinct IVs and in the online phase if it hits any of these IV's in the first half of its T/B (i.e., in roughly $T/2$ of the queries) walks it finds a collision.

We formally state our result below.

Theorem 4.1. *Let $S, T, B, c, r \in \mathbb{N}$ such that $SB \leq 2^{c-1}$, $T \leq \min\{(2^{c-1}, 2^{r-1})\}$, $T \geq 2B$. There exists an (S,T) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) \geq \left\lfloor \frac{S}{c+2r} \right\rfloor \left\lfloor \frac{B}{2} - 1 \right\rfloor \frac{T}{2^{c+3}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}}\ .$$

Proof. The adversary is $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ defined in Figure 5.

We first note that the probability that the preprocessing phase aborts in any of the iterations is at most

$$\frac{\prod_{i=0}^{2^r-1}(2^{c+r} - i2^c)}{\prod_{i=0}^{2^r-1}(2^{c+r} - i)}\ .$$

This follows since the numerator indicates the number of ways to sample $\Pi(\cdot, \mathsf{IV})$ such that no two outputs match in the first r bits, while the denominator is the total number of ways to sample $\Pi(\cdot, \mathsf{IV})$. The above product is upper bounded by

$$\frac{\prod_{i=0}^{2^r-1}(2^{c+r} - i2^c)}{\prod_{i=0}^{2^r-1} 2^{c+r-1}} = \prod_{i=0}^{2^r-1}(1 - i2^{1-r}) \leq e^{-2^{1-r}\sum_{i=0}^{2^r-1} i} = e^{-(2^r-1)}\ .$$

Algorithm $\mathcal{A}_1(\Pi)$

1. $t \leftarrow \lfloor S/(c+2r) \rfloor$
2. $B' \leftarrow \lfloor B/2 \rfloor - 1$
3. For $i = 1, \ldots, t$
4. $\quad \mathsf{IV}_{p,i,0} \leftarrow\!\!\$\ \{0,1\}^c$
5. $\quad$ For $j = 0, \ldots, B'$:
6. $\quad\quad \mathsf{IV}_{p,i,j} \leftarrow \Pi(0, \mathsf{IV}_{p,i,j-1})[2]$
7. $\quad \mathsf{IV}_{p,i} \leftarrow \mathsf{IV}_{p,i,B/2-1}$
8. $\quad$ Compute (m_i, m_i') such that $\Pi(m_i \,\|\, \mathsf{IV}_{p,i})[1] = \Pi(m_i' \,\|\, \mathsf{IV}_{p,i})[1]$
9. $\quad$ If no such m_i, m_i' found, ABORT
10. Return $\{(\mathsf{IV}_{p,i}, m_i, m_i')\}_{i=1,\ldots,B/2-1}$

Algorithm $\mathcal{A}_2(\mathsf{IV}, \sigma)$

1. $t \leftarrow \lfloor S/(c+2r) \rfloor$
2. $B' \leftarrow \lfloor B/2 \rfloor - 1$
3. $T' \leftarrow \lfloor T/B \rfloor$
4. Parse σ as $(\mathsf{IV}_{p,i}, m_i, m_i')_{i=1}^{t}$
5. If $\exists i \in [t] : \mathsf{IV}_{p,i} = \mathsf{IV}$: return (m_i, m_i')
6. For $k = 1, \ldots, T'$:
7. $\quad m_{k,0} \,\|\, \mathsf{IV}_{k,0} \leftarrow \Pi(k, \mathsf{IV})$
8. $\quad$ For $j = 1, \ldots, B-1$:
9. $\quad\quad$ If $\mathsf{IV}_{k',j'} = \mathsf{IV}_{k,j-1}$ for some $(k', j') \neq (k, j-1)$:
10. $\quad\quad\quad$ Return $(k' \,\|\, m_{k',0} \,\|\, m_{k',1} \,\|\, \ldots \,\|\, m_{k',j'}, k \,\|\, m_{k,0} \,\|\, m_{k,1} \,\|\, \ldots \,\|\, m_{k,j-1})$
11. $\quad\quad$ If $\mathsf{IV}_{k,j-1} = \mathsf{IV}_{p,i}$ for some $i \in [t]$:
12. $\quad\quad\quad$ Return $(k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1} \,\|\, m_i, k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1} \,\|\, m_i')$
13. $\quad\quad m_{k,j} \,\|\, \mathsf{IV}_{k,j} \leftarrow \Pi(0 \,\|\, \mathsf{IV}_{k,j-1})$
14. If $\exists (k, k', j, j') : m_{k,j} = m_{k'j'}$:
15. $\quad$ Return $(k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1}, k' \,\|\, m_{k',0} \,\|\, \ldots \,\|\, m_{k',j'-1})$

Figure 5: Adversary $\mathcal{A}$ used in the proof of Theorem 4.1.

Using the union bound it follows that the preprocessing phase will abort with probability at most $S/e^{(2^r-1)}$.

We next calculate the probability that $\mathcal{A}_2$ succeeds in finding a collision. Let R be the event that the adversary $\mathcal{A}_2$ returns due to the if statements in line 9 or line 14 and let Q be the event that it returns due to the if statement in line 11. Clearly, adversary $\mathcal{A}_2$ succeeds with probability $\Pr[R] + \Pr[Q]$. First we compute $\Pr[R]$. Note that R happens if all the $\mathsf{IV}_{k,j}$'s are not distinct or if all the $\mathsf{IV}_{k,j}$'s are distinct but all the $m_{k,j}$'s are not distinct. The probability that all the $\mathsf{IV}_{k,j}$'s are not distinct is at least

$$1 - \sum_{i=1}^{T'B} \left(1 - \frac{i}{2^c - i}\right) \geq 1 - e^{-\sum_{i=1}^{T'B} \frac{i}{2^c}} \geq 1 - e^{-\frac{(T-B)(T-B-1))}{2^{c+1}}} \geq \frac{(T-B)(T-B-1)}{2^{c+1}}$$

The second inequality follows using $1 - x \leq e^{-x}$ and the third inequality follows using $T'B \geq T - B$, and the last inequality follows using $e^{-2x} \leq 1 - x$ for $x \leq 1/2$ (this can be verified using elementary calculus). It is also easy to see that the probability that all the $\mathsf{IV}_{k,j}$'s are not distinct is at most $T^2/2^{c+1}$ using an union bound. Using $T^2 \leq 2^{c-1}$ we have that this probability is at most $1/4$. Hence the probability that all the $\mathsf{IV}_{k,j}$'s are distinct is at least $3/4$. Similarly we can show that the probability that all the $m_{k,j}$'s are not distinct given all $\mathsf{IV}_{k,j}$'s are distinct is at least $\frac{(T-B)(T-B-1)}{2^{r+1}}$. This would give us that

$$\Pr[R] \geq \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} .$$

It is also easy to see that the probability that all the $m_{k,j}$'s are not distinct is at most $T^2/2^{r+1}$ using an union bound. Using $T^2 \leq 2^{r-1}$ we have that this probability is at most $1/4$. Hence the $\Pr[R] \leq 1/2$.

We next note that

$$\Pr[Q] + \Pr[R] \geq \Pr[Q|\neg R](1 - \Pr[R]) + \Pr[R] \geq 1/2 \Pr[Q|\neg R] + \Pr[R] .$$

Above we used $\Pr[R] \leq 1/2$. Now we shall show a lower bound on $\Pr[Q|\neg R]$. First note that the event $\neg R$ implies that all the $\mathsf{IV}_{k,j}$'s are distinct.

Define

$$A = \{\mathsf{IV}_{p,i,j} : 1 \leq i \leq t, 0 \leq j \leq B'\} .$$

Let $F_{k,j}$ be the event that $\mathsf{IV}_{k,j} \in A$ and for all $k' < k, 0 < j' < B - 1$, $\mathsf{IV}_{k',j'} \notin A$ and for all $j' < j$, $x_{k,j'} \notin A$. We have that

$$\Pr[F_{k,j}|\neg R] = \frac{|A|}{2^c - ((k-1)B + j)} \prod_{i=1}^{(i-1)B+j} \left(1 - \frac{|A|}{2^c - i + 1}\right) .$$

Using $T \leq 2^{c-1}$ we have that $2^c - i + 1 \geq 2^{c-1}$. Therefore

$$\Pr[F_{k,j}|\neg R] \geq \frac{|A|}{2^c} \left(1 - \frac{2|A|}{2^c}\right)^{(k-1)B+j} \geq \frac{|A|}{2^c} \left(1 - \frac{2|A|((k-1)B+j)}{2^c}\right) \geq |A|/2^{c+1} .$$

It is clear that $F_{k,j}$ are disjoint, and given that R does not happen, Q happens if for any $k \in [T']$, in the first $B - B'$ iterations of the loop in j, $x_{i,j} \in A$. Therefore

$$\begin{aligned}\Pr[Q|\neg R] &\geq \sum_{k \in [T']} \sum_{j \in \{0,\ldots,B-1-B'\}} \Pr[F_{k,j}|\neg R] = (B - 1 - B')T'(|A|/2^{c+1}) \\ &= (\lceil B/2 \rceil)\lfloor T/B \rfloor |A|/2^{c+1} \geq T|A|/2^{c+1} .\end{aligned}$$

The last inequality follows using $T \geq 2B$.

Hence we have that the advantage of the adversary $\mathcal{A}$ is

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) &\geq \sum_{i=1}^{t(B'+1)} \frac{\Pr[|A|=i]Ti}{2^{c+2}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}} \\ &= \frac{\mathbb{E}[A]T}{2^{c+2}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}} \ .\end{aligned}$$

The second equality follows from the definition of expectation. We next need to show that $\mathbb{E}[A] \geq \frac{1}{2}\lfloor \frac{S}{c+2r} \rfloor \lfloor \frac{B}{2} - 1 \rfloor$ to finish the proof.

Let $E_{i,j}$ be the event that $\mathsf{IV}_{p,i,j}$ is a new IV discovered while preprocessing. We compute a lower bound on $\Pr[E_{i,j}]$ as follows.

$$\begin{aligned}\Pr[E_{i,j}] &\geq \Pr[E_{i,0} \cap E_{i,1} \cap \ldots \cap E_{i,j}] = \prod_{k=0}^{j} \Pr[E_{i,k} | E_{i,0} \cap \ldots \cap E_{i,k-1}] \\ &\geq \prod_{k=0}^{j} \frac{2^{c+r} - ((i-1)(B'+1)+k)2^r}{2^{c+r} - ((i-1)(B'+1)+k)} \geq \prod_{k=0}^{j} \frac{2^{c+r} - 2^{c+r-1}}{2^{c+r}} = 1/2 \ .\end{aligned}$$

The second inequality uses $SB \leq 2^{c-1}$. Hence we have that

$$\mathbb{E}[A] = \sum_{i=1}^{t} \sum_{j=0}^{B'} \Pr[E_{i,j}] \geq \frac{1}{2} \left\lfloor \frac{S}{c+2r} \right\rfloor \left\lfloor \frac{B}{2} - 1 \right\rfloor \ .$$

This concludes the proof. ■

4.2 Preprocessing Attack for $B = 1$

We give a new AI-RPM attack for finding 1-block collisions in the Sponge construction. The key ingredient in our attack is an (S,T) adversary for a function f finds two distinct pre-images of a random element of the co-domain under f. We construct this adversary in Lemma 4.3 based on the adversary from Lemma 4.5 that finds a single pre-image of a random element of the co-domain under f.

Theorem 4.2. *Let $c, r \in \mathbb{N}$. For any $S, T \in \mathbb{N}$ such that $S \geq 24c$, $2^c \geq 24S$, and $2^c \geq (S/(T-2)) \cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that on input $\{0,1\}^c$ outputs a valid 1-block collision with probability ϵ, where*

$$\epsilon \geq \left(\frac{1}{20 \cdot 288^2 \cdot c^2}\right) \cdot \min\left(1, \frac{S^2(T-2)^2}{2^{2c+2}}, \left(\frac{S^2(T-2)}{2^{2c+1}}\right)^{2/3}\right).$$

Proof. Let $\Pi\colon \{0,1\}^{c+r} \to \{0,1\}^{c+r}$ be a random permutation. Define the function $f_\Pi\colon \{0,1\}^c \to \{0,1\}^c$ as $f_\Pi(x) = \Pi^{-1}(0^r \,\|\, x)[2]$. Note that f_Π is equivalent to the function that outputs the first c bits of the permutation $\Pi'(x \,\|\, 0^r)$, where $\Pi'(x \,\|\, y)$ for $x \in \{0,1\}^c, y \in \{0,1\}^r$ computes $\Pi^{-1}(y \,\|\, x)$ and shifts the first r bits of the output to the end of its output. Thus, we can invoke Lemma 4.3 for the function f_Π, which implies an $(S, T-2)$ attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for finding two distinct pre-images of a random $y \leftarrow\!\!\$\ \{0,1\}^c$. The attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as follows.

- $\mathcal{A}_1(\Pi)$:
  1. Output $\sigma \leftarrow \mathcal{B}_1(f_\Pi)$.
- $\mathcal{A}_2(\mathsf{IV}, \sigma)$:
  1. Compute $(x_1, x_2) \leftarrow \mathcal{B}_1(\mathsf{IV}, \sigma)$.

2. If $f_\Pi(x_1) = f_\Pi(x_2) = \mathsf{IV}$ and $x_1 \neq x_2$, compute $m_1 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_1)$ and $m_2 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_2)$ and output (m_1, m_2).
3. Otherwise, output $\perp$.

For correctness, we note that if $\mathcal{A}_2$ outputs a non-$\perp$ value, then $\mathcal{A}_2$ succeeds in finding a 1-block collision. Recall that $\mathsf{Sp}_\Pi(\mathsf{IV}, m_1) = \Pi(m_1 \,\|\, \mathsf{IV})[1]$ and $\mathsf{Sp}_\Pi(\mathsf{IV}, m_2) = \Pi(m_2 \,\|\, \mathsf{IV})[1]$. By construction, $f_\Pi(x_1) = f_\Pi(x_2) = \mathsf{IV}$ implies $m_1 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_1)$ and $m_2 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_2)$ for some $m_1, m_2 \in \{0,1\}^r$. But this in turn implies that $\Pi(m_1 \,\|\, \mathsf{IV})[1] = \Pi(m_2 \,\|\, \mathsf{IV})[1] = 0^r$. Since Π is a permutation and $x_1 \neq x_2$, it must be the case that $m_1 \neq m_2$, so (m_1, m_2) is a valid 1-block collision, as required.

Whenever $\mathcal{B}$ succeeds, $\mathcal{A}$ succeeds, so the success probability follows immediately from Lemma 4.3. ■

Lemma 4.3. *Let $n \geq 1$, $a \leq n - 3$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^a$ such that $f(x)$ consists of the first a bits output by $\Pi(x \,\|\, 0^{n-a})$. For any $S, T \in \mathbb{N}$ such that $S \geq 24a$, $2^a \geq 24S$, and $2^a \geq (S/T) \cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that on input $y \leftarrow\!\!\$\ \{0,1\}^a$ outputs x_1, x_2 such that $f(x_1) = f(x_2) = y$ and $x_1 \neq x_2$ with probability ϵ, where*

$$\epsilon \geq \left(\frac{1}{20 \cdot 288^2 \cdot a^2}\right) \cdot \min\left(1, \frac{S^2T^2}{2^{2a+2}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{2/3}\right).$$

Proof. Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be an $(S/2, T/2)$ adversary from Lemma 4.5. In the offline phase, $\mathcal{A}_1$ on input the function f runs $\mathcal{B}_1(f)$ twice and gets σ_1, σ_2. $\mathcal{A}_1$ outputs $\sigma = (\sigma_1, \sigma_2)$. In the online phase, $\mathcal{A}_2$ on input σ and $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$, computes $x_1 = \mathcal{B}_2(y, \sigma_1)$ and $x_2 = \mathcal{B}_2(y, \sigma_2)$. If $f(x_1) = f(x_2) = y$ and $x_1 \neq x_2$, $\mathcal{A}_2$ outputs (x_1, x_2) and otherwise outputs $\perp$. It directly follows that $\mathcal{A}$ uses space $|\sigma| = |\sigma_1| + |\sigma_2| \leq S$ and makes at most $2 \cdot (T/2) = T$ queries. So it remains to analyze the advantage of $\mathcal{A}$.

We define the following events that are relevant to the analysis. Let $\mathsf{Success}_1$, $\mathsf{Success}_2$ be the events that $\mathcal{B}_1(f, \sigma_1)$ and $\mathcal{B}_1(f, \sigma_2)$ output a valid pre-image, respectively. Let $\mathsf{Inverse}$ be the event that $|f^{-1}(y)| \geq 2$ for the challenge $y \leftarrow\!\!\$\ \{0,1\}^a$. Let $\mathsf{Distinct}$ be the event that the outputs x_1 and x_2 are distinct. Thus, the probability of success is given by

$$\epsilon = \Pr[\mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse} \wedge \mathsf{Distinct}].$$

Note that $\mathsf{Success}_1$ and $\mathsf{Success}_2$ are identical and independently distributed given a fixed value for y. Thus, we can rewrite the success probability as

$$\begin{aligned}\epsilon &= \Pr[\mathsf{Inverse}] \cdot \Pr[\mathsf{Success}_1 \mid \mathsf{Inverse}] \cdot \Pr[\mathsf{Success}_2 \mid \mathsf{Inverse}] \\ &\quad \cdot \Pr[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}]. \\ &= \Pr[\mathsf{Inverse}] \cdot \Pr[\mathsf{Success}_1 \mid \mathsf{Inverse}]^2 \\ &\quad \cdot \Pr[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}]\end{aligned}$$

We analyze each of these terms separately.

In Claim 4.4, we show that $\Pr[\mathsf{Inverse}] \geq 1/10$ as long as $a \leq n - 3$. $\Pr[\mathsf{Success}_1 \mid \mathsf{Inverse}]$ is given in Lemma 4.5 using $S' = S/2$ and $T' = T/2$. As $|f^{-1}(y)| \geq 2$ by assumption, it holds that

$$\Pr[\mathsf{Success}_1 \mid \mathsf{Inverse}] \geq \left(\frac{1}{288 \cdot a}\right) \cdot \min\left(1, \frac{ST}{2^{a+1}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{1/3}\right).$$

For the event $\mathsf{Distinct}$, note that in the worst case $|f^{-1}(y)| = 2$. In this case, it is equally as likely that $x_1 = x_2$ compared to $x_1 \neq x_2$ since there are only two equally likely values for x_1, x_2. Thus,

$$\Pr[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}] = 1/2.$$

Combining the above, we conclude that the attackers probability of success is at least

$$\epsilon \geq \frac{1}{2} \cdot \frac{1}{10} \cdot \left(\frac{1}{288^2 \cdot a^2}\right) \cdot \min\left(1, \frac{S^2T^2}{2^{2a+2}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{2/3}\right),$$

as required. ■

Claim 4.4. *Let $n \geq 1$, $a \leq n-3$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^a$ such that $f(x)$ consists of the first a bits output by $\Pi(x \,\|\, 0^{n-a})$. Then, $\Pr[y \leftarrow\!\!\$\, \{0,1\}^a : |f^{-1}(y)| \geq 2] \geq 1/10$.*

Proof. We analyze this as a balls-into-bins problem. Specifically, we imagine each output $y \in \{0,1\}^a$ as a bin, and for each $x \in \{0,1\}^a$, we put a ball in the bin corresponding to $f(x)$. Let $x_1, \ldots, x_{2^a}$ be an arbitrary ordering of $\{0,1\}^a$ that denotes the order we place the balls into the bins. Let $X_{i,b}$ be a random variable corresponding to the set of bins with i balls inside after placing b balls via this process. Since $\sum_{i=0}^{2^a} |X_{i,2^a}| = 2^a$, to get a lower bound on $\sum_{i=2}^{2^a} \mathbb{E}[|X_{i,2^a}|]$, it suffices to upper bound $\mathbb{E}[|X_{0,2^a}| + |X_{1,2^a}|]$.

First, note that $X_{0,0} = 2^a$. In general when adding the bth ball, $X_{0,b}$ either decreases if the ball hits $X_{0,b-1}$ or stays the same otherwise. This gives the following recursive expression for $\mathbb{E}[|X_{0,b}|]$.

$$\mathbb{E}[|X_{0,b}|] = \mathbb{E}[|X_{0,b-1}|] - \Pr[f(x_b) \in X_{0,b-1}].$$

We claim that $\Pr[f(x_b) \in X_{0,b-1}] = |X_{0,b-1}| \cdot (2^{n-a})/(2^n - (b-1))$. This is because each $y \in X_{0,b-1}$ has not been mapped to, so there are 2^{n-a} values $y \,\|\, u$ that $\Pi(x_b \,\|\, 0^{n-a})$ can take that cause $f(x_b) = y$ out of $2^n - (b-1)$ total values to choose from. We can thus rewrite the recursive expression as

$$\begin{aligned}\mathbb{E}[|X_{0,b}|] &= \mathbb{E}[|X_{0,b-1}|] - \mathbb{E}[|X_{0,b-1}|] \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right) \\ &= \mathbb{E}[|X_{0,b-1}|] \cdot \left(1 - \frac{2^{n-a}}{2^n - (b-1)}\right).\end{aligned}$$

Solving for this expression, we get

$$\begin{aligned}\mathbb{E}[|X_{0,b}|] &= \mathbb{E}[|X_{0,0}|] \cdot \prod_{i=1}^{b} \left(1 - \frac{2^{n-a}}{2^n - (i-1)}\right) \\ &\leq 2^a \cdot \left(1 - \frac{2^{n-a}}{2^n}\right)^b = 2^a \cdot \left(1 - \frac{1}{2^a}\right)^b.\end{aligned}$$

Thus, $\mathbb{E}[|X_{0,2^a}|] \leq 2^a/e$.

For $X_{1,b}$, we first note that $X_{1,0} = 0$. Using our result for $X_{0,b}$, we can again write a recursive expression for $\mathbb{E}[|X_{1,b}|]$. The key point is that $X_{1,b-1}$ increases by one if $f(x_b)$ lands in $X_{0,b-1}$ and decreases by one if $f(x_b)$ lands in $X_{1,b-1}$.

$$\mathbb{E}[|X_{1,b}|] = \mathbb{E}[|X_{1,b-1}|] - \Pr[f(x_b) \in X_{1,b-1}] + \Pr[f(x_b) \in X_{0,b-1}].$$

We already have an expression for $\Pr[f(x_b) \in X_{0,b-1}]$. For $\Pr[f(x_b) \in X_{1,b-1}]$, we note that now each $y \in X_{1,b-1}$ has been mapped to exactly once. So there are only $2^{n-a}-1$ possible $y \,\|\, u$ values that $\Pi(x_b \,\|\, 0^{n-a})$ can take to cause $f(x_b) = y$. It follows that $\Pr[f(x_b) \in X_{1,b-1}] = |X_{1,b-1}| \cdot (2^{n-a} - 1)/(2^{n-a} - (b-1))$, so we can rewrite the recursive expression as

$$\begin{aligned}\mathbb{E}[|X_{1,b}|] &= \mathbb{E}[|X_{1,b-1}|] - \mathbb{E}[|X_{1,b-1}|] \cdot \left(\frac{2^{n-a}-1}{2^n - (b-1)}\right) + \mathbb{E}[|X_{0,b-1}|] \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right) \\ &= \mathbb{E}[|X_{1,b-1}|] \cdot \left(1 - \frac{2^{n-a}-1}{2^n - (b-1)}\right) + 2^a \cdot \prod_{i=1}^{b-1} \left(1 - \frac{2^{n-a}}{2^n - (i-1)}\right) \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right).\end{aligned}$$

We use the fact that $(1 - \frac{2^{n-a}}{2^n - i}) \leq (1 - \frac{2^{n-a}-1}{2^n - i})$ and $X_{1,0} = 0$, we can solve for an upper bound on this recurrence as follows.

$$\begin{aligned}
\mathbb{E}[|X_{1,b}|] &\leq \mathbb{E}[|X_{1,0}|] \cdot \prod_{i=1}^{b} \left(1 - \frac{2^{n-a}-1}{2^n - (i-1)}\right) \\
&\quad + \left(\sum_{i=1}^{b} \frac{2^n}{2^n - (i-1)}\right) \cdot \prod_{i=1}^{b} \left(1 - \frac{2^{n-a}-1}{2^n - (i-1)}\right). \\
&= \left(\sum_{i=1}^{b} \frac{2^n}{2^n - (i-1)}\right) \cdot \prod_{i=1}^{b} \left(1 - \frac{2^{n-a}-1}{2^n - (i-1)}\right) \\
&\leq \frac{b \cdot 2^n}{2^n - 2^a} \cdot \left(1 - \frac{2^{n-a}-1}{2^n}\right)^b \\
&\leq \frac{8b}{7} \cdot \left(1 - \frac{7}{8 \cdot 2^a}\right)^b,
\end{aligned}$$

where the last inequality follows since $a \leq n - 3$. Thus,

$$\mathbb{E}[|X_{1,2^a}|] \leq 2^a \cdot (8/7) \cdot e^{-7/8} \leq 2^a/2.$$

It follows that

$$\begin{aligned}
\sum_{i=2}^{2^a} \mathbb{E}[|X_{i,2^a}|] &= 2^a - \mathbb{E}[|X_{0,2^a}|] - \mathbb{E}[|X_{0,2^a}|] \\
&\geq 2^a \cdot (1 - 1/2 - 1/e) \\
&\geq 2^a/10.
\end{aligned}$$

Define the set S_i to be the set of functions f such that $|\{y : |f^{-1}(y)| \geq 2\}| = i$. Then

$$\begin{aligned}
\Pr[|f^{-1}(y)| \geq 2] &= \sum_{i=2}^{2^a} \Pr[f \in S_i] \cdot \frac{i}{2^a} \\
&= \frac{1}{2^a} \cdot \mathbb{E}[|\{y : |f^{-1}(y)| \geq 2\}|] \\
&\geq \frac{1}{2^a} \cdot \frac{2^a}{10} = \frac{1}{10},
\end{aligned}$$

as required. ■

4.3 Time-Space Tradeoffs for Inverting a Restricted Permutation

In this section, we prove a time-space tradeoff for inverting a restricted permutation. Let $n \in \mathbb{N}$, $a, b < n$, and let $\Pi \leftarrow \mathsf{Perm}(n)$ be a randomly chosen permutation. Consider the function $f\colon \{0,1\}^a \to \{0,1\}^b$ defined such that $f(x)$ outputs the first b bits of $\Pi(x \,\|\, 0^{n-a})$. We show that there exists an (S, T) adversary $\mathcal{A}$ that inverts f with advantage roughly $\Omega(\min(1, ST/2^{\min(a,b)}, (S^2T/2^{2\min(a,b)})^{1/3}))$. Additionally, we show that on input $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, $\mathcal{A}$ outputs a uniformly random pre-image $x' \in f^{-1}(y)$ if it succeeds.

We note that our attack closely follows the approach of Hellman [Hel80] and its extension from Fiat and Naor [FN99]. We provide the full details of the attack and analysis for completeness. We emphasize that our analysis differs from Hellman's analysis since our function f is not quite a random function. Still, we do not need the full generality of the result of Fiat and Naor that works for arbitrary functions. We also note that we show how to instantiate and analyze the "g" functions (see the proof for full details) used in Hellman's attack using only pairwise independence, whereas Fiat and Naor's result for arbitrary functions required k-wise independence for $k \approx T$.

Lemma 4.5. *Let $n \geq 1$, $a, b \leq n$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^b$ such that $f(x)$ consists of the first b bits output by $\Pi(x \,\|\, 0^{n-a})$. For any $S, T \in \mathbb{N}$ such that $S \geq 24\max(a,b)$, $2^{\min(a,b)} \geq 24S$, and $2^{\min(a,b)} \geq (S/T) \cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in inverting f on input $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$ with probability ϵ, where*

$$\epsilon \geq \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right).$$

Additionally, the following hold:

- *If the attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ succeeds at inverting $y = f(x)$, it outputs a uniform pre-image $x' \in f^{-1}(y)$ over the randomness of $\mathcal{A}_1$ and $\mathcal{A}_2$.*
- *For any fixed $x \in \{0,1\}^a$ and $y = f(x)$, the attack succeeds with probability at least*

$$\epsilon \geq \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST \cdot |f^{-1}(y)|}{2^a}, \left(\frac{S^2T|f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right).$$

Proof. We start by describing a "Hellman table," which is the main building block in our attack. It is helpful to think of a Hellman table as the preprocessing phase of an (S,T) adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for a restricted range of S and T. Our full adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ uses many Hellman tables to boost its success probability for any range of S and T.

Hellman tables. For any $m, \ell \in \mathbb{N}$, a Hellman table consists of m chains each of length ℓ and makes use of a common function $g\colon \{0,1\}^b \to \{0,1\}^a$ that we provide as input to $\mathcal{B}_1$ and $\mathcal{B}_2$. We will specify the function g to be used in our full construction. Let $h\colon \{0,1\}^b \to \{0,1\}^b$ be the function such that $h(y) = f(g(y))$. Denote by $h^{(j)}(y)$ the jth iteration of h on input y, so $h^{(j)}(y) = h(h(\ldots h(y)\ldots))$ consisting of j invocations of h. For the ith of m chains, we sample a random $y_{i,0} \leftarrow\!\!\$\ \{0,1\}^b$, and compute the remaining ℓ entries $y_{i,j} = h^{(j)}(y_{i,0})$ for all $j \in [\ell]$. The preprocessing phase $\mathcal{B}_1$ outputs the starting point $y_{i,0}$ and the final chain entry $y_{i,\ell}$ for each chain, $\sigma = \{(y_{i,0}, y_{i,\ell}) : i \in [m]\}$.

The online attacker $\mathcal{B}_2$ receives as input σ, some $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, and the description of a function g. $\mathcal{B}_2$ computes $h^{(j)}(y)$ for each $j \in [\ell]$ number of steps. If, for any $j \geq 1$, $h^{(j)}(y) = y_{i,\ell}$ for any final chain entry stored in σ, $\mathcal{B}_2$ computes $y_{i,k} = h^{(k)}(y_{i,0})$ for each $k \in [\ell]$. If $y_{i,k} = y$ for some $k \in [\ell]$, $\mathcal{B}_2$ outputs $x' = g(y_{i,k-1})$, and outputs $\perp$ if no such k is found.

To see the correctness guarantee, consider the set of points $\mathsf{Table} = \{y_{i,j} : i \in [m], j \in [\ell]\}$. We claim that if the challenge y received by $\mathcal{B}_2$ is in the set Table, then $\mathcal{B}_2$ outputs a valid pre-image $x' \in f^{-1}(y)$. Suppose $y \in S$. Then, by definition of S, we know that $y = h^{(j)}(y_{i,0})$ for some $i \in [m]$, $j \in [\ell]$. But that implies that $y_{i,\ell} = h^{(\ell)}(y_{i,0}) = h^{(\ell-j)}(y)$, so $\mathcal{B}_2$ will compute $y_{i,\ell}$ during its online attack. At this point, $\mathcal{B}_2$ will then compute $y_{i,k} = h^{(k)}(y_{i,0})$ for each $k \in [\ell]$ and see that $y_{i,j} = y$. It outputs $x' = g(y_{i,j-1})$, and it holds that $f(x') = y$ by construction, as required.

The full construction. Our full adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is as follows. For any space bound S and time bound T, we compute the parameter $\gamma \in \mathbb{N}$ as follows,

$$\gamma = \left\lceil \max\left(\frac{ST}{2^{\min(a,b)}}, \left(\frac{ST^2}{2^{\min(a,b)}}\right)^{1/3}\right)\right\rceil.$$

In the pre-processing phase, we sample functions $g_1, \ldots, g_\gamma\colon \{0,1\}^b \to \{0,1\}^a$ from a pairwise independent hash family where additionally g_i and g_j are independent for all $i \neq j \in [\gamma]$. This requires at most $6 \cdot \max(a,b)$ bits to describe, using the construction of Claim 4.9. Using these functions, we construct γ Hellman tables of m chains each of length ℓ where,

$$m = \left\lfloor \frac{S - 6\max(a,b)}{2\gamma b} \right\rfloor, \ell = \left\lfloor \frac{T}{6\gamma} \right\rfloor - 1.$$

For each $t \in [\gamma]$, the function g_t induces a new function $h_t(x) = f(g_t(x))$ used to construct the chains for the tth Hellman table. For all $t \in [\gamma]$, let $\sigma_t \leftarrow \mathcal{B}_1(f, g_t)$ be the output of $\mathcal{B}_1$, which depends on the function g_t as well as the choice of starting points $y_{t,1,0}, \ldots, y_{t,m,0}$ sampled by $\mathcal{B}_1$ for the tth table. $\mathcal{A}_1$ outputs $\{\sigma_t : t \in [\gamma]\}$ plus the descriptions of $g_1, \ldots, g_\gamma$.

The online attacker $\mathcal{A}_2$ receives pre-processing $\sigma = (\{\sigma_t : t \in [\gamma]\}, (g_1, \ldots, g_\gamma))$ and the challenge $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$. For all $t \in [k]$, it computes a candidate pre-image $x' = \mathcal{B}_2(\sigma_t, y, g_t)$. If all such x' are invalid, $\mathcal{A}_2$ outputs $\perp$. Otherwise $\mathcal{A}_2$ outputs the first x' such that $f(x') = y$.

By the correctness guarantee of $\mathcal{B}$, it follows that if $\mathcal{A}_2$ outputs a non-$\perp$ value, it is valid pre-image of y. Let $x' \neq \perp$ be the pre-image output by $\mathcal{A}_2$, we claim that x' is a uniformly random pre-image of y. Suppose x' was first output by $\mathcal{B}_2(\sigma_t, y, g_t)$. Let $P = f^{-1}(y)$. We want to show that for all $\tilde{x} \in P$, $\Pr[x' = \tilde{x}] = 1/|P|$. By construction, it holds that $x' = g_t(y_{t,i,j-1})$ for some index $j \in [\ell]$ in chain $i \in [m]$. Thus, x' is uniformly distributed in $\{0,1\}^a$, so it is equally like to be equal to any $\tilde{x} \in P$, as required.

Analyzing the attack's efficiency. For space efficiency, we remark that $|\sigma_t| = 2mb$ for each table $t \in [k]$. Additionally, by Claim 4.9, the functions $g_1, \ldots, g_k$ requires $6 \cdot \max(a, b)$ bits to store. So, the space usage overall is

$$|\sigma| = 2mb\gamma + 6 \cdot \max(a, b) \leq S,$$

since $m \leq (S - 6 \cdot \max(a, b))/(2\gamma b)$ by construction.

To analyze the time efficiency, consider the attacker $\tilde{\mathcal{A}}_2$ that does not stop at T queries. We will revisit the analysis of $\mathcal{A}_2$ as opposed to $\tilde{\mathcal{A}}_2$ at the end of the proof after analyzing $\tilde{\mathcal{A}}_2$'s success probability. Let $\tilde{T}$ be the random variable describing the number of queries that $\tilde{\mathcal{A}}_2$ makes. Let $\widetilde{\mathsf{Success}}$ be the event that $\tilde{\mathcal{A}}_2$ finds a valid pre-image in this experiment. We analyze the expected number of queries $\tilde{T}$ conditioned on $\tilde{\mathcal{A}}_2$ finding a valid pre-image, $\mathbb{E}[\tilde{T} \mid \widetilde{\mathsf{Success}}]$.

For each $t \in [\gamma]$, $\mathcal{B}_2(\sigma_t, y, g_t)$ always makes at least ℓ oracle calls to f in computing $h_t^{(k)}(y)$ for all $k \in [\ell]$. For each $i \in [m]$ and $j \in [\ell]$ such that $h_t^{(j)}(y) = y_{t,i,\ell}$, $\mathcal{B}_2$ potentially makes ℓ more oracle calls to f. However, each time that $\mathcal{B}_2$ starts to search for y in one of these chains, it is either because (a) y is actually in Table or (b) $\mathcal{B}_2$ got unlucky and hit some $y_{t,i,\ell}$ value that was not preceded by y in the chain. Whenever case (a) occurs, $\mathcal{B}_2$ can terminate early at a cost of at most ℓ extra queries to f. Case (b) is known as a "false alarm" [Hel80] and causes $\mathcal{B}_2$ to make ℓ more queries to f. We show in Claim 4.7 that the probability a single chain in any table results in a false alarm is at most $(\ell+1)^2/(2^{\min(a,b)})$, so the expected number of false alarms per table is at most $m(\ell+1)^2/(2^{\min(a,b)})$. Thus, given that $\tilde{\mathcal{A}}_2$ succeeds, it will make at most $2\ell\gamma$ queries plus ℓ times the number of false alarms across all tables, so

$$\mathbb{E}[\tilde{T} \mid \widetilde{\mathsf{Success}}] \leq 2\ell\gamma + \frac{\gamma m \ell \cdot (\ell+1)^2}{2^{\min(a,b)}}.$$

Since $m \leq (S/(2\gamma b))$ and $\ell + 1 \leq T/(6\gamma)$, it follows that $m(\ell+1)^2 \leq (S/(2\gamma b)) \cdot (T^2/(36\gamma^2)) \leq N$, by construction. This implies that

$$\mathbb{E}[\tilde{T} \mid \widetilde{\mathsf{Success}}] \leq 3\ell\gamma \leq T/2,$$

where the last inequality follows since $\ell \leq T/(6\gamma)$.

Analyzing the attack's success probability. Again, we first analyze the hypothetical attack $\tilde{\mathcal{A}}_2$ that does not stop at T queries, and we will analyze $\mathcal{A}_2$'s success probability at the end of the proof. Recall that $\widetilde{\mathsf{Success}}$ is the probability that $\tilde{\mathcal{A}}_2$ outputs a pre-image $x' \in f^{-1}(y)$.

For each table $t \in [k]$, let $\mathsf{Table}_t = \{y_{t,i,j} = h_t^{(j)}(y_{t,i,0}) : i \in [m], j \in [\ell], y_{t,i,0} \in \sigma_t\}$ be the set of points that table t covers. For the full construction, we define $\mathsf{Table} = \cup_{t \in [\gamma]} \mathsf{Table}_t$. Since $\mathcal{A}_2$ succeeds whenever some $\mathcal{B}_2(\sigma_t, y, g_t)$ succeeds, it follows that $\mathcal{A}_2$ succeeds if $y \in \mathsf{Table}_t$ for any $t \in [\gamma]$. We first analyze the success probability for $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, and then describe why it holds for any fixed x as well. Note that over a random permutation Π, $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$ is uniformly distributed over the range of f. Therefore, it follows that $\Pr[\widetilde{\mathsf{Success}}] \geq \mathbb{E}[|\mathsf{Table}|]/2^{\min(a,b)}$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, so it

suffices to lower bound the expected size of $|\mathsf{Table}|$. By Claim 4.8, it follows that

$$\Pr[\widetilde{\mathsf{Success}}] = \frac{\mathbb{E}[|\mathsf{Table}|]}{2^{\min(a,b)}} \geq \frac{\gamma m \ell}{6 \cdot 2^{\min(a,b)}}.$$

Now for any fixed $x \in \{0,1\}^a$ and $y = f(x)$, recall that each $y_{t,i,j}$ value added to Table from Claim 4.8 is of the form $f(g_t(y_{t,i,j-1}))$. So $g_t(y_{t,i,j-1})$ is a uniformly random value in $\{0,1\}^a$. Thus, the probability that $f(x) = y_{t,i,j}$ is at least $|f^{-1}(y)|/2^a$, and the probability that $f(x) \in \mathsf{Table}$ is at least $\mathbb{E}[|\mathsf{Table}|] \cdot |f^{-1}(y)|/2^a$.

Putting it all together. Let ϵ be the probability that $\mathcal{A}_2$ succeeds at inverting $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$.

$$\begin{aligned}
\epsilon &= \Pr[\mathsf{Success}] = \Pr[\widetilde{\mathsf{Success}} \wedge T' < T] \\
&= \Pr[T' < T \mid \widetilde{\mathsf{Success}}] \cdot \Pr[\widetilde{\mathsf{Success}}] \\
&\geq \left(1 - \frac{\mathbb{E}[T' \mid \widetilde{\mathsf{Success}}]}{T}\right) \cdot \frac{\gamma m \ell}{6 \cdot 2^{\min(a,b)}} \\
&\geq \frac{\gamma m \ell}{12 \cdot 2^{\min(a,b)}} \\
&\geq \frac{\gamma}{12 \cdot 2^{\min(a,b)}} \cdot \frac{S - 6\max(a,b)}{2\gamma b} \cdot \left(\frac{T}{6\gamma} - 1\right) \\
&\geq \frac{ST - 6\gamma S - 6T\max(a,b)}{144\gamma b \cdot 2^{\min(a,b)}} \\
&\geq \frac{ST}{288\gamma b \cdot 2^{\min(a,b)}},
\end{aligned}$$

where the last line holds if $T \geq 24\gamma$ and $S \geq 24\max(a,b)$. We note that $T \geq 24\gamma$ as long as

$$T \geq 24 \cdot \lceil \max(ST/2^{\min(a,b)}, (ST^2/2^{\min(a,b)})^{1/3}) \rceil,$$

which holds as long as $2^{\min(a,b)} \geq 24S$ and $2^{\min(a,b)} \geq (S/T) \cdot 24^3$. We assume these hold in the statement of our lemma. Note that for a fixed $x \in \{0,1\}^a$ and $y = f(x)$, the bound holds for $\epsilon \geq \frac{ST \cdot |f^{-1}(y)|}{288\gamma b \cdot 2^a}$.

Finally, we note that either $\gamma = 1$, or we set γ such that

$$\gamma \geq \lceil \max(ST/2^{\min(a,b)}, (ST^2/2^{\min(a,b)})^{1/3}) \rceil .$$

$\gamma = 1$ corresponds to the case where $ST^2 \leq 2^{\min(a,b)}$. In this case, we get,

$$\epsilon \geq \left(\frac{1}{288 \cdot b}\right) \cdot \left(\frac{ST}{2^{\min(a,b)}}\right)$$

Otherwise if $\gamma > 1$, which corresponds to the case where $ST^2 > 2^{\min(a,b)}$, we get the following bound,

$$\begin{aligned}
\epsilon &\geq \min\left(\frac{ST}{288 \cdot (ST/2^{\min(a,b)}) \cdot b \cdot 2^{\min(a,b)}}, \frac{ST}{288 \cdot ((ST^2/2^{\min(a,b)})^{1/3}) \cdot b \cdot 2^{\min(a,b)}}\right) \\
&= \frac{1}{288 \cdot b} \cdot \min\left(1, \left(\frac{S^2 T}{2^{2\min(a,b)}}\right)^{1/3}\right).
\end{aligned}$$

However, note that if $ST^2 > 2^{\min(a,b)}$, then

$$\left(\frac{S^2 T}{2^{2\min(a,b)}}\right)^{1/3} > \left(\frac{ST}{2^{\min(a,b)}}\right),$$

so the second bound dominates in this case. This implies that the probability of success is always at least

$$\epsilon \geq \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right),$$

as required.

Doing the same analysis for fixed $x \in \{0,1\}^a$ and $y = f(x)$, we get that the probability of success it as least

$$\epsilon \geq \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST|f^{-1}(y)|}{2^a}, \left(\frac{S^2T|f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right).$$

■

Claim 4.6. *Let $n \geq 1$, $a, b \leq n$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^b$ such that $f(x)$ consists of the first b bits output by $\Pi(x \,\|\, 0^{n-a})$. Then*

$$\Pr\left[\begin{array}{l} x_1, x_2 \leftarrow\!\!\$\ \{0,1\}^a, \\ \Pi \leftarrow \mathsf{Perm}\left(\{0,1\}^n\right) \end{array} : f(x_1) = f(x_2)\right] \leq \frac{1}{2^a} + \frac{1}{2^b}.$$

Proof. Let Π be a random permutation, $x_1, x_2 \leftarrow\!\!\$\ \{0,1\}^a$, and $y \,\|\, u = \Pi(x_1 \,\|\, 0^{n-a})$ for $y \in \{0,1\}^b$. First, if $x_1 = x_2$, then clearly $f(x_1) = f(x_2)$. This happens with probability $1/2^a$. Otherwise, for any $x_2 \in \{0,1\}^a$ such that $x_1 \neq x_2$, we note that $\Pi(x_2 \,\|\, 0^{n-a})$ can take any value other that $y \,\|\, u$. There are $(2^{n-b} - 1)$ such values of the form $y \,\|\, u'$ for $u' \neq u$, so the probability over a random Π that the first b bits of $\Pi(x_2 \,\|\, 0^{n-a}) = y$ is $\frac{2^{n-b}-1}{2^n-1}$. This is at most $1/2^b$ since

$$\frac{1}{2^b} = \frac{2^{n-b}-1}{2^n-1} \cdot \frac{2^n-1}{2^n-2^b},$$

and $(2^n - 1)/(2^n - 2^b) \geq 1$ for $b \geq 1$. Thus,

$$\Pr[f(x_1) = f(x_2)] \leq \Pr[x_1 = x_2] + \Pr[f(x_1) = f(x_2) \mid x_1 \neq x_2] \leq \frac{1}{2^a} + \frac{1}{2^b},$$

as required. ■

Claim 4.7. *The probability that a chain results in a false alarm is at most $\frac{(\ell+1)^2}{2^{\min(a,b)}}$.*

Proof. Let $t \in [k]$ be the table considered and $i \in [m]$ the chain in the table. Let $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$ be the challenge. Let $\tilde{y}_0 = y$ and define $\tilde{y}_j = h_t^{(j)}(y)$ for all $j \in [\ell]$ be the steps computed in the online phase. Recall that $y_{t,i,0} \leftarrow\!\!\$\ \{0,1\}^a$ is the initial point of the chain, and $y_{t,i,j} = h_t^{(j)}(y_{t,i,0})$ for each $j \in [\ell]$ are the subsequent points.

A false alarm occurs if $\tilde{y}_j = y_{t,i,j'}$ for any $j \in [\ell]$ and $j' \geq j$. This happens if the initial chain points are equal, $\tilde{y}_0 = y_{t,i,0}$, since then the chains immediately collide and we cannot find a pre-image for y. Note that if $y = \tilde{y}_0 = y_{t,i,j'}$ for some $j' > 0$, then this is not a false alarm as this allows the attacker to invert. Otherwise, there is a false alarm whenever $\tilde{y}_j = y_{t,i,j'}$ for any $j' \geq j > 0$, given that the preceding values are not equal. By a union bound, we upper bound this probability as

$$\begin{aligned} &\Pr[\tilde{y}_0 = y_{t,i,0}] + \Pr[\exists j \in [\ell], j' \geq j, \tilde{y}_j = y_{t,i,j'}] \\ &= \Pr[\tilde{y}_0 = y_{t,i,0}] + \sum_{j' \geq j > 0} \Pr[\tilde{y}_j = y_{t,i,j'} \mid \forall \hat{j}' \geq \hat{j} > 0 \text{ s.t. } \hat{j}' < j' \wedge \hat{j} < j,\ \tilde{y}_{\hat{j}} \neq y_{t,i,\hat{j}'}]. \end{aligned}$$

First, note that the probability that $\tilde{y}_0 = y_{t,i,0}$ is at most $1/2^b$, since $y_{t,i,0}$ is a uniformly random value in $\{0,1\}^b$. Next, for any $j' \geq j > 0$, $y_{t,i,j'} = f(g_t(y_{t,i,j'-1}))$ and $\tilde{y}_j = f(g_t(\tilde{y}_{j-1}))$. Given that $y_{t,i,j'-1} \neq \tilde{y}_{j-1}$,

it holds that $g_t(y_{t,i,j'-1})$ and $g_t(\tilde{y}_{j-1})$ are uniform and independent by pairwise independence of g_t. Thus, they collide with probability at most $1/2^a + 1/2^b$. It follows that each term in the sum above is at most $1/2^a + 1/2^b$, so the probability that a chain results in a false alarm is at most

$$\left(1 + \frac{\ell \cdot (\ell+1)}{2}\right) \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right) \leq \frac{(\ell+1)^2}{2^{\min(a,b)}}.$$

■

Claim 4.8. $\mathbb{E}[|\mathsf{Table}|] \geq \gamma m \ell/6$.

Proof. We consider the process of adding each element in each chain for each table one at a time. To capture this, we define two sets, $\mathsf{Prog}_{t,i,j}$ and $\mathsf{TableProg}_{t,i,j}$. For each table t, chain $i \in [m]$ with starting point $y_{t,i,0} \in \sigma_t$, and value $j \in [\ell]$, recall that $y_{t,i,j} = h_t^{(j)}(g_t(y_{t,i,0}))$ is the jth element in the chain. Let $\mathsf{Prog}_{t,i,j}$ be the set containing all preceding $y_{t',i',j'}$ values before $y_{t,i,j}$ is computed. We also define a similar set per table, $\mathsf{TableProg}_{t,i,j}$, which is the set of $y_{t',i',j'}$ values for fixed $t' = t$ before $y_{t,i,j}$ is computed. For convenience, we define $\mathsf{Prog}_{t,i,0} = \mathsf{Prog}_{t,i,1}$ and $\mathsf{TableProg}_{t,i,0} = \mathsf{TableProg}_{t,i,1}$. Thus, it follows that the expected size of Table is the probability that each value $y_{t,i,j}$ is new and not contained in the set $\mathsf{Prog}_{t,i,j}$, so

$$\begin{aligned}
\mathbb{E}[|\mathsf{Table}|] &= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell} \Pr[y_{t,i,j} \notin \mathsf{Prog}_{t,i,j}] \\
&= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell} \Pr[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1} \wedge y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \\
&= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell} \Pr[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1}] \cdot \Pr[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}],
\end{aligned}$$

where the last equality holds since the function g_t is independent of each previous $g_{t'}$ for $t' < t$. We analyze each of these two terms in the summation separately.

First, we analyze $\Pr[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1}]$. Recall that $y_{t,i,j} = f(g_t(y_{t,i,j-1}))$ and consider $y_{t',i',j'} = f(g_{t'}(y_{t',i',j'-1}))$ for any $t' < t, i' \in [m], j' \in [\ell]$. Since the function g_t and $g_{t'}$ are universal and pairwise independent for all $t' < t$, it holds that $g_t(y_{t,i,j-1})$ and $g_t(y_{t',i',j'-1})$ are independent and uniform on $\{0,1\}^a$. Thus, by Claim 4.6,

$$\Pr[y_{t,i,j} = y_{t',i',j'}] = \Pr[f(g_t(y_{t,i,j-1})) = f(g_{t'}(y_{t',i',j'-1}))] \leq \frac{1}{2^a} + \frac{1}{2^b}.$$

Taking a union bound over all such t', i', j', we have that

$$\Pr[\exists t' < t, i \in [m], j \in [\ell], y_{t,i,j} = y_{t',i',j'}] = \Pr[y_{t,i,j} \in \mathsf{Prog}_{t,1,1}] \leq (t-1)m\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right).$$

Next, we analyze $\Pr[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}]$. Note that $y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}$ holds only if $y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}$ for all $1 \leq j' < j$ and $y_{t,i,0} \notin \mathsf{TableProg}_{t,i,1}$, since otherwise the chain has already collided with a previous chain in the table. It follows that, for all $t \in [\gamma]$, $i \in [m]$,

$$\begin{aligned}
&\Pr[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \\
&\geq \Pr[\forall 0 \leq j' \leq j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}] \\
&= \Pr[y_{t,i,0} \notin \mathsf{TableProg}_{t,i,0}] \\
&\quad \cdot \Pr[y_{t,i,1} \notin \mathsf{TableProg}_{t,i,1} \mid y_{t,i,0} \notin \mathsf{TableProg}_{t,i,0}] \cdot \ldots \\
&\quad \cdot \Pr[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j} \mid \forall 0 \leq j' < j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}].
\end{aligned}$$

We analyze each individual term in the product above. Specifically, consider the term corresponding to $\hat{j}$, $\Pr[y_{t,i,\hat{j}} \notin \mathsf{TableProg}_{t,i,\hat{j}} \mid \forall j' < \hat{j}, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}]$. First, we consider the case where $\hat{j} = 0$. In this case, $y_{t,i,0} \leftarrow\!\!\$\ \{0,1\}^b$ is a uniformly random value. So the probability is it in $\mathsf{TableProg}_{t,i,0}$ is at most $|\mathsf{TableProg}_{t,i,0}|/2^b \leq i\ell/2^b$. For $\hat{j} > 0$, we are given that $y_{t,i,\hat{j}-1} \notin \mathsf{TableProg}t, i, \hat{j} - 1$. Thus, by pairwise independence of g_t, we have that $g_t(y_{t,i,\hat{j}-1})$ and $g_t(y_{t,i',j'})$ are uniform and independent for all earlier values of i', j' in table t. Thus, $\Pr[y_{t,i,\hat{j}} = y_{t,i',j'}] = \Pr[f(g_t(y_{t,i,j-1})) = f(g_{t'}(y_{t',i',j'-1}))] \leq \frac{1}{2^a} + \frac{1}{2^b}$ by Claim 4.6, and by a union bound

$$\Pr[y_{t,i,j} \in \mathsf{TableProg}_{t,i,j} \mid \forall 0 \leq j' < j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}] \leq i\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right).$$

Multiplying out all $j+1$ terms in the product, we get that

$$\Pr[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \geq \left(1 - i\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right)\right)^{j+1}.$$

Combining the previous two results, we get that

$$\begin{aligned}
\mathbb{E}[|\mathsf{Table}|] &\geq \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell}\left(1 - (t-1)m\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right)\right) \cdot \left(1 - i\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right)\right)^{j+1} \\
&\geq \sum_{t=1}^{\gamma}\left(1 - \frac{2(t-1)m\ell}{2^{\min(a,b)}}\right) \cdot \sum_{i=1}^{m}\sum_{j=1}^{\ell}\left(1 - \frac{2i\ell}{2^{\min(a,b)}}\right)^{j+1} \\
&\geq \sum_{t=1}^{\gamma}\left(1 - \frac{2(t-1)m\ell}{2^{\min(a,b)}}\right) \cdot \sum_{i=1}^{m}\sum_{j=1}^{\ell} e^{-(2i\ell(j+1))/(2^{\min(a,b)}-2i\ell)} \\
&\geq \gamma m\ell \cdot \left(1 - \frac{(\gamma-1)m\ell}{2^{\min(a,b)}}\right) \cdot e^{-(4m\ell^2)/(2^{\min(a,b)}-2m\ell)}.
\end{aligned}$$

In the third inequality above, we use the fact that $(1-1/n)^{n-1} > 1/e$ for all $n \in \mathbb{N}$, so if we set $n = 2i\ell/2^{\min(a,b)}$, it follows that $\left(1 - \frac{2i\ell}{2^{\min(a,b)}}\right)^{(2^{\min(a,b)}-2i\ell)/(2i\ell)} \geq e^{-1}$.

For the above, we note that $e^{-(4m\ell^2)/(2^{\min(a,b)}-2m\ell)} \geq 1/e$ by the following sequence of inequalities,

$$4m\ell^2 + 2m\ell \leq 6m\ell^2 \leq 6\left(\frac{S}{2\gamma b}\right)\left(\frac{T^2}{36\gamma^2}\right) \leq \frac{ST^2}{\gamma^3} \cdot \frac{1}{12b} \leq 2^{\min(a,b)},$$

where the last inequality holds since $\gamma \geq ((ST^2)/(2^{\min(a,b)}))^{1/3}$. Next, we note that $(1-(\gamma-1)m\ell/(2^{\min(a,b)}) \geq 1/2$ since

$$2(\gamma-1)m\ell \leq 2\gamma\left(\frac{S}{2\gamma b}\right)\left(\frac{T}{6\gamma}\right) \leq \frac{ST}{6\gamma} \leq 2^{\min(a,b)},$$

where the last inequality holds since $\gamma \geq ST/2^{\min(a,b)}$. Combining these two facts, it follows that

$$\mathbb{E}[|\mathsf{Table}|] \geq \gamma m\ell \cdot (1/2) \cdot (1/e) \geq \gamma m\ell/6,$$

as required. ∎

Claim 4.9 ([FN99])**.** *Let $a, b \in \mathbb{N}$. For any $q \in \mathbb{N}$, there exists a family G of universal q-wise independent functions from $\{0,1\}^b$ to $\{0,1\}^a$ such that:*

1. *k functions $g_1, \ldots, g_k$ from G can be sampled using $2(q+1) \cdot \max(a,b)$ bits, and*
2. *for each $i \neq j \in [k]$, g_i and g_j are independent.*

Proof. The construction is taken from [FN99]. Consider the finite field $\mathbb{F}$ of size $2^{\max(a,b)}$. The construction samples $2(q+1)$ random elements from $\mathbb{F}$, $\alpha_0, \ldots, \alpha_q$ and $\beta_0, \ldots, \beta_q$. Let g'_i be the polynomial over F whose jth coefficient is $\alpha_j \cdot i + \beta_j$, and then set $g_i(x) = g'_i(x) \mod 2^a$. Pairwise independence follows since each g_i is a random degree-2 polynomial. Independence of g_i, g_j for $i \neq j$ holds since the coefficients themselves are uniform and independent between any two functions. ■

5 Impossibility Results

We next give impossibility results for attacks for 1-block and 2-block collisions for sponge hashing. This consists of upper bounding the best possible advantage of any (S, T) adversary.

5.1 Advantage Upper Bound for B=1

We prove an upper bound for the advantage of an adversary in finding a 1-block collision for the sponge construction. We use the result of [CDG18] which relates the advantage upper bound of an adversary in the AI-RPM to that in BF-RPM (bit-fixing RPM). We also argue why the bit-fixing technique cannot be used to prove a better advantage upper bound for $B = 2$ than the upper bounded for collisions of unbounded length.

In the BF model, the first stage of a two stage adversary fixes P points of the permutation Π. The permutation Π is then sampled uniformly at random from the set of permutations with the restriction of the points fixed by the first stage adversary. The second stage adversary can make T queries to Π, Π^{-1} and has to accomplish some given task involving Π. In Fig. 6, we formally define the game $\mathsf{G}^{\mathsf{bf\text{-}cr}}$ which captures the the bounded length collision resistance in the BF-RPM.

For parameters $c, r, B \in \mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of sponge in the BF-RPM is

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) = \Pr\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}(\mathcal{A}) = \mathsf{true}\right] .$$

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

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) \right\},$$

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

We restate the following proposition from [CDG18] that relates upper bounds $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T)$ in terms of $\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T)$.

Proposition 5.1. For any $S, T, P, B \in \mathbb{N}$ and $\gamma > 0$ such that $P \geq (S + \log \gamma^{-1})T$ we have that

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T) \leq \mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T) + \gamma .$$

We prove the following theorem that upper bounds the advantage of finding 1-block collisions for sponge in the AI-RPM.

Theorem 5.2. *For all $S, T, c, r \in \mathbb{N}$*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,1}(S,T) \leq \frac{2(S+c)T+1}{2^c} + \frac{T^2}{2^r} .$$

Proof. The proof of the theorem follows directly from from setting $P = (S+c)T$, $\gamma = 2^{-c}$ in Proposition 5.1 and the claim below.

Claim 5.3. *For all $P, T, c, r \in \mathbb{N}$*

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,1}(P,T) \leq P/2^c + T^2/2^{r+1} .$$

Game $\mathsf{G}^{\mathsf{bf\text{-}cr}}_{c,r,B}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

1. $(I, Q) \leftarrow \mathcal{A}_1$ // $|I| = |Q|$
2. $S \leftarrow \emptyset$
3. For $j \in [|I|]$ set $\Pi(I[j]) \leftarrow Q[j]$, $\mathsf{S}_1 \leftarrow \mathsf{S}_1 \cup \{Q[j]\}$, $\mathsf{S}_2 \leftarrow \mathsf{S}_2 \cup \{I[j]\}$
4. For $j \in [2^{c+r}] \setminus \mathsf{S}_2$ sample $\Pi[j] \leftarrow\!\!\$\ [2^{c+r}] \setminus \mathsf{S}_1$, set $\mathsf{S}_1 \leftarrow \mathsf{S}_1 \cup \{\Pi[j]\}$
5. $(\alpha, \alpha') \leftarrow\!\!\$\ \mathcal{A}_2^{\Pi, \Pi^{-1}}(I, Q, \mathsf{IV})$
6. Return true if:
7. $\alpha \neq \alpha'$ and $|\alpha|, |\alpha'|$ are at most B blocks long, and $\mathsf{Sp}_\Pi(\mathsf{IV}, \alpha) = \mathsf{Sp}_\Pi(\mathsf{IV}, \alpha')$

Figure 6: The bounded-length collision resistance game of salted sponge based hash in the BF-RPM, denoted $\mathsf{G}^{\mathsf{bf\text{-}cr}}_{c,r,B}$.

Proof. If IV which is given as input to $\mathcal{A}_2$ is such that $(*, \mathsf{IV})$ not among any of the P points, IV's then the probability that $\mathcal{A}_2$ finds a 1-block collision is bounded by the birthday bound i.e., $T^2/2^{r+1}$. The probability that $(*, \mathsf{IV})$ is among the P fixed points is at most $P/2^c$ and hence the claim follows. ■

■

Bit-fixing upper bound for $B = 2$ is trivial. We note that we cannot prove an advantage upper bound better than $O(ST^2/2^c)$ for $B = 2$, since there is a (P, T) adversary that has advantage $PT/2^c$ in finding 2-block collisions. This adversary chooses $P/2$ tuples (y_i, x_i, x_i') for $i \in [P/2]$ such that $y_i \in \{0,1\}^c$, $x_i, x_i' \in \{0,1\}^r$, $x_i \neq x_i'$. It sets the P points of Π such that $\Pi(x_i, y_i)[1] = \Pi(x_i', y_i)[1]$ for $i \in [P/2]$.

The second stage of the adversary on getting IV as input queries (m, IV) for T different m's. With probability roughly $PT/2^c$ it succeeds in finding m such that $\Pi(m, \mathsf{IV}) = (*, y_i)$ for some y_i chosen by the first stage adversary. In this case it can output a 2 block collision of $(m, \Pi(m, \mathsf{IV})[1] \oplus x_i), (m, \Pi(m, \mathsf{IV})[1] \oplus x_i')$ with probability $\Omega(PT/2^c)$. This would translate to an upper bound no better than $O(ST^2/2^c)$.

5.2 Advantage Upper Bound for $B = 2$

In this section we prove an upper-bound the advantage of an adversary in finding a 2-block collision for the sponge construction in the AI-RPM, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$ described in Fig. 4. First, without loss of generality, in what follows we assume that the adversary is deterministic. This is because we can transform any probabilistic attacker into a deterministic one by hard-wiring the best randomness (see Adleman [Adl78]).

We reduce the task of bounding the advantage of an attacker in finding a 2-block collision in the sponge construction, to a "multi-instance" game where the adversary does not have a preprocessing phase but rather only has non-uniform auxiliary input, chosen *before* the random permutation Π. The latter game is easier to analyze. This is in line with the work of Akshima et al. [ACDW20].

We define the following "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,B,u}(\sigma, \mathcal{A}_2)$, 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. $\Pi \leftarrow\!\!\$\ \mathsf{Perm}(\{0,1\}^{c+r})$
2. $U \leftarrow\!\!\$\ \binom{\{0,1\}^c}{u}$
3. Define $\mathcal{A}_1$ to be the algorithm that always outputs the string σ.
4. Return true if $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)) = \mathsf{true}$ for every $\mathsf{IV} \in U$. Otherwise, return false.

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

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

Lemma 5.4 (Reducing the problem to the multi-instance game)**.** *Fix $c, r, B, S, T, u \in \mathbb{N}$. Then,*

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

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

We refer the reader to [GK22] for a proof.

We next prove an upper bound on the advantage of any auxiliary-input adversary in finding a 2-block collision for the sponge construction. The main theorem is stated next.

Theorem 5.5. *For any $c, r, S, T \in \mathbb{N}$ and fixing $\hat{S} := S + c$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,2}(S,T) \leq \left(2^7 e \cdot \max \left\{ \frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{\hat{S}T}{2^{c-3}}, \frac{\hat{S}^2T^4}{2^{2c-2}} \right\} \right) + \frac{1}{2^c} .$$

Theorem 5.5 follows as a direct corollary of Lemma 5.4 together with the following lemma, setting $u = S + c$ and observing that **(1)** the lemma holds trivially when $\frac{T^2}{2^{r-1}} > 1$ and **(2)** $\frac{uT^3}{2^{c+r-2}} \leq \frac{uT}{2^{c-1}}$ whenever $\frac{T^2}{2^{r-1}} \leq 1$.

Lemma 5.6 (Hardness of the multi-instance game)**.** *Fix c, r, T, $u \in \mathbb{N}$ and $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,2,u}(\sigma,\mathcal{A}_2) \leq \left(2^7 \cdot e \cdot \max \left\{ \frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}} \right\} \right)^u .$$

The rest of this section is devoted to the proof of Lemma 5.6.

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 collisions of length 2 in a sponge with respect to the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,2,u}(\sigma,\mathcal{A}_2)$. Recall that in this game $\mathcal{A}_2$ needs to find proper collisions for u randomly chosen IVs, denoted U. The main idea in the proof is to use any such adversary $(\sigma,\mathcal{A}_2)$ in order to represent the permutation Π *as well as* the set of random IVs U with as few bits as possible. If the adversary is "too good to be true" we will get an impossible representation, contradicting Proposition 3.2.

Setup. Denote

$$\zeta^* := \log \left(\left(2^5 \cdot 4e \cdot \max \left\{ \frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}} \right\} \right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})! \right).$$

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

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,2,u}(\mathcal{A}) := \zeta > \zeta^*. \tag{5}$$

Define $\mathcal{G}$ to be the set of permutations-sets of IV pairs for which the attacker succeeds in winning the game for every IV in the set relative to the permutation, That is,

$$\mathcal{G} = \left\{ (U,\Pi) \;\middle|\; \begin{array}{c} U \in \binom{\{0,1\}^c}{u}, \\ \Pi \in \mathsf{Perm}(\{0,1\}^{c+r}), \end{array} \forall \mathsf{IV} \in U : \mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A}) = \mathsf{true} \right\}.$$

Recall that ζ is defined to be the advantage of $\mathcal{A}$ in the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,2,u}(\mathcal{A})$ in which Π and U are chosen uniformly, and then $\mathcal{A}$ needs to find a collision with respect to every one of the u IVs in U. Therefore,

$$|\mathcal{G}| = \zeta \cdot \binom{2^c}{u} \cdot (2^{c+r})!.$$

In what follows we define an encoding and a decoding procedure such that the encoding procedure gets as input U, Π such that $U \in \binom{\{0,1\}^c}{u}$ and $\Pi \in \mathsf{Perm}(\{0,1\}^{c+r})$, and it outputs an L bit string, where $L = \log\left(\zeta^* \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right)$. The decoding procedure takes as input the string L and outputs U^*, Π^*. It will hold that $U^* = U$ and $\Pi^* = \Pi$ with probability ζ.[8] Using Proposition 3.2, this would give us that

$$\log \zeta \leq L - \log\left(\binom{2^c}{u} \cdot (2^{c+r})!\right) \implies \zeta \leq \zeta^*$$

which is a contradiction to the assumption (see (5)).

Using $\mathcal{A}$, we shall define procedures $\mathsf{Encode}, \mathsf{Decode}$ such that for every $(U, \Pi) \in \mathcal{G}$, $\mathsf{Decode}(\mathsf{Encode}(U, \Pi)) = (U, \Pi)$ and the size of the output of $\mathsf{Encode}(U, \Pi)$ is at most L bits where

$$L = \log\left(\left(2^5 \cdot 4e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right) .$$

Using Proposition 3.2, this would give us that

$$\epsilon \leq \left(2^7 \cdot e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u .$$

This immediately gives the bound claimed in the statement of the lemma. The rest of the proof of the lemma would define $\mathsf{Encode}, \mathsf{Decode}$, show an upper bound on the size of the output of Encode and that $\mathsf{Decode}(\mathsf{Encode}(U, \Pi)) = (U, \Pi)$ for all $(U, \Pi) \in \mathcal{G}$.

Notation and Definitions. Fix $(U, \Pi) \in \mathcal{G}$. Let $U = \{\mathsf{IV}_1, \ldots, \mathsf{IV}_u\}$ where the IV_i's are ordered lexicographically. Let $\mathsf{Qrs}(\mathsf{IV}) \in (\{0,1\}^{r+c})^T$ be the list of queries that $\mathcal{A}_2$ makes to Π or Π^{-1} when executed with input (σ, IV). Namely, for $\mathsf{IV} \in \{0,1\}^c$,

$$\mathsf{Qrs}(\mathsf{IV}) = \left\{s \in \{0,1\}^{c+r} \mid \mathcal{A}_2(\sigma, \mathsf{IV}) \text{ queries } \Pi \text{ or } \Pi^{-1} \text{ on } s\right\}$$

Note that $\mathsf{Qrs}(\mathsf{IV})$ 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 remember all of its past queries).

Let $\mathsf{Ans}(\mathsf{IV}) \in (\{0,1\}^{r+c})^T$ be the list of answers to the queries of that $\mathcal{A}_2$ to Π or Π^{-1} when executed with input (σ, IV). Namely, for $\mathsf{IV} \in \{0,1\}^c$,

$$\mathsf{Ans}(\mathsf{IV}) = \left\{s \in \{0,1\}^{c+r} \mid \mathcal{A}_2(\sigma, \mathsf{IV}) \text{ queries } \Pi \text{ or } \Pi^{-1} \text{ on } s\right\}$$

We say that $\mathsf{IV}' \in \mathsf{SIVs}(\mathsf{IV})$ if there is some $s[2] \in \{0,1\}^r$ such that $s[2] \,\|\, \mathsf{IV}'$ is an entry in $\mathsf{Qrs}(\mathsf{IV})$ or $\mathsf{Ans}(\mathsf{IV})$. Namely, for $\mathsf{IV}, \mathsf{IV}' \in \{0,1\}^c$,

$$\mathsf{IV}' \in \mathsf{SIVs}(\mathsf{IV}) \iff \exists s[2] \in \{0,1\}^r \text{ s.t. } s[2] \,\|\, \mathsf{IV}' \in \mathsf{Qrs}(\mathsf{IV}) \cup \mathsf{Ans}(\mathsf{IV}).$$

We define the set of *fresh* IVs in U. An IV IV_i for $i \in [u]$ is called fresh if it was never an IV in either input or output of any query performed by $\mathcal{A}_2$ while being executed on IV_j for $j \leq i-1$ which are fresh. The first IV IV_1 is always fresh. An IV IV_i for $i \geq 2$ is fresh if for any fresh IV_j for $j \leq i-1$, $\mathsf{IV}_i \notin \mathsf{SIVs}(\mathsf{IV}_j)$. Namely, denoting the set of fresh IVs by U_{fresh}, we have the following inductive (on $i \in [u]$) definition:

$$\mathsf{IV}_i \in U_{\mathsf{fresh}} \iff \forall j \leq i-1, \mathsf{IV}_j \in U_{\mathsf{fresh}} \colon \mathsf{IV}_i \notin \mathsf{SIVs}(\mathsf{IV}_j).$$

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

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

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

Denote

$$\forall i \in [F]\colon \mathsf{Q}_i := \mathsf{Qrs}(\mathsf{IV}'_i) \quad \text{and} \quad \mathsf{Q}_{\mathsf{fresh}} := \mathsf{Q}_1 \,\|\, \mathsf{Q}_2 \,\|\, \ldots \,\|\, \mathsf{Q}_F,$$

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 $\mathsf{IV} \in U \setminus U_{\mathsf{fresh}}$, let t_{IV} be the minimum value such that $\mathsf{Q}_{\mathsf{fresh}}[t_{\mathsf{IV}}]$ is a query either with input or output of the form $(*, \mathsf{IV})$. Let $b_{\mathsf{IV}} = 0$ if input of $\mathsf{Q}_{\mathsf{fresh}}[t_{\mathsf{IV}}]$ was of the form $(*, \mathsf{IV})$ and 1 otherwise. Define the set of *prediction* queries as

$$\mathsf{P} := \{2t_{\mathsf{IV}} - b_{\mathsf{IV}} \mid \mathsf{IV} \in U \setminus U_{\mathsf{fresh}}\}.$$

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

Structure of collisions. Since adversary $\mathcal{A}_2$ succeeds on all of the IVs in U, it holds that for every $j \in [F]$, the output of the adversary is (α_j, α'_j) such that $\alpha_j \neq \alpha'_j$, $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha_j) = \mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha'_j)$ and both $\alpha_j \neq \alpha'_j$. We can assume without loss of generality that the last blocks of α_j and α'_j are distinct (because otherwise we can trim α_j, α'_j to obtain a shorter collision).

Definition 5.7 (Crucial queries)**.** The queries to Π, Π^{-1} in Q_j include a subset of queries that we call the *crucial queries*. The subset consists of earliest appearing queries in Q_j that are required to compute $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha_j)$ and $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha'_j)$. It follows that for 2-block collisions, this subset consists of at most four queries.

We say that a query made by running while running on (σ, IV'_j) is NEW if either of the following hold.

- the query is $\Pi(m, \mathsf{IV})$ with answer (m', IV') and neither $\Pi(m, \mathsf{IV})$ or $\Pi^{-1}(m', \mathsf{IV}')$ had been queried by $\mathcal{A}_2$ while running on $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_{j-1}$.
- the query is $\Pi^{-1}(m, \mathsf{IV})$ with answer (m', IV') and neither $\Pi^{-1}(m, \mathsf{IV})$ or $\Pi(m', \mathsf{IV}')$ had been queried by $\mathcal{A}_2$ while running on $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_{j-1}$.

If a query is not NEW we classify it into one of 2 types: REPEATEDUSED, and REPEATEDUNUSED. A REPEATEDUSED query is one such that it was a crucial query for IV'_i where $i < j$. A REPEATEDUNUSED query is one such that it is not a NEW or a REPEATEDUSED query.

Our goal is to compress (U, Π) and we are going to 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 $\tilde{\Pi}$ with some entries removed and some additional lists and sets. We will be describing the details of these lists and sets below and which entries we remove from $\tilde{\Pi}$. It is instructive to think about how decoding would work to understand the intuition behind out encoding strategy. The decoding procedure will run $\mathcal{A}_2$ on the IVs in U_{fresh} and answer its queries using the entries in $\tilde{\Pi}$ and the other lists and sets it gets as input. We need to encode enough information in these lists and sets so that decoding procedure can correctly answer the queries whose answers will be removed from $\tilde{\Pi}$. The IVs in $U \setminus U_{\mathsf{fresh}}$ will be recovered using P. Our main goal is to show that when we remove entries of $\tilde{\Pi}$ and instead using additional lists and set, we are actually compressing. Our ways to compress will depend on the crucial queries in each Q_j for $j \in [F]$.

We classify the IV'_jth for each $j \in [F]$ into the first of the following cases it satisfies, e.g., if the crucial queries for IV'_j satisfies both cases 1 and 2, we categorize it into 1.

1. One of the crucial queries for IV'_j is a query such that the last c bits of the answer is IV'_j
2. All of the crucial queries to are NEW.
3. At least one of the crucial queries is REPEATEDUSED.
4. There is exactly one REPEATEDUNUSED crucial query.
5. There are exactly two REPEATEDUNUSED crucial queries.

Claim 5.8. *We claim that each* IV'_j *will be categorized into one of the above cases.*

Proof. To begin with, observe that given how we define *fresh* IV'_j, there will have to be a NEW crucial query such that either it is a query to Π with input of the form $(*, \mathsf{IV}'_j)$ or it is a query to Π^{-1} with answer of the form $(*, \mathsf{IV}'_j)$, because we have assumed without loss of generality that $\mathcal{A}_2$ makes all the queries while running on IV'_j required to find the collision. If there is a NEW crucial query to Π^{-1} with answer of the form $(*, \mathsf{IV}'_j)$ then case 1 is satisfied. Also observe that there are at most two crucial queries that are not NEW since we are looking at 2-block collisions because any query whose input or output is of the form $(*, \mathsf{IV}'_j)$ is NEW for all IV'_j as they are fresh. So it follows if IV'_j is not categorized into 1, it will either have all NEW crucial queries (case 2) or have at least one REPEATEDUSED query (case 3) or have one (case 4) or two REPEATEDUNUSED (case 5) queries. This proves the claim. ∎

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

$$L = \log\left(\left(2^5 \cdot 4e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right)$$

bits.

In other words, we need to show that the encoding procedure saves at least

$$\begin{aligned} u \cdot \min\{ &\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4uT, \\ &c + r - 2 - \log uT - 2\log T, 2c - 2 - 2\log T - 2\log uT\} \\ &- 5u - u\log 4e \end{aligned} \tag{6}$$

bits overall.

We make the following claim.

Claim 5.9. *To show that the compression indeed achieves the savings from* (6)*, for every* IV *in* U_{fresh}*, it suffices to save at least the following number of bits.*

$$\begin{aligned} \min\{ &\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\ &c + r - 2 - \log FT - 2\log T, 2c - 2 - 2\log T - 2\log FT\}\ . \end{aligned}$$

Proof. First we look at the amount of savings we get from $\tilde{\Pi}$.

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

First we compute the amount of bits we save for encoding U, then that would give us the amount of bits we need to save for encoding Π. 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 $\log\binom{2^c}{u}$ bits, can be encoded using $\log\left(\binom{2FT}{u-F}\binom{2^c}{F}\right)$ bits. Therefore, the saving in bits in the description of U is at least

$$\begin{aligned} &\log\binom{2^c}{u} - \log\left(\binom{2FT}{u-F}\binom{2^c}{F}\right) \geq \log\left(\frac{\left(\frac{2^c}{u}\right)^u}{\left(\frac{2eFT}{u-F}\right)^{u-F}\left(\frac{e2^c}{F}\right)^F}\right) \\ &= \log\left(\frac{2^{c(u-F)}(u-F)^{u-F}F^F}{e^u 2^{u-F}(FT)^{u-F}u^u}\right) = (u-F)\log\left(\frac{2^{c-1}}{FT}\right) + \log\left(\frac{(u-F)^{u-F}F^F}{e^u u^u}\right) \\ &= (u-F)\log\left(\frac{2^{c-1}}{FT}\right) - \log\left(e^u\left(\frac{u}{F}\right)^F\left(\frac{u}{u-F}\right)^{u-F}\right) \\ &\geq (u-F)\log\left(\frac{2^{c-1}}{uT}\right) - u\log 4e\ . \end{aligned} \tag{7}$$

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

By subtracting (7) (how much we save in U) from (6) (how much we need to save in total), it suffices to show that we save at least

$$u \cdot \min\{ \min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4uT, c+r-2-\log uT - 2\log T, \\ 2c-2-2\log T - 2\log uT\} - 5u - (u-F)\log\left(\frac{2^c}{uT}\right) \tag{8}$$

bits while encoding Π. Observe that

$$\log(2^c/uT) = c - \log uT \geq \min\{\min\{(c-1),(r-1)\} - 2\log T, \\ (c-1) - \log 4uT, c+r-2-\log uT - 2\log T, 2c-2-2\log T - 2\log uT\}\ .$$

Using this observation and $u \geq F$, we have that the expression in (8) is at least

$$F \cdot \min\{ \min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\ c+r-2-\log FT - 2\log T, 2c-2-2\log T - 2\log FT\} - 5u \tag{9}$$

bits.

We will label each of the IVs in F with few bits that described its "type" (according to the cases described above – case 1 will have 2 subcategories, case 2 will have 4 subcategories, case 3 will have 2 subcategories, case 4 will have 13 subcategories and case 5 will have 8 subcategories), and for this 5 bits will suffice. This will cost, in total, another $5u$ bits, and therefore for each IV in U_{fresh} it suffices to save the following number of bits.

$$\min\{ \min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\ c+r-2-\log FT - 2\log T, 2c-2-2\log T - 2\log FT\}\ .$$

■

We make the following observation- the values of Π which are never queried by the adversary will be stored at the end of $\tilde{\Pi}$ and these will be at least $2^{c+r} - T \cdot 2^c$ values (since on every IV there will be T queries made). Since we have assumed $r \geq 2$ and $T \leq 2^{r/2}$, it follows that at least 2^{c+r-1} values will be at the end of $\tilde{\Pi}$- meaning for every value of Π we will remove from $\tilde{\Pi}$ to recover from m bits of other data structures, we make a saving of $c+r-1-m$ bits.

For each of these cases we will devise strategies to compress the amount of specified amount of bits. Before diving into the details, we give a brief glimpse of the complexity of this task in terms of handling a large number of subcases for some of the cases. Cases 1 to 3 are fairly simple in this regard (they do not lead to a large number of sub-cases). Handling cases 4 and 5 requires way more care since they give rise to many subcases.

For instance in case 4 where there is one crucial query that is REPEATEDUNUSED, one of these possibilities arise: namely **a)** there are only two crucial queries **b)** there are three crucial queries, or **c)** there are four crucial queries. For the possibility **b)** we have three different scenarios depending on the structure formed by the crucial queries – the collision could be caused by either the answer of the two NEW queries sharing common bits with the input and answer of the REPEATEDUNUSED query, or the input of one NEW query sharing bits with the input/answer of the REPEATEDUNUSED query and the output of the same NEW query sharing bits with the answer/input of the REPEATEDUNUSED query or the answer of one NEW query sharing common bits with the answer or input of the other NEW query.

Further depending on whether queries were made in the forward direction or the reverse direction, even more possibilities come up. Similarly for case 5, there are numerous possibilities that arise. We need to give a strategy covering all of these possibilities, making the compression argument extremely cumbersome by design.

We now describe the details of how we handle each case. Assuming that the crucial queries for IV'_j satisfies a particular case, we describe the encoding and decoding procedures, and the amount of savings. We provide this locally verifiable view for ease of parsing- in Appendix A, we also include the full descriptions of the encoding and decoding procedures for the sake of completeness.

Handling case 1. Assume that the query for which the answer is of the form (m, IV'_j) is a query to Π^{-1} on (m', IV'). This query has to be NEW since IV'_j is fresh. Let the index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be i. The encoding procedure stores (i, m) in a list and removes the entry for $\Pi(m, \mathsf{IV}'_j)$ from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 1, then it removes the front element (i, m) the list. It answers the ith query made while running on IV'_j with (m, IV'_j), and appropriately sets $\Pi(m, \mathsf{IV}'_j)$ to (m', IV').

If the query for which the answer is of the form $(*, \mathsf{IV}'_j)$ is a query to Π, the encoding and decoding procedure will be similar with appropriate changes (e.g., what is removed from $\tilde{\Pi}$ etc).

In this case we save at least $(c + r - 1) - r - \log T = (c - 1) - \log T$ bits, which is more than what we need.

Handling case 2. If IV'_j gets categorized into this case then there will always be two NEW queries q_1, q_2 with q_1 queried before q_2 such that either the last c bits in the answer of q_2 is the same as the last c bits in the input or answer of q_1 or the first r bits in the answer of q_2 is the same as the first r bits in the input or answer of q_1 (giving rise to four sub-cases as claimed).

The encoding procedure stores in a list the tuple consisting of the indices of q_1, q_2, and the other c or r bits. It removes the entry corresponding to q_2 from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 2, the front element of list is removed and the information is sufficient to answer the NEW query whose entry was removed from $\tilde{\Pi}$.

In this case we save at least $\min\{(c - 1), (r - 1)\} - 2 \log T$ bits, which suffices.

Handling case 3. In this case it is easy to see that there is a NEW query to Π on (m', IV'_j) such that the last c bits of the answer is same as that of the last c bits of the answer or input of a REPEATEDUSED query q (giving rise to two sub-cases as claimed). Let the index of the query $\Pi(m', \mathsf{IV}'_j)$ among all the queries made by $\mathcal{A}_2$ when running on IV'_j be i and its answer be (m, IV). Let the REPEATEDUSED query q be a crucial query for IV'_k such that $k < j$ and let it be the nth crucial query $(n < 4)$.

The encoding procedure stores (i, k, n, m) in a list and removes the entry for $\Pi(m', \mathsf{IV}'_j)$ from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 3, then it removes the front element (i, k, n, m) the list. It answers the ith query made while running on IV'_j with (m', IV'_j), where IV'_j is computed using (k, n) (and the relevant sub-case categorization) and appropriately sets $\Pi(m', \mathsf{IV}'_j)$ to (m, IV).

In this case we save at least $(c + r - 1) - \log T - \log F - \log 4 - r = (c - 1) - \log 4FT$ bits, which suffices.

Handling case 4. First we observe that if IV'_j is categorized into this case, then there are five following possibilities.

(a) There is a NEW crucial query q_1 of the form $(*, \mathsf{IV}'_j)$ to Π with answer (m', IV') and there is only one other crucial query q_2 which is REPEATEDUNUSED such that either it is a query to Π with input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ or it is a query to Π^{-1} with input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ (i.e., two sub-sub-cases).

(b) There are two NEW crucial queries q_1, q_2 to Π with input $(*, \mathsf{IV}'_j)$ and a REPEATEDUNUSED crucial query q_3 such that if q_3 is a query with input of the form $(*, \mathsf{IV}')$ and output of the form $(m', *)$ then the answer of q_1 is of the form $(*, \mathsf{IV}')$ and the answer to q_2 is of the form $(m', *)$.

(c) There is a NEW crucial query q_1 to Π with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$ and there is a second NEW crucial query q_2 and a REPEATEDUNUSED crucial query q_3 such that one of the following happen.

(i) q_2 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$.

(ii) q_2 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$.

(iii) q_2 is a query to Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$.

(iv) q_2 is a query Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$.

(d) There are two NEW crucial queries q_1, q_2 such that q_1 is made before q_2 and the last c bits of the answer of q_2 is same as the last c bits of the input or answer of q_1

(e) There are two NEW crucial queries q_1, q_2 to Π both with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$, $(*, \mathsf{IV}'')$ respectively and there is a third NEW crucial query q_3 made after q_1, and a REPEATEDUNUSED crucial query q_4 such that one of the following happen.

(i) q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$.

(ii) q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$.

(iii) q_3 is a query to Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and has answer $(m', *)$.

(iv) q_3 is a query to Π^{-1} on input $(m', *)$ and has answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and has answer $(*, \mathsf{IV}'')$.

First we argue why these possibilities are exhaustive for an IV'_j that has gotten categorized in case 4. Since we are looking for 2-block collisions, there can be only one or two other crucial queries that do not involve IV'_j- these will be REPEATEDUNUSED or NEW in this case. If there are only two crucial queries- one NEW query and one REPEATEDUNUSED query then it will satisfy sub-case (a). Otherwise if there are three crucial queries: two NEW queries and one REPEATEDUNUSED query such that both new queries are to Π and of the form $(*, \mathsf{IV}'_j)$, then it will satisfy sub-case (b). It is easy to see that sub-case (c) covers the other possibility of two NEW queries and one REPEATEDUNUSED query when there are three crucial queries because for a 2-block collision, the NEW query which is not of the form $(*, \mathsf{IV}'_j)$ and the REPEATEDUNUSED query will share either the the first c or last r bits in input or output (hence four sub-sub cases) and share the c bits of either input or output with the NEW query of the form $(*, \mathsf{IV}'_j)$. Similarly it is easy to see that if there are four crucial queries- three NEW and one REPEATEDUNUSED then it is covered by sub-cases (d), (e).

For 4(a), the encoding algorithm stores in a list the tuple consisting of the index of the NEW crucial query among the queries made while running on IV'_j and the index of the first occurrence of the REPEATEDUNUSED crucial query among all queries made by $\mathcal{A}_2$. The answer of the NEW crucial query is removed from $\tilde{\Pi}$.

During decoding the REPEATEDUNUSED and NEW queries can be correctly identified and from the input and answer of the REPEATEDUNUSED query both m', IV' can be inferred and the NEW query can be answered correctly.

It follows that savings for this sub-case is at least $c + r - 2 - \log FT - \log T$ bits.

For 4(b), the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the index of the first occurrences of the queries q_3 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_1, q_2 respectively . It removes the answers of q_1, q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, q_3 can be identified correctly using the information from the encoding and q_1, q_2 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $c + r - 2 - 2\log T - \log FT$ bits.

For 4(c), the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the index of the first occurrences of the queries q_3 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_1, q_2 (depending on the sub-sub case) . It removes the answers of q_1, q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, q_3 can be identified correctly using the information from the encoding and q_1, q_2 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{2c-2, c+r-2\} - 2\log T - \log FT$ bits.

For 4(d), the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the remaining c or r bits of the answer of q_2 (depending on the sub-sub case). It removes the answer of q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, can be identified correctly using the information from the encoding and q_2 can be correctly answered using the answer of q_1 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{c-1, r-1\} - 2\log T$ bits.

For 4(e), the encoding algorithm stores in a list the tuple consisting of the indices of q_2, q_3 among the queries made while running on IV'_j and the indices of the first occurrence of the query q_4 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_2, q_3 (depending on the sub-sub case) . It removes the answers of q_2, q_3 from $\tilde{\Pi}$.

During decoding queries q_2, q_3, q_4 can be identified correctly using the information from the encoding and q_2, q_3 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{2c-2, c+r-2\} - 2\log T - \log FT$ bits.

To conclude, in this case we save at least the following number of bits which suffices.

$$\min\{\min\{2c-2, c+r-2\} - \log FT - 2\log T, \min\{c-1, r-1\} - 2\log T\} .$$

Handling case 5. First we observe that if IV'_j is categorized into this case, then one of the following must happen.

(a) There is a NEW crucial query q_1 of the form $(*, \mathsf{IV}'_j)$ to Π with answer (m', IV') and there are two other crucial queries q_2, q_3 (q_2 is made for the first time before q_3 is made for the first time) both of which are REPEATEDUNUSED such that one of the following happen.

(i) q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$.

(ii) q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$.

(iii) q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$.

(iv) q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$.

(b) There are two NEW crucial queries q_1, q_2 to Π both with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$, $(*, \mathsf{IV}'')$ respectively and there are two crucial REPEATEDUNUSED queries q_3, q_4 (q_3 is made for the first time before q_4 is made for the first time) such that one of the following happen.

(i) q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$.

(ii) q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$.

(iii) q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$.

(iv) q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$.

First we argue why these possibilities are exhaustive for an IV'_j that has gotten categorized in case 5. Since we are focusing on two block collisions and we are not in the case 3, the queries other than NEW queries will be REPEATEDUNUSED queries. Again from the definition of fresh IVs and since IV'_j did not get categorized into 1, there must be one or two crucial NEW queries (sub-cases 5(a), 5(b)) on $(*, \mathsf{IV}'_j)$. Since we are looking for 2-block collisions, there can be only two other crucial queries that do not involve IV'_j- these will be REPEATEDUNUSED in this case. It is easy to see that these two REPEATEDUNUSED queries will satisfy one of the four sub-sub-cases of 5(a) and 5(b). Hence this categorization is exhaustive for this case.

For 5(a), the encoding algorithm stores in a list the tuple consisting of the index of q_1 among the queries made while running on IV'_j and the indices of the first occurrences of the queries q_2, q_3 (assume wlog that q_2 is queried earlier) among all queries made by $\mathcal{A}_2$, the message m' and r or c bits of rest of the answer of q_2 or q_3 depending the on the sub-sub-case. The answer of q_1 and q_3 is removed from $\tilde{\Pi}$.

During decoding the queries q_1, q_2, q_3 can be inferred from what is present in the encoding and hence the answers of q_1 and whichever of q_2 or q_3 was removed can be computed correctly.

It follows that savings for this sub-case is at least $\min\{2c-2, c+r-2\} - \log T - 2\log FT$ bits.

For 5(b), the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the indices of the first occurrences of the queries q_3, q_4 among all queries made by $\mathcal{A}_2$, the messages m', m'' and r or c bits of rest of the answer of q_3 or q_4 depending the on the sub-sub-case.

During decoding the queries q_1, q_2, q_3, q_4 can be inferred from what is present in the encoding and hence the answers of q_1 and q_2 can be computed correctly using things present in the encoding and answers of q_3, q_4.

The decoding for this is similar to that for 5(a).

It follows that savings for this sub-case is at least $2c - 2 - 2\log T - 2\log FT$ bits.

In this case we save at least the following number of bits which suffices.

$$\min\{c + r - 2 - \log T - 2\log FT, 2c - 2 - 2\log T - 2\log FT\} .$$

Acknowledgements

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 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 and Cody Freitag's work was done during an internship at NTT Research. Cody Freitag is also 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).

References

[AAC+17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond Hellman's time-memory trade-offs with applications to proofs of space. In *Advances in Cryptology - ASIACRYPT*, pages 357–379, 2017. 8

[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. 5, 6, 7, 8, 12, 13, 15, 29

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 75–83, 1978. 29

[AGL22] Akshima, Siyao Guo, and Qipeng Liu. Time-space lower bounds for finding collisions in Merkle-Damgård hash functions. To appear in CRYPTO, 2022. 5

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology - CRYPTO*, pages 1–21, 2006. 8

[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, 4

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007. 3, 4

[BDPVA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the security of the keyed sponge construction. In *Symmetric Key Encryption Workshop*, volume 2011, 2011. 3

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In *Advances in Cryptology - ASIACRYPT*, pages 321–340, 2013. 8

[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. 4, 5, 6, 7, 8, 11, 13, 14, 28

[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. 5, 7, 15

[CGK18] Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 415–447, 2018. 8

[CK19] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *Theory of Cryptography - TCC*, pages 393–421, 2019. 8

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology - EUROCRYPT*, pages 203–216, 1987. 3

[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. 5, 11

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *Advances in Cryptology - CRYPTO*, pages 157–186, 2010. 8, 15

[FJM14] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, even-mansour and PRINCE. In *Advances in Cryptology - ASIACRYPT*, pages 420–438, 2014. 8

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. 4, 5, 6, 8, 10, 21, 27, 28

[GGH+20] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. Data structures meet cryptography: 3SUM with preprocessing. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 294–307, 2020. 8

[GK22] Ashrujit Ghoshal and Ilan Komargodski. On time-space tradeoffs for bounded-length collisions in Merkle-Damgård hashing, 2022. 5, 30

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science, FOCS*, pages 305–313. IEEE Computer Society, 2000. 7, 8, 11

[GT16] Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In *Advances in Cryptology - EUROCRYPT*, pages 87–116, 2016. 3

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. 4, 5, 6, 8, 10, 15, 21, 23

[Mer82] Ralph C. Merkle. *Secrecy, Authentication and Public Key Systems*. PhD thesis, UMI Research Press, Ann Arbor, Michigan, 1982. 3

[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. 3

[Mih10] Joseph Mihalcik. An analysis of algorithms for solving discrete logarithms in fixed groups. Technical report, Naval Postgraduate School Monterey CA, 2010. 8

[MMR17] Pat Morin, Wolfgang Mulzer, and Tommy Reddad. Encoding arguments. *ACM Comput. Surv.*, 50(3):46:1–46:36, 2017. 7

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO*, pages 617–630, 2003. 4, 5

[ST79] Robert H. Morris Sr. and Ken Thompson. Password security - A case history. *Commun. ACM*, 22(11):594–597, 1979. 8

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *Advances in Cryptology - CRYPTO*, pages 205–223, 2007. 4, 5, 7, 11, 14

[Wee05] Hoeteck Wee. On obfuscating point functions. In *37th Annual ACM Symposium on Theory of Computing, STOC*, pages 523–532, 2005. 8, 11

[Yao90] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *STOC*, pages 84–94, 1990. 4, 5, 8

A Encoding and decoding procedure pseudocodes for Theorem 5.5.

The encoding algorithm $\mathsf{Encode}(\sigma, U, \Pi)$:

- Parse U as $\{\mathsf{IV}_1, \ldots, \mathsf{IV}_u\}$ where $\mathsf{IV}_1, \ldots, \mathsf{IV}_u$ are in lexicographical order.
- Set $j \leftarrow 0$.
- Initialize the set P to be empty, the lists Cases and $\tilde{\Pi}$ to be empty
- For $i = 1, \ldots, u$, do:

– If $\mathsf{IV}_i \in \mathsf{SIVs}(a)$ for some $a \in U_{\mathsf{fresh}}$, let $r \in [jT]$ be the smallest index such that either $\mathsf{Q}_{\mathsf{fresh}}[r]$ or its answer is of the form $(*, \mathsf{IV}_j)$. If $\mathsf{Q}_{\mathsf{fresh}}[r] = (*, \mathsf{IV}_j)$ then set $b \leftarrow 1$ and 0. Add $2r - b$ to P, and continue to next iteration.

– Otherwise, increment j, set $\mathsf{IV}'_j \leftarrow \mathsf{IV}_i$. Run $\mathcal{A}_2$ on (σ, IV'_j), answering its queries to Π, Π^{-1}. If the query was one to Π with input (m, IV), add $\Pi(m, \mathsf{IV})$ to the list $\tilde{\Pi}$ if it was not already added earlier. If the query was one to Π^{-1} with output (m, IV), add $\Pi(m, \mathsf{IV})$ to the list $\tilde{\Pi}$ if it was not already added earlier. Classify the queries as NEW, REPEATEDUSED, REPEATEDUNUSEDas defined above.

– Determine the crucial queries as specified in Definition 5.7. Determine which category IV'_j belongs to among Cases 1 to 5 based on the crucial queries. According to the categorization do the following.

  * If IV'_j gets categorized into case 1
    (a) If the query for which the answer is of the form (m', IV'_j) is a query to Π^{-1} on (m'', IV'), then let index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be d. Set $\mathsf{Cases}[j] \leftarrow 1a$, add (d, m') to a list L_{1a} and remove the entry for $\Pi(m', \mathsf{IV}'_j)$ from $\tilde{\Pi}$.
    (b) If the query for which the answer is of the form (m', IV'_j) is a query to Π on (m'', IV'), then let index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be d. Set $\mathsf{Cases}[j] \leftarrow 1a$, add (d, m') to a list L_{1b} and remove the entry for $\Pi(m'', \mathsf{IV}')$ from $\tilde{\Pi}$.
    Continue to next iteration
  * Else if IV'_j gets categorized into case 2, define q_1, q_2 as specified above under heading "Handling case 2". Let the input of q_1 be (m', IV') and its answer be (m'', IV''). Let the indices of queries q_1, q_2 among the queries made by $\mathcal{A}_2$ when running on IV'_j be d_1, d_2.
    (a) If the answer of q_2 is (m''', IV') for some m''', set $\mathsf{Cases}[j] \leftarrow 2a$, add (d_1, d_2, m''') to list L_{2a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2 (i.e. if q_2 was a query to Π on (m, IV) remove $\Pi(m, \mathsf{IV})$ and if q_2 was a query to Π^{-1} with answer (m, IV) remove $\Pi(m, \mathsf{IV})$ from $\tilde{\Pi}$).
    (b) If the answer of q_2 is (m', IV''') for some IV''', set $\mathsf{Cases}[j] \leftarrow 2b$, add $(d_1, d_2, \mathsf{IV}''')$ to list L_{2a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.
    (c) If the answer of q_2 is (m''', IV'') for some m''', set $\mathsf{Cases}[j] \leftarrow 2c$, add (d_1, d_2, m''') to list L_{2c}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.
    (d) If the answer of q_2 is (m'', IV''') for some IV''', set $\mathsf{Cases}[j] \leftarrow 2d$, add (d_1, d_2, m''') to list L_{2d}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.
    Continue to the next iteration.
  * Else if IV'_j gets categorized into case 3, define q as specified above under heading "Handling case 3". Let q be a crucial query for IV'_k and let it be the n-th crucial query (in query order) among the crucial queries for IV'_k. Let the input of q be (m', IV') and its answer be (m'', IV''). Let the NEW crucial query for IV'_j the last c bits of whose answer is same as the last c bits of the answer or input of q. Let the index of query q' among the queries made by $\mathcal{A}_2$ when running on IV'_j be d.
    (a) If the answer of q' is (m''', IV') for some m''', set $\mathsf{Cases}[j] \leftarrow 3a$, add (d, k, n, m''') to list L_{3a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q'.
    (b) If the answer of q' is (m''', IV'') for some m''', set $\mathsf{Cases}[j] \leftarrow 3b$, add (d, k, n, m''') to list L_{3b}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q'.
  * Else if IV'_j gets categorized into case 4, determine which of the sub-cases IV'_j belongs to according to the categorization specified above under heading "Handling case 4".
    (a) If IV'_j gets categorized into sub-case (a), define $q_1, q_2, (m', \mathsf{IV}')$ as specified above for the sub-case. Let d be the index of q_1 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_2 in $\mathsf{Q}_{\mathsf{fresh}}$.

(i) If q_2 was a Π query with input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$, set $\mathsf{Cases}[j] \leftarrow 4a1$, add (d, e) to the list L_{4a1}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_1.

(ii) If q_2 was a Π^{-1} query with input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4a2$, add (d, e) to the list L_{4a2}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_1.

(b) If IV'_j gets categorized into sub-case (b), define $q_1, q_2, q_3, m', \mathsf{IV}'$ as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Suppose the answer of q_1 is (m'', IV') and that of q_2 is (m', IV'').Set $\mathsf{Cases}[j] \leftarrow 4b$, add $(d_1, d_2, e, m'', \mathsf{IV}'')$ to the list L_{4b}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(c) If IV'_j gets categorized into sub-case (c), define $q_1, q_2, q_3, \mathsf{IV}'$ as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m''', IV').

(i) If q_2 is a query to Π on input (m'', IV') and with answer (m', IV'') and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4c1$, add $(d_1, d_2, e, m''', \mathsf{IV}'')$ to the list L_{4c1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(ii) If q_2 is a query to Π on input (m'', IV') and with answer (m', IV'') and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$, then set $\mathsf{Cases}[j] \leftarrow 4c2$, add $(d_1, d_2, e, m''', \mathsf{IV}'')$ to the list L_{4c2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(iii) If q_2 is a query to Π^{-1} on input (m', IV'') and with answer (m'', IV') and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4c3$, add (d_1, d_2, e, m''', m'') to the list L_{4c3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(iv) If q_2 is a query to Π^{-1} on input (m', IV'') and with answer (m'', IV') and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$, then set $\mathsf{Cases}[j] \leftarrow 4c4$, add (d_1, d_2, e, m''', m'') to the list L_{4c4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(d) If IV'_j gets categorized into sub-case (d), define q_1, q_2 as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j.

(i) If the answer of q_2 is (m', IV') and the input of q_1 is of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4d1$, add (d_1, d_2, m') to list L_{4d1}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

(ii) If the answer of q_2 is (m', IV') and the answer of q_1 is of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4d2$, add (d_1, d_2, m') to list L_{4d2}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

(e) If IV'_j gets categorized into sub-case (e), define q_1, q_2, q_3, q_4 as specified above for the sub-case. Let d_1, d_2 be the indices of q_2, q_3 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e be the smallest index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$. Suppose the answer of q_2 is (m'', IV'').

(i) If q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and with answer (m', IV''') and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4e1$, add $(d_1, d_2, e, m'', \mathsf{IV}''')$ to the list L_{4e1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

(ii) If q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and with answer (m', IV''') and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 4e2$, add $(d_1, d_2, e, m'', \mathsf{IV}''')$ to the list L_{4e2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

(iii) If q_3 is a query to Π^{-1} on input of the form $(m', *)$ and with answer (m''', IV') and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4e3$, add

(d_1, d_2, e, m'', m''') to the list L_{4e3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

(iv) If q_3 is a query to Π^{-1} on input of the form $(m', *)$ and with answer (m''', IV') and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 4e4$, add (d_1, d_2, e, m'', m''') to the list L_{4e4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

$*$ Else if IV'_j gets categorized into case 5, determine which of the sub-cases IV'_j belongs to according to the categorization specified above under heading "Handling case 5".

(a) If IV'_j gets categorized into sub-case (a), define q_1, q_2, q_3 as specified above for the sub-case. Let d be the index of q_1 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e_1, e_2 be the smallest indices of q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m'', IV').

(i) If q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer (m', IV''), then set $\mathsf{Cases}[j] \leftarrow 5a1$, add $(d, e_1, e_2, m'', \mathsf{IV}'')$ to the list L_{5a1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

(ii) If q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer (m''', IV'), then set $\mathsf{Cases}[j] \leftarrow 5a2$, add (d, e_1, e_2, m'', m''') to the list L_{5a2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

(iii) If q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer (m', IV''), then set $\mathsf{Cases}[j] \leftarrow 5a3$, add $(d, e_1, e_2, m'', \mathsf{IV}'')$ to the list L_{5a3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

(iv) If q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer (m''', IV'), then set $\mathsf{Cases}[j] \leftarrow 5a4$, add (d, e_1, e_2, m'', m''') to the list L_{5a4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

(b) If IV'_j gets categorized into sub-case (b), define q_1, q_2, q_3, q_4 as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e_1, e_2 be the smallest indices of q_3, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m'', IV') and that of q_2 be (m''', IV'').

(i) If q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 5b1$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(ii) If q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 5b2$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(iii) If q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 5b3$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(iv) If q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 5b4$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

– Add all the evaluations of $\Pi(m, \mathsf{IV})$ to $\tilde{\Pi}$ in lexicographical order for all (m, IV)'s such that no Π query was not made on them and no Π^{-1} query had (m, IV) as answer.

– Output F, U', P, L_{1a},L_{1b}, L_{2a}, L_{2b}, L_{2c}, L_{2d}, L_{3a}, L_{3b}, L_{4a1}, L_{4a2}, L_{4b}, L_{4c1}, L_{4c2}, L_{4c3}, L_{4c4}, L_{4d1}, L_{4d2}, L_{4e1}, L_{4e2}, L_{4e3}, L_{4e4}, L_{5a1}, L_{5a2}, L_{5a3}, L_{5a4}, L_{5b1}, L_{5b2}, L_{5b3}, L_{5b4}, $\tilde{\Pi}$.

The decoding algorithm Decode:

1. Parse U' as $\{\mathsf{IV}'_1, \ldots, \mathsf{IV}'_F\}$ where $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_F$ are in lexicographical order.

2. Set $j \leftarrow 0$.

3. For $j = 1, \ldots, F$ do:

   (i) Run $\mathcal{A}_2$ on (σ, IV'_j), answering its queries to Π, Π^{-1}.

   (ii) Answer the ith query made by $\mathcal{A}_2$ on (m, IV) as follows.

   (iii) If $2 \cdot ((j-1) \cdot T + i) \in \mathsf{P}$ then add IV to U.

   (iv) If $2 \cdot ((j-1) \cdot T + i) - 1 \in \mathsf{P}$ then add the IV of the answer to U before returning it.

   (v) If $(*, d, c, *, \mathsf{IV}'') \in L_{5a1}$ then let the answer of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, return (m', IV'')

   (vi) If $(*, d, c, *, m''') \in L_{5a2}$ then let the input of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(*, \mathsf{IV}')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, return (m''', IV')

   (vii) If $(*, d, c, *, \mathsf{IV}'') \in L_{5a3}$ then let the input of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, return (m', IV'')

   (viii) If $(*, d, c, *, m''') \in L_{5a4}$ then let the answer of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(*, \mathsf{IV}')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, return (m''', IV')

   (ix) If $\mathsf{Cases}[j] = 1a$ and the tuple in front of list L_{1a} is (i, m'). Set $\Pi(m', \mathsf{IV}'_j) \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{1a}. Return (m', IV'_j).

   (x) Else if $\mathsf{Cases}[j] = 1b$ and the tuple in front of list L_{1b} is (i, m'). Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'_j)$, and remove the tuple from L_{1b}. Return (m', IV'_j).

   (xi) Else if $\mathsf{Cases}[j] = 2a$ and the tuple in front of list L_{2a} is (d, i, m'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{2a}. Return (m''', IV').

   (xii) Else if $\mathsf{Cases}[j] = 2b$ and the tuple in front of list L_{2a} is (d, i, IV'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{2b}. Return (m', IV''').

   (xiii) Else if $\mathsf{Cases}[j] = 2c$ and the tuple in front of list L_{2a} is (d, i, m'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{2c}. Return (m''', IV'').

   (xiv) Else if $\mathsf{Cases}[j] = 2d$ and the tuple in front of list L_{2a} is (d, i, IV'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}''')$, and remove the tuple from L_{2d}. Return (m'', IV''').

   (xv) Else if $\mathsf{Cases}[j] = 3a$ and the tuple in front of list L_{3a} is (i, k, n, m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{3a}. Return (m''', IV').

   (xvi) Else if $\mathsf{Cases}[j] = 3b$ and the tuple in front of list L_{3b} is (i, k, n, m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{3b}. Return (m''', IV'').

   (xvii) Else if $\mathsf{Cases}[j] = 4a1$ and the tuple in front of list L_{4a1} is (i, e), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$, and remove the tuple from L_{4a1}. Return (m', IV').

   (xviii) Else if $\mathsf{Cases}[j] = 4a2$ and the tuple in front of list L_{4a2} is (i, e), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$ and input $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$, and remove the tuple from L_{4a2}. Return (m', IV').

(xix) Else if $\mathsf{Cases}[j] = 4b$ and the tuple in front of list L_{4b} is $(i, d_2, e, m'', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{4b} if $d_2 < i$. Return (m'', IV').

(xx) Else if $\mathsf{Cases}[j] = 4b$ and the tuple in front of list L_{4b} is $(d_1, i, e, m'', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4b} if $d_1 < i$. Return (m', IV'').

(xxi) Else if $\mathsf{Cases}[j] = 4c1$ and the tuple in front of list L_{4c1} is $(i, d_2, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c1} if $d_2 < i$. Return (m''', IV').

(xxii) Else if $\mathsf{Cases}[j] = 4c1$ and the tuple in front of list L_{4c1} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4c1} if $d_1 < i$. Return (m', IV'').

(xxiii) Else if $\mathsf{Cases}[j] = 4c2$ and the tuple in front of list L_{4c2} is $(i, d_2, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c2} if $d_2 < i$. Return (m''', IV').

(xxiv) Else if $\mathsf{Cases}[j] = 4c2$ and the tuple in front of list L_{4c2} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4c2} if $d_1 < i$. Return (m', IV'').

(xxv) Else if $\mathsf{Cases}[j] = 4c3$ and the tuple in front of list L_{4c1} is (i, d_2, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c3} if $d_2 < i$. Return (m''', IV').

(xxvi) If $\mathsf{Cases}[j] = 4c3$ and the tuple in front of list L_{4c3} is (d_1, i, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m'', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4c3} if $d_1 < i$. Return (m'', IV').

(xxvii) Else if $\mathsf{Cases}[j] = 4c4$ and the tuple in front of list L_{4c4} is (i, d_2, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c4} if $d_2 < i$. Return (m''', IV').

(xxviii) Else if $\mathsf{Cases}[j] = 4c4$ and the tuple in front of list L_{4c4} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m'', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4c4} if $d_1 < i$. Return (m'', IV').

(xxix) Else if $\mathsf{Cases}[j] = 4d1$ and the tuple in front of list L_{4d1} is (d_1, i, m'), let the query with index d_1 in Q_j have input $(*, \mathsf{IV}')$. If the query is a query to Π, set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$ else set $\Pi(m', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4d1}. Return (m', IV').

(xxx) Else if $\mathsf{Cases}[j] = 4d2$ and the tuple in front of list L_{4d2} is (d_1, i, m'), let the query with index d_1 in Q_j have answer $(*, \mathsf{IV}')$. If the query is a query to Π, set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$ else set $\Pi(m', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4d2}. Return (m', IV').

(xxxi) Else if $\mathsf{Cases}[j] = 4e1$ and the tuple in front of list L_{4e1} is $(i, d_2, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e1} if $d_2 < i$. Return (m'', IV'').

(xxxii) Else if $\mathsf{Cases}[j] = 4e1$ and the tuple in front of list L_{4e1} is $(d_1, i, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{4e1} if $d_2 < i$. Return (m', IV''').

(xxxiii) Else if $\mathsf{Cases}[j] = 4e2$ and the tuple in front of list L_{4e2} is $(i, d_2, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e2} if $d_2 < i$. Return (m'', IV'').

(xxxiv) Else if $\mathsf{Cases}[j] = 4e2$ and the tuple in front of list L_{4e2} is $(d_1, i, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{4e2} if $d_2 < i$. Return (m', IV''').

(xxxv) Else if $\mathsf{Cases}[j] = 4e3$ and the tuple in front of list L_{4e3} is (i, d_2, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e3} if $d_2 < i$. Return (m'', IV'').

(xxxvi) Else if $\mathsf{Cases}[j] = 4e3$ and the tuple in front of list L_{4e3} is (d_1, i, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4e3} if $d_2 < i$. Return (m''', IV').

(xxxvii) Else if $\mathsf{Cases}[j] = 4e4$ and the tuple in front of list L_{4e4} is (i, d_2, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e4} if $d_2 < i$. Return (m'', IV'').

(xxxviii) Else if $\mathsf{Cases}[j] = 4e4$ and the tuple in front of list L_{4e4} is (d_1, i, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4e4} if $d_2 < i$. Return (m''', IV'').

(xxxix) Else if $\mathsf{Cases}[j] = 5a1$ and the tuple in front of list L_{5a1} is $(i, e_1, e_2, m'', \mathsf{IV}'')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a1}. Return (m'', IV').

(xl) Else if $\mathsf{Cases}[j] = 5a2$ and the tuple in front of list L_{5a2} is (i, e_1, e_2, m'', m'''), let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a2}. Return (m'', IV').

(xli) Else if $\mathsf{Cases}[j] = 5a3$ and the tuple in front of list L_{5a3} is $(i, e_1, e_2, m'', \mathsf{IV}'')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a3}. Return (m'', IV').

(xlii) Else if $\mathsf{Cases}[j] = 5a4$ and the tuple in front of list L_{5a4} is (i, e_1, e_2, m'', m'''), let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a4}. Return (m'', IV').

(xliii) Else if $\mathsf{Cases}[j] = 5b1$ and the tuple in front of list L_{5b1} is $(i, d_2, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b1} if $i > d_2$. Return (m'', IV').

(xliv) Else if $\mathsf{Cases}[j] = 5b1$ and the tuple in front of list L_{5b1} is $(d_1, i, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b1}if $i > d_1$. Return (m''', IV'').

(xlv) Else if $\mathsf{Cases}[j] = 5b2$ and the tuple in front of list L_{5b2} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b2} if $i > d_2$. Return (m'', IV').

(xlvi) Else if $\mathsf{Cases}[j] = 5b2$ and the tuple in front of list L_{5b2} is $(d_1, i, e_1, e_2, m'', m''', m'''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b2} if $i > d_1$. Return (m''', IV'').

(xlvii) Else if $\mathsf{Cases}[j] = 5b3$ and the tuple in front of list L_{5b3} is $(i, d_2, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b3} if $i > d_2$. Return (m'', IV').

(xlviii) Else if $\mathsf{Cases}[j] = 5b3$ and the tuple in front of list L_{5b3} is $(d_1, i, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b3} if $i > d_1$. Return (m''', IV'').

(xlix) Else if $\mathsf{Cases}[j] = 5b4$ and the tuple in front of list L_{5b4} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b4} if $i > d_2$. Return (m'', IV').

(l) Else if $\mathsf{Cases}[j] = 5b4$ and the tuple in front of list L_{5b4} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b4} if $i > d_1$. Return (m''', IV'').

4. Populate the rest of Π by remaining elements in $\tilde{\Pi}$ in lexicographical order.

5. Output (U, Π).