Lower Bound on Number of Compression Calls of a Collision-Resistance Preserving Hash

Debasmita Chakraborty[1] and Mridul Nandi[1]

Indian Statistical Institute, Kolkata
debasmitachakraborty1@gmail.com,mridul.nandi@gmail.com

Abstract. The collision-resistant hash function is an early cryptographic primitive that finds extensive use in various applications. Remarkably, the Merkle-Damgård and Merkle tree hash structures possess the collision-resistance preserving property, meaning the hash function remains collision-resistant when the underlying compression function is collision-resistant. This raises the intriguing question of whether reducing the number of underlying compression function calls with the collision-resistance preserving property is possible. In pursuit of addressing these inquiries, we prove that for an ℓn-to-sn-bit collision-resistance preserving hash function designed using r tn-to-n-bit compression function calls, we must have $r \geq \lceil(\ell - s)/(t - 1)\rceil$. Throughout the paper, all operations other than the compression function are assumed to be linear (which we call linear hash mode).

Keywords: Merkle-Damgård, Merkle Tree, Collision-Resistance Preserving Hash, Compression Function.

1 Introduction

Hash functions serve as fundamental components in cryptography, and the task of designing a hash function that is both secure and efficient is a longstanding challenge in the field. Hash functions can be classified based on their internal construction, and while many hash functions are based on compression functions, there are also hash functions that do not rely on a distinct compression step, such as sponge functions [BDPVA07] (these are a class of hash functions that absorb input data and then "squeeze" out the hash value), polyhash [BJKS93] (this is a polynomial evaluation-based hash function that operates by evaluating a polynomial over the input data), etc.

The process of creating a hash function based on a compression function can generally be broken down into two steps. The first step involves constructing a compression function, which takes inputs of a fixed length and generates smaller outputs than the inputs. The primary goal of a compression function is collision resistance, which means it should be computationally infeasible to find two different inputs that result in the same output. The second step entails designing a domain-extending algorithm, allowing longer messages to be transformed into a fixed-length output through a series of iterations utilizing the underlying compression functions. This algorithm enables the hash function to accommodate inputs of varying lengths, mapping them consistently to a fixed-size hash value.

Historically, block ciphers and permutations have been widely utilized in constructing compression functions [BRS02, BDPA11, RS08, RS07, BCS09]. Initially, block ciphers stood out as the preferred primitives for building compression functions, exemplified by the foundational designs of MD5, SHA1, and SHA2 hash functions. Notably, among the domain-extending algorithms, the Merkle-Damgård [Dam89, Mer89] (MD) and Merkle tree constructions are the most prominent examples.

The Merkle-Damgård hash function technique involves creating collision-resistant cryptographic hash functions by utilizing collision-resistant compression functions. Merkle trees were introduced by Ralph Merkle in 1980 as a method for verifying large public files [Mer80]. Since then, they have found wide-ranging applications in cryptography. These include but are not limited to parallel hashing, ensuring the integrity of large files, long-term data storage, various signature schemes [Ben20, BGD+06, BDK+07, BHK+19], time-stamping mechanisms [HS91], protocols based on zero-knowledge proofs [GGPR13, BCTV14], and even in the development of anonymous cryptocurrencies [BCG+14]. These applications demonstrate the versatility and importance of Merkle trees in modern cryptographic systems.

A collision-resistant hash function is one of the earliest primitives with plenty of cryptography use. Starting from digital signatures [Rab79, M.O78], proof of membership (Merkle trees [Mer89]), encryption, authentication, etc. The Merkle-Damgård construction gained popularity because Merkle and Damgård demonstrated that if the compression function used is resistant to collisions, the resulting hash function will also be collision-resistant. We can call this property *collision-resistance preserving* property. Merkle-Damgård and Merkle tree hash have the collision-resistance preserving property, i.e., if the compression function is collision-resistant, then so is the hash function constructed using it. The number of compression function calls for both constructions is the same. For example, we can process ℓ block messages by making ℓ or $(\ell - 1)$ calls to the underlying $2n$-to-n bit compression function for both constructions. Now, the question arises:

Can we improve Merkle-Damgard and Merkle tree constructions?

In a recent publication [DKMN21], the authors introduced a novel $5n$-to-n bit hash function denoted as T_5. This function utilizes three $2n$-to-n compression functions, surpassing the current standards set by Merkle-Damgård and Merkle trees. Notably, it achieves this improvement by handling an extra message block while maintaining the same count of compression function calls. Moreover, in [ABR21], a perfect binary tree hash function (ABR) of height ℓ, which processes $(2^\ell + 2^{\ell-1} - 1)$ message blocks using $(2^\ell - 1)$ calls to underlying $2n$-to-n-bit compression function, was introduced. The collision security of T_5, and ABR hash function with height 3 (a hash function that processes 11 blocks message using seven calls to $2n$-to-n bit compression functions) was proved under the ideal model assumption of the compression functions in [DDN22, DKMN21]. The authors in [ABR21] claimed optimal birthday-bound collision resistance of ABR hash of height ℓ for any ℓ under the random oracle model. However, the proof is incorrect and reported [DDN22], and the claim is still unproven.

In the ideal model, Stam [Sta08] conjectured that for an ℓn-to-n bit hash function using r calls to tn-to-n bits compression functions, the minimum number of compression function calls required to achieve optimal birthday security is given by $r \geq (2\ell - 1)/(2t - 1)$. This bound is popularly known as Stam's bound, and it was later proven in two works by Steinberger [Ste10] and by Steinberger, Sun and Yang [SSY12]. In these works, the authors used an ideal oracle and proved that one can break the constructed primitive in unbounded time but with a bounded number of queries. Therefore, in the extensively researched scenario where $t = 2$, the established minimum value is $r \geq (2\ell - 1)/3$ as a lower limit. This results in a 1.5 times efficiency gap compared to the efficiency of Merkle-Damgård and Merkle tree.

Are we done?

Studying the collision resistance property of a hash function, assuming the underlying compression function is collision-resistant, can be an exciting research area. It can provide insights into the robustness and security guarantees of the hash function and potentially help develop more secure cryptographic constructions or evaluate existing ones.

Hence, investigating the potential for the most optimal ℓn-to-n bit collision-resistant hash function becomes an exciting research direction, presuming the underlying compression functions possess collision-resistant properties. This examination aims to discern whether there are opportunities for enhancing the Merkle-Damgård and Merkle tree constructions within such a framework. We already know that for a general hash function based on a compression function, Steinberger, Sun, and Yang [Ste10, SSY12] proved a lower bound on the number of compression function calls to achieve birthday-bound collision security in the random oracle model conjectured by Stam [Sta08], which motivates us to propose some lower bound on the number of compression function calls to achieve collision-resistance preserving property for a general hash function. Hence, the following questions remain open.

Main Problem of the Paper: Can we propose any collision-resistance preserving hash function improving over the state-of-the-art Merkle-Damgård and Merkle-Tree hash? What about the lower bound on the number of compression function calls for collision-resistance preserving hash functions?

1.1 Our Contributions

To address the problem, we investigate the collision-resistance preserving properties of various hash modes such as T_5, ABR hash mode, and the extended version of SS Hash [**?**]. Our findings reveal that these modes do not have collision-resistance preserving property. Given the significance of collision-resistance preserving properties in cryptographic hash modes, this paper aims to establish the lower bound on the number of compression function calls required for collision-resistant hash modes, assuming that the underlying compression functions are collision-resistant.

Collision-Resistance Preserving Property: Lower Bound. The above results motivate us to think about the lower bound on the number of underlying compression function calls to achieve the collision-resistance preserving property of a hash mode. Analyzing the structures of various hash modes, including the well-known Merkle-Damgård and Merkle tree constructions, as well as the recently proposed T_5 and ABR constructions, we observe that all functions, apart from the underlying compression functions, are linear in these hash modes (e.g., Figure 4, Figure 3). We refer to this category of hash modes as *linear hash modes*. A detailed description is provided in Subsection 3.3. Notably, to the best of our knowledge, all existing hash modes fall into the category of linear hash modes. Consequently, our initial focus is determining a lower bound on the number of compression function calls required to achieve the collision-resistance preserving property in a linear hash mode. In particular, we prove the following statement:

Let H be an ℓn-to-n bit linear hash mode using r many tn-to-n bit compression function calls, satisfies either (i) $r < \lceil(\ell-1)/(t-1)\rceil$, if $t \geq 2$ or (ii) $\ell \geq 2$, if $t = 1$ with $(tr+\ell+r) < 2^n$. Then H is not collision-resistant, only assuming that the underlying compression functions are collision-resistant.

Our objective is to demonstrate that a hash mode H, which is an ℓn-to-n bit linear hash mode utilizing r calls to tn-to-n bit compression function does not possess the collision-resistance preserving property under specific conditions. Specifically, if $r < \lceil(\ell-1)/(t-1)\rceil$, if $t \geq 2$ or $\ell \geq 2$, if $t = 1$, and $(tr+\ell+r) < 2^n$, then H fails to be collision-resistance preserving. To prove this, it suffices to construct a collision-resistant compression function f and show that H^f is not collision-resistant[1]. Specifically, we establish that for a linear hash mode H meeting the specified conditions, a particular type of collision-resistant compression function f exists such that H^f is not collision-resistant. Constructing such

[1] It is important to note that a collision-resistance preserving mode must ensure a secure hash for all collision-resistant compression functions (e.g., Merkle-Damgård hash), not just random ones.

an f requires the *assumption* that collision-resistant hashes exist. We state and prove Theorem 1 in Section 4.
As we already know, Merkle-Damgård and Merkle tree hash functions process ℓ block messages using $\lceil(\ell-1)/(t-1)\rceil$ many tn-to-n bit compression function calls with $t \geq 2$, and they have collision-resistance preserving property, which implies that we cannot improve the construction of Merkle-Damgård and Merkle tree hash function in light of the collision-resistance preserving property. Hence, these two constructions are optimum while considering the class of linear hash mode.

As of the second contribution of our paper, we also prove that if an ℓn-to-sn-bit linear hash mode is designed using r many tn-to-n-bit compression function calls with $(tr+\ell+r) < 2^n$, and $t, s \geq 2$ then $r \geq \lceil(\ell-s)/(t-1)\rceil$ to achieve collision-resistance preserving property. More precisely, we prove the following statement:

Let H be an ℓn-to-sn bit linear hash mode using r many tn-to-n bit compression function calls, satisfies $r < \lceil(\ell-s)/(t-1)\rceil$, with $t, s \geq 2$, and $(tr+\ell+r) < 2^n$. Then H does not have collision-resistance preserving property.

It is important to note that all existing hash modes are linear, which is best known to us. This indicates that the linear hash modes represent a substantial class of hash functions, reflecting the importance of our results.

Organization of the Paper. In Section 2, we provide a mathematical background of hash function and collision security (both in uniform and non-uniform setup). We also describe white-box and black-box reductions of collision security for hash modes. In Section 3, we describe linear hash mode and its security. In Section 4, we state and prove our main result Theorem 1. In Section 5, we provide proof of Lemma 3, which is used in proving our main theorem. Finally, in Section 6, we extend our lower bound results for linear hash modes with more than one output block.

2 Background

Notations. Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ be the set of natural numbers and for $k \in \mathbb{N}$, we write $[k] = \{1, \ldots, k\}$. For any two integers $a \leq b$, we write $[a..b] = \{a, a+1, \ldots, b\}$. We write a k-tuple as $a = (a_1, a_2, \ldots, a_k) = (a_i)_{i \in [k]}$ or as $(a_i : i \in [k])$. For $I \subseteq [k]$, we define the subtuple with all indices of I as $a_I := (a_i : i \in I)$. We write the concatenated vector or tuple as $a \parallel b := (a_1, \ldots, a_k, b_1, \ldots, b_\ell)$.

In this paper, $n \in \mathbb{N}$ is considered to be a security parameter. We call the elements of $\{0,1\}^n$ blocks. For any nonempty subset $\mathcal{L} \subseteq \mathbb{N}$, we write $\{0,1\}^{\mathcal{L} \cdot n} = \bigcup_{l \in \mathcal{L}} \{0,1\}^{ln}$. Let $\mathbb{F} := \mathbb{F}_{2^n}$ denote the Galois field over $\{0,1\}^n$ with bitwise addition $a+b$ and field multiplication $a \cdot b$. We write $\mathbf{0} := 0^n$ and $\mathbf{1} = 0^{n-1}1$ (the additive and multiplicative identities of the field, respectively). For a statement $P(m)$ we write $\forall^* m\ P(m)$ if there exists a positive integer M such that for all integers $m > M$, $P(m)$ is true. In other words, $P(m)$ is true for all sufficiently large m. A non-negative function $\epsilon(\cdot)$ is called negligible if $\forall k$, $\forall^* m$, $\epsilon(m) \leq m^{-k}$.

Complexity Model. We fix a reasonable computational model (polynomial equivalent to the Turing machine), and the runtime of all algorithms is computed under that model. The runtime also includes the size of the algorithm's description, which would help to avoid storing arbitrarily large advise strings implicitly. It also includes reading the input and writing its output. Any algorithm A has an input set of the form $\cup_{n \in \mathbb{N}}(\{1^n\} \times D_n)$ for some $D_n \subseteq \{0,1\}^*$. The run time of A is said to be $t(n)$ if the runtime for computing $A(1^n, x)$ is at most $t(n)$ for all $x \in D_n$. If $t(n)$ is a polynomial, we call the algorithm A and the function $A(\cdot)$ realized by the algorithm *polynomial time computable* (or simply "efficient").

2.1 Hash Function

A most general hash function is defined over $\{0,1\}^*$. In the first step, an appropriate padding rule is applied to the message so that the size of the padded message is a multiple of n. Then, the hash computation is applied to the padded message with the help of some other building blocks. For the sake of simplicity, we ignore the padding rule and restrict the domain of our hash functions to $\{0,1\}^{\mathcal{L}\cdot n}$ for an appropriate set $\mathcal{L}$. We also restrict the hash output to n-bit or a multiple of n-bits.[2] An algorithm or function A is called (D,R)-function if for all $x \in D$, $A(x) \in R$.

Definition 1. *An $\mathcal{L}$-to-s computation F (or simply c-to-s computation when $\mathcal{L} = \{c\}$) is an efficiently computable function F such that for all $n \in \mathbb{N}$, $F_n := F(1^n, \cdot)$ is a $(\{0,1\}^{\mathcal{L}\cdot n}, \{0,1\}^{s\cdot n})$-function.*

Generalized Hash Mode. A hash mode is a computation that uses some building blocks (a relatively smaller domain) as oracles. A tuple of functions $H = (g_1, \ldots, g_r, g)$ is called (ℓ, r, t)*-mode* if

- g, called a *final output function*, is a $(\ell + r)$-to-1 computation, and
- g_i's, called *intermediate processing functions*, are $(\ell + i - 1)$-to-t computations, $i \in [r]$.

An (ℓ, r, t)-mode induces an ℓ-to-1 computation by applying r executions of t-to-1 computations as oracles. More formally, given r many t-to-1 computation oracles $\mathcal{O}_1, \ldots, \mathcal{O}_r$, we define the ℓ-to-1 computation $H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}$ as

$$H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(1^n, M) = g(1^n, x_1, \ldots, x_{\ell+r})$$

where $M = (x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$ and for $i \in [r]$, $\overrightarrow{y^{(i)}} \in \mathbb{F}^t, x_{i+\ell} \in \mathbb{F}$ (called *intermediate chaining inputs and outputs*, respectively) are calculated as follows (see Figure 1):

```
for $i = 1$ to $r$
    – $\overrightarrow{y^{(i)}} = g_i(1^n, M, x_{\ell+1}, \ldots, x_{\ell+i-1})$;
    – $x_{i+\ell} = \mathcal{O}_i(1^n, \overrightarrow{y^{(i)}})$;
```

We also write the intermediate chaining outputs as $\text{OUT}_H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(M) = (x_1, \ldots, x_{\ell+r})$ (whenever understood, we skip the notation H). For notational simplicity, we write $\mathcal{O}^r$ to denote r-tuple of oracles $(\mathcal{O}_1, \ldots, \mathcal{O}_r)$ and we skip 1^n (whenever n is understood or fixed in the context).

For fixed t-to-1 functions $f_1, \ldots, f_r$, $H^{f_1,\ldots,f_r}$ is a ℓ-to-1 hash function. A hash mode essentially transforms several small domain hash functions into a larger domain hash function (provided ℓ is larger than t).

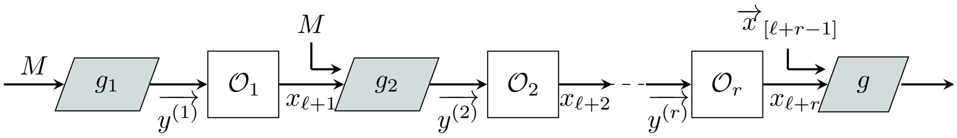


Figure 1: General Hash mode with ℓ message blocks based on r calls to underlying t-to-1 oracles.

[2] One can similarly consider other sizes of hash outputs and a similar analysis of our paper would follow.

2.2 Collision Security

UNIFORM VS. NON-UNIFORM COLLISION FINDER. We call A uniform collision-finder (or simply collision finder) if for all n, it returns a pair (M, M') of polynomial-bounded size. In notation, $(M, M') \leftarrow A(1^n)$. Let $\alpha_n \in D_n$ be a fixed string for each n. We call A non-uniform collision finder with an advise string $(\alpha_n)_n$ if for all n, $(M, M') \leftarrow A(1^n, \alpha_n)$. We also denote a non-uniform algorithm by a pair $(A, (\alpha_n))$. When α_n is constant for all n, it becomes a uniform collision finder, and we skip the notation (α_n) because it can be hard-coded in constant size inside the description of A.

COLLISION SECURITY OF A HASH FUNCTION. In the following we adopt the definition from [Rog06]. A hash computation or hash function is also a computation whose collision security is considered. For an $\mathcal{L}$-to-s hash computation H, we define the *collision advantage of* A (uniform) as

$$\mathbf{CP}_{H,n}(A) = \Pr[(M, M') \leftarrow A(1^n), H(1^n, M) = H(1^n, M')].$$

We similarly define the collision advantage for non-uniform collision finders as

$$\mathbf{CP}_{H,n}(A, (\alpha_n)_n) = \Pr[(M, M') \leftarrow A(1^n, \alpha_n), H(1^n, M) = H(1^n, M')].$$

We say that $(A, \alpha := (\alpha_n)_n)$ (or simply A in the case of the uniform algorithm) is a successful collision finder of H if A runs in polynomial time and $\mathbf{CP}_{H,n}(A, \alpha)$ is non-negligible. We write

$$\mathbf{CP}_{H,n}(t, \alpha) = \max_A \mathbf{CP}_{H,n}(A, \alpha), \quad \mathbf{CP}_{H,n}(t) = \max_A \mathbf{CP}_{H,n}(A)$$

where the maximum is taken over all $t(n)$-time collision finder A.

Definition 2 (**Collision-Resistant Hash Function** (against uniform and non-uniform adversary))**.** *A hash function H is called $(t(n), \epsilon(n), \alpha)$* ***non-uniform collision-resistant*** *if $\boldsymbol{CP}_{H,n}(t(n), \alpha) \le \epsilon(n)\ \forall^* n$. We call H collision-resistant against an advise string (α_n) if for all polynomial $t(n)$, $\boldsymbol{CP}_{H,n}(t, \alpha)$ is negligible. (In other words, there is no efficient successful (non-uniform with the advise string α) collision finder of H.) When α_n is constant we call H* ***uniform collision-resistant*** *and we skip the notation α_n.*

Collision Security Model and Assumption. Informally, a hash function H is called *collision-resistant* if it is *hard* to find a collision pair (M, M') of H (i.e., $M \neq M'$ and $H(M) = H(M')$). It is easy to see there is no meaningful way to formalize the notion of collision resistance for a single hash function. Every function H with domain size larger than range (we call those functions compression functions) must be non-injective; hence, a collision pair exists. So, a short and fast program that outputs (M, M'), which are hardwired in the program, finds a collision pair. In other words, an efficient collision finding algorithm always exists, even if we currently may not know how to write it down. To overcome this issue, one may consider a family of hash functions $H = \{H_K : K \in \mathcal{K}\}$, and the definition says that when a uniformly random public key is used, no adversary can find a collision with non-negligible probability. Another possibility is to consider a sequence of hash functions written as $H(1^n, \cdot)$, which is associated with each choice of a security parameter n, and so there exists collision-resistant hash function against a uniform adversary as one cannot simply hardwire all collision pairs into a uniform adversary for all n. So, we must assume existence of collision resistant hash function against uniform adversaries.

CLASSICAL COLLISION-RESISTANT ASSUMPTION. There exists a 2-to-1 collision-resistant function against all uniform collision finder algorithms. Using known collision-resistance preserving modes such as MD Hash, we can also assume a collision-resistant c-to-1 compression function H exists for every $c \ge 2$. We prove our main result (Theorem 1) in the uniform setting under this classical collision-resistant assumption. Then, we discuss our result in the non-uniform setting as well.

Note that for any hash function H (so a collision pair (M_n, M'_n) exists for all n), a non-uniform algorithm based on the advise string $(M_n, M'_n)_n$ can return a collision pair for all n. In other words, there cannot exist a collision secure hash function against all non-uniform collision finders. However, till now any universal advise string (α^*_n) is known such that for every H there exists A (depending on H) so that $A(1^n, \alpha^*_n)$ finds a collision pair for H for all n. This motivates us to pose the following assumption (a general assumption than the classical assumption).

Collision-Resistant Assumption against Non-uniform Collision Finder. For every advise string (α_n), a collision-resistant 2-to-1 compression function H against (α_n) exists. When α_n is constant this is same as the classical collision resistant assumption for uniform adversaries.

2.3 Reduction of Collision Security

In [Rog06], the author explained that, if a cryptographic protocol Π employs a hash function (keyed or unkeyed) H, then to prove the security of Π using a reduction-based approach, one can demonstrate the statement in existential form (C0): *If there is an effective algorithm A for attacking protocol Π, then there's an effective algorithm B for finding collisions in H.* For the unkeyed hash function H, this statement is trivially true. Therefore, the author in [Rog06] stated constructive reductions defined as follows:

Code-Constructive Form (C1) [Rog06]: *If you know an effective algorithm A for attacking protocol Π then you know an effective algorithm B for finding collisions in H.*

Black-Box-Constructive Form (C2) [Rog06]: *If you possess effective means A to attack the protocol Π, then you have effective means B to find collisions in H.*

Black-box and White-Box Reduction for Collision Security. Based on our current knowledge, it is infeasible to prove unconditionally that a hash function is collision-resistant or to prove the collision-resistant assumptions. However, we can prove a reduction that shows collision security of H given that H' is collision-resistant, where H' is used to define H. In particular, we consider two types of reduction, namely black-box reduction (C2 form) and white-box reduction (C1 form) as defined below.

Definition 3 (Black-box and White-box Reduction). *Let H, H' be some hash functions and α, α' be some advise strings.*

1. *We call collision security of H is α* black-box reduced *to $H'_1, \ldots, H'_r$ if there is an efficient algorithm A such that for any sequence of collision pairs (M_n, M'_n) of $H(1^n, \cdot)$, $A(1^n, (\alpha_n, M_n, M'_n)_n)$ is a successful collision finder of $H'_i(1^n, \cdot)$, for some $i \in [r]$.*
2. *We call collision security of H is (α, α')* white-box reduced *to $H'_1, \ldots, H'_r$ if there is an efficient algorithm A such that for any successful collision finder $A'(1^n, \alpha')$ of $H(1^n, \cdot)$, $A(1^n, (\textsc{code}(A'), \alpha_n, \alpha'_n)_n)$ is also a successful collision finder of $H'_i(1^n, \cdot)$ for some $i \in [r]$, where $\textsc{code}(A')$ represents the algorithmic description of A'.*

We skip the notations α and α' whenever these are constant.

A black-box reduction algorithm (C2) finds a collision of H whenever a collision pair of H' is given to it. Whereas, a white-box reduction (C1) algorithm finds a collision of H whenever a code of an efficient collision-finder of H' is given. So, the existence of black-box reduction implies the existence of white-box reduction. For example, it is very well known that the Merkle-Damgård hash function, Merkle tree hash function, etc., are collision-resistant given that the underlying compression function is collision-resistant. This is essentially a black-box reduction. Those reductions can be easily converted to white-box also.

Remark 1. According to the general definition of black-box reduction, a hash function H is considered black-box reduced to H' if an efficient algorithm B can attack the protocol H, enabling the construction of an algorithm A to find collisions in H'. In this specific context of collision, we simplify the definition of black-box reduction. If there is an efficient algorithm B to find collisions in H, B simply returns the collision pairs for H. The efficient algorithm A then uses those collision pairs from H (provided by algorithm B) to construct collision pairs for H'.

2.4 Input-Output Fixing Collision-Resistant Compression Function

An *input-output fixing function* f is a function that compresses its input while preserving specific fixed input-output pairs. Let $\alpha = (\alpha_1, \ldots, \alpha_p) \in D^p$ and $\beta = (\beta_1, \ldots, \beta_p) \in R^p$, consisting of distinct values for some fixed p. A function $f : D \to R$ is called $\alpha \mapsto \beta$ input-output fixing mapping if for all i, $f(\alpha_i) = \beta_i$. We now extend this definition to an asymptotic setup. Let $\alpha = (\alpha_n)_n$ and $\beta = (\beta_n)_n$ where $\alpha_n = (\alpha_{n,1}, \ldots, \alpha_{n,p}) \in (\{0,1\}^{nt})^p$ and $\beta_n = (\beta_{n,1}, \ldots, \beta_{n,p}) \in (\{0,1\}^n)^p$ consisting of distinct values for all n. We call any such pair (α, β) (t,p)-in-out pair. A (t,p)-in-out pair is called computable if there is an efficient algorithm G such that $G(1^n)$ returns (α_n, β_n) for all n.

Definition 4 (**Input-Output Fixing Function**). *Let (α, β) be (t,p)-in-out pair. A t-to-1 function f is called an $\alpha \mapsto \beta$ input-output fixing mapping, if*

$$f(1^n, \alpha_{n,i}) = \beta_{n,i}, \text{ for all } i \in [p], n \in \mathbb{N}.$$

In other words, for all n, the function $f_n := f(1^n, \cdot)$ is $\alpha_n \mapsto \beta_n$ input-output fixing mapping.

Proposition 1. *Let (α, β) be a (t,p)-in-out computable pair. Suppose the classical collision-resistant assumption is true. Then, there is a $\alpha \mapsto \beta$ input-output fixing collision-resistant compression function.*

Proof. Let $t \geq 2$ be a fixed integer, and let $c \geq 2t$ be some fixed integer. Suppose h' is a c-to-1 collision-resistant compression function (it exists by our classical collision-resistant assumption). For all sufficiently large n, we define $k := \lfloor n/2 \rfloor > \lceil \log p \rceil$ and $n' = n - k$. Suppose $q \in \{0,1\}^k$ is different from the last k bits of all $\beta_{n,i}$'s and be the smallest possible value among all k-bit values when considered as an integer (note that it can be computed efficiently). Now we define

$$h_{\alpha,\beta}(1^n, x) = \begin{cases} \beta_{n,i} & \text{if } x = \alpha_{n,i}, \text{ for all } i \in [p] \\ h'(1^{n'},\ x \parallel 0^{n(c-t)-ck}) \parallel q & \text{otherwise} \end{cases}$$

Clearly, $h_{\alpha,\beta}$ is a $\alpha \mapsto \beta$ input-output fixing mapping. Now, we prove that $h_{\alpha,\beta}$ is collision-resistant. The function $h_{\alpha,\beta}$ is easily seen to be efficiently computable.

Let an adversary A on the collision resistance of $h_{\alpha,\beta}$ be given, which returns a pair (M, M'). Now, whenever (M, M') is a collision pair, then both M, and M' can not be $\alpha_{n,i}$, $\forall i \in [p]$ as α_n, and β_n are two tuples of distinct values, for all n. Moreover, it can not be possible that M is one of $\alpha_{n,i}$ for some $i \in [p]$, and M' is other than $\alpha_{n,j}$, $\forall j \in [p]$ as q is different from the last k bits of all $\beta_{n,i}$'s. Therefore, $(M \parallel 0^{n(c-t)-ck}, M' \parallel 0^{n(c-t)-ck})$ is a collision pair of h'_{n-k}. □

Therefore, we have shown that given a collision-resistant compression function, we can define another compression function that maps to certain outputs for some fixed inputs.

We call such a function an input-output fixing collision-resistant compression function. Moreover, this proof shows the black-box reduction (and hence white-box reduction) from h to h'.

3 Linear Hash Modes

3.1 Linear Algebra Lemma

Vectors. Given two m-dimensional vectors $\overrightarrow{u} = (u_1, \ldots, u_m), \overrightarrow{v} = (v_1, \ldots, v_m) \in \mathbb{F}^m$, we compute the dot-product

$$\overrightarrow{u} \cdot \overrightarrow{v} := \sum_{i=1}^{m} u_i \cdot v_i.$$

Given a set of m-dimensional vectors $\mathcal{V}$, we denote the following terminologies widely used in linear algebra:

- $\mathsf{span}(\mathcal{V})$: the set of all vectors which are linearly dependent on the vectors of $\mathcal{V}$;
- $\mathsf{null}(\mathcal{V})$: the set of all m-dimensional vectors $\overrightarrow{x}$ such that $\overrightarrow{v} \cdot \overrightarrow{x} = \mathbf{0}$ for all $\overrightarrow{v} \in \mathcal{V}$;
- $\mathsf{rank}(\mathcal{V})$: the maximal number of linearly independent vectors in $\mathcal{V}$;

We note that $\mathsf{rank}(\mathsf{span}(\mathcal{V})) = \mathsf{rank}(\mathcal{V})$. The *rank-nullity theorem* says that

$$\mathsf{rank}(\mathcal{V}) + \mathsf{rank}(\mathsf{null}(\mathcal{V})) = m.$$

A function $f : \mathbb{F}^m \to \mathbb{F}^t$ is called m-to-t function.

Lemma 1 (**Linear Algebra Lemma**). *Let $\mathcal{S}, \mathcal{N} \subseteq \mathbb{F}^m \setminus \{\mathbf{0}^m\}$ be two nonempty sets such that (i) $\mathcal{N}$ and $\mathsf{span}(\mathcal{S})$ are disjoint and (ii) $|\mathcal{N}| < 2^n$. Then,*

$$\textit{there exists } \overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \textit{ such that } \forall \overrightarrow{v} \in \mathcal{N},\ \overrightarrow{v} \cdot \overrightarrow{y} \neq \mathbf{0}.$$

Equivalently, $\exists \overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \setminus \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})$.

Proof. Note that for all $\overrightarrow{v} \in \mathcal{N}$, $rank(\mathcal{S} \cup \{\overrightarrow{v}\}) = r + 1 \leq m$ where $r = rank(\mathcal{S})$. So, by rank-nullity theorem $|\mathsf{null}(\mathcal{S})| = (2^n)^{m-r}$ and $|\mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})| = (2^n)^{m-r-1}$ for all $v \in \mathcal{N}$. Hence,

$$|\mathsf{null}(\mathcal{S}) \setminus \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})| \geq 2^{nm-nr} - |\mathcal{N}| 2^{nm-nr-n} \geq 2^{nm-nr} \left[1 - \frac{|\mathcal{N}|}{2^n}\right] > 0.$$

This shows that the above set is nonempty and hence the proof follows. □

Remark 2. Given the set of vectors $\mathcal{S}, \mathcal{N}$, in order to find such

$$\overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \setminus \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\}),$$

it is sufficient to find a basis of $\mathsf{null}(\mathcal{S})$, and $\mathsf{null}(\mathcal{S} \cap \{\overrightarrow{v}\})$ for all $\overrightarrow{v} \in \mathcal{N}$. A basis of the null space of a set of vectors may be computed by Gaussian elimination or may be computed by the Bareiss algorithm [Bar68], which may work more efficiently than Gaussian elimination. The efficiency of this algorithm depends on the cardinalities of $\mathcal{S}$ and $\mathcal{N}$.[3]

Let $\overrightarrow{u} \in \mathbb{F}^m, \overrightarrow{v} \in \mathbb{F}^{m'}$ and $\mathcal{Z} \subseteq [m]$. We write $\overrightarrow{u}|_{\mathcal{Z}} = (u'_1, \ldots, u'_m)$ where $u'_i = u_i$ for all $i \notin \mathcal{Z}$, otherwise $u'_i = \mathbf{0}$. We call $\overrightarrow{u}|_{\mathcal{Z}}$, *$\mathcal{Z}$-projected vector* of $\overrightarrow{u}$. Note that

$$\text{for all } \overrightarrow{x} \in \mathbb{F}^m, \quad \overrightarrow{u}|_{\mathcal{Z}} \cdot \overrightarrow{x} = \overrightarrow{u_{\bar{\mathcal{Z}}}} \cdot \overrightarrow{x_{\bar{\mathcal{Z}}}}.$$

[3] In our main proof, where we use the results of Lemma 1, the cardinality of $\mathcal{S}$ and $\mathcal{N}$ is upper bounded by approximately $(t + 1)r + \ell + 1$, where the linear hash mode processes ℓ blocks of messages using r many tn-to-n bit compression function calls.

3.2 Linear Functions and Tuples of Vectors

m-to-1 Linear Function. To each vector $\overrightarrow{u} \in \mathbb{F}^m$, we associate a m-to-1 linear function u, defined as $u(\overrightarrow{x}) = \overrightarrow{u} \cdot \overrightarrow{x}$. For $i \in [m]$, the linear function $e_i(x_1, x_2, \ldots, x_m) = x_i$ has an equivalent vector representation $\overrightarrow{e_i}$, called the i^{th} *coordinate vector*, whose i^{th} element is $\mathbf{1}$, and all the remaining elements are $\mathbf{0}$. It is well known that every linear function over $\mathbb{F}$ can be uniquely represented by a function $u(\cdot)$ as defined above for some $\overrightarrow{u} \in \mathbb{F}^m$. So, we use the convention $\overrightarrow{u}$ to denote the corresponding vector for a linear function u and vice-versa. For $u : \mathbb{F}^\ell \to \mathbb{F}$ and $v : \mathbb{F}^r \to \mathbb{F}$, we define the joint linear function

$$(u \parallel v) : \mathbb{F}^{\ell+r} \to \mathbb{F}, \quad \overrightarrow{x} \mapsto u(x_1, \ldots, x_\ell) + v(x_{\ell+1}, \ldots, x_{\ell+r}), \ \overrightarrow{x} \in \mathbb{F}^{\ell+r}.$$

Note that $\overrightarrow{u \parallel v} = \overrightarrow{u} \parallel \overrightarrow{v}$, hence the concatenation notation | is consistent in both vector and linear function representation.

m-to-t Linear Function. An m-to-t linear function $\mathbf{u} = (u_1, \ldots, u_t)$, can be represented by a t-tuple of vectors $\overrightarrow{\mathbf{u}} := (\overrightarrow{u_1}, \ldots, \overrightarrow{u_t}) \in (\mathbb{F}^m)^t$ such that for $\overrightarrow{x}$, we have

$$\mathbf{u}(\overrightarrow{x}) = (\overrightarrow{u_1} \cdot \overrightarrow{x}, \ldots, \overrightarrow{u_t} \cdot \overrightarrow{x}) := (u_1(\overrightarrow{x}), \ldots, u_t(\overrightarrow{x})).$$

Note that a pair of vectors $(\overrightarrow{u_1}, \overrightarrow{u_2})$ is different from the concatenation $\overrightarrow{u_1} \parallel \overrightarrow{u_2}$ as they correspond to two different linear functions.

For $k \in [t]$, we write $\overrightarrow{u_k} = (u_{k,1}, \ldots, u_{k,m}) \in \mathbb{F}^m$ (indices corresponding to components of vectors appear at the end). We use bold letters like $\mathbf{u}$ to denote m-to-t linear functions and vector arrow over bold letters like $\overrightarrow{\mathbf{u}}$ to denote t-tuples of m-dimensional vectors.

Tuple of m-to-t Linear Function. Now, we further generalize and consider a r-tuple of m-to-t linear function $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$, i.e., for $i \in [r]$, $\mathbf{u}^{(i)}$ is a m-to-t linear function. This actually induces a linear function, abusing notation, $\mathcal{U} : \mathbb{F}^m \to (\mathbb{F}^t)^r$, such that for all $\overrightarrow{x} \in \mathbb{F}^m$,

$$\mathcal{U}(\overrightarrow{x}) = (\mathbf{u}^{(1)}(\overrightarrow{x}), \ldots, \mathbf{u}^{(r)}(\overrightarrow{x})).$$

We use calligraphic fonts to denote such tuples of m-to-t linear functions. We write

$$\overrightarrow{\mathbf{u}^{(i)}} := (\overrightarrow{u_1^{(i)}}, \ldots, \overrightarrow{u_t^{(i)}}), \quad \overrightarrow{u_j^{(i)}} := (u_{j,1}^{(i)}, \ldots, u_{j,m}^{(i)}) \in \mathbb{F}^m, i \in [r], j \in [t].$$

Definition 5 (**Triangular-Dependent**)**.** *An m-to-t linear function* $\mathbf{u}$ *and its vector representation* $\overrightarrow{\mathbf{u}} = (\overrightarrow{u_1}, \ldots, \overrightarrow{u_t})$ *are called i-onward independent if* $u_{k,i} = \ldots = u_{k,m} = \mathbf{0}$*, for all* $k \in [t]$*.*

Let $r < m$. A r-tuple of m-to-t linear functions $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$ *is called* triangular-dependent *if for all* $i \in [r]$*,* $\mathbf{u}^{(i)}$ *is* $(i + (m - r))$*-onward independent. We call* $\mathcal{U}$ (ℓ, r, t)*-triangular-dependent where* $\ell = m - r$*.*

If $\mathbf{u}$ is a m-to-t i-onward independent linear function then for all $\overrightarrow{x}, \overrightarrow{y} \in \mathbb{F}^m$ with $x_1 = y_1, \ldots, x_{i-1} = y_{i-1}$, we have $\mathbf{u}(\overrightarrow{x}) = \mathbf{u}(\overrightarrow{y})$. In other words, $\mathbf{u}(x_1, \ldots, x_m)$ is functionally independent of $x_i, \ldots, x_m$. This justifies the term "i-onward independent". So, for $1 \leq i \leq m$, any $(i-1)$-to-t linear function $\mathbf{u}'$ is equivalent to a m-to-t i-onward independent linear function $\mathbf{u}$ such that

$$\mathbf{u}(x_1, \ldots, x_m) = \mathbf{u}'(x_1, \ldots, x_{i-1}), \quad \forall \overrightarrow{x} \in \mathbb{F}^m.$$

Convention. Suppose $\mathbf{u}^{'(i)}$ is $(\ell + i - 1)$-to-t linear function, $i \in [r]$. As discussed above, we can equivalently represent $\mathbf{u}^{'(i)}$ by i-onward independent m-to-t $\mathbf{u}^{(i)}$ linear function. Hence, we equivalently represent $\mathcal{U}' := \left(\mathbf{u}^{'(1)}, \ldots, \mathbf{u}^{'(r)}\right)$ by a triangular dependent r-tuple of m-to-t linear functions $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$. We follow this convention while we define linear hash mode.

3.3 Linear Hash Mode

(ℓ, r, t) Linear Hash Mode. *Linear Hash Mode* is a generalized hash mode $(g_1, \ldots, g_r, g)$ where the intermediate processing functions $g_i(1^n, \cdot)$'s and the final output function $g(1^n, \cdot)$ are linear for all n. We equivalently represent a linear hash mode by a pair $H(1^n, \cdot) := (\mathcal{U}_n := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}), g(1^n, \cdot))$ where $\mathcal{U}_n$ is a triangular dependent r-tuple of m-to-t linear functions and g is a m-to-1 linear function, where $m = \ell + r$. We sometimes skip the notation 1^n for the sake of simplicity.

1. We call H *simple* if $g(x_1, \ldots, x_m) = x_m$. A simple linear hash mode is defined by $(\mathcal{U}_n := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}))_n$ and denoted as $\text{NLH}_{\mathcal{U}}$.

2. On the other hand, we call H *non-simple linear hash mode* if

$$g(x_1, \ldots, x_m) = c_1 x_1 + c_2 x_2 + \ldots + c_m x_m$$

where $c_m \neq \mathbf{0}$, and there exists at least one $i \in [m-1]$ such that $c_i \neq \mathbf{0}$. A non-simple linear hash mode is defined by $(\mathcal{U}_n, g)_n$ and is denoted as $\text{CLH}_{(\mathcal{U},g)}$.

Example 1 (**Merkle-Damgård Hash Function**)**.** Let $f : \mathbb{F}^t \to \mathbb{F}$ be a t-to-1 compression function. The Merkle-Damgård (MD) hash function derived from f, denoted $(\text{MD})^f$ is a hash function defined over $\mathbb{F}^\ell$ using $r = \lceil (\ell-1)/(t-1) \rceil$, t-to-1 compression function calls that works as follows:

$$(MD)^f(m_0, m_1, \ldots, m_{\ell-1}) = f\big(\cdots f(f(m_0, m_1, \ldots, m_{t-1}), m_t, \ldots, m_{2t-2}), \cdots, m_{\ell-1}\big)$$

Here, $m_i \in \mathbb{F}$, for all $0 \leq i \leq \ell - 1$. Merkle-tree variant for a binary tree (based on a 2-to-1 compression function) is a hash function using $(\ell - 1)$ compression function calls where $\ell = 2^d$ for $d \in \mathbb{N}$. Both Merkle-Damgård and Merkle tree hash are simple linear hash modes.

Example 2 (**T_5 & ABR Hash Function**)**.** T_5 hash function, described in [DKMN21], is designed to handle a message input consisting of five blocks, which requires three 2-to-1 compression function calls. As discussed in [ABR21], the ABR hash function operates as a perfect binary tree hash with a height denoted by ℓ. It processes a total of $m = (2^\ell + 2^{\ell-1} - 1)$ message blocks and executes $(2^\ell - 1) = (m - 2^{\ell-1})$ 2-to-1 compression function calls. It is important to highlight that both are non-simple linear hash modes.

Example 3 (**SS Hash Function**)**.** Shrimpton-Stam construction [SS08] is a $2n$-to-n bit compression function based on three n-to-n-bit non-compressing primitives. Now, using three $2n$-to-n-bit compression functions, we can compress five message blocks, adding one block for each compression function (trivially extended version[4] of Shrimptom-Stam construction). It's worth noting that Shrimpton-Stam hash mode falls under the category of non-simple linear hash modes.

4 Our Main Result

In this section, we present our main result, which establishes the lower bound on the number of compression function calls required to achieve collision-resistance preserving property in an (ℓ, r, t)-linear hash mode.

Theorem 1. *[**Main Theorem**] Let H be an (ℓ, r, t) linear hash mode satisfying either (i) $t \geq 2$, $r < \lceil (\ell-1)/(t-1) \rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr + \ell + r) < 2^n$. Then, there exists*

[4] Note that this construction has not been proposed formally.

- *t-to-1 collision-resistant compression functions* $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$[5] *and*
- *an uniform collision finder A such that* $\boldsymbol{CP}_{H^{f_1, f_2, \ldots, f_r}, n}(A) = 1, \forall^* n$.

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

We already know (ℓ, r, t) linear hash mode having Merkle-Damgård and Merkle tree hash structures are collision-resistant assuming underlying compression functions are collision-resistant, and in this case $r = \lceil (\ell - 1)/(t - 1) \rceil$, for $t \geq 2$. The above result says we cannot have a more efficient collision-resistance preserving hash mode than Merkle-Damgård and Merkle tree hash mode. Recently, T_5, ABR hash were shown to be collision-resistant in the random oracle model [ABR21, DKMN21]. The above theorem shows that it is impossible to prove the collision security of ABR and T_5 based on only the collision security assumption of the underlying compression function. We present the detailed attack on T_5 in Subsection 4.1. For the detailed attack on ABR hash mode, please refer to Subsection 8.1 (as the attack procedure is almost similar to the attack procedure of T_5, we choose supplementary material for detailed explanation). It is important to note that Shrimpton-Stam construction also does not have collision-resistance preserving property. Similar reasoning can be applied as we do for T_5 and ABR constructions. All known hash modes are linear, indicating that linear hash modes represent a substantial class of hash functions.

4.1 Absence of Collision-Resistance Preserving Property in T_5

Our goal is to prove that T_5 does not have collision-resistance preserving property, i.e., there exists some collision-resistant compression function f, such that T_5^f is not collision-resistant. We start our proof by establishing an observation on T_5 hash mode. We notice that, if the underlying compression function f of the T_5 hash mode satisfy the following constraints:

$$f(\mathbf{0}, \Delta) = \Delta, \; f(\mathbf{0}, \mathbf{0}) = \mathbf{0} \tag{1}$$

then, two messages $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$, and $(\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ with non-zero message difference $(\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$ construct a collision pair for T_5^f (see Figure 2). It is important to note that the compression function f meeting the mentioned properties (Equation 1) does not need to be collision-resistant. Therefore, the task remains to construct some collision-resistant compression function, say F, and identify a message pair that produces a collision in the T_5^F hash mode.

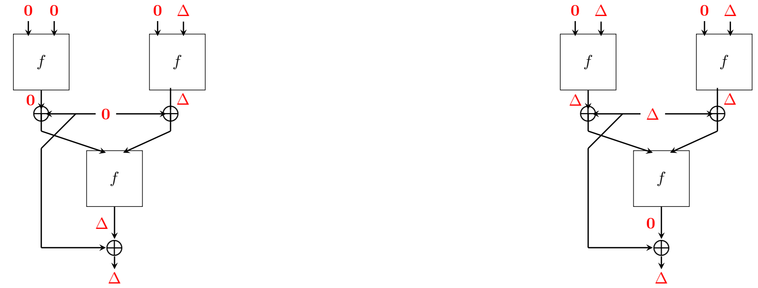
Based on this observation, we define a 2-to-1 compression function F as follows:

$$F(x) = \begin{cases} \mathbf{0} & \text{if } \; x = (\mathbf{0}, \mathbf{0}) \\ \Delta & \text{if } \; x = (\mathbf{0}, \Delta) \\ f'(x \parallel 0^{n-3k}) \parallel q & \text{Otherwise} \end{cases}$$

where f' is a 3-to-1 collision-resistant compression function (it exists by our classical collision-resistant assumption) and $q \in \{0,1\}^k \setminus \{0^k\}$ is different from the last k bits of Δ, and is the smallest possible value among all k-bit values when considered as an integer with $k := \lceil n/2 \rceil$.

For instance, F can also be seen as $h_{\alpha,\beta}$ defined as in Proposition 2.4 for $\alpha = (\alpha_1, \alpha_2) \in \{0,1\}^{4n}$ such that $\alpha_1 = (\mathbf{0}, \mathbf{0})$, and $\alpha_2 = (\mathbf{0}, \Delta)$, and $\beta = (\beta_1, \beta_2) = (\mathbf{0}, \Delta) \in \{0,1\}^{2n}$,

[5] The phrase "there exists t-to-1 collision-resistant compression functions" refers to a specific category of collision-resistant compression functions for which we can prove that H is not collision-resistant, rather than any arbitrary collision-resistant compression function. To demonstrate the existence of this particular type of collision-resistant compression function, we must rely on the assumption that collision-resistant hashes exist.



(a) Processing of message $m = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ through T_5^f

(b) Processing of message $m' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ through T_5^f

Figure 2: Processing of non-zero message difference $(\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$ through T_5^f, where f satisfies Equation 1

with $p = t = 2$, $c = 3$. Therefore, Using Proposition 1, we can easily conclude that F is collision-resistant. Finally, for the T_5 hash construction using F as its compression function, i.e., for T_5^F, if one process $M = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ and $M' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ in T_5^F, then

$$\mathrm{T}_5^F(M) = \mathrm{T}_5^F(M') = \Delta \neq \mathbf{0}.$$

Remark 3. To prove the general lower bound on linear hash mode H to achieve collision-resistance preserving property, mentioned in Subsection 1.1, we have to construct some particular type of collision-resistant compression function f, a similar construction is necessary as described in T_5. Hence, we establish such construction generally in Proposition 2.4 and also prove that such construction is collision-resistant.

4.2 Proof of Theorem 1

In this section, we demonstrate the proof of Theorem 1. First, we state and explain some necessary definitions and results to finally prove the Theorem 1.

Difference Propagation through (ℓ, r, t) Linear Hash Mode. Let H be an (ℓ, r, t) linear hash mode where $\mathcal{U} := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$. Suppose we process two messages $M = (x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$, and $M' = (x_1', \ldots, x_\ell') \in \mathbb{F}^\ell$ through $H^{\mathcal{O}^r}$ and let the intermediate chaining outputs and inputs be

$$\begin{aligned} \textsc{out}_H^{\mathcal{O}^r}(M) = (x_1, \ldots, x_m), &\quad \mathcal{U}(\overrightarrow{x}) = \mathcal{Y} := (\overrightarrow{y^{(1)}}, \ldots, \overrightarrow{y^{(r)}}) \\ \textsc{out}_H^{\mathcal{O}^r}(M') = (x_1', \ldots, x_m'), &\quad \mathcal{U}(\overrightarrow{x'}) = \mathcal{Y}' := (\overrightarrow{y'^{(1)}}, \ldots, \overrightarrow{y'^{(r)}}) \end{aligned}$$

Let $m = \ell + r$, $\delta x_i = x_i \oplus x_i'$ for all $i \in [m]$. Clearly,

$$\mathbf{u}^{(i)}(\delta x_1, \ldots, \delta x_m) = \overrightarrow{\delta y^{(i)}} := \overrightarrow{y^{(i)}} \oplus \overrightarrow{y'^{(i)}}.$$

Note that

$$\overrightarrow{\delta y^{(i)}} = \mathbf{0}^t \Rightarrow \delta x_{i+\ell} = \mathbf{0}, \quad \forall i \in [r]$$

(zero differences in inputs of the oracles would lead to zero difference in the output). However, to avoid collisions in the inputs and outputs of the oracles, we also need non-zero differences in the inputs to lead to non-zero differences in the outputs, and hence, we require

$$\overrightarrow{\delta y^{(i)}} = \mathbf{0}^t \iff \delta x_{i+\ell} = \mathbf{0}, \quad \forall i \in [r]. \tag{2}$$

Definition 6 (**Compatible Vectors**). *We call an m-dimensional vector $\overrightarrow{\delta}$, $\mathcal{U}$-*compatible *if*

$$\forall i \in [r],\ \delta_{i+\ell} = \mathbf{0} \text{ if and only if } \overrightarrow{z^{(i)}} = \mathbf{0}^t, \ \text{ where } \mathbf{u}^{(i)}(\delta_1, \dots, \delta_m) = \overrightarrow{z^{(i)}}.$$

Moreover, it is called a ***collision-compatible vector*** *or* ***collision $\mathcal{U}$-compatible*** *if $\overrightarrow{\delta}$ is non-zero and $\delta_m = \mathbf{0}$. If $\overrightarrow{\delta}$ is non-zero $\mathcal{U}$-compatible, then $\delta_{[\ell]}$ is also non-zero (since, otherwise, the whole vector becomes zero vector using the definition of compatibility.*

Now, we show that the existence of a collision-compatible vector can lead to a collision of the hash mode provided the underlying oracles satisfy certain input-output constraints. More formally, we have the following lemma.

Lemma 2. *Let $\overrightarrow{\delta}$ be a $\mathcal{U}$-compatible m-dimensional vector for an (ℓ, r, t) linear hash mode $H := (\mathcal{U}, g)$. $\mathcal{O}_i(\overrightarrow{y^{(i)}}) = x_{i+\ell}$ and $\mathcal{O}_i(\overrightarrow{y^{'(i)}}) = x'_{i+\ell}$ where $\mathcal{U}(\overrightarrow{x}) = \mathcal{Y}$ and $\mathcal{U}(\overrightarrow{x'}) = \mathcal{Y}'$, we have*

$$\textsc{out}_H^{\mathcal{O}^r}(M) = \overrightarrow{x} = (x_1, \dots, x_{\ell+r})$$
$$\textsc{out}_H^{\mathcal{O}^r}(M') = \overrightarrow{x'} = (x'_1, \dots, x'_{\ell+r}).$$

Moreover, we have the following:

1. *If $\overrightarrow{\delta}$ is a collision-compatible, then*
$$\textsc{nlh}_{\mathcal{U}}^{\mathcal{O}_1,\dots,\mathcal{O}_r}(M) = x_{\ell+r} = x'_{\ell+r} = \textsc{nlh}_{\mathcal{U}}^{\mathcal{O}_1,\dots,\mathcal{O}_r}(M').$$

2. *If $g(\overrightarrow{\delta}) = \mathbf{0}$, then*
$$H^{\mathcal{O}_1,\dots,\mathcal{O}_r}(M) = g(\overrightarrow{x}) = g(\overrightarrow{x'}) = H^{\mathcal{O}_1,\dots,\mathcal{O}_r}(M'),.$$

The proof of the above lemma is straightforward from the definition. The last statement on the intermediate output is easy to observe. From this, the collision of hash output follows as the hash function is simple hash. We also note that in the above lemma, $M \neq M'$. Since otherwise, the intermediate inputs of M and M' are the same and hence contradicting $\overrightarrow{x} \oplus \overrightarrow{x'} = \overrightarrow{\delta} \neq \mathbf{0}^m$. This Lemma describes collision pairs of a linear hash mode $H = (\mathcal{U}, g)$, given that a collision-compatible differential vector $\overrightarrow{\delta}$ exists.

Now, we are ready to prove our main result with the above definitions and results. We know that the linear hash mode can be classified into two types based on its structure: simple and non-simple. Therefore, we first prove Theorem 1 for simple linear hash mode, followed by non-simple linear hash mode. More formally, for simple linear hash mode, we aim to prove the following statement:

Proposition 2. *[**Proposition for Simple Linear Hash Mode**] Let $\textsc{nlh}_{\mathcal{U}}$ be an (ℓ, r, t)-linear simple hash mode satisfying either (i) $t \geq 2$, $r < \lceil (\ell - 1)/(t - 1) \rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr + \ell + r) < 2^n$. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \dots, f_r(1^n, \cdot)$ and*
- *an uniform collision finder A such that $\boldsymbol{CP}_{H,n}(A) = 1, \forall^* n$ where $H = \textsc{nlh}_{\mathcal{U}}^{f_1,\dots,f_r}$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \dots, f_r)$ does not exist.

Proof of Proposition 2. $\text{NLH}_{\mathcal{U}}$ be an (ℓ, r, t)-linear simple hash mode satisfying either (i) $t \geq 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr + \ell + r) < 2^n$. Therefore, by definition, we can represent $\text{NLH}_{\mathcal{U}}$ as $\mathcal{U} = (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$, where $\mathcal{U}$ be a triangular dependent tuple of m-to-t linear function with $m = \ell + r$. Therefore, we state and prove the following Lemma:

Lemma 3. *Let $\mathcal{U} := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$ be a triangular dependent tuple of m-to-t linear function where $m = r + \ell$ and either (i) $t \geq 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr + \ell + r) < 2^n$. Then, a collision $\mathcal{U}$-compatible vector exists, which can be computed efficiently.*

The proof of this lemma is intricate and lengthy, requiring several steps. Consequently, we have dedicated a separate section (Section 5) to its proof.

Now, we can see the above Lemma (Lemma 3) ensures the existence of efficiently computable collision $\mathcal{U}$-compatible vector $\overrightarrow{\delta}$. We split $\overrightarrow{\delta}$ as $\overrightarrow{x} \oplus \overrightarrow{x'} = \overrightarrow{\delta}$. Then, by using Lemma 2, we have an efficient algorithm A (which simply returns (M, M') computed from $\overrightarrow{x}, \overrightarrow{x'}$ respectively) such that $\mathbf{CP}_{H,n}(A) = 1$. This holds for any hash function as long as oracles satisfy certain input-output constraints. By Proposition 1, we have such collision-resistant compression functions. Note that the vectors $\overrightarrow{x}, \overrightarrow{x'}$ can be computed efficiently (given the description of linear hash mode).

Now, it remains to prove Theorem 1 for the case of non-simple linear hash mode. Precisely, we prove the following:

Proposition 3. *[**Proposition for Non-simple Linear Hash Mode**] Let $H := \text{CLH}_{(\mathcal{U},g)}$ be an (ℓ, r, t) non-simple linear hash mode satisfying either (i) $t \geq 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr + \ell + r) < 2^n$. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$ and*
- *an uniform collision finder A such that $\boldsymbol{CP}_{H^{f^r},n}(A) = 1, \forall^* n$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

Proof of Proposition 3. For non-simple linear hash mode $\text{CLH}_{(\mathcal{U},g)}$ satisfying the conditions specified above, we can choose the index set $Z = \emptyset$, and construct the following two sets of vectors as follows:

$$\mathcal{S} = \{\overrightarrow{g}\}$$
$$\mathcal{N} = \{\overrightarrow{u_j^{(i)}} : i + \ell \in [\ell + r], j \in [t]\} \cup \{\overrightarrow{e_k} \ : \ k \in [\ell + r]\}$$

Note that $\mathsf{span}(\mathcal{S})$ and $\mathcal{N}$ are disjoint as $g(x_1, \ldots, x_{\ell+r})$ has non-zero coefficient for $x_{\ell+r}$, whereas $x_{\ell+r}$ is not present in any vector of $\mathcal{N}$. Now $\mathcal{N}$ contains $(tr + \ell + r) < 2^n$ vectors. Hence, using Lemma 1, there always exists $\overrightarrow{\delta} \in \mathsf{null}(\mathcal{S})$ such that $\forall \overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{\delta} \cdot \overrightarrow{v} \neq \mathbf{0}$ (and hence for all $i \in [\ell + r]$, $\delta_i \neq \mathbf{0}$) and $g(\overrightarrow{\delta}) = \mathbf{0}$. We already know that the vectors $\overrightarrow{u_j^{(i)}}$ belongs to $\mathcal{N}$, for all $i + \ell \in [\ell + r]$, then we can easily conclude that, for all $j \in [t]$,

$$\overrightarrow{u_j^{(i)}} \cdot \overrightarrow{\delta} \neq \mathbf{0} \Rightarrow \mathbf{u}^{(i)}(\overrightarrow{\delta}) \neq \mathbf{0}^t \Rightarrow \overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t$$

where $\mathcal{U}(\overrightarrow{\delta}) = (\overrightarrow{\beta^{(1)}}, \ldots, \overrightarrow{\beta^{(r)}})$. This immediately implies that $\overrightarrow{\delta}$ is $\mathcal{U}$-compatible (since all intermediate input and output vectors are non-zero).

Now we split $\overrightarrow{\delta} = \overrightarrow{x} \oplus \overrightarrow{x'}$. Note that the vectors $\overrightarrow{x}, \overrightarrow{x'}$ can be computed efficiently (given the description of linear hash mode). Therefore, by using Lemma 2 and Proposition 1 (similar to the proof of the simple linear has mode), we can have an efficient algorithm A, which returns a message pair (M, M') computed from $\overrightarrow{x}$, $\overrightarrow{x'}$ such that $\mathbf{CP}_{H,n}(A) = 1$.

Non-Uniform Setting. Based on the collision-resistant assumption against a non-unifrom collision finder, we can state our result as follows:

Theorem 2. *Let H be an (ℓ, r, t) linear hash mode satisfying either (i) $t \geq 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geq 2$ with $(tr+\ell+r) < 2^n$ and let $\alpha := (\alpha_n)_n$ be an advise string. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$ against α and*
- *an uniform collision find,er A such that $\boldsymbol{CP}_{H^{f_1,f_2,\ldots,f_r},n}(A) = 1, \forall^* n$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

The proof approach is similar to what we have done in the uniform setup.

5 Proof of Lemma 3

$\mathcal{U}$-consistent Index set. Let $Z = \{i \in [\ell + r] : x_i = \mathbf{0}\}$ be called a *zero-index* set of $\overrightarrow{x}$. Now, for all $i + \ell \in Z$, $\mathbf{u}^{(i)}(\overrightarrow{x}) = \mathbf{0}^t$ and so for all $1 \leq j \leq t$, $\overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{x} = \mathbf{0}$. Let

$$\mathcal{S} = \{\overrightarrow{u_j^{(i)}}|_Z : i + \ell \in Z, j \in [t]\}.$$

Thus, if for some $k \in [\ell + r]$, $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S})$ then $x_k = \overrightarrow{e_k} \cdot \overrightarrow{x} = \mathbf{0}$, and hence $k \in Z$. So, we state our first necessary condition:

N1: $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \Rightarrow k \in Z$.

By using the similar argument (using the condition, $\overrightarrow{y^{(k)}} = \mathbf{0}^t$ if and only if $k + \ell \in Z$) we have

N2: $i + \ell \notin Z \Rightarrow \exists j \in [t], \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S})$. A contra-positive statement says that $\forall j \in [t], \overrightarrow{u_j^{(i)}}|_Z \in \mathsf{span}(\mathcal{S}) \Rightarrow i + \ell \in Z$.

Any set Z satisfying the above two necessary conditions is called $\mathcal{U}$-consistent or simply *consistent* whenever the tuple $\mathcal{U}$ is understood. Moreover, Z is called *non-trivial* if $Z \neq [\ell + r]$ (note that $[\ell + r]$ is trivially consistent which leads to the compatible vector $\mathbf{0}^{\ell+r}$). Now, we show that the above necessary conditions are indeed sufficient.

Lemma 4 (Linear Algebra Technical Lemma). *Let $\mathcal{U}$ be a triangular dependent tuple of $\ell + r$-to-t linear functions such that $tr + \ell + r < 2^n$. If $Z \subseteq [\ell + r]$ is a non-trivial $\mathcal{U}$-consistent index set, then there exists a non-zero $\mathcal{U}$-compatible vector $\overrightarrow{\alpha}$ with the zero-index set as Z.*

The proof of Lemma 4 relies primarily on the outcomes derived from Lemma 1. We have chosen to defer the detailed proof to the supplementary section. Now, given such $\mathcal{U}$, we describe an efficient algorithm (Algorithm 1) that yields a non-trivial $\mathcal{U}$-consistent index set Z.

```
Algorithm 1: Calculating an index set $Z$ from $\mathcal{U}$
    Input: The tuple of $\ell + r$-to-$t$ linear functions $\mathcal{U}$
    Output: The index set $Z$
Declare two empty index sets $Z$, and $N$;
Initialize $Z \leftarrow \{\ell + r\}, \quad N \leftarrow \{1, 2, \ldots, \ell + r - 1\}$;
Declare a set of vectors $\mathcal{S} \leftarrow \{\overrightarrow{u_j^{(r)}}|_Z \mid j \in [t]\}$;
while $\exists k \in N$ such that $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S})$ or $\overrightarrow{u_j^{(k-\ell)}}|_Z \in \mathsf{span}(\mathcal{S})$ for all $j \in [t]$, and
 $k \geq \ell + 1$ do
    Choose the largest such $k$;
    $Z \leftarrow Z \cup \{k\}$;
    $N \leftarrow N \setminus \{k\}$;
    if $k = i + \ell$ for some $i \in [r]$ then
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{\overrightarrow{u_j^{(i)}}|_Z \mid j \in [t]\}$;
return $Z$;
```

Algorithm 1. For a triangular dependent tuple of $\ell+r$-to-t linear functions $\mathcal{U}$, Algorithm 1 starts with two sets of indices Z and N such that Z contains only one index $(\ell + r)$, and N contains all the other indices in the set $[\ell + r]$. The set of vectors $\mathcal{S}$ contains the vectors $\overrightarrow{u_j^{(r)}}|_Z$, for all $j \in [t]$. Now, we continue to transfer the indices from N to Z if any index k in N satisfies any of the following two conditions:

$$\begin{cases} \textbf{Condition 1 : } \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}). \\ \textbf{Condition 2 : } \overrightarrow{u_j^{(k-\ell)}}|_Z \in \mathsf{span}(\mathcal{S}), \ \forall j \in [t], \ k \geq (\ell + 1). \end{cases}$$

We can define a index set $Z_\ell \subseteq Z$ as follows:

$$Z_\ell = \{i + \ell \in Z \ : \ i \in [r]\}.$$

Therefore at any instance of Algorithm 1, the set of vectors $\mathcal{S} = \{\overrightarrow{u_j^{(i)}}|_Z \ : \ i+\ell \in Z_\ell, \ j \in [t]\}$. Furthermore, whenever we find that any index $k \in N$ satisfies at least one of the conditions above, we transfer k from N to Z. So, we can partition the index set Z_ℓ into two subsets as follows:

$$Z_\ell = Z_\ell^1 \sqcup Z_\ell^2.$$

Here, the index set Z_ℓ^i consisting of those indices k that are transferred from N to Z_ℓ after satisfying **Condition *i***, for all $i \in [2]$. Using the index set Z_ℓ^1, we define $\mathcal{S}_1$ as follows:

$$\mathcal{S}_1 = \{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1, \ j \in [t]\}$$

Next, we examine key properties derived from Algorithm 1 to ultimately prove the Lemma 3. The properties are as follows:

Property 1. $\mathsf{rank}(\mathcal{S}_1) \leq t|Z_\ell^1|$.

This property directly follows from the fact that, $\mathcal{S}_1$ contains total $t|Z_\ell^1|$ vectors which implies $\mathsf{rank}(\mathcal{S}_1)$ can be at most $t|Z_\ell^1|$.

Property 2. For any index $k \in Z$, where $k \leq \ell$, we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Property 3. For any index $k \in Z \setminus \{\ell + r\}$, where $k \geq (\ell + 1)$, we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Property 4. Algorithm 1 always returns $\mathcal{U}$-consistent index set Z.

The proof of these aforementioned properties has been deferred to the (Subsubsection 8.3.1, Subsubsection 8.3.2, Subsubsection 8.3.3), respectively.

Finally, to establish the non-trivial $\mathcal{U}$-consistent nature of the index set Z constructed through Algorithm 1, we formulate and subsequently prove the following lemma:

Lemma 5. *Let $\mathcal{U}$ be a triangular dependent tuple of $\ell + r$-to-t linear function such that either (1) $r < \lceil (\ell - 1)/(t - 1) \rceil$, $t \geq 1$ or (2) $t = 1, \ell \geq 2$. If Z is an index set constructed using Algorithm 1, then Z is a non-trivial $\mathcal{U}$-consistent index set.*

Proof. Let $\mathcal{U}$ be a triangular dependent tuple of $\ell + r$-to-t linear function, and Z is the output of Algorithm 1. From Property 4, one can easily conclude that Z is $\mathcal{U}$-consistent index set. Now, we closely look into the Algorithm 1, Property 1, Property 2, and Property 3, and try to see some relations between the cardinalities of the index sets Z, and Z_ℓ.

We already know that the set of indices Z_ℓ^1 consisting of those indices k which are transferred from N to Z after satisfying **Condition 1**, and $\mathcal{S}_1$ is the set of vectors $\overrightarrow{u_j^{(i)}}$, for all $i+\ell \in Z_\ell^1$, and for all $j \in [t]$. Then, from Property 1, we have,

$$\mathsf{rank}(\mathcal{S}_1) \leq t|Z_\ell^1| \tag{3}$$

Moreover, from the properties of Algorithm 1, we can easily conclude that $\mathsf{span}\{\overrightarrow{e_i} \ : \ i \in Z \setminus \{\ell + r\}\}$ is a subspace of $\mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$. Therefore, taking the dimensions of both these vector spaces, we can easily conclude that

$$\begin{aligned}
|Z| - 1 &\leq \mathsf{rank}(\mathcal{S}_1) + |Z_\ell^2| \\
&\leq \mathsf{rank}(\mathcal{S}_1) + t|Z_\ell^2| \quad \text{[For } t \geq 1\text{]} \\
&\leq t|Z_\ell^1| + t|Z_\ell^2| \qquad \text{[Using } \textit{Equation } 3\text{]} \\
&\leq t|Z_\ell|
\end{aligned}$$

which leads us to the fact that $|Z| \leq t|Z_\ell| + 1$.
Now, we construct the proof using a contradiction-based method. Therefore, we assume that Z is a trivial $\mathcal{U}$-consistent index set i.e., $Z = [\ell + r]$. Now, the following two cases arise:

Case-A If $t = 1$, and $Z = [\ell + r]$, then we can easily say that $|Z| \leq |Z_\ell| + 1$, which implies

$$\ell + r \leq r + 1 \Rightarrow \ell \leq 1$$

However, it contradicts the given condition, as we know that if $t = 1$, then $\ell \geq 2$.

Case-B In this case, as $t \geq 2$, and $Z = \ell + r$ which directly implies that

$$|Z| \leq t|Z_\ell| + 1 \Rightarrow \ell + r \leq tr + 1 \Rightarrow r \geq (\ell - 1)/(t - 1)$$

It is given that when $t > 1$, it should be $r < \lceil (\ell - 1)/(t - 1) \rceil$,, and we arrive at a contradiction. Therefore, Z is non-trivial $\mathcal{U}$-consistent index set. □

Now, from Algorithm 1, and Lemma 5, we have that for a triangular dependent tuple $\mathcal{U}$ of $\ell+r$-to-t linear function, satisfying either (1) $r < \lceil(\ell-1)/(t-1)\rceil$, $t > 1$ or (2) $t = 1, \ell \geq 2$ with $(tr + \ell + r) < 2^n$, one can construct a non-trivial $\mathcal{U}$-consistent index set Z with at least one index $(\ell + r)$. Therefore, using Lemma 4 (*Linear Algebra Technical Lemma*), we can easily conclude that there exists a non-zero $\mathcal{U}$-compatible vector $\overrightarrow{\alpha}$ with the zero index set as Z.

Furthermore, as $\overrightarrow{\alpha} \neq \mathbf{0}^{\ell+r}$ is $\mathcal{U}$-compatible, then $\overrightarrow{\alpha}_{[\ell]} \neq \mathbf{0}^{\ell}$. Furthermore, as the index set Z contains $m = \ell + r$, and Z is the zero-index set of $\overrightarrow{\alpha}$, therefore, $\overrightarrow{\alpha}$ is non-zero collision $\mathcal{U}$-compatible vector. Hence, the result follows.

6 Collision-Resistance Preserving Security Bound for Linear Hash Mode having s $(s \geq 2)$ Output Blocks

In this section, we extend our lower bound result for hash functions with multiple hash outputs. The main idea of the lower bound is to convert a s-output hash function to a single block hash function by applying a suitable postprocessor. Then we can apply our previous result to obtain a lower bound on the compression function calls of s-output hash function for collision-resistance preserving property. Precisely, we state and prove the following theorem:

Theorem 3. *Let $H : \mathbb{F}^\ell \to \mathbb{F}^s$ be a linear hash mode which computes $H(M)$ for a given message $M = (m_1, m_2, \ldots, m_\ell) \in \mathbb{F}^\ell$ via access to r many t-to-1 compression function calls with $t \geq 2$. If*

$$r < \lceil(\ell - s)/(t-1)\rceil,$$

then this construction does not have white-box (and hence black-box) reduction.

Proof. We prove this result by contradiction. Let, $H : \mathbb{F}^\ell \to \mathbb{F}^s$ be a linear hash mode with $r < \lceil(\ell - s)/(t-1)\rceil$, t-to-1 compression function calls and H has white-box (black-box) reduction.

As we know that given any integers a and b, with $a > 0$, there exist unique integers p and q such that $b = pa + q$, $0 \leq q < a$. So, for the integers $(\ell-1), (s-1)$, and $(t-1)$, where $t \geq 2$, there exist unique integers p, p', q, and q' such that

$$\begin{cases} \ell - 1 = (t-1)p + q \\ s - 1 = (t-1)p' + q' \\ \ell - s = (t-1)(p - p') + (q - q') \end{cases} \tag{4}$$

where $0 \leq q, q' < (t-1)$. Therefore, from Equation 4, we can easily conclude the following:

$$\begin{cases} \lceil(\ell-s)/(t-1)\rceil = \lceil(\ell-1)/(t-1)\rceil - \lceil(s-1)/(t-1)\rceil & \textit{if either } q = q' \textit{ or } q' > q > 0 \\ & \textit{or } q > 0,\ q' = 0 \\ \lceil(\ell-s)/(t-1)\rceil > \lceil(\ell-1)/(t-1)\rceil - \lceil(s-1)/(t-1)\rceil & \textit{if either } q = 0,\ q' > 0 \\ & \textit{or } q > q' > 0 \end{cases} \tag{5}$$

We will now examine these two cases in detail. Suppose,

$$\lceil(\ell-s)/(t-1)\rceil = \lceil(\ell-1)/(t-1)\rceil - \lceil(s-1)/(t-1)\rceil. \tag{6}$$

Therefore from H, we can construct $H_1 : \mathbb{F}^\ell \to \mathbb{F}$ using $r_1 = r + \lceil(s-1)/(t-1)\rceil$ many t-to-1 compression function calls, same as in (Figure 7):

$$H_1(m_1, m_2, \ldots, m_\ell) = MD(H(m_1, m_2, \ldots, m_\ell)) \tag{7}$$

where MD is the Merkle-Damgård hash structure, which takes s block messages and outputs a message of length n using $\lceil (s-1)/(t-1) \rceil$ many t-to-1 compression function calls. As, we know that both H and the Merkle-Damgård hash functions have a black-box (hence white-box) reduction (as existence of black-box reduction implies the existence of white-box reduction), then we can easily prove that H_1 defined in Equation 7 has black-box (hence white-box) reduction using r_1 many t-to-1 compression function calls. As, $r < \lceil (\ell - s)/(t-1) \rceil$, therefore,

$$r_1 < \lceil (\ell - s)/(t-1) \rceil + \lceil (s-1)/(t-1) \rceil$$

which implies $r_1 < \lceil (\ell - 1)/(t-1) \rceil$ (using Equation 6).

If $\lceil (\ell - s)/(t-1) \rceil > \lceil (\ell - 1)/(t-1) \rceil - \lceil (s-1)/(t-1) \rceil$, in this case $q' > 0$ (Equation 5) which means $(s-1)$ is not multiple of $(t-1)$. For this case, we can have a integer k such that $q' + k = t - 1$. Therefore, we can write $s + k - 1 = (t-1)(p'+1)$. Now, all together we have

$$\begin{cases} s + k - 1 = (t-1)(p'+1) \\ \ell - s = (t-1)(p-p') + (q-q') \\ \ell + k - 1 = (t-1)(p+1) + (q-q') \end{cases} \tag{8}$$

Hence, we can easily conclude that,

$$\lceil (\ell - s)/(t-1) \rceil = \lceil (\ell + k - 1)/(t-1) \rceil - \lceil (s + k - 1)/(t-1) \rceil.$$

Then, from H, we will construct $H_2 : \mathbb{F}^{(\ell+k)} \to \mathbb{F}$ using r_2 many t-to-1 compression function calls in the following way:

$$H_2(m_1, m_2, \ldots, m_{\ell+k}) = MD(H(m_1, m_2, \ldots, m_\ell), m_{\ell+1}, \ldots, m_{\ell+k}) \tag{9}$$

where MD is the Merkle-Damgård hash function which takes here $s + k$ block messages and uses $\lceil (s + k - 1)/(t-1) \rceil$ compression function calls (Figure 6), i.e.,

$$r_2 = r + \lceil (s + k - 1)/(t-1) \rceil.$$

Therefore, as we know that both H and the Merkle-Damgård hash have black-box (hence white-box) reduction, we can easily prove that H_2 defined in Equation 9 has black-box (hence white-box) reduction using r_2 many t-to-1 compression function calls. As, $r < \lceil (\ell - s)/(t-1) \rceil$, therefore we can conclude that

$$r_2 < \lceil (\ell - s)/(t-1) \rceil + \lceil (s + k - 1)/(t-1) \rceil = \lceil (\ell + k - 1)/(t-1) \rceil$$

which implies that from H we can construct a linear hash mode $H_2 : \mathbb{F}^{(\ell+k)} \to \mathbb{F}$ which has black-box (hence white-box) reduction using r_2 many t-to-1 compression function call where $r_2 < \lceil (\ell + k - 1)/(t-1) \rceil$.

Thus, in both the scenarios, we have demonstrated that if there exists a linear hash mode, denoted as $H : \mathbb{F}^\ell \to \mathbb{F}^s$, with r t-to-1 compression function calls, where $r < \lceil (\ell - s)/(t-1) \rceil$, and H possesses black-box (hence white-box) reduction, then it is possible to construct another linear hash mode, denoted as $H' : \mathbb{F}^{\ell'} \to \mathbb{F}$, with black-box reduction (hence white-box reduction) using r' t-to-1 compression function calls, where $r' < \lceil (\ell' - 1)/(t-1) \rceil$, for some $\ell' > \ell$. However, this construction contradicts the result we proved earlier in Proposition 2 and Proposition 3. Hence, the result follows. □

6.1 Tightness of Lower Bound

To construct a linear hash mode $H : \mathbb{F}^\ell \to \mathbb{F}^s$, we design it in such a way that processing an ℓn-bit input necessitates $\lceil(\ell-s)/(t-1)\rceil$ calls to t-to-1 compression functions. Subsequently, we establish the proof that H possesses black-box reduction, where ℓ, s, and n are integers greater than or equal to 1. We construct $H : \mathbb{F}^\ell \to \mathbb{F}^s$ in the following way:

$$H(m_1, m_2, \ldots, m_\ell) = (MD(m_1, m_2, \ldots, m_{\ell-s+1}), m_{\ell-s+2}, \ldots, m_\ell) \tag{10}$$

where MD is the Merkle-Damgård hash function which takes here $\ell - s + 1$ block messages and uses $\lceil(\ell - s)/(t - 1)\rceil$ many t-to-1 compression function calls. So, we prove that H (defined in Equation 10) has black-box reduction as follows:

Lemma 6. *Given that the Merkle-Damgård hash function has a black-box reduction, H defined in Equation 10 also has a black-box reduction.*

Proof. Assuming H defined in Equation 10 does not have black-box collision reduction whenever $(M, M') \in \mathbb{F}^\ell \times \mathbb{F}^\ell$ is a collision pair of $H^{f_1,f_2,\ldots,f_r}$ for some collision-resistant t-to-1 compression functions $f_1, f_2, \ldots, f_r$ where $r = \lceil(\ell - s)/(t - 1)\rceil$, $(M_1, M_2) = (m_1, m_2, \ldots, m_{\ell-s+1}, m'_1, m'_2, \ldots, m'_{\ell-s+1}) \in \mathbb{F}^{(\ell-s+1)} \times \mathbb{F}^{(\ell-s+1)}$ is a collision pair of Merkle-Damgård hash mode processing $\ell - s + 1$ block messages using $r = \lceil(\ell - s)/(t - 1)\rceil$ calls to the underlying compression function. This leads us to the conclusion that the Merkle-Damgård hash mode does not have a black-box reduction, which is a contradiction. Hence the result follows. □

Therefore, using the result proved in Theorem 3, and from the construction defined in Equation 10, we have the result that for a linear hash mode $H : \mathbb{F}^\ell \to \mathbb{F}^s$ to have black-box reduction, it should call at least $\lceil(\ell - s)/(t - 1)\rceil$ many t-to-1 compression functions. Please note that, the existence of black-box reduction implies the existence of white-box reduction.

7 Conclusion and Future Work

In this paper, we have initially demonstrated the impossibility of proving collision security for the recently constructed hash modes ABR [ABR21] and T_5 [DKMN21] solely based on the collision security assumption of the underlying compression function. Despite T_5 being proven collision-resistant under the random oracle model, we have established a lower bound on the number of calls to the underlying compression function required to achieve collision-resistance preserving properties of a hash function when assuming the intermediate processing functions possess linear characteristics. Here, we encounter an engaging and challenging problem: Determining the lower bound on the compression function calls of a hash function when the intermediate processing functions within the hash structure are non-linear.

References

[ABR21] Elena Andreeva, Rishiraj Bhattacharyya, and Arnab Roy. Compactness of hashing modes and efficiency beyond merkle tree. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 92–123. Springer, 2021.

[Bar68] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. *IACR Cryptol. ePrint Arch.*, page 349, 2014.

[BCS09] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. *J. Cryptol.*, 22(3):311–329, 2009.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 781–796. USENIX Association, 2014.

[BDK$^+$07] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, ACNS '07, page 31–45, Berlin, Heidelberg, 2007. Springer-Verlag.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 200, page 21, 2007.

[Ben20] David Benjamin. Batch Signing for TLS. Internet-Draft draft-ietf-tls-batch-signing-00, Internet Engineering Task Force, January 2020. Work in Progress.

[BGD$^+$06] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. Cmss – an improved merkle signature scheme. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006*, pages 349–363, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BHK$^+$19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The sphincs$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2129–2146. ACM, 2019.

[BJKS93] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben J. M. Smeets. On families of hash functions via geometric codes and concatenation. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 1993.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International*

Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[Dam89] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[DDN22] Chandranan Dhar, Yevgeniy Dodis, and Mridul Nandi. Revisiting Collision and Local Opening Analysis of ABR Hash. In Dana Dachman-Soled, editor, *3rd Conference on Information-Theoretic Cryptography (ITC 2022)*, volume 230 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[DKMN21] Yevgeniy Dodis, Dmitry Khovratovich, Nicky Mouha, and Mridul Nandi. T5: hashing five inputs with three compression calls. *IACR Cryptol. ePrint Arch.*, page 373, 2021.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptol.*, 3(2):99–111, 1991.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 122–134. IEEE Computer Society, 1980.

[Mer89] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[M.O78] Rabin M.O. Digitalized signatures. *Foundations of Secure Computation*, pages 155–168, 1978.

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, USA, 1979.

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[RS07] Thomas Ristenpart and Thomas Shrimpton. How to build a hash function from any collision-resistant function. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory* *and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2007.

[RS08] Phillip Rogaway and John P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2008.

[SS08] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2008.

[SSY12] John P. Steinberger, Xiaoming Sun, and Zhe Yang. Stam's conjecture and threshold phenomena in collision resistance. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 384–405. Springer, 2012.

[Sta08] Martijn Stam. Beyond uniformity: Better security/efficiency tradeoffs for compression functions. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

[Ste10] John P. Steinberger. Stam's collision resistance conjecture. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 597–615. Springer, 2010.

8 Appendix

8.1 Collision Attack on General ABR Hash

ABR [ABR21] mode of height ℓ with 2^ℓ leaf message inputs, with $r = 2^\ell - 1$ compression function calls and a total of $(2^\ell + 2^{\ell-1} - 1)$ input blocks. Therefore, we represent the input message blocks as $m_1, m_2, \ldots, m_{2^\ell+2^{\ell-1}-1}$, while the compression functions are denoted as $f_{i,j}$. Here, i ranges from 1 to ℓ, and for a given value of i, j takes on values from 1 to $2^{\ell-i}$ ($\ell = 3$ case is illustrated in Figure 3). Here, we would like to prove that the general ABR hash does not have black-box reduction.

In order to do this, first, we prove that ABR_2 does not have black-box reduction. Similarly, in ABR_2 hash mode, we notice that some non-zero message difference $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ leads us to zero hash difference, provided the underlying compression function f satisfy certain non-zero differential patterns ($f(\mathbf{0}, \Delta) = \Delta$, $f(\mathbf{0}, \mathbf{0}) = \mathbf{0}$) for some $\Delta \neq \mathbf{0}$. Therefore, if we process $M = (\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$, and $M' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ in ABR_2^f where 2-to-1 collision-resistant compression function f is defined as in Figure 4.1), then $\mathrm{ABR}_2^f(M) = \mathrm{ABR}_2^f(M') = \Delta$. Hence, we conclude that ABR_2 does not have black-box reduction.

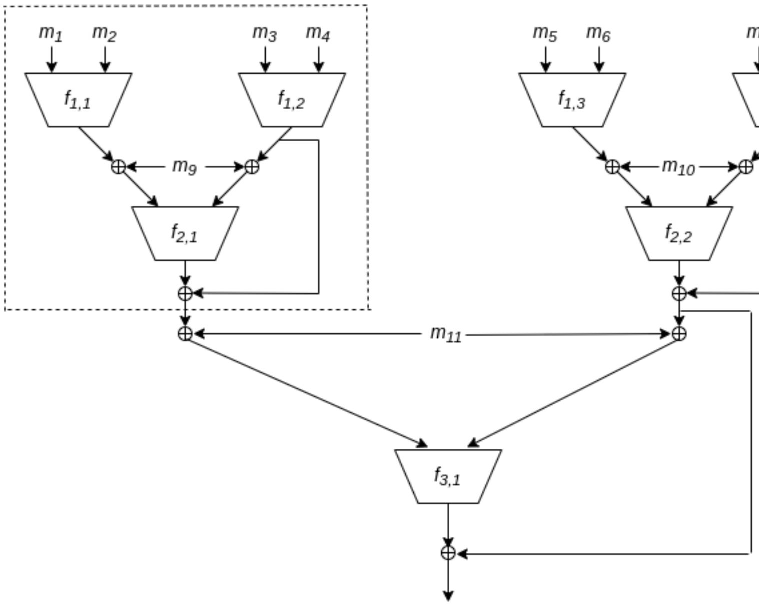


Figure 3: ABR mode of height 3 with 11 message blocks based on 7 calls to underlying 2-to-1 compression functions.

Therefore, we can extend this attack to ABR hash mode of height ℓ for any integer $\ell \geq 3$. Now, the general ABR hash mode of height ℓ using f as its compression function calls where 2-to-1 collision-resistant compression function f is defined in Eqn. (4.1), can be written as

$$ABR_\ell^f(m_1, m_2, \ldots, m_{2^\ell+2^{\ell-1}-1}) = H(ABR_2^f(m_1, m_2, m_3, m_4, m_{2^\ell+1}), m_5, \ldots, m_{2^\ell+2^{\ell-1}-1}) \tag{11}$$

where H is a hash function that takes an input of $(2^\ell + 2^{\ell-1} - 5)$ blocks and outputs one block, and it uses f as its compression function calls ($\ell = 3$ case is illustrated in Figure Figure 3, where the dotted line implies the function ABR_2). Specifically, H contains all the other functions of ABR_ℓ^f except the leftmost ABR_2^f. If we consider the case of $\ell = 3$, H is a hash function which includes all the functions of ABR_3^f except the dotted part in Figure Figure 3. This implies H takes input as $ABR_2^f(m_1, m_2, m_3, m_4, m_9), m_5, m_6, m_7, m_8, m_{10}, m_{11}$, and outputs $h = ABR_3^f(m_1, \ldots, m_{11})$.

Now, as we know, (M, M') is a collision pair of ABR_2^f as defined above, then we can construct a pair $(\bar{M}, \bar{M}')$ defined as follows:

$$\begin{cases} \bar{m}_j = m_j,\ \bar{m}'_j = m'_j & for\ j = 1,\ 2,\ 3,\ 4,\ 2^\ell + 1 \\ \bar{m}_j = \bar{m}'_j = \mathbf{0} & otherwise \end{cases} \tag{12}$$

Therefore, from Eqn. (11), using Eqn. (12) we can conclude that $(\bar{M}, \bar{M}')$ is a collision pair of ABR_ℓ^f which implies the general ABR hash mode of height ℓ does not have black-box reduction.

Remark 4. As elaborated in Example 3, the extended version of the Shrimption-Stam construction involves the processing of five message blocks through three calls to $2n$-to-n

bit compression function. It is crucial to recognize that demonstrating collision resistance for this construction based solely on the collision-resistant property of the underlying compression function is unattainable. Similar reasoning can be applied as we do when establishing the impossibility of collision reduction for T_5 and ABR constructions.

8.2 Proof of Lemma 3 (Linear Algebra Technical Lemma)

We set $Z^c = [\ell + r] \setminus Z$. We prove this result using Lemma 1. To do so, we define:

$$
\begin{aligned}
I &= \{(i+\ell, j) \in Z^{\mathsf{c}} \times [t] \;:\; \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S})\} \\
\mathcal{S} &= \{\overrightarrow{u_j^{(i)}}|_Z : i+\ell \in Z, j \in [t]\} \\
\mathcal{S}' &= \{\overrightarrow{u_j^{(i)}}|_Z : (i+\ell, j) \in I\}
\end{aligned}
$$

We moreover define $\mathcal{S}_1 = \mathcal{S} \cup \{\overrightarrow{e_k} \;:\; k \in Z\}$ and $\mathcal{N} = \mathcal{S}' \cup \{\overrightarrow{e_k} \;:\; k \notin Z\}$.

Claim. $\mathsf{span}(\mathcal{S}_1)$, and $\mathcal{N}$ are disjoint.

Proof of Claim. We prove this by contradiction. Let there exist some $\overrightarrow{v} \in \mathcal{N}$ such that $\overrightarrow{v} \in \mathsf{span}(\mathcal{S}_1)$. This implies *either* $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z$, for some $(i+\ell, j) \in I$ *or* $\overrightarrow{v} = \overrightarrow{e_k}$, for some $k \notin Z$. Now, if $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z$, for some $(i+\ell, j) \in I$, then $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z \in \mathsf{span}(\mathcal{S}_1)$. Therefore, we can write

$$
\overrightarrow{u_j^{(i)}}|_Z = \sum_{(p+\ell,q) \in Z \times [t]} c_{p,q} \overrightarrow{u_q^{(p)}}|_Z + \sum_{s \in Z} c_s \overrightarrow{e_s}
$$

where $c_{p,q}, c_s \in \mathbb{F}$. Now, if $c_s = \mathbf{0}$, $\forall s \in Z$, then $\overrightarrow{u_j^{(i)}} \in \mathsf{span}(\mathcal{S})$ which cannot be true as $(i+\ell, j) \in I$. Furthermore, if there exists some $s \in Z$ s.t. $c_s \neq \mathbf{0}$, then $\overrightarrow{v}$ contains some non-zero entry in the s-th index which cannot be possible. So,

$$
\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S}_1), \text{ for all } (i+\ell, j) \in I.
$$

On the other hand, if $\overrightarrow{v} = \overrightarrow{e_k}$, for some $k \notin Z$, then $\overrightarrow{v} = \overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$ using the condition N1. But according to our assumption $\overrightarrow{v} = \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1)$, where $k \notin Z$. Therefore, we write,

$$
\overrightarrow{e_k} = \sum_{(i+\ell,j) \in Z \times [t]} d_{i,j} \overrightarrow{u_j^{(i)}}|_Z + \sum_{q \in Z} d_q \overrightarrow{e_q}
$$

where $d_{i,j}, d_q \in \mathbb{F}$. As, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$, then there exists some $q \in Z$ s.t $d_q \neq \mathbf{0}$. However, this cannot be true, as the vector $\overrightarrow{e_k}$ does not have any non-zero entry at the q-th index where $q \in Z$. So,

$$
\overrightarrow{v} = \overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S}_1), \text{ for all } k \notin Z.
$$

This completes the proof of the claim.

Note, $\mathcal{N}$ can have at most $(tr + \ell + r) < 2^n$ vectors. Hence, using Lemma 1, there exists $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$ such that $\forall \overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$. Now, as $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$, then we have, for all $k \in Z$,

$$
\overrightarrow{e_k} \cdot \overrightarrow{\alpha} = \mathbf{0} \Rightarrow \alpha_k = \mathbf{0}.
$$

Furthermore, as for all $\overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$, we have, for all $k \notin Z$,

$$
\overrightarrow{e_k} \cdot \alpha \neq \mathbf{0} \Rightarrow \alpha_k \neq \mathbf{0}.
$$

Hence, Z is zero-index set of $\overrightarrow{\alpha}$. As, $Z \neq [\ell + r]$, then $\overrightarrow{\alpha} \neq \mathbf{0}^m$. Now, the only remaining task is to prove that $\overrightarrow{\alpha}$ is $\mathcal{U}$-compatible. As, $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$, then we have, for all $(i+\ell) \in Z$ with $i \in [r]$,

$$\begin{aligned} & \overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{\alpha} = \mathbf{0}, \ \forall j \in [t] \\ \Rightarrow\ & \mathbf{u}^{(i)}(\overrightarrow{\alpha}) = \mathbf{0}^t \\ \Rightarrow\ & \overrightarrow{\beta^{(i)}} = \mathbf{0}^t \ \ \text{[Here we denote, } \mathcal{U}(\overrightarrow{\alpha}) = (\overrightarrow{\beta^{(1)}}, \ldots, \overrightarrow{\beta^{(r)}})]. \end{aligned}$$

Therefore, we have, for all $i \in [r]$, if $\alpha_{i+\ell} = \mathbf{0}$, then $\overrightarrow{\beta^{(i)}} = \mathbf{0}^t$. Finally, as we have for all $\overrightarrow{v} \in \mathcal{S}_2$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$, then for all $(i + \ell) \notin Z$, there exists some $j \in [t]$, such that

$$\overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{\alpha} \neq \mathbf{0} \Rightarrow \mathbf{u}^{(i)}(\overrightarrow{\alpha}) \neq \mathbf{0}^t \Rightarrow \overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t.$$

Thus, we can easily conclude that for all $i \in [r]$ with $(i + \ell) \notin Z$ i.e., $\alpha_{i+\ell} \neq \mathbf{0}$ implies $\overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t$. This immediately implies that $\overrightarrow{\alpha}$ is $\mathcal{U}$-compatible. Hence, the result follows.

8.3 Proof of Properties

In this section, we prove the properties stated above which have been used to prove our Proposition.

8.3.1 Proof of Property 2

Consider that instance of Algorithm 1, when the index set Z contains $k \geq \ell + 1$ which implies that $Z = Z_\ell$. At that instance, suppose for the first time one find out some index $k \leq \ell$ such that,

$$\begin{aligned} & \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}}|_Z \ : \ i \in Z_\ell, \ j \in [t]\}) \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell, \ j \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell\}) \end{aligned}$$

Now, if $Z = Z_\ell = \mathcal{Z}_\ell^1$, then we know that for all $i \in Z_\ell^1$, $\overrightarrow{e_i} \in \mathsf{span}(\mathcal{S}_1)$. Therefore, we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1, \ j \in [t]\}) = \mathsf{span}(\mathcal{S}_1).$$

Next, if $Z = Z_\ell = Z_\ell^1 \cup Z_\ell^2$, then we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$$

as we know if $Z = Z_\ell$, then for all $i+\ell \in Z_\ell^2$, and $j \in [t]$, we have $\overrightarrow{u_j^{(i)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$ (Lemma 7 in supplementary material). Therefore, according to Algorithm 1, we transfer the index k from N to Z, and update $\mathcal{S}$ accordingly.

From here, we can easily conclude that, for any index $k \in Z$, where $k < \ell + 1$, then,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Hence, the result follows.

8.3.2 Proof of Property 3

At any instance of Algorithm 1, the index set Z can have indices $i \geq (\ell+1)$ which are transferred from N to Z after satisfying both the conditions, and some indices k satisfying $k \leq \ell$ as well.[6] Therefore, for any index $k \in Z_\ell^1 \setminus \{(\ell+r)\}$, we have,

$$\begin{aligned}
&\overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell\} \cup \{\overrightarrow{e_i} \ : \ i \in Z \setminus \{k\}\}) \\
\Rightarrow \ &\overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}) \text{ [Using } \textit{Property } 2\text{]} \\
\Rightarrow \ &\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})
\end{aligned}$$

Hence, the result follows.

8.3.3 Proof of Property 4

Let Algorithm 1 return an index set say Z. Therefore, from Algorithm 1, we have, for all $k \notin Z$, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$, and for all $k \notin Z$ where $k \geq (\ell+1)$, there exists $j \in [t]$ such that $\overrightarrow{u_j^{(k-\ell)}}|_Z \notin \mathsf{span}(\mathcal{S}$. Now, for all $k \notin Z$, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$ implies that the index set satisfies the first necessary condition N1:

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \Rightarrow k \in Z.$$

Furthermore, for all $i+\ell \notin Z$, there exists $j \in [t]$ such that $\overrightarrow{u_j^{(k-\ell)}}|_Z \notin \mathsf{span}(\mathcal{S})$ immediately implies that the index set satisfies the second necessary condition N2. Hence, Z is $\mathcal{U}$-consistent index set.

8.3.4 Proof of Lemma 7

Lemma 7. *If the index set $Z = Z_\ell$, then for all $k+\ell \in Z_\ell^2$, and $j \in [t]$, we have,*

$$\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Proof. Consider that instance of Algorithm 1, when the index set $Z = Z_\ell^1$ which means at that instance Z_ℓ^2 is empty. Now, at that instance, suppose for the first time we find out some index $k_1 = k+\ell \geq \ell+1$ such that,

$$\begin{aligned}
&\overrightarrow{u_j^{(k)}}|_Z \in \mathsf{span}(\mathcal{S}), \ \forall j \in [t] \\
\Rightarrow \ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^1\}), \ \forall j \in [t] \\
\Rightarrow \ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\}), \ \forall j \in [t] \\
\Rightarrow \ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}^1), \ \forall j \in [t]
\end{aligned}$$

Therefore, according to Algorithm 1 we transfer the index k_1 from N to Z (precisely the index k_1 is added to Z_ℓ^2), and update $\mathcal{S}$ accordingly.

Next, if we consider any instance of Algorithm 1, where the index set $Z = Z_\ell$, then for all $k_1 = k+\ell \in Z_\ell^2$,

$$\begin{aligned}
\Rightarrow \ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in \mathcal{Z}_\ell\}), \ \forall j \in [t] \\
\Rightarrow \ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}), \ \forall j \in [t]
\end{aligned}$$

Hence, the result follows. □

[6] At any instance of Algorithm 1, any index $k \leq \ell$ can be transferred from N to Z after satisfying **Condition 1** only.

8.4 Figures

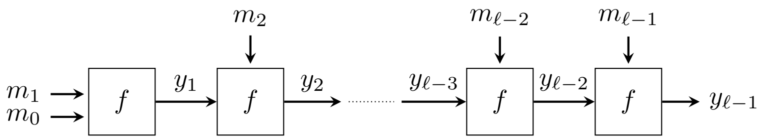


Figure 4: Merkle-Damgård hash mode processing ℓ block messages with $(\ell - 1)$ 2-to-1 compression function calls

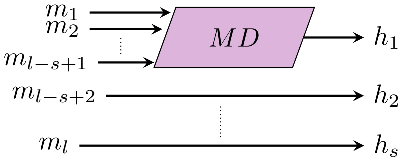


Figure 5: Linear Hash Mode H using $\lceil (l - s)/(t - 1) \rceil$ many t-to-1 Compression Function Calls.

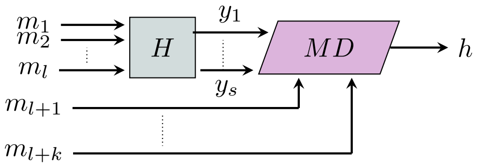


Figure 6: Linear Hash Mode H_2 using $r + \lceil (s + k - 1)/(t - 1) \rceil$ many t-to-1 Compression Function Calls where MD is Merkle-Damgård hash function.

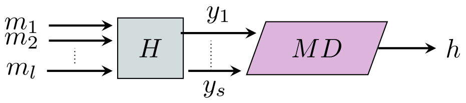


Figure 7: Linear Hash Mode H_1 using H.