

Linicrypt in the Ideal Cipher Model ^{*}

Zahra Javar  and Bruce M. Kapron 

University of Victoria
Victoria, BC, CANADA
zahrajavar@uvic.ca bmkapron@uvic.ca

Abstract. We extend the Linicrypt framework for characterizing hash function security as proposed by McQuoid, Swope, and Rosulek (TCC 2018) to support constructions in the ideal cipher model. In this setting, we give a characterization of collision- and second-preimage-resistance in terms of a linear-algebraic condition on Linicrypt programs, and present an efficient algorithm for determining whether a program satisfies the condition. As an application, we consider the case of the block cipher-based hash functions proposed by Preneel, Govaerts, and Vandewalle (Crypto 1993), and show that the semantic analysis of PGV given by Black et. al. (J. Crypto. 2010) can be captured as a special case of our characterization. In addition, We model hash functions constructed through the Merkle-Damgård transformation within the Linicrypt framework. Finally, we apply this model to an analysis of how various attacks on the underlying compression functions can compromise the collision resistance of the resulting hash function.

Keywords: Collision-resistant hash function, Compression function, Ideal cipher model, Linicrypt

1 Introduction

Two fundamental properties of cryptographic hash functions which are the basis for their cryptographic application are *collision resistance* and *2nd-preimage resistance*. Applications of cryptographic hash functions include message authentication code [2] and hash-based signatures [6,12,14,16]. One basic approach to build hash functions is by the iteration of a fixed-length *compression function*.

In this work, we extend the approach of [15] to characterize collision-resistance properties of compression functions constructed in the ideal cipher model and demonstrate that such an approach is amenable to automated validation and generation.

In [15] the *Linicrypt* formalism [8] is applied to give a characterization of collision- and 2nd-preimage resistance of hash functions constructed via straight-line algebraic programs with access to a random oracle. In this paper, as suggested by [15], we extend the characterization given in that paper to the ideal cipher model. As the ideal cipher model is a standard setting for the construction of cryptographic hash functions, in particular modeling block-cipher-based construction of compression functions, this is a natural and relevant extension of the previous work.

A well-studied group of block-cipher-based compression functions was proposed by Preneel, Govaerts, and Vandewalle (PGV) [17]. They proposed a systematic way to construct *rate-1* block-cipher-based compression functions which make a single call to the ideal cipher, using only simple algebraic operations.

In a generalization of [17], Black, Rogaway, and Shrimpton [4] introduced 64 rate-1 compression functions $h : \{0, 1\}^l \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ utilizing a single call to a block-cipher $E : \{0, 1\}^l \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ of the form $h^E(h, m) = E_a(b) \oplus c$ where $a, b, c \in \{h, m, h \oplus m, v\}$, and v is a fixed constant, which they term *PGV compression functions*. They then prove that of the 64 PGV compression functions, 12 of them, referred to as *group-1*, are collision-resistant and preimage resistant up to the birthday bound, and 8 of them, which are called *group-2* are only collision resistant after some iteration. The proofs in [4] are given on a per-function basis, with canonical examples and an indication of how these could be generalized to any function in the

^{*} Work supported in part by NSERC Discovery Grant RGPIN-2021-02481. This is an expanded and emended version of [13]

corresponding group. In subsequent work, [5] characterized group-1 and group-2 PGV compression functions via a more general approach which considers fundamental combinatorial properties of the definitions, based on *pre-* and *post-processing functions*. A related work by Stam [19] presents these properties in an algebraic setting, partly anticipating the approach presented in the current paper. In section 4, we model the PGV compression functions in Linicrypt and show the properties proposed in [5] may in fact be obtained as a special case of our general characterization.

Contributions Our main contributions are summarized as follows:

1. A formulation in the ideal cipher model of the notion of *collision structure*, introduced in [15] for the random oracle model (Definition 4).
2. A characterization showing a Linicrypt program is collision-resistant (and 2nd-preimage resistant) if and only if it does not have a collision structure (Theorem 1).
3. An efficient algorithm for finding collision structures (Algorithm 1).
4. In the rate-1 setting, giving an alternate proof the collision-resistance of *group-1* PGV compression functions as originally established in [4] using our characterization (Theorem 2.)
5. We generalize the definition of group 2 PGV compression functions to encompass any Linicrypt program that, while not inherently collision-resistant, yields a collision-resistant hash function. Furthermore, we formalize the structure of iterated hash functions in Linicrypt, providing a comprehensive definition, and investigate potential attacks on group 3 PGV compression functions.

While our approach largely follows that of [15], the extension to the ideal cipher model is non-trivial, and the application to the PGV functions presents an interesting case where the Linicrypt approach sheds new light on existing approaches to hash function security.

Relation to previous work This paper is an expansion of [13]. Item (5) above is a new contribution. Also, the Linicrypt model for ideal ciphers presented in this paper corrects the model of [13] by including nonces to ensure independence in multiple calls to the ideal cipher. This only impacts the results of [13] involving programs that make more than one call. In particular, the analysis of PGV compression functions in that paper remains unchanged. We thank Frederik Semmel for bringing this oversight to our attention.

2 Preliminaries

In our presentation, we largely follow the approach of [15]. Programs are defined over a finite field \mathbb{F} , elements of which are denoted by lower-case non-bold letters¹, vectors over \mathbb{F} by lowercase bold letters and matrices over \mathbb{F} by uppercase bold letters. We write $\mathbf{a} \cdot \mathbf{b}$ or \mathbf{ab} for the inner product and $\mathbf{M} \times \mathbf{a}$ or $\mathbf{M}\mathbf{a}$ for the matrix-vector product. Note that we will sometimes think of matrices as a column vector of row vectors (so we may write $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_k)^\top$.)

2.1 Ideal Cipher Model

It is often challenging to design cryptographic primitives which provably provide a needed security property, even in the presence of computational assumptions. One approach to deal with this problem is to assume the existence of an ideal primitive, such as an ideal block cipher or a random oracle which may be used to prove the security of new primitives. The new primitive is then implemented using a real-world instantiation of the ideal primitive. While this approach is not sound in general — there are primitives that can be proven secure when using a truly random oracle but not when using a non-ideal hash function [7] — it is widely used in practice and is considered to provide some formal assurance of security.

The idea of modeling a block cipher as a random permutation appears as early as the work of Shannon [18]. In the *ideal cipher model*, the adversary has access to oracles E and E^{-1} , where E is a random block cipher $E : \{0, 1\}^k \times \{0, 1\}^n$ and E^{-1} is its inverse. Thus each key $k \in \{0, 1\}^n$ determines a uniformly selected

¹ We usually use Roman letters, but may also use Greek or script letters depending on the setting.

permutation $E_k = E(k, \cdot)$ on $\{0, 1\}^n$, and the adversary is given oracles for E and E^{-1} . The latter, on input (k, y) , returns the x such that $E_k(x) = y$. As is standard in this setting (see, e.g., [3],) programs used to construct hash functions are given access to E .

2.2 LiniCrypt

The LiniCrypt framework [8] was introduced by Carmer and Rosulek to formally model cryptographic algorithms with access to a random oracle, and using linear operations. In that work, they give an algebraic condition to efficiently decide if two LiniCrypt programs induce computationally indistinguishable distributions. As mentioned above, in [15] the framework is applied to characterize collision-resistance properties of hash functions.

A LiniCrypt program is a straight-line program over a fixed vector (v_1, \dots, v_m) of *program variables*, where the first k are designated as *inputs*. A program is a sequence of lines specifying assignments to (non-input) program variables where the right-hand side of each assignment is either²

1. A call to the random oracle on a previously assigned variable or input
2. A \mathbb{F} -linear combination of previously assigned variables and inputs

This defines a function $\mathcal{P}^H : \mathbb{F}^k \rightarrow \mathbb{F}^r$ for some k, r which on input vector $\mathbf{x} = (x_1, \dots, x_k)$ returns output vector $\boldsymbol{\ell} = (l_1, \dots, l_r)$. The field is parameterized by a *security parameter* λ . In particular $\mathbb{F} = \mathbb{F}_{p^\lambda}$ for some prime p . In this work, we assume a *uniform* model in which there is a single program \mathcal{P} (with constants from \mathbb{F}_p .) As discussed in [8], it is also possible to consider families of programs depending on λ .

As described in [8,15], LiniCrypt programs can be given a purely *algebraic* representation. We will give a slightly modified (but equivalent) version of this as presented in [15]. We first note that in the straight-line program representation, if there are no assignments of random field elements to variables, then all assignments of the second form may be eliminated, via successive in-lining. In this case, a program \mathcal{P} over a field \mathbb{F} is given by a set of *base variables* $\mathbf{v}_{base} = (v_1, \dots, v_{k+n})^\top$, where $v_i \in \mathbb{F}$ and the first k variables are \mathcal{P} 's input and the last n variables correspond to the results of oracle queries. This means that, algebraically, for each program variable v_i we can write $v_i = \mathbf{e}_i \mathbf{v}_{base}$ where \mathbf{e}_i denotes the i th canonical basis vector over \mathbb{F}^{k+n} . Similarly, the output may be given as a vector of \mathbb{F} -linear combinations of program base variables, and this may be specified by the *output matrix* $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_r)^\top$, where $\mathbf{m}_i \in \mathbb{F}^{k+n}$ and $\mathbf{M} \mathbf{v}_{base} = (l_1, \dots, l_r)$ is the vector of program outputs. Finally, each oracle call $v_i = H(t, v_{i_1}, \dots, v_{i_m})$ where t is a *nonce* and $v_i \in \mathbb{F}$, is represented by an *oracle constraint* $c = (t, \mathbf{Q}, \mathbf{a})$, indicating that when H is called on input $(t, \mathbf{Q} \times \mathbf{v}_{base}) = (t, v_{i_1}, \dots, v_{i_m})$, the returned value is $\mathbf{a} \cdot \mathbf{v}_{base} = v_i$.

This representation allows us to abstract away from the straight-line syntax, and reason about programs in a purely algebraic fashion. In particular, as noted in [8,15], if we let \mathcal{C} denote the set of oracle constraints, then the behavior of a program \mathcal{P} is completely specified by its *algebraic representation* $(\mathbf{M}, \mathcal{C})$. In particular, the characterization of collision-resistance properties is completely determined by algebraic properties of $(\mathbf{M}, \mathcal{C})$.

The following is an example of a two-input LiniCrypt program with random oracle H

$$\begin{aligned} & \underline{\mathcal{P}^H(v_1, v_2)} : \\ & \quad v_3 := H(t_1, v_1) \\ & \quad v_4 := H(t_2, v_2) \\ & \quad v_5 := v_3 + v_4 \\ & \quad \text{return } v_5 \end{aligned}$$

In the algebraic presentation, the program is specified by:

$$\mathbf{v}_{base} = (v_1, v_2, v_3, v_4)^\top$$

² The LiniCrypt model, introduced in [8], also allows the assignment of a random field element to a variable. Here, as in [15], we consider only deterministic programs.

$$\mathbf{M} = [0 \ 0 \ 1 \ 1]$$

$$\mathcal{C} = \langle (t_1, \mathbf{q}_1, \mathbf{a}_1), (t_2, \mathbf{q}_2, \mathbf{a}_2) \rangle$$

where

$$\mathbf{q}_1 = (1, 0, 0, 0) \quad \mathbf{q}_2 = (0, 1, 0, 0)$$

$$\mathbf{a}_1 = (0, 0, 1, 0) \quad \mathbf{a}_2 = (0, 0, 0, 1)$$

In moving to programs in the ideal cipher model, for assignments of the first type, we now have calls to an ideal cipher $E(t, \cdot, \cdot)$ on a pair of values which are either program inputs or previously assigned variables. These inputs to E correspond to the key and input of an encryption query. Thus, supposing \mathcal{P} has n oracle calls, for $i \in [n]$, each call is of the form $\mathbf{a}_i \cdot \mathbf{v}_{base} = E(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}_{base}, \mathbf{q}_{X_i} \cdot \mathbf{v}_{base})$ and can be represented by an *oracle constraint* $c = (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ where $\mathbf{q}_K, \mathbf{q}_X, \mathbf{a} \in \mathbb{F}^{k+n}$ and t is a nonce. To simplify the presentation we define $K_i := \mathbf{q}_{K_i} \cdot \mathbf{v}_{base}$, $X_i := \mathbf{q}_{X_i} \cdot \mathbf{v}_{base}$ and $Y_i := \mathbf{a}_i \cdot \mathbf{v}_{base}$. Letting \mathcal{C} denote the set of oracle constraints and \mathbf{M} the output matrix, we again have an algebraic representation $(\mathbf{M}, \mathcal{C})$. The following is a simple example of such a program:

$$\begin{aligned} & \underline{\mathcal{P}^E(v_1, v_2)} : \\ & \quad v_4 := E(t, v_1, v_2) \\ & \quad \text{return } v_4 + v_2 \end{aligned}$$

$$\mathbf{M} = (0, 1, 0, 1), \quad \mathcal{C} = (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}), \quad \mathbf{v}_{base} = (v_1, v_2, v_4)^\top$$

$$\mathbf{q}_K = (1, 0, 0, 0) \quad \mathbf{q}_X = (0, 1, 0, 0) \quad \mathbf{a} = (0, 0, 1, 0)$$

Further examples are given in the Appendices. We also refer the reader to [8,15] for a more detailed introduction to Linicrypt.

In this scenario, the adversary exchanges the values of v_1 and v_2 to find a second preimage. Nevertheless, if the two ideal cipher calls were independent, the adversary would not be able to carry out this attack.

Nonces While the use of nonces is unusual for the ideal cipher, it does not change the model in a significant way. In particular, it is already the case that for a given key k , $E(k, \cdot)$ is an independently chosen random permutation. Thus introducing nonces can be viewed as extending the key space to pairs (t, k) . In practice this will incur an extra cost for extending the key space.

Constant Values In practice, the definition of a hash function may depend on the use of a constant value from the underlying field or domain, typically referred to as an *initialization vector (IV)*. Such definitions involve affine expressions and so, strictly speaking, are beyond the model provided by Linicrypt. One approach to deal with this problem is to utilize some underlying algebraic property of operations involving constant values. This is the approach taken in the algebraic analysis of [19], where it is noted that translation by a constant preserves bijectivity. We take a more general approach, treating constants *parametrically*. Namely, a constant c used in a program \mathcal{P} is treated as an additional input and also as an output of the program, making it a fixed parameter. In particular, \mathcal{P} has base variables v_1, \dots, v_{k+n} and inputs v_1, \dots, v_k the modified program with a constant c has base variables v_1, \dots, v_{k+n+1} where $v_{k+1} := c$ and \mathbf{M} and \mathcal{C} are updated appropriately. Finally, the single row \mathbf{e}_{k+1} is appended to \mathbf{M} (indicating that c is an output.) By following this convention, we can analyze the security properties of hash functions defined by Linicrypt programs using constants without making any modifications to our definitions and proofs. Moreover, any property which does not depend on a particular property (e.g., the bit-level representation) of a constant value used in a program will be preserved by this convention. While this does not capture implementation-level details, it provides a level of analysis consistent with works such as [5].

Security Definitions We assume the number of oracle queries the adversary makes to E is q_E and to E^{-1} is q_D .

Definition 1 ([15] **Definition 2**). Program \mathcal{P} is (q, ϵ) -collision resistant if any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$(\mathbf{x}, \mathbf{x}') \leftarrow \mathcal{A}^{E, E^{-1}}(\lambda); \text{ return } (\mathbf{x} \neq \mathbf{x}') \text{ and } \mathcal{P}^E(\mathbf{x}) = \mathcal{P}^E(\mathbf{x}') \quad (1)$$

Definition 2 ([15] **Definition 3**). Program \mathcal{P} is (q, ϵ) -2nd-preimage resistant if any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$\mathbf{x} \leftarrow \mathbb{F}^k; \mathbf{x}' \leftarrow \mathcal{A}^{E, E^{-1}}(\mathbf{x}, \lambda); \text{ return } (\mathbf{x} \neq \mathbf{x}') \text{ and } \mathcal{P}^E(\mathbf{x}) = \mathcal{P}^E(\mathbf{x}') \quad (2)$$

3 Characterizing Collision Resistance

The LiniCrypt framework described above gives a purely algebraic presentation of LiniCrypt programs. Our goal now is to provide an algebraic condition that characterizes collision resistance and 2nd-preimage resistance for LiniCrypt programs in the ideal cipher model (Definition 4). Before giving this definition we note that some programs fail trivially to be collision-resistant because two different inputs produce exactly the same queries to the oracle. This situation is formalized as follows:

Definition 3. Program $\mathcal{P} = (M, \mathcal{C})$ is degenerate if

$$\begin{aligned} \text{span}(\{\mathbf{e}_1, \dots, \mathbf{e}_{k+n}\}) \not\subseteq & \text{span}(\{\mathbf{q}_K \mid (\mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \cup \{\mathbf{q}_X \mid (\mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \\ & \cup \{\mathbf{a} \mid (\mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \cup \text{rows}(M)) \end{aligned}$$

Lemma 1. If \mathcal{P} is degenerate then 2nd-preimages can be found with probability 1.

Proof. Assume the adversary \mathcal{A} is given a preimage \mathbf{x} , and it determines the base vector \mathbf{v}_{base} in the execution

of $\mathcal{P}(\mathbf{x})$. We define the matrix $\mathbf{P} = \begin{bmatrix} \mathbf{Q}_K \\ \mathbf{Q}_X \\ \mathbf{A} \\ \mathbf{M} \end{bmatrix}$ where $\mathbf{Q}_K, \mathbf{Q}_X, \mathbf{A}$ are matrices whose rows correspond to the

components of the elements of \mathcal{C} (ordered arbitrarily.) If the adversary can determine a 2nd-preimage $\mathbf{x}' \neq \mathbf{x}$ where $\mathbf{P}\mathbf{v}_{base} = \mathbf{P}\mathbf{v}'_{base}$ where \mathbf{v}'_{base} is the base vector in the calculation of $\mathcal{P}(\mathbf{x}')$ then $\mathcal{P}(\mathbf{x}) = \mathcal{P}(\mathbf{x}')$ and \mathcal{A} wins. Since program \mathcal{P} is degenerate, the rows of \mathbf{P} cannot span all $k + n$ basis vectors which means $\text{rank}(\mathbf{P}) < k + n$, and thus, for some $\mathbf{v} \neq \mathbf{0}$, $\mathbf{P}\mathbf{v} = \mathbf{0}$. The adversary can solve for this \mathbf{v} and set $\mathbf{v}'_{base} = \mathbf{v}_{base} + \mathbf{v}$. Then $\mathbf{P}(\mathbf{v}'_{base} - \mathbf{v}_{base}) = \mathbf{0}$, so $\mathbf{v}'_{base} \neq \mathbf{v}_{base}$ and $\mathbf{P}\mathbf{v}'_{base} = \mathbf{P}\mathbf{v}_{base}$. In particular, this allows \mathcal{A} to compute $\mathbf{x}' \neq \mathbf{x}$ such that $\mathcal{P}(\mathbf{x}') = \mathcal{P}(\mathbf{x})$. \square

Motivating example The following example gives the idea of characterizing collision structure for LiniCrypt programs and how a collision attack can be applied to a program. A more formal and precise algorithm to find a collision is given in the proof of Lemma 2.

$$\begin{aligned} & \underline{\mathcal{P}^E(v_1, v_2)} : \\ & \quad v_3 := E(t_1, v_1, v_2) \\ & \quad v_4 = E(t_2, v_1, v_3) \\ & \quad \text{return } v_4 + v_1 \\ & \mathbf{q}_{K_1} = (1, 0, 0, 0) \quad \mathbf{q}_{X_1} = (0, 1, 0, 0) \quad \mathbf{a}_1 = (0, 0, 1, 0) \\ & \mathbf{q}_{K_2} = (1, 0, 0, 0) \quad \mathbf{q}_{X_2} = (0, 0, 1, 0) \quad \mathbf{a}_2 = (0, 0, 0, 1) \\ & \quad \mathbf{m} = (1, 0, 0, 1) \end{aligned}$$

This program is not collision resistant and the adversary can find a collision as follows

- The adversary \mathcal{A} picks an arbitrary input $\mathbf{x} = (x_1, x_2) \in \mathbb{F}^2$ and runs the program on \mathbf{x} and gets the output l .
- \mathcal{A} picks arbitrary $x'_1 \neq x_1 \in \mathbb{F}$ and makes the query $v'_3 = E^{-1}(t_2, x'_1, l + x'_1)$.
- \mathcal{A} makes the query $x'_2 = E^{-1}(t_1, x'_1, v'_3)$ to find x'_2 .
- \mathcal{A} returns $\mathbf{x}' = (x'_1, x'_2)$.

This attack was possible because of the following,

- x'_1 was independent of the output in other words $\mathbf{q}_{K_1} \neq \mathbf{m}$.
- In step two after fixing x'_1 and the output l the value of v'_3 was not fixed which in algebraic representation means $\mathbf{q}_{X_1} \notin \text{span}(\mathbf{q}_{K_1}, \mathbf{m})$ so the adversary could determine this value by making a decryption query.
- In the third step because $\mathbf{q}_{X_2} \notin \text{span}(\mathbf{q}_{K_1}, \mathbf{q}_{K_2}, \mathbf{q}_{X_1}, \mathbf{a}_1, \mathbf{a}_2, \mathbf{m})$ the value of x'_2 is not fixed so via a decryption query the adversary finds a compatible value for x'_2 .

The following definition gives a syntactic condition on programs that will be used to characterize collision resistance. Intuitively, for a program to have a collision $\mathbf{x} \neq \mathbf{x}'$, there must first be a query for which the adversary can pick an arbitrary input. This means at least two of K, X, Y need to be independent of all other fixed values. Secondly, to get the same output value on this different input \mathbf{x}' , the results of the remaining queries must be independent of other fixed values which implies one of Y or X must be independent of the previous queries and output values. This leads to the following definition:

Definition 4. Let $\mathcal{P} = (M, \mathcal{C})$ be a LiniCrypt program. A collision structure for \mathcal{P} is a tuple (i^*, c_1, \dots, c_n) where c_1, \dots, c_n is an ordering of \mathcal{C} and $i^* \in [1, n]$, such that for $i = i^*$ at least two of the following conditions are true, and for all $i > i^*$ at least one of (C2) or (C3) is true.

- (C1) $\mathbf{q}_{K_i} \notin \text{span}(\{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_{i-1}}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i-1}}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_{i-1}\}, \text{rows}(M))$
- (C2) $\mathbf{q}_{X_i} \notin \text{span}(\{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i-1}}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_i\}, \text{rows}(M))$
- (C3) $\mathbf{a}_i \notin \text{span}(\{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_i}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_{i-1}\}, \text{rows}(M))$

Here is an example that illustrates the critical importance of nonces in the Ideal Cipher Model and how, without the use of nonces, adversaries can implement certain attacks that defy characterization within a collision structure.

$$\begin{aligned} & \underline{\mathcal{P}^E(v_1, v_2)} : \\ & v_3 := E(v_1, v_1) \\ & v_4 := E(v_2, v_2) \\ & \text{return } v_3 + v_4 \end{aligned}$$

Even though this program lacks a collision structure, the adversary can manipulate the input elements to produce collisions like (v_1, v_2) and (v_2, v_1) for any arbitrary values of v_1 and v_2 .

Lemma 2. If a LiniCrypt program \mathcal{P} with n constraints has a collision structure (i^*, c_1, \dots, c_n) then there exists a collision adversary \mathcal{A} with access to E and E^{-1} which given an input \mathbf{x} makes at most $2n$ queries and returns $\mathbf{x}' \neq \mathbf{x}$ such that $\mathcal{P}^E(\mathbf{x}') = \mathcal{P}^E(\mathbf{x})$, and so has success probability of 1 in Game 2.

Proof. The adversary \mathcal{A} first determines a setting of the base variables \mathbf{v} by running $\mathcal{P}^E(\mathbf{x})$, and creates linear constraints on unknowns \mathbf{v}' as follows:

- add constraint $M\mathbf{v}' = M\mathbf{v}$
- for $i < i^*$, add constraints $\mathbf{q}_{K_i} \cdot \mathbf{v}' = \mathbf{q}_{K_i} \cdot \mathbf{v}$, $\mathbf{q}_{X_i} \cdot \mathbf{v}' = \mathbf{q}_{X_i} \cdot \mathbf{v}$ and $\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v}$
- For $i \geq i^*$,
 - if (C1) holds, choose $K'_i \in \mathbb{F}$ so that $K'_i \neq \mathbf{q}_{K_i} \cdot \mathbf{v}$ and add the constraint $\mathbf{q}_{K_i} \cdot \mathbf{v}' = K'_i$
 - if (C2) holds, set $X'_i := E^{-1}(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}', \mathbf{a}_i \cdot \mathbf{v}')$ and add the constraint $\mathbf{q}_{X_i} \cdot \mathbf{v}' = X'_i$

- if (C3) holds, set $Y'_i := E(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}', \mathbf{q}_{X_i} \cdot \mathbf{v}')$ and add the constraint $\mathbf{a}_i \cdot \mathbf{v}' = Y'_i$
- if (C2) and (C3) both hold, choose $X'_i \in \mathbb{F}$ such that $X'_i \neq \mathbf{q}_{X_i} \cdot \mathbf{v}$, set $Y'_i := E(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}', X'_i)$ and add the constraints $\mathbf{q}_{X_i} \cdot \mathbf{v}' = X'_i$ and $\mathbf{a}_i \cdot \mathbf{v}' = Y'_i$

We claim that the constraints have a unique solution $\mathbf{v}' \neq \mathbf{v}$ such that if $x'_i = \mathbf{e}_i \cdot \mathbf{v}'$, $1 \leq i \leq k$, then $\mathbf{x}' \neq \mathbf{x}$ and $\mathcal{P}^E(\mathbf{x}') = \mathcal{P}^E(\mathbf{x})$.

To see that $\mathbf{v}' \neq \mathbf{v}$, note that for $i = i^*$, either (C1) holds, or both (C2) and (C3) hold. The choice of K'_{i^*} in the first case and X'_{i^*} in the second, ensure $\mathbf{v}' \neq \mathbf{v}$.

The constraints that are added for the output matrix and for $i < i^*$ are consistent, as they already have a solution, namely \mathbf{v} . For $i \geq i^*$, a new constraint is added only in the case that the corresponding \mathbf{q}_{K_i} , \mathbf{q}_{X_i} or \mathbf{a}_i is independent of the vectors added in previous constraints, and so consistency is maintained as constraints are added. Once all constraints are added, nondegeneracy ensures that \mathbf{v}' is unique.

Finally, \mathbf{v}' is consistent with the values returned by E and E^{-1} . This means that \mathbf{v}' corresponds to the setting of base variables resulting from evaluating $\mathcal{P}^E(\mathbf{x}')$, so from $\mathbf{v}' \neq \mathbf{v}$ we conclude $\mathbf{x}' \neq \mathbf{x}$, and from the \mathbf{M} constraint, $\mathcal{P}^E(\mathbf{x}') = \mathcal{P}^E(\mathbf{x})$. \square

Lemma 3. *Let \mathcal{P} be a Linicrypt program with n constraints. If there is an adversary \mathcal{A} for \mathcal{P} making at most N oracle queries with success probability $> N^{2n}/|\mathbb{F}|$ in the collision-resistance game (Game 1) or success probability $> N^n/|\mathbb{F}|$ in the 2nd-preimage game (Game 2) then the \mathcal{P} is either degenerate or has a collision structure (i^*, c_1, \dots, c_n) .*

Proof. We may assume the following without loss of generality:

1. \mathcal{A} does not repeat a query or make the inverse of a query it has already made. This can be achieved by recording queries as they are made.
2. For queries made in the execution of $\mathcal{P}(\mathbf{x})$ and $\mathcal{P}(\mathbf{x}')$, \mathcal{A} makes either the query or its corresponding inverse query before returning. For Game 1, this is achieved by having \mathcal{A} run $\mathcal{P}(\mathbf{x})$ and $\mathcal{P}(\mathbf{x}')$ before returning and making the corresponding queries subject to restriction (1). For Game 2 this is achieved by having \mathcal{A} initially make all the queries that result from running $\mathcal{P}(\mathbf{x})$ and also running $\mathcal{P}(\mathbf{x}')$ before returning, and making any corresponding query, subject to restriction (1), before returning.
3. \mathcal{A} actually returns \mathbf{v}, \mathbf{v}' which are the settings of base variables determined by the execution of $\mathcal{P}(\mathbf{x})$ and $\mathcal{P}(\mathbf{x}')$, respectively.

The assumptions imply that for an oracle constraint $c = (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ occurring in \mathcal{P} , \mathcal{A} determines the value of triples $(t, \mathbf{q}_K \cdot \mathbf{v}, \mathbf{q}_X \cdot \mathbf{v}, \mathbf{a} \cdot \mathbf{v})$ through exactly one of its N queries, which is either a E -query or E^{-1} -query. Based on this fact, we define two mappings $T, T' : \mathcal{C} \rightarrow [N]$ where \mathcal{C} is the set of constraints in \mathcal{P} and the $T(c_i)$ th and $T'(c_i)$ th adversary queries correspond to constraint c_i in the computation of $\mathcal{P}(\mathbf{x})$ and $\mathcal{P}(\mathbf{x}')$, and determine the triple $(t, \mathbf{q}_K \cdot \mathbf{v}, \mathbf{q}_X \cdot \mathbf{v}, \mathbf{a} \cdot \mathbf{v})$ and $(t, \mathbf{q}_K \cdot \mathbf{v}', \mathbf{q}_X \cdot \mathbf{v}', \mathbf{a} \cdot \mathbf{v}')$ respectively. In Game 1, $T(c_i)$ and $T'(c_i)$ are each mapped to one of N queries made by \mathcal{A} , so the number of possible mappings (T, T') is N^{2n} . In Game 2, Assumption 2 implies that T is fixed, so the number of possible mappings (T, T') is N^n . Using the pigeonhole principle and \mathcal{A} 's assumed advantage in each game, there is a specific mapping (T, T') for which \mathcal{A} 's advantage when using this mapping is at least $1/|\mathbb{F}|$. We will assume that the adversary is using this mapping — for any other mapping, it returns \perp as its last action.

Using the same terminology as [15], a query $c \in \mathcal{C}$ is *convergent* if $T(c) = T'(c)$, and *divergent* otherwise. Because $\mathbf{x} \neq \mathbf{x}'$ is a collision and \mathcal{P} is nondegenerate there is at least one divergent constraint. Define $\mathbf{finish}(c) = \max\{T(c), T'(c)\}$. Since each constraint has a distinct nonce, two different program constraints can not be mapped to the same adversary query, thus, function \mathbf{finish} is injective. Note a non-injective function \mathbf{finish} makes some attacks possible that can not be covered by a collision structure. Two examples of attacks can be found in Appendix 3. Arrange the oracle constraints in \mathcal{C} as (c_1, \dots, c_n) , with the convergent queries taking precedence, followed by the divergent queries sorted in ascending order based on their \mathbf{finish} and letting i^* denote the index of the first divergent constraint, we claim that (i^*, c_1, \dots, c_n) is a collision structure for \mathcal{P} .

For $i < i^*$, since each c_i is convergent we have $\mathbf{q}_{K_i} \cdot \mathbf{v}' = \mathbf{q}_{K_i} \cdot \mathbf{v}$, $\mathbf{q}_{X_i} \cdot \mathbf{v}' = \mathbf{q}_{X_i} \cdot \mathbf{v}$ and $\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v}$ and because $\mathcal{P}(\mathbf{x}) = \mathcal{P}(\mathbf{x}')$ we have $\mathbf{M}\mathbf{v}' = \mathbf{M}\mathbf{v}$.

For $i = i^*$ the query c_i is divergent thus at least one of the following inequalities holds $\mathbf{q}_{K_i} \cdot \mathbf{v}' \neq \mathbf{q}_{K_i} \cdot \mathbf{v}$, $\mathbf{q}_{X_i} \cdot \mathbf{v}' \neq \mathbf{q}_{X_i} \cdot \mathbf{v}$ or $\mathbf{a}_i \cdot \mathbf{v}' \neq \mathbf{a}_i \cdot \mathbf{v}$. If $\mathbf{a}_i \cdot \mathbf{v}' \neq \mathbf{a}_i \cdot \mathbf{v}$ or $\mathbf{q}_{X_i} \cdot \mathbf{v}' \neq \mathbf{q}_{X_i} \cdot \mathbf{v}$ then at least one of the other inequalities hold since the ideal cipher is a permutation when the key is fixed.

This gives five possible cases. Without loss of generality, in all cases, we assume that $T(c_i) < T'(c_i)$.

1. $K_i = K'_i$ and $X_i \neq X'_i$ and $Y_i \neq Y'_i$.

In this case, we prove both conditions (C2) and (C3) hold. By way of contradiction assume (C3) does not hold, say

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_j \mathbf{q}_{K_j} + \sum_{j \leq i} \beta_j \mathbf{q}_{X_j} + \sum_{j < i} \gamma_j \mathbf{a}_j + \delta \mathbf{M}, \quad (3)$$

for some $\alpha, \beta, \gamma, \delta$. After multiplying both side of the equation by $(\mathbf{v}' - \mathbf{v})$ and considering that all the queries before i^* are convergent, $\mathbf{M}(\mathbf{v}' - \mathbf{v}) = 0$, and $K_i = K'_i$ we have

$$\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v} + \beta_i \mathbf{q}_{X_i} \cdot (\mathbf{v}' - \mathbf{v})$$

Also because $Y_i \neq Y'_i$ and $X_i \neq X'_i$ we know $\beta_i \neq 0$. If $T'(c_i)$ is an encryption query then the right-hand side of the equation is a fixed value but the left-hand side is a query result, so the advantage of the adversary is $\leq 1/|\mathbb{F}|$ contrary to assumption. If $T'(c_i)$ is a decryption query then isolating $\mathbf{q}_{X_i} \cdot \mathbf{v}'$ gives us

$$\mathbf{q}_{X_i} \cdot \mathbf{v}' = \mathbf{q}_{X_i} \cdot \mathbf{v} + \frac{1}{\beta_i} \mathbf{a}_i \cdot (\mathbf{v}' - \mathbf{v})$$

and again the right-hand side of the equation is determined while the left-hand side is a random value, again giving a contradiction. The proof for condition (C2) is similar.

2. $K_i \neq K'_i$ and $X_i = X'_i$ and $Y_i \neq Y'_i$.

Because $K_i \neq K'_i$, condition (C1) holds. We want to show $T'(c_i)$ is an encryption query and (C3) holds. If $T'(c_i)$ is not an encryption then X_i is fixed and X'_i random so $X_i = X'_i$ holds with probability $\leq 1/|\mathbb{F}|$. If condition (C3) does not hold then 3 holds. Multiplying the equation to $(\mathbf{v}' - \mathbf{v})$ and applying $X_i = X'_i$ and $\mathbf{M}(\mathbf{v} - \mathbf{v}') = 0$ gives

$$\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v} + \alpha_i \mathbf{q}_{K_i} \cdot (\mathbf{v}' - \mathbf{v})$$

The left-hand side of the above equation is a random value and the right-hand side is a fixed value, so the adversary advantage again is $\leq 1/|\mathbb{F}|$.

3. $K_i \neq K'_i$ and $X_i \neq X'_i$ and $Y_i = Y'_i$.

Because $K_i \neq K'_i$, condition (C1) holds. We want to show $T'(c_i)$ has to be a decryption query and (C2) holds. If it is not a decryption query then the probability of a random value Y'_i being equal to the fixed value Y_i would be $1/|\mathbb{F}|$, which contradicts the assumption, so we assume $T'(c_i)$ is a decryption query. If condition (C2) does not hold then we have

$$\mathbf{q}_{X_i} = \sum_{j \leq i} \pi_j \cdot \mathbf{q}_{K_j} + \sum_{j < i} \rho_j \cdot \mathbf{q}_{X_j} + \sum_{j \leq i} \varsigma_j \cdot \mathbf{a}_j + \tau \mathbf{M}$$

Multiplying the equation to $(\mathbf{v}' - \mathbf{v})$ and applying $Y_i = Y'_i$ and $\mathbf{M}(\mathbf{v} - \mathbf{v}') = 0$ and canceling convergent queries, gives

$$\mathbf{q}_{X_i} \cdot \mathbf{v}' = \mathbf{q}_{X_i} \cdot \mathbf{v} + \pi_i \mathbf{q}_{K_i} \cdot (\mathbf{v}' - \mathbf{v})$$

The left-hand side of the above equation is a random value and the right-hand side is a fixed value, so the adversary advantage again is at most $1/|\mathbb{F}|$ which is a contradiction.

4. $K_i \neq K'_i$ and $X_i \neq X'_i$ and $Y_i \neq Y'_i$.

Because $K_i \neq K'_i$, condition (C1) holds. We show if $T'(c_i)$ is an encryption query then (C3) holds and if it is a decryption query then (C2) holds. In the first case, assume for contradiction that (C3) does not hold, implying Equation 3. Applying the assumption $\mathbf{M}(\mathbf{v} - \mathbf{v}') = 0$ and canceling all the queries before i^* gives,

$$\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v} + \alpha_i \mathbf{q}_{K_i} \cdot (\mathbf{v}' - \mathbf{v}) + \beta_i \mathbf{q}_{X_i} \cdot (\mathbf{v}' - \mathbf{v})$$

Here when the adversary is making query $T'(c_i)$, all the values on the right-hand side of the equation are fixed and so the adversary's advantage is $\leq 1/|\mathbb{F}|$, contrary to assumption. The case that $T'(c_i)$ is a decryption query and (C2) holds is similar.

5. $K_i \neq K'_i$ and $X_i = X'_i$ and $Y_i = Y'_i$.

The probability of this case occurring is $\leq 1/|\mathbb{F}|$.

For $i > i^*$ by way of contradiction, assume (C2) and (C3) both fail:

$$\mathbf{q}_{X_i} = \sum_{j \leq i} \pi_j \cdot \mathbf{q}_{K_j} + \sum_{j < i} \rho_j \cdot \mathbf{q}_{X_j} + \sum_{j \leq i} \varsigma_j \cdot \mathbf{a}_j + \tau \mathbf{M}$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_j \cdot \mathbf{q}_{K_j} + \sum_{j \leq i} \beta_j \cdot \mathbf{q}_{X_j} + \sum_{j < i} \gamma_j \cdot \mathbf{a}_j + \delta \mathbf{M}$$

Multiplying both sides by $(\mathbf{v}' - \mathbf{v})$ and canceling the terms having index less than i^* and noting $\mathbf{M}(\mathbf{v} - \mathbf{v}') = 0$ we get,

$$\mathbf{q}_{X_i} \cdot \mathbf{v}' = \mathbf{q}_{X_i} \cdot \mathbf{v} + \sum_{i^* \leq j \leq i} \pi_j \mathbf{q}_{K_j} \cdot (\mathbf{v}' - \mathbf{v}) + \sum_{i^* \leq j < i} \rho_j \mathbf{q}_{X_j} \cdot (\mathbf{v}' - \mathbf{v}) + \sum_{i^* \leq j \leq i} \varsigma_j \mathbf{a}_j \cdot (\mathbf{v}' - \mathbf{v}) \quad (4)$$

$$\mathbf{a}_i \cdot \mathbf{v}' = \mathbf{a}_i \cdot \mathbf{v} + \sum_{i^* \leq j \leq i} \alpha_j \mathbf{q}_{K_j} \cdot (\mathbf{v}' - \mathbf{v}) + \sum_{i^* \leq j \leq i} \beta_j \mathbf{q}_{X_j} \cdot (\mathbf{v}' - \mathbf{v}) + \sum_{i^* \leq j < i} \gamma_j \mathbf{a}_j \cdot (\mathbf{v}' - \mathbf{v}) \quad (5)$$

Now, if the adversary is making query $T'(c_i)$ as an encryption query then in Equation 5 all the values on the right-hand side of the equation are fixed and the left-hand side is random so the adversary advantage is at most $1/|\mathbb{F}|$, and if it is a decryption query, Equation 4 will give the same contradiction. So at least one of the conditions (C2) or (C3) has to hold. □

Combining the Lemmas of this section, we obtain:

Theorem 1. (Main Theorem) Suppose \mathcal{P} is a nondegenerate Linicrypt program in the ideal cipher model over \mathbb{F}_λ , with n constraints. For sufficiently large λ the following are equivalent

- \mathcal{P} is $(N, N^{2n}/|\mathbb{F}|)$ -collision resistant
- \mathcal{P} is $(N, N^n/|\mathbb{F}|)$ -2nd preimage resistant
- \mathcal{P} is $(2n, 1)$ -2nd preimage resistant
- \mathcal{P} does not have a collision structure

3.1 Efficiently Finding Collision Structures

An immediate benefit of the characterization given in the proceeding section is provided by Algorithm 1, which gives an efficient procedure for deciding whether a program has a collision structure. The algorithm splits the constraints into two stacks using two loops. In the beginning, all the constraints \mathcal{C} are in the LEFT stack and the first loop runs until all the constraints that satisfy at least one of (C2) or, (C3) are assigned to the RIGHT stack. In the second loop, those constraints that don't satisfy at least two of (C1), (C2), or (C3) will be pushed back to the LEFT stack.

Each loop makes at most n iterations, where each iteration involves several span computations. This gives a total running time of $O(n^{\omega+1})$, where ω is the exponent in the complexity of matrix multiplication.

Lemma 4. *Algorithm **FindColStruct** \mathcal{P} returns a collision structure for \mathcal{P} iff one exists.*

Proof. First, we prove if the algorithm returns (i^*, c_1, \dots, c_n) , this is a collision structure. Note that after the second loop ends,

$$V = \{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_{i^*-1}}\} \cup \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i^*-1}}\} \cup \{\mathbf{a}_1, \dots, \mathbf{a}_{i^*-1}\} \cup \text{rows}(M).$$

Also, the query c_{i^*} is still in RIGHT, so at least two of the conditions from Definition 4 hold, otherwise c_{i^*} would be sent back to LEFT.

For $i > i^*$, immediately before moving c_i from LEFT to RIGHT in the first loop, sLEFT still included $\{c_1, \dots, c_{i-1}\}$ which means

$$V = \{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\} \cup \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_i}\} \cup \{\mathbf{a}_1, \dots, \mathbf{a}_i\} \cup \text{rows}(M),$$

and $\mathbf{q}_X \notin \text{span}(V \setminus \{\mathbf{q}_X\})$ or $\mathbf{a} \notin \text{span}(V \setminus \{\mathbf{a}\})$. Hence,

Algorithm 1 FindColStruct $\mathcal{P}(M, \mathcal{C})$

LEFT := \mathcal{C}

RIGHT := empty stack

$V := \{\mathbf{q}_K | (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \cup \{\mathbf{q}_X | (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \cup \{\mathbf{a} | (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \mathcal{C}\} \cup \text{rows}(M)$

while $\exists (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \text{LEFT}$ where $\mathbf{q}_X \notin \text{span}(V \setminus \{\mathbf{q}_X\})$ or $\mathbf{a} \notin \text{span}(V \setminus \{\mathbf{a}\})$ **do**

 remove $(t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ from LEFT

 push $(t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ to RIGHT

 reduce multiplicity of $\mathbf{q}_K, \mathbf{q}_X$ and \mathbf{a} in V by 1

end while

while $\exists (t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}) \in \text{RIGHT}$ where

$(\mathbf{q}_K \in \text{span}(V) \wedge \mathbf{q}_X \in \text{span}(V \cup \mathbf{q}_K \cup \mathbf{a}))$ or

$(\mathbf{q}_K \in \text{span}(V) \wedge \mathbf{a} \in \text{span}(V \cup \mathbf{q}_K \cup \mathbf{q}_X))$ or

$(\mathbf{q}_X \in \text{span}(V \cup \mathbf{q}_K \cup \mathbf{a}) \wedge \mathbf{a} \in \text{span}(V \cup \mathbf{q}_K \cup \mathbf{q}_X))$ **do**

 remove $(t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ from RIGHT

 add $(t, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ to LEFT

 increase multiplicity of $\mathbf{q}_K, \mathbf{q}_X$ and \mathbf{a} in V by 1

end while

if RIGHT is nonempty **then**

$i^* := |\text{LEFT}| + 1$

 let LEFT = (c_1, \dots, c_{i^*-1}) where order doesn't matter

 let RIGHT = (c_{i^*}, \dots, c_n) in reverse order of insertion

 return (i^*, c_1, \dots, c_n)

else return \perp

end if

$$\mathbf{q}_{X_i} \notin \text{span}(\{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\} \cup \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i-1}}\} \cup \{\mathbf{a}_1, \dots, \mathbf{a}_i\} \cup \text{rows}(\mathbf{M}))$$

or

$$\mathbf{a}_i \notin \text{span}(\{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\} \cup \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_i}\} \cup \{\mathbf{a}_1, \dots, \mathbf{a}_{i-1}\} \cup \text{rows}(\mathbf{M}))$$

satisfying one the conditions (C2), (C3) from Definition 4.

To prove the other direction, we prove if there is a collision structure for \mathcal{P} then the second phase c_{i^*} is not sent back from RIGHT to LEFT, and so RIGHT $\neq \perp$. By contradiction suppose the algorithm adds c_{i^*} to LEFT. Denote by S the set of indices of constraints in LEFT immediately before c_{i^*} is added.

Then, for c_{i^*} to be sent back, at least 2 of the following conditions must hold

$$\mathbf{q}_{K_{i^*}} = \sum_{j \in S} \kappa_j \mathbf{q}_{K_j} + \sum_{j \in S} \lambda_j \mathbf{q}_{X_j} + \sum_{j \in S} \mu_j \mathbf{a}_j + \nu \mathbf{M} \quad (6)$$

$$\mathbf{q}_{X_{i^*}} = \sum_{j \in S \cup \{i^*\}} \pi_j \mathbf{q}_{K_j} + \sum_{j \in S} \rho_j \mathbf{q}_{X_j} + \sum_{j \in S \cup \{i^*\}} \zeta_j \mathbf{a}_j + \tau \mathbf{M} \quad (7)$$

$$\mathbf{a}_{i^*} = \sum_{j \in S \cup \{i^*\}} \alpha_j \mathbf{q}_{K_j} + \sum_{j \in S \cup \{i^*\}} \beta_j \mathbf{q}_{X_j} + \sum_{j \in S} \gamma_j \mathbf{a}_j + \delta \mathbf{M} \quad (8)$$

Since, after the first loop $\{c_{i^*}, \dots, c_n\} \subseteq \text{RIGHT}$, and if any c_j for $j > i^*$ is sent back to LEFT it means at least 2 of $\{\mathbf{q}_{K_j}, \mathbf{q}_{X_j}, \mathbf{a}_j\}$ were already in the right-hand side of the above equations which is the span of vectors in LEFT and $\text{rows}(\mathbf{M})$, so we can rewrite Equations 6, 7 and 8 as follows where the $S_1 \cup S_2 \cup S_3 = \{i^*, \dots, n\}$ and each index appears at least 2 times in the unions of these three sets.

$$\mathbf{q}_{K_{i^*}} = \sum_{j \in S \setminus S_1} \kappa'_j \mathbf{q}_{K_j} + \sum_{j \in S \setminus S_2} \lambda'_j \mathbf{q}_{X_j} + \sum_{j \in S \setminus S_3} \mu'_j \mathbf{a}_j + \nu' \mathbf{M} \quad (9)$$

$$\mathbf{q}_{X_{i^*}} = \sum_{j \in S \setminus S_1} \pi'_j \mathbf{q}_{K_j} + \sum_{j \in S \setminus S_2} \rho'_j \mathbf{q}_{X_j} + \sum_{j \in S \cup \{i^*\} \setminus S_3} \zeta'_j \mathbf{a}_j + \rho' \mathbf{M} \quad (10)$$

$$\mathbf{a}_{i^*} = \sum_{j \in S \setminus S_1} \alpha'_j \mathbf{q}_{K_j} + \sum_{j \in S \cup \{i^*\} \setminus S_2} \beta'_j \mathbf{q}_{X_j} + \sum_{j \in S \setminus S_3} \gamma'_j \mathbf{a}_j + \delta' \mathbf{M} \quad (11)$$

Assume 2 of these equations hold. If in these equations j_1 , j_2 , and j_3 be the maximum indices in $S \setminus S_1$, $S \setminus S_2$ and $S \setminus S_3$ then there are 2 cases,

If $j_1, j_2, j_3 \leq i^*$ then all the indices on the right-hand side are less than i^* and because at least two of the Equations 9, 10 and 11 are true, this contradicts with the condition for i^* in Definition 4.

If j_3, j_2 , and j_1 are more than i^* then the coefficients with these indices in the above equations are nonzero. If the $\max\{j_1, j_2, j_3\} = j_1$ it means $\mathbf{q}_{K_{j_1}}$ was not in the span of V but both $\mathbf{q}_{X_{j_1}}$ and \mathbf{a}_{j_1} were in the span of constraints with smaller indices and $\text{rows}(\mathbf{M})$ which is a contradiction otherwise they would not be in the RIGHT after the first loop. Thus, the $\max\{j_1, j_2, j_3\}$ is either j_2 or j_3 . If j_3 is the max then \mathbf{a}_{j_3} was not in the span of LEFT and $\text{rows}(\mathbf{M})$ so the other two vectors $\mathbf{q}_{K_{j_3}}$ and $\mathbf{q}_{X_{j_3}}$ had to be in the span of LEFT and $\text{rows}(\mathbf{M})$ and all the vectors in LEFT have smaller index than j_3 thus for c_{j_3} to be sent to RIGHT in the first loop, \mathbf{a}_{j_3} had to be independent of previous constraints and the output (condition (C3)), but we can rewrite Equation 9 as follow,

$$\mathbf{a}_{j_3} = -\frac{1}{\gamma'_{j_3}} \left(\sum_{j \in S \setminus S_1} \alpha'_j \mathbf{q}_{K_j} - \mathbf{q}_{K_{i^*}} + \sum_{j \in S \setminus S_2} \beta'_j \mathbf{q}_{X_j} + \sum_{j \in S \setminus \{S_3 \cup j_3\}} \gamma'_j \mathbf{a}_j + \delta' \mathbf{M} \right), \quad (12)$$

contradicting condition (C3) of Definition 4. The case for j_2 is similar. \square

Discussion Beyond validating the correspondence between our characterization of collision resistance for rate-1 compression functions and the well-known notion of group-1 for PGV, Theorem 2 situates our understanding of the group-1 functions as part of a general framework for collision resistance using Lincrypt. As an immediate application of Theorem 2 and Algorithm 1, we can automatically generate all group-1 PGV compression functions.³ In particular, this provides a purely *syntactic* characterization using algebraic properties of the defining program \mathcal{P} , including the possibility of automated identification using **FindColStruct**. However, we note that [4,5] provide a finer analysis of the PGV compression functions, also identifying the *group-2* functions which, although not collision-resistant as compression functions, are still suitable for constructing collision-resistant hash functions and analyzing the preimage resistance of group-1 and -2 PGV functions. We leave an extended analysis of this sort in the Lincrypt setting to future work. We give some examples of definitions of PGV compression functions from [4] and their analysis on the same GitHub page.

3.2 Discussion

We note that the significance of our results is somewhat limited by the fact that a characterization of collision-resistance for rate-1 (which is the most significant from a practical perspective) was already provided by [4]. However, our more general setting does provide some advantages. Our characterization is uniform and based solely on the syntax of programs (expressed algebraically). We have noted that the approach of [4] is ad-hoc, while that of [5] depends on semantic properties (i.e. bijectiveness) of the component functions. One benefit of our approach is that it immediately gives an efficient automated enumeration technique. We also note that in order to obtain security properties beyond collision resistance (see below) we may need to consider programs that make more than one call to the ideal cipher. The general characterization for collision resistance may be viewed as a step towards characterizations of other properties. Finally, our results demonstrate that the utility of the Lincrypt framework is not limited to the random oracle model.

4 Group-1 PGV Compression Functions

A compression function is *rate-1* if it uses one call to an underlying primitive, such as a random oracle or ideal cipher. In the latter case, we assume (without loss of generality) that there is one call to the ideal encryption function. The systematic study of such compression functions has a long history. Building on the initial work of [17], [4] gives a definition of 64 possible rate-1 compression functions mapping $D \times D \rightarrow D$, where $D = \mathbb{GF}(2^\lambda)$, that is definable using \oplus and a constant value from D . Typically, these functions are referred to as *PGV compression functions*. Among its results, [4] identifies a subset of these functions, the *group-1* functions, and proves that these are exactly the PGV compression functions that are collision-resistant.

Our goal in this section is to give a characterization of the group-1 functions in Lincrypt. In particular, we will show that a PGV compression function is group-1 iff it does not have a collision structure. Thus the characterization of [4] may be viewed as a special case of our general characterization.

The results of [4] are revisited in [5], and proven via a more unified approach, building on the approach of [19]. This is based on the factorization of a compression function f into two component functions f' and f'' . In particular, for a *message* m and *chaining value* h , $f(h, m) = f''(h, m, y)$, where $y = E_k(x)$ and $(k, x) = f'(h, m)$. In the case that f' is bijective, the function f^* is defined by $f^*(k, x, y) = f''(h, m, y)$ where $(h, m) = f^{-1}(k, x)$. Using f' , f'' , and f^* , the following properties of f are defined:

- P1 f' is bijective.
- P2 $f''(h, m, \cdot)$ is bijective for all (h, m) .
- P3 $f^*(k, \cdot, y)$ is bijective for all (k, y) .
- P4 $f'(h, \cdot)|_1$ is bijective for all h .

In [4,5], a stronger notion of collision-resistance for compression functions is used:

³ A sample implementation in Octave is available at <https://github.com/zahrajava/PGVCollisionResistantCompressionFunctions.git>

Definition 5 ([5] Definition 3). Program \mathcal{P} is (q, ϵ) -collision resistant if for any $h_0 \in \mathbb{F}$, any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$(\mathbf{x}, \mathbf{x}') \leftarrow \mathcal{A}^{E, E^{-1}}(\lambda); \text{ return } (\mathbf{x} \neq \mathbf{x}') \text{ and } \mathcal{P}^E(\mathbf{x}) = \mathcal{P}^E(\mathbf{x}') \text{ or } \mathcal{P}^E(\mathbf{x}) = h_0 \quad (13)$$

Letting T1 denote the conjunction of P1, P2, and P3, we have

Lemma 5 ([5] Lemma 3). Suppose $f : \mathbb{GF}(2^\lambda) \times \mathbb{GF}(2^\lambda) \rightarrow \mathbb{GF}(2^\lambda)$ is a PGV compression function satisfying T1. Then for any $h_0 \in \mathbb{GF}(2^\lambda)$ the advantage of any adversary \mathcal{A} making q queries in Game 13 is at most $q(q+1)/2^\lambda$.

Remark 1. The compression functions satisfying T1 are exactly the *group 1* functions defined in [4].

In the Lincrypt framework, a PGV compression function f is specified via

- An output matrix $\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \end{bmatrix}$, where \mathbf{m}_2 is always $(0, 0, 1, 0)$ (corresponding to the fixed value c , as described below,) while \mathbf{m}_1 is one of $(1, 0, 0, 1)$, $(0, 1, 0, 1)$, $(1, 1, 0, 1)$ or $(0, 0, 1, 1)$.
- A single query constraint $\left(\begin{bmatrix} \mathbf{q}_K \\ \mathbf{q}_X \end{bmatrix}, \mathbf{a} \right)$, where each \mathbf{q}_i , $i \in \{K, X\}$, is one of $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(1, 1, 0, 0)$ or $(0, 0, 1, 0)$, and \mathbf{a} is $(0, 0, 0, 1)$.

Here use \mathbf{m}_2 to capture the use of a constant value in the Lincrypt setting, as described in Section 2.2.

Up to our convention regarding the constant value c , f'' corresponds to \mathbf{m}_1 while f' corresponds to $\begin{bmatrix} \mathbf{q}_K \\ \mathbf{q}_X \end{bmatrix}$.

In particular, writing $\mathbf{q}_i = (q_i^1, q_i^2, q_i^3, q_i^4)$, $i \in \{K, X\}$, we have $f'(h, m) = \begin{bmatrix} \mathbf{q}'_K \\ \mathbf{q}'_X \end{bmatrix} \times (h, m, c)$, where $\mathbf{q}'_i = (q_i^1, q_i^2, q_i^3)$, $i \in \{K, X\}$. We also have $f''(h, m, y) = \mathbf{m}_1 \cdot (h, m, c, y)$.

We will use the following simple fact in several proofs below:

Proposition 1. Over any field \mathbb{F} , a function of the form $g(x) = rx + s$, where $r, s \in \mathbb{F}$, is bijective iff $r \neq 0$.

In the following, let f denote a compression function defined by $\mathbf{m}_1, \mathbf{m}_2, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a}$, as described above, with respect to a fixed constant c . Define the following matrices

$$\mathbf{R} = \begin{bmatrix} \mathbf{q}_K \\ \mathbf{q}_X \\ \mathbf{m}_2 \\ \mathbf{a} \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} \mathbf{q}_K \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{a} \end{bmatrix}$$

Lemma 6. Writing $\mathbf{m}_1 = (m_1^1, m_1^2, m_1^3, m_1^4)$, f satisfies P2 iff $m_1^4 \neq 0$.

Proof. For fixed h, m , $f'(h, m, y) = \mathbf{m}_1 \cdot (h, m, c, y) = m_1^4 y \oplus s$, where s is fixed. □

We note that every PGV compression function satisfies $m_1^4 \neq 0$ and hence P2.

Lemma 7. f satisfies P1 iff \mathbf{R} is nonsingular.

Proof. Let $\mathbf{R}' = \mathbf{R}[1-3; 1-3]$ be the 3×3 principle submatrix of \mathbf{R} . Given the possible values of \mathbf{q}_K and \mathbf{q}_X , \mathbf{R} is nonsingular iff \mathbf{R}' is nonsingular. For any h, m , $f'(h, m) = \mathbf{R}' \times (h, m, c)$, which is a bijection iff \mathbf{R}' is nonsingular. □

Lemma 8. Assume \mathbf{R} is nonsingular. Then f satisfies P3 iff \mathbf{S} is nonsingular.

Proof. First note that $f^*(k, x, y) = \mathbf{m}_1 \cdot (h, m, c, y) = \mathbf{m}_1 \cdot (\mathbf{R}^{-1} \times (k, x, c, y)) = (\mathbf{m}_1 \mathbf{R}^{-1}) \times (k, x, c, y)$. Let $\mathbf{u} = (u_1, u_2, u_3, u_4) = \mathbf{m}_1 \mathbf{R}^{-1}$. Then for fixed k, y , $f^*(k, x, y) = u_2 x \oplus s$, where s is fixed. Thus, it is enough to show \mathbf{S} is nonsingular iff $u_2 \neq 0$. Since \mathbf{R} is nonsingular, \mathbf{S} is nonsingular iff $\mathbf{S} \mathbf{R}^{-1}$ is nonsingular. But $\mathbf{S} \mathbf{R}^{-1} = (\mathbf{e}_1, \mathbf{u}, \mathbf{e}_3, \mathbf{e}_4)$, which is nonsingular iff $u_2 \neq 0$. □

Together, the preceding Lemmas give the following

Lemma 9. *A PGV function f satisfies T1 iff \mathbf{R} and \mathbf{S} are nonsingular.*

In the rate-1 setting conditions (C1), (C2), (C3) become

- (C1) $\mathbf{q}_K \notin \text{span}(\{\mathbf{m}_1, \mathbf{m}_2\})$
- (C2) $\mathbf{q}_X \notin \text{span}(\{\mathbf{q}_K, \mathbf{a}, \mathbf{m}_1, \mathbf{m}_2\})$
- (C3) $\mathbf{a} \notin \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\})$

Given the possible values of $\mathbf{q}_K, \mathbf{m}_1$ and \mathbf{m}_2 , (C1) may be further simplified to

- (C1) $\mathbf{q}_K \neq \mathbf{m}_2$

Lemma 10. *Suppose f is a PGV compression function specified by $\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}$. If both \mathbf{R} and \mathbf{S} are nonsingular, then two of the conditions (C1), (C2), (C3) must fail.*

Proof. If (C1) fails, then \mathbf{S} is necessarily singular, so we must show that under the assumption, both (C2) and (C3) fail. Clearly, if \mathbf{S} is nonsingular,

$$\mathbf{q}_X \in \text{span}(\text{rows}(\mathbf{S})) = \text{span}(\{\mathbf{q}_K, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) \quad (*)$$

so (C2) fails. Now since \mathbf{R} is nonsingular $\mathbf{q}_X \notin \text{span}(\{\mathbf{q}_K, \mathbf{m}_2, \mathbf{a}\})$, which in combination with (*) means that $\mathbf{m}_1 \in \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_2, \mathbf{a}\})$ (**). Noting that the last component of \mathbf{m}_1 is always nonzero, we have $\mathbf{m}_1 \notin \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_2\})$. Combining this last fact with (**), we conclude

$$\mathbf{a} \in \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\}),$$

so that (C3) also fails. □

Lemma 11. *Suppose f is a nondegenerate PGV compression function specified by $\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}$. If one of \mathbf{R}, \mathbf{S} is singular, then two of the conditions (C1), (C2), (C3) must hold.*

Proof. First, suppose (C2) fails, so $\mathbf{q}_X \in \text{span}(\{\mathbf{q}_K, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\})$. Then, if \mathbf{S} is singular,

$$\text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) = \text{span}(\text{rows}(\mathbf{S})) \not\supseteq \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\},$$

which means f is degenerate. Thus \mathbf{S} is nonsingular. As in the proof of Lemma 10, this means $\mathbf{q}_K \neq \mathbf{m}_2$. Also if \mathbf{S} is nonsingular then by the assumption, \mathbf{R} is singular, so we must have $\mathbf{q}_K = \mathbf{q}_X$ or $\mathbf{q}_X = \mathbf{m}_2$. If the former holds, $\mathbf{a} \in \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\})$ implies $\mathbf{q}_K = \mathbf{q}_X = \mathbf{a} \oplus \mathbf{m}_1$, and so

$$\text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) = \text{span}(\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) \not\supseteq \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\},$$

If the latter holds then $\mathbf{a} \in \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\})$ implies

$$\text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) = \text{span}(\{\mathbf{q}_K, \mathbf{m}_1, \mathbf{m}_2\}) \not\supseteq \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\},$$

So in both cases, by nondegeneracy $\mathbf{a} \notin \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\})$, and we have that if (C2) fails, both (C1) and (C3) must hold.

Now suppose (C2) holds and (C1) fails. Then

$$\text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\}) = \text{span}(\{\mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\}),$$

and so, if $\mathbf{a} \in \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\})$,

$$\begin{aligned} \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2, \mathbf{a}\}) &= \text{span}(\{\mathbf{q}_K, \mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\}) \\ &= \text{span}(\{\mathbf{q}_X, \mathbf{m}_1, \mathbf{m}_2\}) \not\supseteq \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4\}. \end{aligned}$$

In conclusion, if (C2) holds, then one of (C1) or (C3) must hold. □

Combining Lemmas 9, 10, and 11, we obtain

Theorem 2. *A PGV compression function is group-1 iff it does not have a collision structure.*

5 Beyond Group-1 Compression Functions

In [5], a categorization of the 64 PGV compression functions into three groups is given. Group 1, discussed in Section 4, comprises collision-resistant functions. Group 2 includes 8 PGV compression functions that, while not inherently collision-resistant, yield collision-resistant hash functions when subjected to the Merkle-Damgård transformation. Extending this terminology, we will designate the remaining functions as being in group 3.

In this section, we will first present a model for hash functions resulting from the iteration of a program $\mathcal{P}(m, \mathcal{C})$ via the Merkle-Damgård transformation and analyze their collision-resistant property by examining the characteristics of the underlying compression functions. Next, we describe various attacks on compression functions, focusing on aspects which are not captured by collision structures. In particular, we provide characterizations of different groups of programs, each susceptible to specific types of attacks as introduced in Preneel (1994) [17]. Additionally, we concentrate on attacks that lead to collision vulnerabilities within the iterated hash functions. Specifically, we explore the vulnerabilities tied to the *direct attack* and *permutation attack*. We provide characterizations of programs susceptible to these attacks and illustrate how possessing any of these characteristics can result in non-collision-resistant iterated hash functions. Furthermore, we generalize group-2 PGV functions as those programs $\mathcal{P}(m, \mathcal{C})$ that, while not inherently collision-resistant, yield a collision-resistant hash function. These are also identified as group-2 Linicrypt programs. We elucidate the structure of hash functions constructed through the Merkle-Damgård construction from a program $\mathcal{P}(m, \mathcal{C})$ in the Linicrypt framework. Moreover, we demonstrate that none of the attacks discussed in this section apply to group-2 programs.

5.1 Iterated Hash Functions in Linicrypt

Here, we present a model for the hash function $\mathcal{H}^{\mathcal{P}}$, which is constructed by iterating the program \mathcal{P} over a message \mathbf{W} formatted as a $k \times b$ matrix. During the i th iteration, the input provided to the program is the i th row of this matrix. Let program $\mathcal{P} : \mathbb{F}^k \rightarrow \mathbb{F}$ be a Linicrypt program in the ideal cipher model and $IV = \ell_0 \in \mathbb{F}$ is a fixed value. By iterating \mathcal{P} on input vectors $\mathbf{x}_i \in \mathbb{F}^k$ with MD chaining, we get construction $\mathcal{H}^{\mathcal{P}} : \mathbb{F}^* \rightarrow \mathbb{F}$ where $l_i = \mathcal{P}(\mathbf{x}_i)$ for $i = 1, \dots, b$ is the chaining value. Note, vector \mathbf{x}_i is the input of program \mathcal{P} in i th iteration and the first element of each $\mathbf{x}_i = (x_i^1, \dots, x_i^k)$ is the chaining value so $x_{i+1}^1 = l_i$ for $i \in [0, b-1]$. And the construction $\mathcal{H}^{\mathcal{P}}$ runs on $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_b)^\top$ where $\mathbf{w}_i = (x_i^2, \dots, x_i^k)$ and $\mathcal{H}^{\mathcal{P}}(\mathbf{W}) = l_b$.

Security Definitions We assume the number of oracle queries the adversary makes to E and E^{-1} are q_E and q_D respectively.

Definition 6. Construction $\mathcal{H}^{\mathcal{P}}$ is (q, ϵ) -collision resistant if any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$(\mathbf{W}, \mathbf{W}') \leftarrow \mathcal{A}^{E, E^{-1}}(\lambda); \text{ return } (\mathbf{W} \neq \mathbf{W}') \text{ and } \mathcal{H}^{\mathcal{P}}(\mathbf{W}) = \mathcal{H}^{\mathcal{P}}(\mathbf{W}') \quad (14)$$

Definition 7. Program \mathcal{P} is (q, ϵ) -preimage resistant for a fixed first input if any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$\ell_0 \leftarrow \mathbb{F}; \ell \leftarrow \mathbb{F}^r; \mathbf{x} \leftarrow \mathcal{A}^{E, E^{-1}}(\ell_0, \ell, \lambda); \text{ return } \mathcal{P}^E(\mathbf{x}) = \ell \text{ and } (x^1 = \ell_0) \quad (15)$$

We refer to this game, the *direct attack* by the adversary.

Definition 8. Program \mathcal{P} is (q, ϵ) -2nd-preimage resistant for fixed first inputs if any oracle adversary \mathcal{A} making at most $q = q_E + q_D$ queries has probability of success at most ϵ in the following game:

$$\ell_0, \ell'_0 \leftarrow \mathbb{F}; \mathbf{w} \leftarrow \mathbb{F}^{k-1}; \ell = \mathcal{P}(\ell_0, \mathbf{w}); \mathbf{w}' \leftarrow \mathcal{A}^{E, E^{-1}}(\ell_0, \ell'_0, \mathbf{w}, \lambda); \text{ return } \mathcal{P}^E(\ell'_0, \mathbf{w}') = \ell \quad (16)$$

We refer to this game, the *forward attack* by the adversary.

5.2 Characterizing Certain Attacks on Group 3 Functions

The degenerate programs are inherently vulnerable, making them susceptible to a forward or direct attack. Therefore, we exclude these programs from further consideration, and for the remainder of this section, we focus only on nondegenerate programs.

Another attack, introduced in Preneel (1994) [17] and known as the *permutation attack*, results in a collision for the hash function constructed from certain compression functions. The programs vulnerable to this attack exhibit a linear dependency on the chaining value, allowing an adversary to generate a collision by rearranging the message blocks \mathbf{w}_i . In these programs, although a direct attack may be difficult due to the program's linear dependence on the chaining value, the resulting hash value $\mathcal{H}^{\mathcal{P}}(\mathbf{W}) = \ell_b$ remains unaffected by the order of the blocks. The following definition will outline the characteristics of programs that are vulnerable to the permutation attack.

Definition 9. Program \mathcal{P} is linearly-chained if

$$\mathbf{e}_1 \notin \text{span}(\text{rows}(\mathbf{Q}_K) \cup \text{rows}(\mathbf{Q}_X) \cup \{\mathbf{e}_2, \dots, \mathbf{e}_k\})$$

Lemma 12. If $\mathcal{P} = (m, \mathcal{C})$ is linearly-chained then there exist an adversary \mathcal{A} making at most $2n$ queries and succeeds with probability 1 in game 14

Proof. The program queries are independent of the chaining value so the adversary changes the order of \mathbf{w}_i 's and gets the same output $\mathcal{H}^{\mathcal{P}}(\mathbf{W})$ always.

The next definition provides a characterization of programs vulnerable to the direct attack 2. The hash functions constructed from these programs are susceptible to a collision attack.

Definition 10. Let $\mathcal{P} = (m, \mathcal{C})$ be a LiniCrypt program with n constraints, where the first element of \mathcal{P} 's input corresponds to the chaining value. A direct attack structure for \mathcal{P} is a tuple (c_1, \dots, c_n) where c_1, \dots, c_n is an ordering of \mathcal{C} where for every $i \in [1, n]$ at least one of (C1) or (C2) is true.

- (C1) $\mathbf{q}_{X_i} \notin \text{span}(\{\mathbf{e}_1\}, \{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i-1}}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_i\}, \text{rows}(\mathbf{M}))$
(C2) $\mathbf{a}_i \notin \text{span}(\{\mathbf{e}_1\}, \{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_i}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_i}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_{i-1}\}, \text{rows}(\mathbf{M}))$

The following lemma shows that if a program possesses a direct attack structure, then it is vulnerable to a direct attack.

Lemma 13. If a LiniCrypt program $\mathcal{P} = (m, \mathcal{C})$ with n constraints has a direct attack structure (c_1, \dots, c_n) then there exists an adversary \mathcal{A} that makes at most n queries and succeeds with the probability of 1 in the direct attack game 15.

Proof. Assume (ℓ_0, ℓ) is given to the adversary. The adversary \mathcal{A} creates the following constraints to determine the unknown elements of vector $\mathbf{v} = (\ell_0, v_2, \dots, v_{k+n})^\top$.

- add $m\mathbf{v} = \ell$.
- For every $i \in [1, n]$,
 - If $\mathbf{q}_{K_i} \notin \text{span}(\{\mathbf{e}_1\}, \{\mathbf{q}_{K_1}, \dots, \mathbf{q}_{K_{i-1}}\}, \{\mathbf{q}_{X_1}, \dots, \mathbf{q}_{X_{i-1}}\}, \{\mathbf{a}_1, \dots, \mathbf{a}_{i-1}\}, \text{rows}(\mathbf{M}))$, choose $K_i \in \mathbb{F}$ add the constraint $\mathbf{q}_{K_i} \cdot \mathbf{v} = K_i$.
 - If (C1) holds, set $X_i := E^{-1}(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}, \mathbf{a}_i \cdot \mathbf{v})$ and add the constraint $\mathbf{q}_{X_i} \cdot \mathbf{v} = X_i$
 - If (C2) holds, set $Y_i := E(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}, \mathbf{q}_{X_i} \cdot \mathbf{v})$ and add the constraint $\mathbf{a}_i \cdot \mathbf{v} = Y_i$
 - if (C1) and (C2) both hold, choose $X_i \in \mathbb{F}$ and set $Y_i := E(t_i, \mathbf{q}_{K_i} \cdot \mathbf{v}, X_i)$ and add the constraints $\mathbf{q}_{X_i} \cdot \mathbf{v} = X_i$ and $\mathbf{a}_i \cdot \mathbf{v} = Y_i$

We claim that the constraints have a unique solution \mathbf{v} such that if $\mathbf{x}^\top = (\mathbf{e}_1, \dots, \mathbf{e}_k)^\top \cdot \mathbf{v}$, then $\mathcal{P}^E(\mathbf{x}) = \ell$.

The constraints that are added for the output vector and for every $i \in [1, n]$ are consistent, as a new constraint is added only in the case that the corresponding \mathbf{q}_{K_i} , \mathbf{q}_{X_i} or \mathbf{a}_i is independent of the vectors added for previous constraints, the image and \mathbf{e}_1 , so consistency is maintained. Once all constraints are added, nondegeneracy ensures that \mathbf{v} is unique.

Finally, \mathbf{v} is consistent with the values returned by E and E^{-1} . This means that \mathbf{v} corresponds to the setting of base variables resulting from evaluating $\mathcal{P}^E(\mathbf{x})$, so from the $m\mathbf{v} = \ell$ constraint, $\mathcal{P}^E(\mathbf{x}) = \ell$.

The next lemma shows if there is a direct attack adversary for a program \mathcal{P} with n constraints then there is a direct attack structure (c_1, \dots, c_n) for \mathcal{P} .

Lemma 14. *Let \mathcal{P} be a Linicrypt program with n constraints. If there is a direct adversary \mathcal{A} for \mathcal{P} making at most N oracle queries with success probability $> (N/n)^n / |\mathbb{F}|$ then \mathcal{P} has direct attack structure (c_1, \dots, c_n) .*

Proof. Without loss of generality, we assume

1. \mathcal{A} does not repeat a query or make the inverse of a query it has already made. This can be achieved by recording queries as they are made.
2. For queries made in the execution of $\mathcal{P}^E(\mathbf{x})$, \mathcal{A} makes either the query or its corresponding inverse query before returning. For Game 15, this is achieved by having \mathcal{A} run $\mathcal{P}^E(\mathbf{x})$ before returning and making the corresponding queries subject to restriction (1).
3. \mathcal{A} returns $\mathbf{v} = (v_1, \dots, v_k)$ which are respectively, the settings of base variables determined by the execution of $\mathcal{P}^E(\mathbf{x})$.

The assumptions (1),(2),(3) imply that for an oracle constraint $c_i = (t_i, \mathbf{q}_K, \mathbf{q}_X, \mathbf{a})$ occurring in \mathcal{P} , \mathcal{A} determines the value of triples $(\mathbf{q}_K \cdot \mathbf{v}, \mathbf{q}_X \cdot \mathbf{v}, \mathbf{a} \cdot \mathbf{v})$ through exactly one of its N_i queries with nonce t_i , which is either a E -query or E^{-1} -query.

Based on this fact, we define mappings $T : \mathcal{C} \rightarrow [N]$ where \mathcal{C} is the set of constraints in \mathcal{P} and the $T(c_i)$ th adversary queries correspond to constraint c_i in the computation of $\mathcal{P}(\mathbf{x})$ and determines the triple $(\mathbf{q}_K \cdot \mathbf{v}_i, \mathbf{q}_X \cdot \mathbf{v}_i, \mathbf{a} \cdot \mathbf{v}_i)$. Letting N_i denote the number of queries made by \mathcal{A} using nonce t_i , $1 \leq i \leq n$, we have that there are $\prod_{i=1}^n N_i$ possible mappings. Since $\sum_{i=1}^n N_i \leq N$, the product is maximized when $N_i = N/n$, so that there are at most $(N/n)^n$ possible mappings. Using the pigeonhole principle and \mathcal{A} 's advantage in the direct attack game, there is a specific mapping T for which \mathcal{A} 's advantage when using this mapping is at least $1/|\mathbb{F}|$. We will assume that the adversary is using this mapping — for any other mapping, it returns \perp as its last action.

Knowing there are no useless constraints in \mathcal{P} , each query result is used as part of the input to another useful query, or as part of the final output of the program. Thus having ℓ_1 and ℓ_0 fixed, for each query, either part of the input or the output is fixed. By way of contradiction, we assume none of (C2) and (C3) hold for a constraint c_i , $1 \leq i \leq n$ in \mathcal{P} ;

$$\begin{aligned} \mathbf{q}_{X_i} &= \sum_{j \leq i} \pi_j \mathbf{q}_{K_j} + \sum_{j < i} \rho_j \mathbf{q}_{X_j} + \sum_{j \leq i} \varsigma_j \mathbf{a}_j + \tau \mathbf{M} + \kappa \mathbf{e}_1 \\ \mathbf{a}_i &= \sum_{j \leq i} \alpha_j \mathbf{q}_{K_j} + \sum_{j \leq i} \beta_j \mathbf{q}_{X_j} + \sum_{j < i} \gamma_j \mathbf{a}_j + \delta \mathbf{M} + \nu \mathbf{e}_1 \end{aligned}$$

Multiplying both sides by \mathbf{v}

$$\begin{aligned} \mathbf{q}_{X_i} \cdot \mathbf{v} &= \sum_{j \leq i} \pi_j \mathbf{q}_{K_j} \cdot \mathbf{v} + \sum_{j < i} \rho_j \mathbf{q}_{X_j} \cdot \mathbf{v} + \sum_{j \leq i} \varsigma_j \mathbf{a}_j \cdot \mathbf{v} + \tau \mathbf{M} \cdot \mathbf{v} + \kappa \mathbf{e}_1 \cdot \mathbf{v} \\ \mathbf{a}_i \cdot \mathbf{v} &= \sum_{j \leq i} \alpha_j \mathbf{q}_{K_j} \cdot \mathbf{v} + \sum_{j \leq i} \beta_j \mathbf{q}_{X_j} \cdot \mathbf{v} + \sum_{j < i} \gamma_j \mathbf{a}_j \cdot \mathbf{v} + \delta \mathbf{M} \cdot \mathbf{v} + \nu \mathbf{e}_1 \cdot \mathbf{v} \end{aligned}$$

Since in Game 15, the output of the program and the first element of the input are fixed, the last two terms on the right-hand side of both equations are fixed. If the adversary is making the query $T(c_i)$ as an encryption query then in the second equation all the terms on the right-hand side are fixed and the advantage of the adversary to make the equality is $\leq 1/|\mathbb{F}|$ which is a contradiction. And if $T(c_i)$ is a decryption query then the first equation on the right-hand side is fixed so the adversary advantage is $\leq 1/|\mathbb{F}|$ which is a contradiction.

The following lemma demonstrates that a hash function, constructed from a program which is vulnerable to a direct attack, has a collision.

Lemma 15. *If a Linicrypt program $\mathcal{P} = (\mathbf{m}, \mathcal{C})$ with n constraints has a direct attack structure (c_1, \dots, c_n) then there exists an adversary \mathcal{A} making at most nb queries that success with probability of 1 in Game 14.*

Proof. To find a collision, \mathcal{A} finds $\mathbf{W}' \neq \mathbf{W}$ where $\mathcal{H}^{\mathcal{P}}(\mathbf{W}) = \mathcal{H}^{\mathcal{P}}(\mathbf{W}')$ and the initial value ℓ_0 is fixed.

First, \mathcal{A} picks an vector $\mathbf{x}_1 \in \mathbb{F}^k$ where $x_1^1 = \ell_0$ and the rest $k - 1$ elements are arbitrary then it runs $\mathcal{P}^E(\mathbf{x}_1)$ to get ℓ_1 then it puts $x_2^1 = \ell_1$ and picks the other $k - 1$ elements of \mathbf{x}_2 arbitrary from \mathbb{F} and it runs $\mathcal{P}^E(\mathbf{x}_2)$ to get ℓ_2 . It continues these iterations until it gets ℓ_b for an arbitrary number of blocks $b \in [1, n]$.

To find \mathbf{W}' , the adversary \mathcal{A} selects a $\mathbf{x}'_1 \in \mathbb{F}^k$ such that $\mathbf{x}'_1 \neq \mathbf{x}_1$ but $x_1^1 = \ell_0$, and executes $\mathcal{P}^E(\mathbf{x}'_1)$ to obtain ℓ'_1 . Then, the adversary sets $x_2^1 = \ell'_1$ and arbitrarily chooses the other $k - 1$ elements of \mathbf{x}'_2 from \mathbb{F} , running $\mathcal{P}^E(\mathbf{x}'_2)$ to obtain ℓ'_2 . This process continues until $\ell'_{b'-1}$ is obtained where $b' \leq b$. If any $\ell_i = \ell'_j$, then the adversary has found a collision and completed the attack. Otherwise, possessing $\ell'_{b'-1}$ and ℓ_b , and based on Lemma 13, the adversary can find a preimage $\mathbf{w}'_{b'}$ with probability 1.

If $\mathbf{w}_i = (x_i^2, \dots, x_i^k)$ and $\mathbf{w}'_j = (x_j^2, \dots, x_j^k)$ for $i \in [1, b]$ and $j \in [1, b']$, then $\mathcal{H}^{\mathcal{P}}(\mathbf{W}) = \mathcal{H}^{\mathcal{P}}(\mathbf{W}') = \ell_b$. Furthermore, since $\mathbf{x}'_1 \neq \mathbf{x}_1$, it follows that $\mathbf{W}' \neq \mathbf{W}$.

While we do not have a complete characterization of the group-2 PGV compression functions we note that none of them are degenerate, nor are they linearly-chained or possess a direct attack structure.

6 Conclusion and Future Work

We have demonstrated the utility of the Linicrypt framework beyond the random oracle model by giving characterizations of collision-resistance properties for Linicrypt programs in the ideal cipher model. We also show that in the case of the PGV compression function our characterization is equivalent to the notion of group-1 for the PGV functions. In addition, we delve into the characterization of group-2 Linicrypt programs by analyzing permutation and direct attacks and presenting a characterization of Linicrypt programs susceptible to these attacks.

There are several ways in which this work might be extended. First of all, in the rate-1 setting, we have not addressed the finer analysis provided by [4,5] which also characterizes group-2 functions and compares the pre-image resistance of group-1 and group-2 functions. Can we give a general notion of group-2 for arbitrary Linicrypt programs which generalizes the corresponding notion for rate-1 functions? We note that it is possible to give a characterization of *pre-image awareness*, a stronger notion of hash function security introduced by [10], for Linicrypt programs with random oracles, and it should be possible to extend this characterization to the ideal cipher model. It would also be interesting to consider even stronger properties for hash functions, such as *indifferentiability* [9]. A more ambitious goal would be to extend the analysis provided by Linicrypt beyond purely algebraic constructions. In particular, it would be very useful to consider using bit-string operations, especially truncation, and concatenation, which are used in many constructions such as the sponge construction which is the basis of Keccak/SHA-3. Here it might be useful to revisit [1], which considers equivalence properties of algebraic programs over $\mathbb{GF}(2^\lambda)$ which include bit-string operations but do not have access to random oracles, ideal ciphers or (as in the case of Keccak) random permutations.

References

1. Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In *LPAR 2010*, volume 6355 of *LNCS*, pages 46–63. Springer, 2010.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *CRYPTO 1996*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.
3. John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *FSE 2006*, volume 4047 of *LNCS*, pages 328–340. Springer, 2006.
4. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, 2002.
5. John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An analysis of the blockcipher-based hash functions from pgv. *Journal of Cryptology*, 23(4):519–545, 2010.

6. Jurjen N. Bos and David Chaum. Provably unforgeable signatures. In *CRYPTO 1992*, volume 740 of *LNCS*, pages 1–14. Springer, 1992.
7. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
8. Brent Carmer and Mike Rosulek. Linicrypt: A model for practical cryptography. In *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 416–445. Springer, 2016.
9. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.
10. Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for practical applications. *IACR Cryptol. ePrint Arch.*, page 177, 2009. Full version of [11].
11. Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for practical applications. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 371–388. Springer, 2009.
12. Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *J. Cryptol.*, 9(1):35–67, 1996.
13. Zahra Javar and Bruce M. Kapron. Linicrypt in the ideal cipher model. In Mingwu Zhang, Man Ho Au, and Yudi Zhang, editors, *Provable and Practical Security - 17th International Conference, ProvSec 2023, Wuhan, China, October 20-22, 2023, Proceedings*, volume 14217 of *Lecture Notes in Computer Science*, pages 91–111. Springer, 2023.
14. Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report CSL 98, SRI International, 1979.
15. Ian McQuoid, Trevor Swope, and Mike Rosulek. Characterizing collision and second-preimage resistance in linicrypt. In *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 451–470. Springer, 2019.
16. Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO 1987*, volume 293 of *LNCS*, pages 369–378. Springer, 1987.
17. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO 1993*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.
18. Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.
19. Martijn Stam. Blockcipher-based hashing revisited. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 67–83. Springer, 2009.