Linicrypt in the Ideal Cipher Model

Zahra Javar(✉) and Bruce M. Kapron(✉)

University of Victoria, Victoria, BC, Canada
{zahrajavar,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 Vandewall (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.

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 [6,12,13,15]. 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 [14] 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 [14] 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 [14], 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.

Work supported in part by NSERC Discovery Grant RGPIN-2021-02481.

M. Zhang et al. (Eds.): ProvSec 2023, LNCS 14217, pp. 91–111, 2023.
https://doi.org/10.1007/978-3-031-45513-1_6

A well-studied group of block-cipher-based compression functions was proposed by Preneel, Govaerts, and Vandewalle (PGV) [16]. 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 [16], Black, Rogaway, and Shrimpton [4] introduced 64 rate-1 compression functions $h : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ utilizing a single call to a block-cipher $E : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ 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 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 [18] presents these properties in an algebraic setting, partly anticipating the approach presented in the current paper. In Sect. 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 [14] 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.)

While our approach largely follows that of [14], 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.

2 Preliminaries

In our presentation, we largely follow the approach of [14]. Programs are defined over a finite field $\mathbb{F}$, elements of which are denoted by lower-case non-bold letters[1], vectors over $\mathbb{F}$ by lowercase bold letters and matrices over $\mathbb{F}$ by uppercase

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

bold letters. We write $\boldsymbol{a} \cdot \boldsymbol{b}$ or $\boldsymbol{ab}$ for the inner product and $\boldsymbol{M} \times \boldsymbol{a}$ or $\boldsymbol{Ma}$ for the matrix-vector product. Note that we will sometimes think of matrices as a column vector of row vectors (so we may write $\boldsymbol{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k)^\top$.)

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 [17]. 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 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 [14] the framework is applied to characterize collision-resistance properties of hash functions.

A Linicrypt program is a straight-line program over a fixed vector $(v_1, \ldots, v_m)$ 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 either[2]

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 $\mathcal{P}^H : \mathbb{F}^k \to \mathbb{F}^r$ for some k, r which on input vector $\boldsymbol{x} = (x_1, \cdots, x_k)$ returns output vector $\boldsymbol{\ell} = (l_1, \ldots, l_r)$. 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

[2] The Linicrypt model, introduced in [8], also allows the assignment of a random field element to a variable. Here, as in [14], we consider only deterministic programs.

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 [8,14], Linicrypt programs can be given a purely *algebraic* representation. We will give a slightly modified (but equivalent) version of this as presented in [14]. 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* $\boldsymbol{v}_{base} = (v_1, \ldots, v_{k+n})^\top$, 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 $v_i = \mathbf{e}_i \boldsymbol{v}_{base}$ where $\mathbf{e}_i$ denotes the ith 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* $\boldsymbol{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_r)^\top$, where $\boldsymbol{m}_i \in \mathbb{F}^{k+n}$ and $\boldsymbol{M}\boldsymbol{v}_{base} = (l_1, \cdots, l_r)$ is the vector of program outputs. Finally, each oracle call $v_i = H(t, v_{i_1}, \cdots, v_{i_m})$ where t is a *nonce* and $v_i \in \mathbb{F}$, is represented by an *oracle constraint* $c = (t, \boldsymbol{Q}, \boldsymbol{a})$, indicating that when H is called on input $(t, \boldsymbol{Q} \times \boldsymbol{v}_{base}) = (t, v_{i_1}, \cdots, v_{i_m})$, the returned value is $\boldsymbol{a} \cdot \boldsymbol{v}_{base} = v_i$.

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 [8,14], 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* $(\boldsymbol{M}, \mathcal{C})$. In particular, the characterization of collision-resistance properties is completely determined by algebraic properties of $(\boldsymbol{M}, \mathcal{C})$.

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

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

In the algebraic presentation, the program is specified by:

$$
\begin{aligned}
\boldsymbol{v}_{base} &= (v_1, v_2, v_3, v_4)^\top \\
\boldsymbol{M} &= \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \\
\mathcal{C} &= \langle (t_1, \boldsymbol{q}_1, \boldsymbol{a}_1), (t_2, \boldsymbol{q}_2, \boldsymbol{a}_2) \rangle
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{q}_1 &= (1, 0, 0, 0) \quad \boldsymbol{q}_2 = (0, 1, 0, 0) \\
\boldsymbol{a}_1 &= (0, 0, 1, 0) \quad \boldsymbol{a}_2 = (0, 0, 0, 1)
\end{aligned}
$$

In moving to programs in the ideal cipher model, for assignments of the first type, we now have calls to an ideal cipher $E(.,.)$ 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 $\boldsymbol{a}_i \cdot \boldsymbol{v}_{base} = E(\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}_{base}, \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}_{base})$ and can be represented by an *oracle constraint* $c = (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ where $\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a} \in \mathbb{F}^{k+n}$. To simplify the presentation we define $K_i := \boldsymbol{q}_{K_i} \cdot \boldsymbol{v}_{base}$, $X_i := \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}_{base}$ and $Y_i := \boldsymbol{a}_i \cdot \boldsymbol{v}_{base}$. Letting $\mathcal{C}$ denote the set of oracle constraints and $\boldsymbol{M}$ the output matrix, we again have an algebraic representation $(\boldsymbol{M}, \mathcal{C})$. The following is a simple example of such a program:

$$\begin{array}{l} \underline{\mathcal{P}^E(v_1, v_2):} \\ \quad v_4 := E(v_1, v_2) \\ \quad \mathit{return}\ v_4 + v_2 \end{array}$$

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

Further examples are given in the Appendices. We also refer the reader to [8,14] for a more detailed introduction to Linicrypt.

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 [18], 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 $v_1, \ldots, v_{k+n}$ and inputs $v_1, \ldots, v_k$ the modified program with a constant c has base variables $v_1, \ldots, v_{k+n+1}$ where $v_{k+1} := c$ and $\boldsymbol{M}$ and $\mathcal{C}$ are updated appropriately. Finally, the single row $\mathbf{e}_{k+1}$ is appended to $\boldsymbol{M}$ (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 ([14] **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:*

$$(\boldsymbol{x}, \boldsymbol{x}') \leftarrow \mathcal{A}^{E,E^{-1}}(\lambda);\ \textit{return}\ (\boldsymbol{x} \neq \boldsymbol{x}')\ \textit{and}\ \mathcal{P}^E(\boldsymbol{x}) = \mathcal{P}^E(\boldsymbol{x}') \tag{1}$$

Definition 2 ([14] **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:*

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

3 Characterizing Collision Resistance

In this section, we give an algebraic condition to characterize collision resistance and 2nd-preimage resistance for Linicrypt programs in the ideal cipher model (Definition 4). Before giving the definition we note some programs fail trivially to be collision-resistant because two different inputs produce exactly the same queries to the oracle. This is formalized in the following:

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

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

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 $\boldsymbol{x}$, and it determines the base vector $\boldsymbol{v}_{base}$ in the execution of $\mathcal{P}(\boldsymbol{x})$. We define the matrix $\boldsymbol{P} = \begin{bmatrix} \boldsymbol{Q}_K \\ \boldsymbol{Q}_X \\ \boldsymbol{A} \\ \boldsymbol{M} \end{bmatrix}$ where $\boldsymbol{Q}_K, \boldsymbol{Q}_X, \boldsymbol{A}$ are matrices whose rows correspond to the components of the elements of $\mathcal{C}$ (ordered arbitrarily.) If the adversary can determine a 2nd-preimage $\boldsymbol{x}' \neq \boldsymbol{x}$ where $\boldsymbol{P}\boldsymbol{v}_{base} = \boldsymbol{P}\boldsymbol{v}'_{base}$ where $\boldsymbol{v}'_{base}$ is the base vector in the calculation of $\mathcal{P}(\boldsymbol{x}')$ then $\mathcal{P}(\boldsymbol{x}) = \mathcal{P}(\boldsymbol{x}')$ and $\mathcal{A}$ wins. Since program $\mathcal{P}$ is degenerate, the rows of $\boldsymbol{P}$ cannot span all $k + n$ basis vectors which means $\mathrm{rank}(\boldsymbol{P}) < k + n$, and thus, for some $\boldsymbol{v} \neq \boldsymbol{0}$, $\boldsymbol{P}\boldsymbol{v} = \boldsymbol{0}$. The adversary can solve for this $\boldsymbol{v}$ and set $\boldsymbol{v}'_{base} = \boldsymbol{v}_{base} + \boldsymbol{v}$. Then $\boldsymbol{P}(\boldsymbol{v}'_{base} - \boldsymbol{v}_{base}) = \boldsymbol{0}$, so $\boldsymbol{v}'_{base} \neq \boldsymbol{v}_{base}$ and $\boldsymbol{P}\boldsymbol{v}'_{base} = \boldsymbol{P}\boldsymbol{v}_{base}$. In particular, this allows $\mathcal{A}$ to compute $\boldsymbol{x}' \neq \boldsymbol{x}$ such that $\mathcal{P}(\boldsymbol{x}') = \mathcal{P}(\boldsymbol{x})$. □

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 $\boldsymbol{x} \neq \boldsymbol{x}'$, 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 $\boldsymbol{x}'$, 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 $\mathcal{P} = (\boldsymbol{M}, \mathcal{C})$ be a Linicrypt program. A collision structure for $\mathcal{P}$ is a tuple $(i^*, c_1, \ldots, c_n)$ where $c_1, \cdots, c_n$ 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) $\boldsymbol{q}_{K_i} \notin \mathsf{span}(\{\boldsymbol{q}_{K_1}, \ldots, \boldsymbol{q}_{K_{i-1}}\}, \{\boldsymbol{q}_{X_1}, \ldots \boldsymbol{q}_{X_{i-1}}\}, \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}, \mathsf{rows}(\boldsymbol{M}))$
(C2) $\boldsymbol{q}_{X_i} \notin \mathsf{span}(\{\boldsymbol{q}_{K_1}, \ldots, \boldsymbol{q}_{K_i}\}, \{\boldsymbol{q}_{X_1}, \ldots \boldsymbol{q}_{X_{i-1}}\}, \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_i\}, \mathsf{rows}(\boldsymbol{M}))$
(C3) $\boldsymbol{a}_i \notin \mathsf{span}(\{\boldsymbol{q}_{K_1}, \ldots, \boldsymbol{q}_{K_i}\}, \{\boldsymbol{q}_{X_1}, \ldots \boldsymbol{q}_{X_i}\}, \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}, \mathsf{rows}(\boldsymbol{M}))$

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

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

- add constraint $\boldsymbol{M}\boldsymbol{v}' = \boldsymbol{M}\boldsymbol{v}$
- for $i < i^*$, add constraints $\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}' = \boldsymbol{q}_{K_i} \cdot \boldsymbol{v}$, $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}' = \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}$ and $\boldsymbol{a}_i \cdot \boldsymbol{v}' = \boldsymbol{a}_i \cdot \boldsymbol{v}$
- For $i \geq i^*$,
  - if (C1) holds, choose $K_i' \in \mathbb{F}$ so that $K_i' \neq \boldsymbol{q}_{K_i} \cdot \boldsymbol{v}$ and add the constraint $\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}' = K_i'$
  - if (C2) holds, set $X_i' := E^{-1}(\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}', \boldsymbol{a}_i \cdot \boldsymbol{v}')$ and add the constraint $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}' = X_i'$
  - if (C3) holds, set $Y_i' := E(\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}', \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}')$ and add the constraint $\boldsymbol{a}_i \cdot \boldsymbol{v}' = Y_i'$
  - if (C2) and (C3) both hold, choose $X_i' \in \mathbb{F}$ such that $X_i' \neq \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}$, set $Y_i := E(\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}', X_i')$ and add the constraints $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}' = X_i'$ and $\boldsymbol{a}_i \cdot \boldsymbol{v}' = Y_i'$

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

To see that $\boldsymbol{v}' \neq \boldsymbol{v}$, 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 $\boldsymbol{v}' \neq \boldsymbol{v}$.

The constraints that are added for the output matrix and for $i < i^*$ are consistent, as they already have a solution, namely $\boldsymbol{v}$. For $i \geq i^*$, a new constraint is added only in the case that the corresponding $\boldsymbol{q}_{K_i}, \boldsymbol{q}_{X_i}$ or $\boldsymbol{a}_i$ 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 $\boldsymbol{v}'$ is unique.

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

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 $(i^*, c_1, \ldots, c_n)$.*

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 $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{P}(\boldsymbol{x}')$, $\mathcal{A}$ makes either the query or its corresponding inverse query before returning. For Game 1, this is achieved by having $\mathcal{A}$ run $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{P}(\boldsymbol{x}')$ 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 $\mathcal{P}(\boldsymbol{x})$ and also running $\mathcal{P}(\boldsymbol{x}')$ before returning, and making any corresponding query, subject to restriction (1), before returning.
3. $\mathcal{A}$ actually returns $\boldsymbol{v}, \boldsymbol{v}'$ which are the settings of base variables determined by the execution of $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{P}(\boldsymbol{x}')$, respectively.

The assumptions imply that for an oracle constraint $c = (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ occurring in $\mathcal{P}$, $\mathcal{A}$ determines the value of triples $(\boldsymbol{q}_K \cdot \boldsymbol{v}, \boldsymbol{q}_X \cdot \boldsymbol{v}, \boldsymbol{a} \cdot \boldsymbol{v})$ 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 $T, T' : \mathcal{C} \to [N]$ 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 $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{P}(\boldsymbol{x}')$, and determine the triple $(\boldsymbol{q}_K \cdot \boldsymbol{v}, \boldsymbol{q}_X \cdot \boldsymbol{v}, \boldsymbol{a} \cdot \boldsymbol{v})$ and $(\boldsymbol{q}_K \cdot \boldsymbol{v}', \boldsymbol{q}_X \cdot \boldsymbol{v}', \boldsymbol{a} \cdot \boldsymbol{v}')$ 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 [14], a query $c \in \mathcal{C}$ is *convergent* if $T(c) = T'(c)$, and *divergent* otherwise. Because $\boldsymbol{x} \neq \boldsymbol{x}'$ is a collision and $\mathcal{P}$ is nondegenerate there is at least one divergent constraint. Define $\texttt{finish}(c) = \max\{T(c), T'(c)\}$. Note in contrast to [14], here we do not have unique nonces, so two different constraints can be mapped to the same adversary query, thus `finish` is not an injective function. However, we will show that there is an ordering of $\mathcal{C}$ as $(c_1, \ldots, c_n)$ where the convergent constraints come first, in any order, followed by divergent constraints in some non-decreasing order, and letting i^* denote the index of the first divergent constraint, we claim that $(i^*, c_1, \ldots, c_n)$ is a collision structure for $\mathcal{P}$.

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

For $i = i^*$ the query c_i is divergent thus at least one of the following inequalities holds $\boldsymbol{q}_{K_i} \cdot \boldsymbol{v}' \neq \boldsymbol{q}_{K_i} \cdot \boldsymbol{v}$, $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}' \neq \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}$ or $\boldsymbol{a}_i \cdot \boldsymbol{v}' \neq \boldsymbol{a}_i \cdot \boldsymbol{v}$. If $\boldsymbol{a}_i \cdot \boldsymbol{v}' \neq \boldsymbol{a}_i \cdot \boldsymbol{v}$ or $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}' \neq \boldsymbol{q}_{X_i} \cdot \boldsymbol{v}$ 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

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

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

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

   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 $\boldsymbol{q}_{X_i} \cdot \boldsymbol{v}'$ gives us

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

   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 $(\boldsymbol{v}' - \boldsymbol{v})$ and applying $X_i = X_i'$ and $\boldsymbol{M}(\boldsymbol{v} - \boldsymbol{v}') = 0$ gives

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

   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 $Y_i = Y_i'$.

   This is similar to the preceding case. See Appendix A for details.

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 Eq. 3. Applying the assumption $\boldsymbol{M}(\boldsymbol{v} - \boldsymbol{v}') = 0$ and canceling all the queries before i^* gives,

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

   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:

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

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

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

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

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

Now, if the adversary is making query $T'(c_i)$ as an encryption query then in Eq. 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, Eq. 4 will give the same contradiction. So at least one of the conditions (C2) or (C3) has to hold.

If there are multiple constraints with the same finish, say $\texttt{finish}(c_{i-k}) = \cdots = \texttt{finish}(c_i)$, we want to show at least one ordering of these constraints is a collision structure. Without loss of the generality suppose for these k constraints $\texttt{finish}(c_j) = T'(c_j)$ and this query is an encryption query. Consider the ordering $(i^*, c_1, \ldots, c_{i-k}, \ldots, c_i, \ldots, c_n)$. If this is not a collision structure then there is a

constraint c_s where $i-k \leq s \leq i$ satisfies Eq. 5. If we call all the fixed terms on the right-hand side of this equation f then we can rewrite the equation as

$$\boldsymbol{a}_s \cdot \boldsymbol{v}' = f + \sum_{i-k \leq j \leq s-1} \gamma_j \boldsymbol{a}_j \cdot \boldsymbol{v}'$$

Applying the fact that all $\boldsymbol{a}_j \cdot v'$ in the above equation are equal to $\boldsymbol{a}_s \cdot v'$ we get

$$(1 - \sum_{i-k \leq j \leq s-1} \gamma_j) \boldsymbol{a}_s \cdot \boldsymbol{v}' = f$$

Now in the above equation, all the terms on the right-hand side are fixed and the left-side value is random, so the advantage of the adversary is at most $1/|\mathbb{F}|$. □

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 $\mathbb{F}_\lambda$, 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 $(i^*, c_1, \ldots, c_n)$, this is a collision structure. Note that after the second loop ends,

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

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 $\{c_1, \ldots, c_{i-1}\}$ which means

$$V = \{\boldsymbol{q}_{K_1}, \ldots, \boldsymbol{q}_{K_i}\} \cup \{\boldsymbol{q}_{X_1}, \ldots, \boldsymbol{q}_{X_i}\} \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_i\} \cup \mathsf{rows}(\boldsymbol{M}),$$

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

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

LEFT $\coloneqq \mathcal{C}$
RIGHT $\coloneqq$ empty stack
$V \coloneqq \{\boldsymbol{q}_K | (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a}) \in \mathcal{C}\} \cup \{\boldsymbol{q}_X | (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a}) \in \mathcal{C}\} \cup \{\boldsymbol{a} | (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a}) \in \mathcal{C}\} \cup \mathsf{rows}(M)$

while $\exists (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a}) \in$ LEFT where $\boldsymbol{q}_X \notin \mathsf{span}(V \setminus \{\boldsymbol{q}_X\})$ or $\boldsymbol{a} \notin \mathsf{span}(V \setminus \{\boldsymbol{a}\})$ **do**
  remove $(\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ from LEFT
  push $(\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ to RIGHT
  reduce multiplicity of $\boldsymbol{q}_K, \boldsymbol{q}_X$ and $\boldsymbol{a}$ in V by 1
end while

while $\exists (\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a}) \in$ RIGHT where
  $(\boldsymbol{q}_K \in \mathsf{span}(V) \wedge \boldsymbol{q}_X \in \mathsf{span}(V \cup \boldsymbol{q}_K \cup \boldsymbol{a}))$ or
  $(\boldsymbol{q}_K \in \mathsf{span}(V) \wedge \boldsymbol{a} \in \mathsf{span}(V \cup \boldsymbol{q}_K \cup \boldsymbol{q}_X))$ or
  $(\boldsymbol{q}_X \in \mathsf{span}(V \cup \boldsymbol{q}_K \cup \boldsymbol{a}) \wedge \boldsymbol{a} \in \mathsf{span}(V \cup \boldsymbol{q}_K \cup \boldsymbol{q}_X))$ **do**
  remove $(\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ from RIGHT
  add $(\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{a})$ to LEFT
  increase multiplicity of $\boldsymbol{q}_K, \boldsymbol{q}_X$ and $\boldsymbol{a}$ in V by 1
end while

if RIGHT is nonempty **then**
  $i^* \coloneqq |\text{LEFT}| + 1$
  let LEFT $= (c_1, \ldots, c_{i^*-1})$ where order doesn't matter
  let RIGHT $= (c_{i^*}, \ldots, c_n)$ in reverse order of insertion
  return $(i^*, c_1, \ldots, c_n)$
else return $\perp$
end if

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

or

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

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

$$\boldsymbol{q}_{K_i^*} = \sum_{j \in S} \kappa_j \boldsymbol{q}_{K_j} + \sum_{j \in S} \lambda_j \boldsymbol{q}_{X_j} + \sum_{j \in S} \mu_j \boldsymbol{a}_j + \boldsymbol{\nu} \boldsymbol{M} \tag{6}$$

$$\boldsymbol{q}_{X_i^*} = \sum_{j \in S \cup \{i^*\}} \pi_j \boldsymbol{q}_{K_j} + \sum_{j \in S} \rho_j \boldsymbol{q}_{X_j} + \sum_{j \in S \cup \{i^*\}} \varsigma_j \boldsymbol{a}_j + \boldsymbol{\tau} \boldsymbol{M} \tag{7}$$

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

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

$$\boldsymbol{q}_{K_i^*} = \sum_{j \in S \setminus S_1} \kappa'_j \boldsymbol{q}_{K_j} + \sum_{j \in S \setminus S_2} \lambda'_j \boldsymbol{q}_{X_j} + \sum_{j \in S \setminus S_3} \mu'_j \boldsymbol{a}_j + \boldsymbol{\nu'} \boldsymbol{M} \tag{9}$$

$$\boldsymbol{q}_{X_i^*} = \sum_{j \in S \setminus S_1} \pi'_j \boldsymbol{q}_{K_j} + \sum_{j \in S \setminus S_2} \rho'_j \boldsymbol{q}_{X_j} + \sum_{j \in S \cup \{i^*\} \setminus S_3} \varsigma'_j \boldsymbol{a}_j + \boldsymbol{\rho'} \boldsymbol{M} \tag{10}$$

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

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 Eqs. 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 $\boldsymbol{q}_{K_{j_1}}$ was not in the span of V but both $\boldsymbol{q}_{X_{j_1}}$ and $\boldsymbol{a}_{j_1}$ where in the span of constraints with smaller indices and rows$(\boldsymbol{M})$ 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 $\boldsymbol{a}_{j_3}$ was not in the span of LEFT and rows$(\boldsymbol{M})$ so the other two vectors $\boldsymbol{q}_{K_{j_3}}$ and $\boldsymbol{q}_{X_{j_3}}$ had to be in the span of LEFT and rows$(\boldsymbol{M})$ 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, $\boldsymbol{a}_{j_3}$ had to be independent of previous constraints and the output (condition (C3)), but we can rewrite Eq. 9 as follow,

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

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

4 Rate-1 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 [4,16] gives a definition of 64 possible rate-1 compression functions mapping $D \times D \to D$, 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 Linicrypt. 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 [18]. 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).

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

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:*

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

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

Lemma 5 ([5] Lemma 3). *Suppose $f : \mathbb{GF}(2^\lambda) \times \mathbb{GF}(2^\lambda) \to \mathbb{GF}(2^\lambda)$ 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 Linicrypt framework, a PGV compression function f is specified via

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

Here use $\boldsymbol{m}_2$ to capture the use of a constant value in the Linicrypt setting, as described in Sect. 2.2. Up to our convention regarding the constant value c, f'' corresponds to $\boldsymbol{m}_1$ while f' corresponds to $\begin{bmatrix} \boldsymbol{q}_K \\ \boldsymbol{q}_X \end{bmatrix}$. In particular, writing $\boldsymbol{q}_i = (q_i^1, q_i^2, q_i^3, q_i^4)$, $i \in \{K, X\}$, we have $f'(h,m) = \begin{bmatrix} \boldsymbol{q}'_K \\ \boldsymbol{q}'_X \end{bmatrix} \times (h, m, c)$, where $\boldsymbol{q}'_i = (q_i^1, q_i^2, q_i^3)$, $i \in \{K, X\}$. We also have $f''(h,m,y) = \boldsymbol{m}_1 \cdot (h, m, c, y)$.

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 $\boldsymbol{m}_1$,$\boldsymbol{m}_2$,$\boldsymbol{q}_K$,$\boldsymbol{q}_X$,$\boldsymbol{a}$, as described above, with respect to a fixed constant c. Define the following matrices

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

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

Proof. For fixed h, m, $f'(h,m,y) = \boldsymbol{m}_1 \cdot (h, m, c, y) = m_1^4 y \oplus s$, 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 $\boldsymbol{R}$ is nonsingular.*

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

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

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

Together, the preceding Lemmas give the following

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

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

(C1) $\boldsymbol{q}_K \notin \mathsf{span}(\{\boldsymbol{m}_1, \boldsymbol{m}_2\})$
(C2) $\boldsymbol{q}_X \notin \mathsf{span}(\{\boldsymbol{q}_K, \boldsymbol{a}, \boldsymbol{m}_1, \boldsymbol{m}_2\})$
(C3) $\boldsymbol{a} \notin \mathsf{span}(\{\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{m}_1, \boldsymbol{m}_2\})$

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

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

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

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

$$\boldsymbol{q}_X \in \mathsf{span}(\mathsf{rows}(\boldsymbol{S})) = \mathsf{span}(\{\boldsymbol{q}_K, \boldsymbol{m}_1, \boldsymbol{m}_2, \boldsymbol{a}\}) \qquad (*)$$

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

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

so that (C3) also fails. □

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

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

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

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

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

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

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

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

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

$$\mathsf{span}(\{\boldsymbol{q}_K, \boldsymbol{q}_X, \boldsymbol{m}_1, \boldsymbol{m}_2\}) = \mathsf{span}(\{\boldsymbol{q}_X, \boldsymbol{m}_1, \boldsymbol{m}_2\}),$$

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

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

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.*

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 Linicrypt. 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 Linicrypt setting to future work. We give some examples of definitions of PGV compression functions from [4] and their analysis on the same GitHub page.

5 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

[3] A sample implementation in Octave is available at https://github.com/zahrajavar/PGVCollisionResistantCompressionFunctions.git.

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 Linicrypt framework is not limited to the random oracle model.

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.

There are a number of 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.

A Missing Proofs

Lemma 2 Case 3: 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

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

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

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

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.

B 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.

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

$$\boldsymbol{q}_{K_1} = (1,0,0,0) \quad \boldsymbol{q}_{X_1} = (0,1,0,0) \quad \boldsymbol{a}_1 = (0,0,1,0)$$
$$\boldsymbol{q}_{K_2} = (1,0,0,0) \quad \boldsymbol{q}_{X_2} = (0,0,1,0) \quad \boldsymbol{a}_2 = (0,0,0,1)$$
$$\boldsymbol{m} = (1,0,0,1)$$

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

- The adversary $\mathcal{A}$ picks an arbitrary input $\boldsymbol{x} = (x_1, x_2) \in \mathbb{F}^2$ and runs the program on $\boldsymbol{x}$ and gets the output l.
- $\mathcal{A}$ picks arbitrary $x_1' \neq x_1 \in \mathbb{F}$ and makes the query $v_3' = E^{-1}(x_1', l + x_1')$.
- $\mathcal{A}$ makes the query $x_2' = E^{-1}(x_1', v_3')$ to find x_2'.
- $\mathcal{A}$ returns $\boldsymbol{x}' = (x_1', x_2')$.

This attack was possible because of the following,

- x_1' was independent of the output in other words $\boldsymbol{q}_{K_1} \neq \boldsymbol{m}$.
- In step two after fixing x_1' and the output l the value of v_3' was not fixed which in algebraic representation means $\boldsymbol{q}_{X_1} \notin \mathsf{span}(\boldsymbol{q}_{K_1}, \boldsymbol{m})$ so the adversary could determine this value by making a decryption query.
- In the third step because $\boldsymbol{q}_{X_2} \notin \mathsf{span}(\boldsymbol{q}_{K_1}, \boldsymbol{q}_{K_2}, \boldsymbol{q}_{X_1}, \boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{m})$ the value of x_2' is not fixed so via a decryption query the adversary finds a compatible value for x_2'.

References

1. Barthe, G., Daubignard, M., Kapron, B., Lakhnech, Y., Laporte, V.: On the equality of probabilistic terms. In: Clarke, E.M., Voronkov, A. (eds.) LPAR 2010. LNCS (LNAI), vol. 6355, pp. 46–63. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17511-4_4
2. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_1
3. Black, J.: The ideal-cipher model, revisited: an uninstantiable blockcipher-based hash function. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 328–340. Springer, Heidelberg (2006). https://doi.org/10.1007/11799313_21
4. Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_21
5. Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An analysis of the Blockcipher-based hash functions from PGV. J. Cryptol. **23**(4), 519–545 (2010)
6. Bos, J.N.E., Chaum, D.: Provably unforgeable signatures. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 1–14. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_1
7. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4), 557–594 (2004)
8. Carmer, B., Rosulek, M.: Linicrypt: a model for practical cryptography. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 416–445. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_15
9. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: how to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_26
10. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgard for practical applications. In: IACR Cryptol. ePrint Arch., p. 177 (2009) Full version of [11].
11. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for Practical Applications. In: Joux, A., et al. (eds.) Advances in Cryptology - EUROCRYPT 2009, pp. 371–388. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_22
12. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital signatures. J. Cryptol. **9**(1), 35–67 (1996). https://doi.org/10.1007/BF02254791
13. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report CSL 98, SRI International (1979)
14. McQuoid, I., Swope, T., Rosulek, M.: Characterizing collision and second-preimage resistance in Linicrypt. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 451–470. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_18
15. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
16. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_31

17. Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)
18. Stam, M.: Blockcipher-Based Hashing Revisited. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 67–83. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03317-9_5