Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling the Ideal Cipher in Linicrypt

Master Thesis

Frederik Semmel

September 6, 2022

Advisors: Fabio Banfi, Ueli Maurer

Institute of Theoretical Computer Science, ETH Zürich

Abstract

Linicrypt is a proof model used to analyze programs that only make calls to a random oracle and perform linear operations in a field $\mathbb{F}$. We introduce new abstractions which allow us to extend Linicrypt to work with the ideal ciphers instead of random oracles. Specifically, given Lincrypt oracle constraints $\mathcal{C}$ of dimension base, we define the set of solutions to $\mathcal{C}$ as a subspace of $\mathbb{F}^{\mathsf{base}}$. A Linicrypt program can then be interpreted as a method for finding a solution to $\mathcal{C}$ while fulfilling a given linear constraint. This allows us to characterize a weakness with regard to collision resistance simultaneously for Linicrypt programs in the ideal cipher model and in the random oracle model. If a program has this weakness, which we call a collision structure, one can transform the program into its attack by performing a basis change on the algebraic representation and reversing the roles of the input and output. The characterization is sound and complete in the case of Linicrypt programs making only a single query. We apply these concepts in the derivation of an attack taxonomy regarding the Merkle-Damgård construction for the 64 compression functions introduced by Preneel, Govaerts, and Vandewalle (Crypto 1993).

Contents

Chapter 1

Introduction

The theme of this thesis is to apply a new proof model to well-known and trusted cryptographic constructions. The goal is twofold. On one hand, we hope to increase our understanding of the known cryptographic constructions. On the other hand, we aim to improve and extend the proof model.

1.1 Background and Motivation

It has been well studied, that hash functions can be constructed securely from block ciphers. Since the success of very strong block ciphers such as AES, the trust in the security of modern block cipher constructions is very high. Therefore, it makes sense to use a block cipher that is believed to be secure as a building block for other constructions, and express their security in terms of the security of the block cipher. The ideal cipher model, which was introduced by Shannon in [Sha49], is a useful technique in this context. In this model, we assume that the block cipher is a perfect PRP, and we call it an ideal cipher. This enables a proof of security for a cryptographic construction in absolute terms, depending only on the number of queries an adversary makes to the ideal cipher. The number of queries acts as a proxy for the time and resources needed to break the construction, as we assume that the execution of the ideal cipher program has a fixed cost. In a seminal paper from 1993 [PGV94], the authors Preneel, Govaerts, and Vandewalle analyze the 64 most basic constructions of a compression function from a block cipher. 12 of them were found to lead to secure hash functions in the Merkle-Damgård construction, including e.g. the Davies-Meyer construction. This construction is used, for example, in the popular hash function family SHA-2. A follow-up paper by Black, Rogaway, and Shrimpton [BRS02] in 2002 confirms these results by providing tight security proofs in the ideal cipher model. In addition, they find that 8 insecure compression functions of the 64 PGV constructions still lead to collision resistant Merkle-Damgård hash function constructions.

The PGV compression functions as well as the Merkle-Damgård construction itself consist of linear operations in a field, mixed with calls to a primitive. They are just two of many examples of this type of cryptographic constructions. The Linicrypt model was introduced by Carmer and Rosulek [CR16] with the aim of analyzing such constructions at a higher level of abstraction. The central idea of this proof model is the abstraction of a cryptographic construction from its program representation to an algebraic representation.

The Linicrypt model has several benefits. Firstly, attacks and security proofs tend to work for a big class of constructions. It is easier to create a characterization for a given security definition at the level of the algebraic representation than at the level of the program description. Therefore, one can write proofs that work for all constructions that fulfill this characterization. Secondly, characterizations can often be checked efficiently by a machine. For example, in [CR16] two inputless Linicrypt programs were shown to be indistinguishable if and only if their algebraic representation are equivalent under a relation they define. Another example Theorem 1 from [HRR22]. The authors describe a sufficient condition for a program to be IND$-CPA secure in terms of ts the algebraic representation. Both these characterizations can be computed in polynomial time.

1.2 Our Contribution

This thesis focuses on the results in [MSR19] by McQuoid, Swope, and Rosulek. They find a complete and sound characterization of collision resistance for a Linicrypt program if the program uses only distinct nonces in its calls to a random oracle. It turns out that collision resistance is equivalent to second-preimage resistance. We replace the random oracle with an ideal cipher and adapt the algebraic representation accordingly. Following the philosophy of the Linicrypt model, we try to unify the proofs for both primitives.

In order to do so, we make a set of definitions that differ slightly in the case of the random oracle and the ideal cipher. Some basic lemmas have to be proven separately for both cases, but their statements are identical. Our characterization of the existence of a second-preimage attack is then expressed in terms of these lemmas. This way, the characterization and proof for the attack work for both primitives at the same time. If a program fulfills our condition we say that it has a collision structure, copying [MSR19]. But our more general definition of collision structure includes the case of degeneracy as an edge case, which they had to treat separately. Unfortunately, the security proof from [MSR19] cannot be used directly in the ideal cipher model, because a block cipher does not take a nonce as input. Therefore, we restrict ourselves to single query constructions for our security proof. The existence of a collision structure is then a complete and sound characterization of collision resistance and second-preimage resistance for single query constructions.

We then apply this ideal cipher based Linicrypt model to the 64 compression functions from [PGV94]. Using the new concepts we can derive an attack taxonomy for the Merkle-Damgård constructions based on these compression functions. Such a derivation of the types of attacks was not given in the aforementioned works. Our categorization for second-preimage attacks agrees mostly with the categories developed by [PGV94] and [BRS02] for collision attacks. However, we consider ours to be more fine-grained. We could not find a second-preimage attack for 13 schemes. 8 of them have been proven to be collision resistant by [BRS02], hence also second-preimage resistant. The remaining 5 are not collision resistant due to very specific collisions that occur with inputs of different lengths. We suspect, however, that they are second-preimage resistant.

The high degree of geometric symmetry in the derivation, the 5 remaining schemes, and the schemes vulnerable to an input permutation attack point to possible future research on Linicrypt. In particular, a theorem about collision resistance and second-preimage resistance for general Linicrypt programs using repeated nonces should fill these gaps in the derivation. Towards this goal, we identify two weaknesses a Linicrypt program with repeated nonces can have in addition to a collision structure. The problem of collapsing queries has been briefly described in [MSR19]. Adding to this, we describe a permutation attack that can be carried out when the queries made by the program and its output follow a certain symmetry. This includes the before-mentioned input permutation attack but is not limited to it.

Chapter 2

Preliminaries

2.1 Linicrypt

2.1.1 Definition of a Linicrypt program

The Linicrypt model for cryptographic constructions was introduced by Carmer & Rosulek in [CR16]. Summarizing the formalization from that paper, a pure Linicrypt program $\mathcal{P}$ is a straight line program whose intermediate variables are elements in a field $\mathbb{F}$. The only operations allowed to create an intermediate variable are:

- Retrieve an input, which is in $\mathbb{F}$
- Perform a linear combination of existing internal variables with fixed parameters
- Call a random oracle $H : \{0,1\}^* \times \mathbb{F}^* \to \mathbb{F}$
- Sample from $\mathbb{F}$ uniformly

Finally, the program $\mathcal{P}$ is allowed to output one or more of its variables. One can formalize a Linicrypt program as a sequence of commands, where each command creates a new itermediate variable. The input size and the specification of which variables form the output are also part of the formal description in [CR16].

Below is an example of a Linicrypt program $\mathcal{P}^H$, written in conventional pseudocode on the left and in explicit Linicrypt on the right.

$\mathcal{P}^H(x,y)$		$\mathcal{P}^H(x,y)$
$r \leftarrow\$ \mathbb{F}$	$\rightsquigarrow$	$v_1 := x$
return $H(x+r)+y$		$v_2 := y$
		$v_3 \leftarrow\$ \mathbb{F}$
		$v_4 := v_1 + v_3$
		$v_5 := H(v_4)$
		$v_6 := v_5 + v_2$
		return (v_6)

We will usually work with programs in conventional pseudocode and use suggestive names for the intermediate variables instead of v_i. Nevertheless, one should keep in mind, that such programs could be formalized as a sequence of Linicrypt commands, each generating a new intermediate variable. The superscript is used to highlight that $\mathcal{P}$ has access to a random oracle H. As this is often clear from the context, this superscript will usually be dropped. When a Linicrypt program contains no sampling operations it is called a deterministic Linicrypt program. In the context of collision resistance, this is the case we care about the most.

2.1.2 Type of Adversaries

The Linicrypt model only imposes computational restrictions on the cryptographic constructions, not on the adversaries. We consider computationally unbounded adversaries $\mathcal{A}$, which have bounded access to the random oracle H. Therefore, the behavior of an adversary is described in terms of the number of queries it makes. The additional power granted to an adversary by allowing unbounded computations is usually not helpful in this model. For example, we will show in the collision resistance case: A successful attack is either not possible for information-theoretical reasons, or it can be carried out by another Linicrypt program.

2.1.3 Algebraic Representation

One of the advantages of restricting the computational model is that one can characterize Linicrypt programs with an algebraic representation. We will introduce the concept of the algebraic representation as it was developed in previous Linicrypt papers. Some definitions, in particular the definition of an oracle constraint, will be generalized in the next chapters. Let $\mathcal{P}$ be a Linicrypt program with intermediate variables $v_1, \dots, v_n$. These are sorted in the order in which they are created in the program.

A **base variable** is an intermediate variable that was created by retrieving an input, calling the random oracle H or sampling from $\mathbb{F}$. These are special because they are not intrinsically linearly dependent on other intermediate variables. A **derived variable** is an intermediate variable that is created by performing a linear combination of existing intermediate variables. Note, that derived variables can always be written as a unique linear combination of base variables. Let base be the number of base variables, and let us call them $b_1, \dots, b_{\mathsf{base}} \subset \{v_1, \dots, v_n\}$. We fix the ordering of the base variables by their order in $v_1, \dots, v_n$. We denote by $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ the column vector consisting of the values that the base variables take in a specific execution of $\mathcal{P}$. That is, the ith component of $\boldsymbol{v}$ is set to the value that b_i takes in that execution. One should think of $\boldsymbol{v}$ as a vector containing the whole state of the program execution. An intermediate variable, base or derived, can then be seen as a linear function going from the vector space $\mathbb{F}^{\mathsf{base}}$ to its concrete value in $\mathbb{F}$.

Let v_i be an intermediate variable. We define the **associated row vector** $\boldsymbol{v}_i$ to be the unique row vector in $\mathbb{F}^{1\times\mathsf{base}}$ representing this function. That means, that for every execution of $\mathcal{P}$: If base variables take the values $\boldsymbol{v}$, the variable v_i has the value $\boldsymbol{v}_i\boldsymbol{v}$. Here we use the ordinary matrix product. For example for the ith base variable b_i we have $\boldsymbol{b}_i = \begin{bmatrix}0 & \cdots & 1 & \cdots & 0\end{bmatrix}$ where the 1 is in the ith position. We follow the convention used in the previous papers about Linicrypt to write matrices and vectors using a bold font.

The outputs of $\mathcal{P}$ can be described by a matrix with entries in $\mathbb{F}$. Let $o_1, \ldots, o_l \in \{v_1, \ldots, v_n\}$ be the output variables of $\mathcal{P}$. Then the **output matrix** $\boldsymbol{O}$ of $\mathcal{P}$ is defined by

$$\boldsymbol{O} = \begin{bmatrix}\boldsymbol{o}_1 \\ \vdots \\ \boldsymbol{o}_l\end{bmatrix}.$$

By the definition of the associated vectors $\boldsymbol{o}_i$ we have $\boldsymbol{O}\boldsymbol{v} = \begin{bmatrix}o_1 & \cdots & o_k\end{bmatrix}^\top$. The output matrix describes the linear correlations in the output of $\mathcal{P}$.

In the same way, we also define the **input matrix** of $\mathcal{P}$. If $i_1, \ldots, i_k \in \{v_1, \ldots, v_n\}$ are the intermediate variables created by retrieving an input, then we write

$$\boldsymbol{I} = \begin{bmatrix}\boldsymbol{i}_1 \\ \vdots \\ \boldsymbol{i}_k\end{bmatrix}.$$

As $i_1, \ldots, i_k$ are base variables, the rows of $\boldsymbol{I}$ are canonical basis row vectors. If the Linicrypt program is written such that it first retrieves all its inputs, then $\boldsymbol{i}_m$ is simply the m'th canonical basis row vector.

The input and output matrices describe the linear correlations between the input and output of the program and its base variables. But the base variables are not completely independent of each other. The relationship between the queries and answers to the random oracle H needs to be captured algebraically. Let $a_i = H(t_i, (q_1, \ldots, q_n))$ be an operation in $\mathcal{P}$. The **associated oracle constraint** c of this operation is the tuple

$$c = \left(t_i, \begin{bmatrix}\boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_n\end{bmatrix}, \boldsymbol{a}_i\right) = (t_i, \boldsymbol{Q}_i, \boldsymbol{a}_i).$$

This should be interpreted as a constraint on $\boldsymbol{v}$, which requires $\boldsymbol{a}_i\boldsymbol{v} = H(t_i, \boldsymbol{Q}_i\boldsymbol{v})$. We denote the set of all (associated) oracle constraints of $\mathcal{P}$ by $\mathcal{C}$.

As we want the base variables to be linearly independent of each other, we restrict ourselves to Linicrypt programs which don't make multiple calls to the random oracle with the same input. In the language of the algebraic representation: We assume wlog that no two constraints in $\mathcal{C}$ share the same t and $\boldsymbol{Q}$).

Wrapping up these definitions, we define the **algebraic representation** of the program $\mathcal{P}$ to be the tuple $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$. A natural question that arises at this point is: Does the algebraic representation determine the behavior of $\mathcal{P}$ completely?

The answer is yes; the algebraic representation does not lose any relevant information about the operations executed in $\mathcal{P}$. Informally, this is because the constraints in $\mathcal{C}$ have a particular form, which makes it clear in which order the oracles calls have to be executed. Given the order, and using the input matrix, one can determine which variables used in the calls are retrieved from the input and which have to be sampled. Finally, the output matrix completely describes how to construct the output from the input, the results of the queries and the additional randomly sampled values.

The authors of the original paper on Linicrypt [CR16] establish a much stronger result for inputless programs. First, they define the normalized form of an algebraic representation. It can be efficiently generated from any inputless program description. They prove that two programs are indistinguishable if and only if their normalized algebraic representations are the same up to a basis change. The concept of basis change will be discussed extensively in the next chapters.

2.1.4 Collision Resistance in Linicrypt

In a paper by I. McQuoid, T. Swope and M. Rosulek [MSR19, Characterizing Collision and Second-Preimage Resistance in Linicrypt], the authors introduced a necessary condition for collision resistance and second-preimage resistance for a deterministic Linicrypt program.

They identified two reasons why a *deterministic* Linicrypt $\mathcal{P}$ program can fail to be second-preimage resistant. One can describe them roughly as follows:

1. It is degenerate, meaning that it doesn't use all of its inputs independently
2. It has a collision structure, which means that one can change some intermediate variable and compute what the input needs to be to counteract this change

Below are two example Linicrypt programs, $\mathcal{P}^H_{\text{deg}}$ is degenerate and $\mathcal{P}^H_{\text{cs}}$ has a collision structure.

$\mathcal{P}^H_{\text{deg}}(x, y)$
$v := x + y$
return $H(v)$

$\mathcal{P}^H_{\text{cs}}(x, y)$
$w := x + y$
return $H(w) + x$

Note, that you can set $w' \neq w$ to any value, then find an x' such that the output of $\mathcal{P}^H_{\text{cs}}$ stays the same, and finally solve for y' according to $w' = x' + y'$.

The authors show that for any deterministic Linicrypt program which is degenerate or has a collision structure second-preimage resistance (and hence also collision resistance) is completely broken. The main result of [CR16] is that they show that the converse of this is also true, but only for Linicrypt programs which use distinct nonces in each call to the random oracle. That is, if an adversary wins the collision game against such a Linicrypt program with a certain probability, then the program either has a collision structure or is degenerate. Furthermore, checking for degeneracy and the existence of a collision structure can be done efficiently.

In the following chapters, a similar result will be presented for a variant of Linicrypt where we replace the random oracle H with an ideal cipher $\mathcal{E} = (E, D)$. Along the way, we will merge the concepts of degeneracy and collision structure by considering degeneracy as an edge case of a collision structure.

2.2 Notation

We have already introduced some notational conventions. For reference, we summarize the notation we are going to use throughout the argument. Variables of a Linicrypt program and their values will be denoted with letters from the Latin alphabet in regular font. For example, depending on the context x denotes a variable, or it denotes a concrete value in $\mathbb{F}$ a variable called x takes in a specific execution of the program. Their associated row vectors will have the same letter but in bold font, e.g. $\boldsymbol{x} \in \mathbb{F}^{1\times \mathsf{base}}$. Column vectors and matrices are also denoted by bold letters. We use $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ or $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ to denote the column vector containing all the values for the base variables in the program's execution. In a slight overload of notation, we will denote both the canonical basis column vectors and the canonical basis row vectors by $\boldsymbol{e}_i$. In order to simplify the notation, we will write $\boldsymbol{i} = (i_1, \ldots, i_k)$ to denote the column vector $\begin{bmatrix} i_1 & \cdots & i_k \end{bmatrix}^\top$. We allow a Linicrypt program to take such a column vector as input, writing $\mathcal{P}(\boldsymbol{i})$ to denote the output of $\mathcal{P}$ given the input $i_1, \ldots, i_k$.

Linicrypt statements are often about the span of the rows of matrices. If $\boldsymbol{A} \in \mathbb{F}^{a\times \mathsf{base}}$ for arbitrary $a \in \mathbb{N}$, then we define the rowspace of $\boldsymbol{A}$ as $\mathsf{rowsp}(\boldsymbol{A}) := \mathsf{span}(\mathsf{rows}(\boldsymbol{A})) \subseteq \mathbb{F}^{1\times \mathsf{base}}$. Here we see rows as a function mapping a matrix in $\mathbb{F}^{a\times \mathsf{base}}$ the set of its rows, where each row is in $\mathbb{F}^{1\times \mathsf{base}}$. Note that rowsp is sometimes defined as a subset of $\mathbb{F}^{\mathsf{base}}$ instead of $\mathbb{F}^{1\times \mathsf{base}}$. We do not do this to simplify the notation in many statements, and to emphasize that elements in $\mathsf{rowsp}(\boldsymbol{A})$ should be considered as possible intermediate variables. For convenience, we also define for $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n$ matrices with base columns: $\mathsf{rowsp}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n) := \sum_i^n \mathsf{rowsp}(\boldsymbol{A}_i)$. The concept of rowspace is useful for example in the following style of arguments: If $\boldsymbol{Av}$ has been determined, then $\boldsymbol{wv}$ is determined for any $\boldsymbol{w} \in \mathsf{rowsp}(\boldsymbol{A})$.

The kernel of a matrix $\boldsymbol{A}$ as above is also used in Linicrypt proofs. As a reminder, $\mathsf{ker}(\boldsymbol{A}) := \{\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}} \mid \boldsymbol{Av} = 0\}$. We will use the following facts from Linear Algebra:

$$\begin{aligned} &\mathsf{rowsp}(\boldsymbol{A})^\top = \mathsf{ker}(\boldsymbol{A})^\perp \\ &(V_1 + V_2)^\perp = V_1^\perp \cap V_2^\perp \quad \text{for } V_1 \text{ and } V_2 \text{ subspaces of } \mathbb{F}^{\mathsf{base}} \end{aligned}$$

For example: If $\mathsf{rowsp}(\boldsymbol{A}) + \mathsf{rowsp}(\boldsymbol{B}) = \mathbb{F}^{1\times \mathsf{base}}$, then $\boldsymbol{Av} = \boldsymbol{Av}'$ and $\boldsymbol{Bv} = \boldsymbol{Bv}'$ implies $\boldsymbol{v} = \boldsymbol{v}'$ for any $\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{F}^{\mathsf{base}}$.

2.3 Security Definitions

We will use the following security definitions. We only state them for the ideal cipher model, as that is the one we need in the argument. Let $\mathcal{P}$ be a Linicrypt program taking k inputs. The collision game and the second-preimage game in the ideal cipher model are defined as:

$\mathrm{CR}\mathsf{Game}^{\mathrm{ic}}(\mathcal{P}, \mathcal{A})$
instantiate an ideal cipher $\mathcal{E} = (E, D)$
$(\boldsymbol{i}, \boldsymbol{i}') \leftarrow\$\ \mathcal{A}^{\mathcal{E}}$
return $(\boldsymbol{i} \neq \boldsymbol{i}') \wedge \big(\mathcal{P}^{\mathcal{E}}(\boldsymbol{i}) = \mathcal{P}^{\mathcal{E}}(\boldsymbol{i})\big)$

$\mathrm{SPR}\mathsf{Game}^{\mathrm{ic}}(\mathcal{P}, \mathcal{A})$
instantiate an ideal cipher $\mathcal{E} = (E, D)$
$\boldsymbol{i} \leftarrow\$\ \mathbb{F}^k$
$\boldsymbol{i}' \leftarrow\$\ \mathcal{A}^{\mathcal{E}}(\boldsymbol{i})$
return $(\boldsymbol{i} \neq \boldsymbol{i}') \wedge \big(\mathcal{P}^{\mathcal{E}}(\boldsymbol{i}) = \mathcal{P}^{\mathcal{E}}(\boldsymbol{i})\big)$

Definition 2.1 (Collision Resistance)**.** *Let $\mathcal{A}$ be an adversary and $\mathcal{P}$ be a Linicrypt program. We define $\mathcal{A}$'s advantage as*

$$\mathrm{CR}\mathsf{adv}^{\mathrm{ic}}[\mathcal{A}, \mathcal{P}] = \Pr\big[\mathrm{CR}\mathsf{Game}^{\mathrm{ic}}(\mathcal{P}, \mathcal{A}) = 1\big].$$

We call $\mathcal{P}$ collision resistant if $\mathrm{CR}\mathsf{adv}^{\mathrm{ic}}[\mathcal{A}, \mathcal{P}]$ is negligible for all (possibly not efficient) adversaries $\mathcal{A}$.

Definition 2.2 (Second-preimage Resistance)**.** *Let $\mathcal{A}$ be an adversary and $\mathcal{P}$ be a Linicrypt program. We define $\mathcal{A}$'s advantage as*

$$\mathrm{SPR}\mathsf{adv}^{\mathrm{ic}}[\mathcal{A}, \mathcal{P}] = \Pr\big[\mathrm{SPR}\mathsf{Game}^{\mathrm{ic}}(\mathcal{P}, \mathcal{A}) = 1\big].$$

We call $\mathcal{P}$ second-preimage resistant if $\mathrm{SPR}\mathsf{adv}^{\mathrm{ic}}[\mathcal{A}, \mathcal{P}]$ is negligible for all (possibly not efficient) adversaries $\mathcal{A}$.

These definitions are dependent on the choice of the field $\mathbb{F}$. A result that limits the advantage of an adversary against a program usually depends on the size of the field $\mathbb{F}$. If one wants to have security definitions depending on a security parameter $\lambda \in \mathbb{N}$, then one can choose a family of fields $\mathbb{F}_\lambda$ where $|\mathbb{F}_\lambda|$ is exponential in λ. One can then only talk about the security of a family of programs $\mathcal{P}_\lambda$. If the coefficients are in a subfield of every $\mathbb{F}_\lambda$, e.g. $\{0, 1\}$ as they are for most constructions, then changing λ will usually not affect the relevant properties of the program. For example, if such a $\mathcal{P}_\lambda$ has a collision structure for some λ, then it has a collision structure for all λ.

Chapter 3

A new level of abstraction for Linicrypt

The goal of this chapter is to introduce new concepts which are useful for adapting Linicrypt to the ideal cipher model. We separate some Linicrypt arguments from the actual properties of the oracle model which is used. The oracle model defines which values can be derived from which other values. For example, in the random oracle model, if one has some value $q \in \mathbb{F}$, one can get the value $a \in \mathbb{F}$ from the random oracle such that $a = H(q)$. But going from a to q is difficult in the random oracle model. In contrast, for the ideal cipher model, we can find an x such that $E(k, x) = y$ for given k and y. The adversary has access to the inverse permutaton $D(k, \cdot)$. We make a small set of definitions and lemmas which encapsulate these properties of the oracle. Because the statements of the definitions and lemmas are identical for both oracle models, we can base further argumentation on these building blocks. Our definition for a collision structure, which incorporates the notions of collision structure and degeneracy from [MSR19], is expressed in terms of these concepts. Thus it works for both oracle models at the same time.

3.1 Revisiting Algebraic Representations

In the previous section, we defined the algebraic representation for a Linicrypt program. It consists of the input matrix $\boldsymbol{I} \in \mathbb{F}^{k \times \mathsf{base}}$, the output matrix $\boldsymbol{O} \in \mathbb{F}^{l \times \mathsf{base}}$ and the set of constraints $\mathcal{C} = \{(t_i, \boldsymbol{Q}_i, \boldsymbol{a}_i) \mid i = 1, \ldots, n\}$, with $\boldsymbol{Q}_i \in \mathbb{F}^{m_i \times \mathsf{base}}$ and $\boldsymbol{a}_i \in \mathbb{F}^{1 \times \mathsf{base}}$. These matrices are such that, if one constrains $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ with $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$ for some arbitrary input, the program computes the rest of the base variables and it outputs $\boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$.

Our goal for this section is to find a better understanding of algebraic representations and thus also for collision structures. The algebraic representation suggest that the input and output matrix play similar roles: They are linear constraints for the base variables. Therefore, collision resistance should be expressible in terms of the ability to solve the constraints $\mathcal{C}$ while setting $\boldsymbol{O}\boldsymbol{v}$ to some value. This idea will be formalized in this section. A question that arises is: Which combination of matrices with a structure as described above correspond in essence to a Linicrypt program? To answer this question, we need to define what a random oracle constraint is for arbitrary vectors.

Definition 3.1 (Random oracle constraint)**.** *A **random oracle constraint** of dimension* base *taking* m *inputs is a tuple* $(t, \boldsymbol{Q}, \boldsymbol{a})$ *for* $t \in \{0,1\}^*$*,* $\boldsymbol{Q} \in \mathbb{F}^{m \times \mathsf{base}}$ *and* $\boldsymbol{a} \in \mathbb{F}^{1 \times \mathsf{base}}$*.*

This definition is a generalization of the definition in the preliminaries. When constructing the algebraic representation for a Linicrypt program, we always have the special case that $\boldsymbol{a} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$. We call t the nonce and refer to $\boldsymbol{Q}$ as the query matrix and to $\boldsymbol{a}$ as the answer matrix. If the nonce is the empty string, then we just write $(\boldsymbol{Q}, \boldsymbol{a})$ instead of $(t, \boldsymbol{Q}, \boldsymbol{a})$. Usually we just say "a constraint" when the other variables are clear from the context.

The constraint $(t, \boldsymbol{Q}, \boldsymbol{a})$ encodes the relationship via the random oracle between the base variables in a program. This semantic meaning of the constraints is encoded in the following definition.

Definition 3.2 (Solution of constraints)**.** *Let* $\mathcal{C}$ *be a set of constraints of dimension* base*. We say a vector* $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ ***solves*** $\mathcal{C}$ *if* $\boldsymbol{av} = H(t, \boldsymbol{Qv})$ *for all* $(t, \boldsymbol{Q}, \boldsymbol{a}) \in \mathcal{C}$*. Such a* $\boldsymbol{v}$ *is also called a **solution** of* $\mathcal{C}$*. The set of all solutions to* $\mathcal{C}$ *is called* $\mathsf{sol}(\mathcal{C})$*.*

Because H is a well-defined function, and not just any relation, these requirements extend to the constraints.

Definition 3.3 (Well-defined)**.** *A set of (random oracle) constraints* $\mathcal{C}$ *is **well-defined** if for any pair of constraints* $c, c' \in \mathcal{C}$ *we have* $(t, \boldsymbol{Q}) = (t', \boldsymbol{Q}') \implies \boldsymbol{a} = \boldsymbol{a}'$*.*

When we use a set of constraints, we will implicitly also require that it is well-defined. Highlighting the function like properties of a constraint $(t, \boldsymbol{Q}, \boldsymbol{a})$, we will also use the shorthand $(t, \boldsymbol{Q}) \mapsto \boldsymbol{a}$, or $\boldsymbol{Q} \mapsto \boldsymbol{a}$ if t in case t is an empty string. This notation was introduced in [HRR22].

We want to analyze which sets of constraints have solutions and how these solutions can be computed. First, every Linicrypt program is a method to solve the constraints $\mathcal{C}$ from its algebraic representation. If we run a program $\mathcal{P}$ on some input $(i_1, \ldots, i_k)$, it will query the random oracle and solve each random oracle constraint one by one in the order of the corresponding queries in $\mathcal{P}$. Let us call the resulting vector containing the values of the base variables in this execution $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$. Then it is a solution of $\mathcal{C}$ satisfying $\boldsymbol{Iv} = (i_1, \ldots, i_k)$.

If a set of constraints is well-defined, it might still not correspond to a valid Linicrypt program. Consider the set

$$\mathcal{C} = \left\{ \begin{bmatrix} 1 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 \end{bmatrix} \right\}.$$

Although it is well-defined, it would correspond to the two calls to the random oracle that are not compatible with the Linicrypt model. Specifically, such constraints would correspond to a program which contains the commands $y = H(x)$ and $x = H(y)$, where x, y are intermediate variables. There is no order in which these two calls can be executed without doing a reassignment, which does not exist in Linicrypt. Still, this set of constraints might have solutions. For every $x \in \mathbb{F}$ with the property that $x = H(H(x))$ the vector $(x, H(x))$ is a solution of $\mathcal{C}$. In other words, the set $\mathcal{C}$ expresses a condition on the base variables which cannot be represented by a Linicrypt program.

In order to characterize which constraints can be solved by a Linicrypt program, we need to consider the concept of basis change as it was introduced in [CR16].

Definition 3.4 (Basis change)**.** *Let $\boldsymbol{B}$ be any matrix in $\mathbb{F}^{\mathsf{base} \times \mathsf{base}'}$ and $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a Linicrypt program. We define*

$$\mathcal{C}\boldsymbol{B} := \{(t, \boldsymbol{QB}, \boldsymbol{aB}) \mid (t, \boldsymbol{Q}, \boldsymbol{a}) \in \mathcal{C}\}.$$

*If $\boldsymbol{B}$ is invertible, we say $\mathcal{P}\boldsymbol{B} := (\boldsymbol{IB}, \boldsymbol{OB}, \mathcal{C}\boldsymbol{B})$ is a **pseudo algebraic representation** of $\mathcal{P}$.*

The concept of base change combines well with the definition of the solution set for some constraints. Let $\mathcal{C}$ be any set of constraints and consider the basis change matrix $\boldsymbol{B}$. Note the following equivalence, which follows from the associativity of the matrix product:

$$\begin{aligned} & \boldsymbol{v} \text{ solves } \mathcal{C}\boldsymbol{B} \\ \iff & H(t, \boldsymbol{QBv}) = \boldsymbol{aBv} \quad \text{for all} \quad (t, \boldsymbol{Q}, \boldsymbol{a}) \in \mathcal{C} \\ \iff & \boldsymbol{Bv} \text{ solves } \mathcal{C} \end{aligned}$$

This implies that $\boldsymbol{B}|_{\mathsf{sol}(\mathcal{C}\boldsymbol{B})} : \mathsf{sol}(\mathcal{C}\boldsymbol{B}) \to \mathsf{sol}(\mathcal{C})$ is a well-defined bijection.

We restrict our goal to characterizing which matrices $\boldsymbol{I}$ and $\boldsymbol{O}$ and constraints $\mathcal{C}$ are pseudo algebraic representations of some program $\mathcal{P}$. This is interesting because it enables us to choose an arbitrary input matrix for a given set $\mathcal{C}$ and check whether there exists a Linicrypt program that computes solutions to $\mathcal{C}$. The degeneracy and collision structure attacks described in [MSR19] are both attacks that can be performed as Linicrypt programs.

First, we describe the form $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ need to have such that they are the algebraic representation of a Linicrypt program.

Lemma 3.5. *Let $\mathcal{C}$ be a finite well-defined set constraints of dimension* base *with $|\mathcal{C}| = n$, let $\boldsymbol{I} = \begin{bmatrix} \mathbb{1}_k & 0 \end{bmatrix} \in \mathbb{F}^{k \times \mathsf{base}}$ for $k = \mathsf{base} - n$ and let $\boldsymbol{O} \in \mathbb{F}^{l \times \mathsf{base}}$ for some $l \in \mathbb{N}$.*

$(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ is the algebraic representation of a deterministic Linicrypt program $\mathcal{P}$ if there exists an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ such that for all $i = 1, \ldots, n$:

1. $\mathsf{rows}(\boldsymbol{Q}_i) \subseteq \mathsf{rowsp}(\boldsymbol{e}_1, \dots, \boldsymbol{e}_{k+i-1})$
2. $\boldsymbol{a}_i = \boldsymbol{e}^{k+i}$

Proof Sketch. This is only a sketched proof. In order to formalize it, one first has to formalize the Linicrypt model as is done in [CR16].

Assume there is such an ordering as in the Lemma. We will construct the program $\mathcal{P}$. First, $\mathcal{P}$ will store its k inputs into intermediate variables, thus the input matrix is $\boldsymbol{I} = \begin{bmatrix} \mathbb{1}_k & 0 \end{bmatrix}$. Then each of the random oracle constraints is converted into Linicrypt commands of the form $v_{k+i} = H(q_1, \dots, q_{k+i-1})$. Condition 1. from the Lemma ensures that $q_1, \dots, q_{k+i-1}$ are linear combinations only of previously defined intermediate variables. Condition 2. from the Lemma guarantees that the variable v_{k+i} has not been used previously, so the command is a valid Linicrypt command.

Finally, because $k = \mathsf{base} - n$, we have instantiated all base variables, and we can therefore output according to $\boldsymbol{O}$. □

We will now generalize the condition from this Lemma in order to make it independent of the chosen basis of $\mathbb{F}^{\mathsf{base}}$. This leads to the characterization of pseudo algebraic representations.

Definition 3.6 (Deterministically Solvable)**.** *Let $\mathcal{C}$ be a finite well-defined set of constraints of dimension* base*, and let $\boldsymbol{I} \in \mathbb{F}^{k \times \mathsf{base}}$ for some $k \in \mathbb{N}$. $\mathcal{C}$ is **deterministically solvable fixing $\boldsymbol{I}$** (or fixing $\mathsf{rowsp}(\boldsymbol{I})$) if there exists an ordering $(c_1, \dots, c_n)$ of $\mathcal{C}$ such that for all $i = 1, \dots, n$:*

1. $\mathsf{rows}(\boldsymbol{Q}_i) \subseteq \mathsf{rowsp}(\boldsymbol{I}) + \mathsf{rowsp}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_{i-1})$
2. $\boldsymbol{a}_i \notin \mathsf{rowsp}(\boldsymbol{I}) + \mathsf{rowsp}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_{i-1})$

Additionally we require that $\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$ form a basis of $\mathbb{F}^{1 \times \mathsf{base}}$. We call $(c_1, \dots, c_n)$ a solution ordering of $\mathcal{C}$ fixing $\boldsymbol{I}$ (or fixing $\mathsf{rows}(\boldsymbol{I})$).

This is similar to the definition of collision structure in [MSR19]. Indeed, we will use it to combine the collision structure and degeneracy attack from that work.

Intuitively, $\mathcal{C}$ being deterministically solvable fixing $\boldsymbol{I}$ means the following. We can constrain $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ by $\boldsymbol{I}\boldsymbol{v} = (i_1, \dots, i_k)$ for an arbitrary input $i_1, \dots, i_k \in \mathbb{F}$, and then compute each oracle query deterministically (condition 1 in Definition 3.6) one by one, without creating a contradiction (condition 2 in Definition 3.6).

If we construct the algebraic representation $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ of a deterministic Linicrypt program $\mathcal{P}$, then $\mathcal{C}$ is a deterministically solvable set of constraints fixing $\boldsymbol{I}$. Indeed, the solution ordering of $\mathcal{C}$ fixing $\boldsymbol{I}$ can be exactly the order of the corresponding queries in the execution of $\mathcal{P}$.

Lemma 3.7. *Let $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be the algebraic representation of a deterministic Linicrypt program $\mathcal{P}$ taking k inputs. Then $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}$.*

Proof. Consider a constraint $(t, \boldsymbol{Q}, \boldsymbol{a})$ in $\mathcal{C}$. By definition $\boldsymbol{a} = \boldsymbol{e}^i$ for some i. And $\boldsymbol{Q}$ has only zeros in the columns including and to the right of column i. We can sort $\mathcal{C} = \{c_1, \dots, c_n\}$, such that the $\boldsymbol{a}$'s are sorted. Indeed, this is the same order as the associated oracle calls in the execution of $\mathcal{P}$.

Clearly condition 2 from Definition 3.6 is then fulfilled. Because $\mathcal{P}$ is deterministic, the queries to the oracle are linear combinations of input variables and results of previous queries. Therefore, condition 1 must be fulfilled. Finally, as $\mathcal{P}$ contains no sampling operation, we know $\mathsf{base} = k + n$. The associated vectors $\mathsf{rows}(I) = \{\boldsymbol{i}_1, \dots, \boldsymbol{i}_k\}$ to the input variables are linearly independent, hence it follows that $\{\boldsymbol{i}_1, \dots, \boldsymbol{i}_k, \boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$ is a basis of $\mathbb{F}^{1 \times \mathsf{base}}$. □

The following Lemma shows that the reversed direction also works, meaning that we can recover a deterministic Linicrypt program from deterministically solvable constraints by applying a basis change. The fact that a basis change is necessary is not surprising. In the definition of the algebraic representation, we make a choice in ordering the base variables. Any other choice of ordering should not change the properties of the algebraic representation. This corresponds to rewriting the program in trivial ways, e.g. changing the order in which inputs are retrieved.

Lemma 3.8. *Let $\mathcal{C}$ be a set of deterministically solvable constraints fixing $\boldsymbol{I} \in \mathbb{F}^{k \times \mathsf{base}}$ for some $k \in \mathbb{N}$. Let $\boldsymbol{O} \in \mathbb{F}^{\mathsf{out} \times \mathsf{base}}$ be an arbitrary output matrix for some $\mathsf{out} \in \mathbb{N}$. Then there is a basis change $\boldsymbol{B} \in \mathbb{F}^{\mathsf{base} \times \mathsf{base}}$ and a deterministic Linicrypt program $\mathcal{P}$, such that $(\boldsymbol{IB}, \boldsymbol{OB}, \mathcal{C}\boldsymbol{B})$ is its algebraic representation.*

Proof. Let $(c_1, \dots, c_n)$ be the solution ordering of $\mathcal{C}$ fixing $\boldsymbol{I} = \begin{bmatrix} \boldsymbol{i}_1^\top & \cdots & \boldsymbol{i}_k^\top \end{bmatrix}^\top$. We choose the new basis for $\mathbb{F}^{1 \times \mathsf{base}}$ as $(\boldsymbol{i}_1, \dots, \boldsymbol{i}_k, \boldsymbol{a}_1, \dots, \boldsymbol{a}_n)$, hence the basis change matrix is defined by

$$\boldsymbol{B}^{-1} = \begin{bmatrix} \boldsymbol{i}_1 \\ \vdots \\ \boldsymbol{i}_k \\ \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_n \end{bmatrix}.$$

In the following we denote by $\boldsymbol{e}_i$ the ith canonical row vector in $\mathbb{F}^{1 \times \mathsf{base}}$. By definition of $\boldsymbol{B}$ we have for $i = 1, \dots, n$

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{e}_k \end{bmatrix} \boldsymbol{B}^{-1} \qquad \text{and} \qquad \boldsymbol{a}_i = \boldsymbol{e}_{k+i} \boldsymbol{B}^{-1},$$

which is equivalent to

$$\boldsymbol{I}\boldsymbol{B} = \begin{bmatrix} \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{e}_k \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{a}_i\boldsymbol{B} = \boldsymbol{e}_{k+i}.$$

Additionally, as $\mathcal{C}$ is solvable via the ordering $(c_1, \ldots, c_n)$, we know for any $i \leq n$:

$$\mathsf{rows}(\boldsymbol{Q}_i\boldsymbol{B}) \subset \mathsf{rowsp}(\boldsymbol{i}_1\boldsymbol{B}, \ldots, \boldsymbol{i}_k\boldsymbol{B}, \boldsymbol{a}_1\boldsymbol{B}, \ldots, \boldsymbol{a}_{i-1}\boldsymbol{B}) = \mathsf{rowsp}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$$

This shows that $(\boldsymbol{I}\boldsymbol{B}, \boldsymbol{O}\boldsymbol{B}, \mathcal{C}\boldsymbol{B})$ fulfills the conditions in Lemma 3.5, therefore we obtain the corresponding program $\mathcal{P}$ from the statement. $\square$

In order to complete the abstratction to the level of the algebraic representation, we need the following Lemma and Corollaries.

Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic Linicrypt program taking k inputs and returning l outputs. First, we note that one can see a deterministic Linicrypt program as a function from its input space to its output space. We associate to $\mathcal{P}$ the function $f_{\mathcal{P}} : \mathbb{F}^k \to \mathbb{F}^l$ which maps $(i_1, \ldots, i_k)$ onto the output of $\mathcal{P}(i_1, \ldots, i_k)$.

Lemma 3.9. *Let $\mathcal{C}$ be deterministically solvable fixing some $\boldsymbol{I}$. If we view $\boldsymbol{I}$ as a function $\mathbb{F}^{\mathsf{base}} \to \mathbb{F}^k$, then $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is a bijection.*

Proof. If we set $\boldsymbol{O} = \mathbb{1}_{\mathsf{base}}$ then we can apply Lemma 3.8 to get a deterministic program $\mathcal{P}$ and basis change $\boldsymbol{B}$ such that, $\mathcal{P} = (\boldsymbol{I}\boldsymbol{B}, \boldsymbol{B}, \mathcal{C}\boldsymbol{B})$. The associated function $f_{\mathcal{P}} : \mathbb{F}^k \to \mathbb{F}^{\mathsf{base}}$ is injective, as $\mathcal{P}$ is deterministic. By definition of the algebraic representation, the vector of the values of the base variables in any execution of $\mathcal{P}$ is in $\mathsf{sol}(\mathcal{C}\boldsymbol{B})$. Note that the output matrix of $\mathcal{P}$ is $\boldsymbol{B}$ and $\boldsymbol{B}|_{\mathsf{sol}(\mathcal{C}\boldsymbol{B})} : \mathsf{sol}(\mathcal{C}\boldsymbol{B}) \to \mathsf{sol}(\mathcal{C})$ is a bijection. Therefore, $f_{\mathcal{P}}$ is an injective map from $\mathbb{F}^k$ to $\mathsf{sol}(\mathcal{C})$.

Now we will show that the input matrix $\boldsymbol{I} : \mathbb{F}^{\mathsf{base}} \to \mathbb{F}^k$ is injective when restricted to $\mathsf{sol}(\mathcal{C})$. This would then complete the proof because $\mathbb{F}^k$ is a finite set.

Assume we have $\boldsymbol{v}, \boldsymbol{v}' \in \mathsf{sol}(\mathcal{C})$ with $\boldsymbol{I}\boldsymbol{v} = \boldsymbol{I}\boldsymbol{v}'$. Also, assume that $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}$ using the ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$. We will inductively prove that $\boldsymbol{a}_i\boldsymbol{v} = \boldsymbol{a}_i\boldsymbol{v}'$ for all $i = 1, \ldots, n$.

Assume this is true for all $j \leq i$. Because of the solution ordering of $\mathcal{C}$, we know that $\mathsf{rows}(\boldsymbol{Q}_{i+1}) \subseteq \mathsf{rowsp}(\boldsymbol{I}) + \mathsf{rowsp}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_i)$. This means there is a $\boldsymbol{\lambda}$ such that $\boldsymbol{Q}_{i+1} = \boldsymbol{\lambda} \begin{bmatrix} \boldsymbol{I}^\top & \boldsymbol{a}_1^\top & \cdots & \boldsymbol{a}_n^\top \end{bmatrix}^\top =: \lambda \boldsymbol{A}_i$. Therefore, we have $\boldsymbol{Q}_{i+1}\boldsymbol{v}' = \boldsymbol{\lambda}\boldsymbol{A}_i\boldsymbol{v}' = \boldsymbol{\lambda}\boldsymbol{A}_i\boldsymbol{v} = \boldsymbol{Q}_{i+1}\boldsymbol{v}$. Because $\boldsymbol{v}$ and $\boldsymbol{v}'$ are solutions to $\mathcal{C}$ we know that $\boldsymbol{a}_{i+1}\boldsymbol{v}' = H(\boldsymbol{Q}_{i+1}\boldsymbol{v}') = H(\boldsymbol{Q}_{i+1}\boldsymbol{v}) = \boldsymbol{a}_{i+1}\boldsymbol{v}$.

Because $\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ is a basis, the matrix $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_n \end{bmatrix}$ is invertible.

Using the assumptions and the results from the induction we have shown that $\boldsymbol{Cv'} = \boldsymbol{Cv}$ and therefore $\boldsymbol{v'} = \boldsymbol{v}$. This shows that $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})} : \mathsf{sol}(\mathcal{C}) \to \mathbb{F}^k$ is injective. But because $f_{\mathcal{P}} : \mathbb{F}^k \to \mathsf{sol}(\mathcal{C})$ is also injective and $\mathbb{F}^k$ is finite, both maps are bijections. $\square$

Corollary 3.10. *Let $\mathcal{C}$ be deterministically solvable fixing some $\boldsymbol{I}$. For each element in $\mathbb{F}^k$ its inverse under $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ can be computed with $|\mathcal{C}|$ queries to the random oracle.*

Proof. Consider the function $f_{\mathcal{P}}$ from the proof of Lemma 3.9. It can be computed with $|\mathcal{C}|$ queries to the random oracle. Because $(\boldsymbol{IB}, \boldsymbol{B}, \mathcal{C}\boldsymbol{B})$ is an algebraic representation, we have $\boldsymbol{IB} = \mathbb{1}_k \mid 0$. If we run $\mathcal{P}$ on $(i_1, \dots, i_k)$ we compute the values of the remaining base variables. Let us call the vector containing the values of the base variables $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$. Clearly $\boldsymbol{IBv} = (i_1, \dots, i_k)$. Also, $\mathcal{P}$ outputs $\boldsymbol{Bv}$, so $f_{\mathcal{P}}\big((i_1, \dots, i_k)\big) = \boldsymbol{Bv}$. Therefore,

$$\boldsymbol{IBv} = (\boldsymbol{I} \circ f_{\mathcal{P}})\big((i_1, \dots, i_k)\big) = (i_1, \dots, i_k). \tag{3.1}$$

As $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ and $f_{\mathcal{P}} : \mathbb{F}^k \to \mathsf{sol}(\mathcal{C})$ are bijective, (3.1) shows that $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ and $f_{\mathcal{P}}$ are inverses of each other. $\square$

The benefit of this bijection is that every solution of $\mathcal{C}$ corresponds to a unique input. This implies, that any question regarding the input and output of a Linicrypt program can be translated to the level of the algebraic representation and solutions of $\mathcal{C}$. Explicitly, we can formulate the following corollary.

Corollary 3.11. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic Linicrypt program. The following two statements are equivalent:*

1. *Running $\mathcal{P}$ on input $(i_1, \dots, i_k)$ gives output $(o_1, \dots, o_l)$.*
2. *There exists a $\boldsymbol{v}$ solving $\mathcal{C}$ such that $\boldsymbol{Iv} = (i_1, \dots, i_k)$ and $\boldsymbol{Ov} = (o_1, \dots, o_l)$.*

Proof. The direction 1. $\implies$ 2. is clear, by definition of the algebraic representation.

For the other direction, assume we have such a $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$. If we run $\mathcal{P}$ on $\boldsymbol{Iv} = (i_1, \dots, i_k)$ we compute the base variables $\boldsymbol{v'} \in \mathbb{F}^{\mathsf{base}}$. Again by definition of the algebraic representation $\boldsymbol{v'}$ is a solution to $\mathcal{C}$ with $\boldsymbol{Iv'} = (i_1, \dots, i_k) = \boldsymbol{Iv}$. By Lemma 3.9 $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is bijective, so we have $\boldsymbol{v} = \boldsymbol{v'}$. Running $\mathcal{P}$ on $(i_1, \dots, i_k)$ gives output $\boldsymbol{Ov'} = \boldsymbol{Ov} = (o_1, \dots, o_l)$. $\square$

3.2 Revisiting Collision Structures

Using this language we can argue about the invertibility and second preimage resistance of a Linicrypt program. The goal of this section is to describe collision structures in a more abstract form that is easier to adapt to the ideal cipher model. We will describe collision structures as a program being partially invertible with extra degrees of freedom. As a stepping stone towards this goal, we start by describing a sufficient condition for a Linicrypt program to be invertible.

Lemma 3.12. *$\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic Linicrypt program. If $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{O}$, then $f_{\mathcal{P}}$ is bijective and $f_{\mathcal{P}}^{-1}$ is the associated function of a deterministic Linicrypt program which we call $\mathcal{P}^{-1}$.*

Proof. Assume $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{O}$. Applying the Lemma 3.8 we get a basis change $\boldsymbol{B}$ and a Linicrypt program $\mathcal{P}^{-1}$ with algebraic representation $(\boldsymbol{OB}, \boldsymbol{IB}, \mathcal{C}\boldsymbol{B})$. Note, that $\boldsymbol{O}$ and $\boldsymbol{I}$ have the same number of rows, because both $\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$ and $\mathsf{rows}(\boldsymbol{O}) \cup \{\boldsymbol{a}_1, \dots, \boldsymbol{a}_n\}$ are bases of $\mathbb{F}^{1\times\mathsf{base}}$.

Because of the bijection $\boldsymbol{B} : \mathsf{sol}(\mathcal{C}\boldsymbol{B}) \to \mathsf{sol}(\mathcal{C})$ we have the following equivalence:

$$\boldsymbol{v} \text{ solves } \mathcal{C} \text{ with } \boldsymbol{Iv} = (vdtsik) \text{ and } \boldsymbol{Ov} = (vdtsol)$$
$$\Updownarrow$$
$$\boldsymbol{B}^{-1}\boldsymbol{v} \text{ solves } \mathcal{C}\boldsymbol{B} \text{ with } \boldsymbol{OBB}^{-1}\boldsymbol{v} = (vdtsol) \text{ and } \boldsymbol{IBB}^{-1}\boldsymbol{v} = (vdtsik)$$

Combining this equivalence with the one from Lemma 3.11 we get: Running $\mathcal{P}$ on input $i_1, \dots, i_k$ giving output $o_1, \dots, o_l$ is equivalent to running $\mathcal{P}^{-1}$ on input $o_1, \dots, o_l$ with output $i_1, \dots, i_k$. This means that $f_{\mathcal{P}^{-1}} = f_{\mathcal{P}}^{-1}$, which is what we needed to prove. □

For example, the Lemma applies to the following program $\mathcal{P}$. The proof constructs the algebraic representation of the inverse program $\mathcal{P}^{-1}$.

$\mathcal{P}(x, y)$
return $(y + H(x), x)$

$\mathcal{P}^{-1}(w, z)$
return $(z, w - H(z))$

To simplify the notation we extend the linear algebra operators rowsp and ker to be able to use them naturally with constraints and sets of constraints. We define for a constraint $c = (t, \boldsymbol{Q}, \boldsymbol{a})$:

$$\mathsf{rowsp}(c) = \mathsf{rowsp}(\boldsymbol{Q}) + \mathsf{rowsp}(\boldsymbol{a})$$
$$\mathsf{ker}(c) = \mathsf{ker}(\boldsymbol{Q}) \cap \mathsf{ker}(\boldsymbol{a})$$

For a set of constraints $\mathcal{C}$ we write:

$$\mathsf{rowsp}(\mathcal{C}) = \sum_{c \in \mathcal{C}} \mathsf{rowsp}(c)$$
$$\mathsf{ker}(\mathcal{C}) = \bigcap_{c \in \mathcal{C}} \mathsf{ker}(c)$$

Lemma 3.13. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic Linicrypt program. Assume we can write $\mathcal{C} = \mathcal{C}_{cs} \cup \mathcal{C}_{fix}$ and $\mathcal{C}_{cs}$ is deterministically solvable fixing some $\boldsymbol{I}_{cs}$ such that*

$$\mathsf{rowsp}(\boldsymbol{I}_{cs}) \supsetneq \mathsf{rowsp}(\mathcal{C}_{fix}) + \mathsf{rowsp}(\boldsymbol{O}). \tag{3.2}$$

Then an attacker can find second preimages with $|\mathcal{C}_{cs}|$ queries. We say a program has a collision structure if it fulfills condition (3.2).

This Lemma is similar to Lemma 3.12 in the sense that both describe a condition for being able to find a $\boldsymbol{v}$ solving $\mathcal{C}$ while fixing $\boldsymbol{O}\boldsymbol{v} = (vdtsol)$. The difference is that here we consider the easier case (from an attacker's perspective): We already have a solution $\boldsymbol{v}'$ with $\boldsymbol{O}\boldsymbol{v}' = (vdtsol)$ and only require a different one.

The definition of collision structure in Lemma 3.13 is slightly more general than the one from [MSR19]. It also includes the case of degeneracy, namely it corresponds to choosing $\mathcal{C}_{cs} = \{\}$ and $\boldsymbol{I}_{\text{cs}} = \mathbb{1}_{\mathsf{base}}$. Degeneracy means precisely that $\mathbb{F}^{1\times\mathsf{base}} \supset \mathsf{rowsp}(\mathcal{C}) + \mathsf{rowsp}(\boldsymbol{O})$. Note, that it is crucial that the space on the left is $\supsetneq$ and not only $\supseteq$, as this gives the extra degree of freedom to find a different preimage. It plays the same role as $\boldsymbol{q}_{i^*}$ plays in the definition of collision structure from [MSR19].

Here we only give a proof sketch, a more formal proof will be given in the next section when we adapt Linicrypt to the ideal cipher model.

Proof Sketch. Assume we are given a $\boldsymbol{v}'$ solving $\mathcal{C}$ such that $\boldsymbol{I}\boldsymbol{v}' = (vdtsik)$ and $\boldsymbol{O}\boldsymbol{v}' = (vdtsol)$. The goal is to find a different $\boldsymbol{v}$ solving $\mathcal{C}$ with $\boldsymbol{O}\boldsymbol{v} = (vdtsol)$. By (3.2) we can constrain $\boldsymbol{v}$ to fulfill $\boldsymbol{O}\boldsymbol{v} = \boldsymbol{O}\boldsymbol{v}'$ as well as $\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{v}'$ and $\boldsymbol{a}\boldsymbol{v} = \boldsymbol{a}\boldsymbol{v}'$ for every $(t, \boldsymbol{Q}, \boldsymbol{a}) \in \mathcal{C}_{fix}$. Even doing so, we still have at least one degree of freedom before we have uniquely determined what $\boldsymbol{I}_{cs}\boldsymbol{v}$ is. We can therefore choose these further constraints to be arbitrary such that $\boldsymbol{v} \neq \boldsymbol{v}'$. Because $\mathcal{C}_{cs}$ is deterministically solvable fixing $\boldsymbol{I}_{cs}$, we can uniquely determine $\boldsymbol{v}$ such that it solves $\mathcal{C}_{cs}$. But we chose $\boldsymbol{I}_{cs}\boldsymbol{v}$ in such a way, that $\boldsymbol{v}$ also solves $\mathcal{C}_{fix}$, hence $\boldsymbol{v}$ solves $\mathcal{C} = \mathcal{C}_{cs} \cup \mathcal{C}_{fix}$.

Finally, because $\mathcal{P}$ is deterministic $\boldsymbol{I}$ is injective and therefore $\boldsymbol{I}\boldsymbol{v}$ is a second preimage to $\boldsymbol{I}\boldsymbol{v}'$ for the Linicrypt program $\mathcal{P}$. □

The main benefit of this perspective on the collision structure attack is that it directly describes the collision structure attack as a Linicrypt program. If we apply Lemma 3.8 to $\mathcal{C}_{cs}$ while fixing $\boldsymbol{I}_{cs}$, the resulting "attack program" is one taking as input the desired output, some values determined by the execution on the given preimage, and some extra values which can be set freely.

The following is the running example from [MSR19].

$\mathcal{P}(x, y, z)$

$w = H(x) + H(z) + y$
return $(H(w) + x, H(z))$

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{O} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
$$\boldsymbol{q}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{a}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{q}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{a}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
$$\boldsymbol{q}_3 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$
$$\boldsymbol{a}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We set $\mathcal{C}_{cs} = \{c_1, c_3\}$ and $\mathcal{C}_{\mathit{fix}} = \{c_2\}$. Then (c_3, c_1) is a solution ordering for $\mathcal{C}_{cs}$ fixing

$$\boldsymbol{I}_{cs} = \begin{bmatrix} \boldsymbol{O} \\ \boldsymbol{q}_2 \\ \boldsymbol{q}_3 \end{bmatrix}.$$

Condition (3.2) is fulfilled because

$$\mathsf{rowsp}(\boldsymbol{I}_{cs}) = \big(\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{\mathit{fix}})\big) \oplus \mathsf{rowsp}(\boldsymbol{q}_3).$$

The direct sum means that $\boldsymbol{q}_3$ is not in $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{\mathit{fix}})$. Then Lemma 3.8 gives us the basis change $\boldsymbol{B}$ such that $(\boldsymbol{I}_{cs}\boldsymbol{B}, \boldsymbol{I}\boldsymbol{B}, \mathcal{C}_{cs}\boldsymbol{B})$ is the algebraic representation of some $\mathcal{P}_{cs}$. Here we have

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

which leads to the representation:

$$\mathcal{C}_{cs}\boldsymbol{B} = \left\{ \begin{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \\ \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix} \right\}$$
$$\boldsymbol{I}_{\mathrm{cs}}\boldsymbol{B} = \begin{bmatrix} \mathbb{1}_4 & 0 \end{bmatrix}$$
$$\boldsymbol{I}\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The corresponding Linicrypt program $\mathcal{P}_{cs}$ to this algebraic representation can be written as:

$\mathcal{P}_{cs}(o_1, o_2, q_2, q_3)$

$a_3 = H(q_3)$
$v = H(o_1 - a_3)$
return $(o_1 - a_3, q_3 - o_2 - v, q_2)$

This program computes second preimages to the input (x, y, z) if we set its inputs $(o_1, o_2) = \mathcal{P}(x, y, z)$, $q_2 = z$ and $q_3 \neq H(x) + H(z) + y$ arbitrarily.

Chapter 4

Adapting Linicrypt to the ideal cipher model

In this chapter, we modify the Linicrypt model to make use of the ideal cipher model instead of the random oracle model. This means that a Linicrypt program gets access to a block cipher $\mathcal{E} = (E, D)$ where E and D are functions $\mathbb{F} \times \mathbb{F} \to \mathbb{F}$ instead of the hash function $H : \{0,1\}^* \times \mathbb{F}^* \to \mathbb{F}$. By the definition of a block cipher, $E_k := E(k, \cdot)$ is a permutation of $\mathbb{F}$ for all $k \in \mathbb{F}$ and $D_k := D(k, \cdot)$ is its inverse. In the ideal cipher model, we assume that the block cipher has no weakness. This is modeled by choosing each permutation E_k uniformly at random at the beginning of every security game. We will call these programs with access to a block cipher instead of a hash function ideal cipher Linicrypt programs.

The command $y = E(k, x)$ in an ideal cipher Linicrypt program has to be treated differently from the command $y = H(k, x)$ when considering collision resistance, because an attacker has access to the deterministic Linicrypt program and both directions of the block cipher $\mathcal{E} = (E, D)$. Consider these two programs, $\mathcal{P}^H$ in standard Linicrypt and $\mathcal{P}^{\mathcal{E}}$ in ideal cipher Linicrypt.

$\mathcal{P}^H(k, x)$
return $H(k, x)$

$\mathcal{P}^{\mathcal{E}}(k, x)$
return $E(k, x)$

While $\mathcal{P}^H$ is collision resistant, it is trivial to find second preimages for $\mathcal{P}^{\mathcal{E}}$ For any $k' \in \mathbb{F}$ the pair $(k', D(k', E(k, x)))$ is a second preimage to (k, x).

This invertibility property of block ciphers has to be taken into account in both the algebraic representation and the characterization of collision resistance.

4.1 Algebraic representation for ideal cipher Linicrypt

Parallel to standard Linicrypt, we will define the algebraic representation of an ideal cipher Linicrypt program. Towards this goal, we will make the equivalent definitions to the ones in chapter 3.1.

Definition 4.1 (Ideal cipher oracle constraint)**.** *An **ideal cipher oracle constraint** of dimension* base *is a tuple* $(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$ *for* $\boldsymbol{k}, \boldsymbol{x}$ *and* $\boldsymbol{y}$ *in* $\mathbb{F}^{1\times\mathsf{base}}$*.*

We will just say "constraint" to mean ideal cipher oracle constraint or random oracle constraint depending on the context.

Definition 4.2 (Solution of constraints)**.** *Let* $\mathcal{C}$ *be a set of ideal cipher constraints of dimension* base*. We say a vector* $\boldsymbol{v} \in \mathbb{F}^{\mathsf{base}}$ ***solves*** $\mathcal{C}$ *if* $\boldsymbol{yv} = E(\boldsymbol{kv}, \boldsymbol{xv})$ *for all* $(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{C}$*. Such a* $\boldsymbol{v}$ *is also called a **solution** of* $\mathcal{C}$*. The set of all solutions to* $\mathcal{C}$ *is called* $\mathsf{sol}(\mathcal{C})$*.*

Let $\mathcal{P}$ be an ideal cipher Linicrypt program. Base variables and associated vectors are defined exactly as defined in standard Linicrypt. The only difference is that the base variables can now be instantiated with calls to the ideal cipher instead of calls to the random oracle. For each command in $\mathcal{P}$ of the form $v_3 = E(v_1, v_2)$ we define the **associated ideal cipher constraint** $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$. Note, that the bold variant of the intermediate variable denotes its associated vector. Correspondingly, each query to D of the form $v_3 = D(v_1, v_2)$, is associated with the constraint $(\boldsymbol{v}_1, \boldsymbol{v}_3, \boldsymbol{v}_2)$. We call the set of all associated constraints $\mathcal{C}$. The input matrix $\boldsymbol{I}$ and output matrix $\boldsymbol{O}$ are defined in the same way as in standard Linicrypt. We call $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ the algebraic representation of $\mathcal{P}$. Then, the following statement holds true, which is the trivial direction of Corollary 3.11 in standard Linicrypt.

Lemma 4.3. *Let* $\mathcal{P}$ *be an ideal cipher Linicrypt program with algebraic representation* $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$*. Let* $\boldsymbol{v}$ *denote the values of the base variables in an execution of* $\mathcal{P}$ *with input* $(i_1, \ldots, i_k)$ *and output* $(o_1, \ldots, o_l)$*. Then* $\boldsymbol{v}$ *is a solution to* $\mathcal{C}$ *with* $\boldsymbol{Iv} = (i_1, \ldots, i_k)$ *and* $\boldsymbol{Ov} = (o_1, \ldots, o_l)$

Proof. The statements $\boldsymbol{Iv} = (i_1, \ldots, i_k)$ and $\boldsymbol{Ov} = (o_1, \ldots, o_l)$ follow from definition as in standard Linicrypt. Each constraint in $\mathcal{C}$ comes from a command in $\mathcal{P}$ with a query to the ideal cipher.

Take some $(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{C}$. After renaming the intermediate variables of $\mathcal{P}$, it is associated either to the command $y = E(k, x)$ or to the command $x = D(k, y)$. In the first case we have $\boldsymbol{yv} = y = E(k, x) = E(\boldsymbol{kv}, \boldsymbol{xv})$. In the second case we also have $y = E(k, x)$ by the properties of the ideal cipher $\mathcal{E} = (E, D)$. Therefore, $\boldsymbol{v}$ is a solution to $\mathcal{C}$. □

The main definition that changes with ideal cipher Linicrypt is the one determining if $\mathcal{C}$ is deterministically solvable. This definition captures the properties of the black box that $\mathcal{P}$ and an attacker can access. In standard Linicrypt the black box is a one-way random function. In ideal cipher Linicrypt the black box is a random permutation that can be computed both ways, as both the Linicrypt program and the attacker have full access to the ideal cipher $\mathcal{E} = (E, D)$.

To simplify the notation we extend the linear algebra operators rowsp and ker to be able to use them naturally with constraints and sets of constraints. We define for a constraint $c = (\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$:

$$\begin{aligned}\mathsf{rowsp}(c) &= \mathsf{rowsp}(\boldsymbol{k}) + \mathsf{rowsp}(\boldsymbol{x}) + \mathsf{rowsp}(\boldsymbol{y})\\ \mathsf{ker}(c) &= \mathsf{ker}(\boldsymbol{k}) \cap \mathsf{ker}(\boldsymbol{x}) \cap \mathsf{ker}(\boldsymbol{y})\end{aligned}$$

For a set of constraints $\mathcal{C}$ we write:

$$\begin{aligned}\mathsf{rowsp}(\mathcal{C}) &= \sum_{c\in\mathcal{C}} \mathsf{rowsp}(c)\\ \mathsf{ker}(\mathcal{C}) &= \bigcap_{c\in\mathcal{C}} \mathsf{ker}(c)\end{aligned}$$

Definition 4.4 (Deterministically Solvable)**.** *Let $\mathcal{C}$ be a finite well-defined set of ideal cipher constraints of dimension* base*, and let $\boldsymbol{I} \in \mathbb{F}^{k\times\mathsf{base}}$ for some $k \in \mathbb{N}$. $\mathcal{C}$ is **deterministically solvable fixing $\boldsymbol{I}$** (or fixing $\mathsf{rowsp}(\boldsymbol{I})$) if there exists an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ such that for all $i = 1, \ldots, n$ the following holds.*

Let the components of c_i be called $(\boldsymbol{k}_i, \boldsymbol{x}_i, \boldsymbol{y}_i)$. One can write either

$$\boldsymbol{Q}_i = \begin{bmatrix}\boldsymbol{k}_i\\ \boldsymbol{x}_i\end{bmatrix} \text{ and } \boldsymbol{a}_i = \boldsymbol{y}_i \text{ (Case a) or } \boldsymbol{Q}_i = \begin{bmatrix}\boldsymbol{k}_i\\ \boldsymbol{y}_i\end{bmatrix} \text{ and } \boldsymbol{a}_i = \boldsymbol{x}_i \text{ (Case b),}$$

such that:

1. $\mathsf{rows}(\boldsymbol{Q}_i) \subseteq \mathsf{rowsp}(\boldsymbol{I}) + \mathsf{rowsp}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1})$
2. $\boldsymbol{a}_i \notin \mathsf{rowsp}(\boldsymbol{I}) + \mathsf{rowsp}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1})$

Additionally we require that $\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ form a basis of $\mathbb{F}^{1\times\mathsf{base}}$. We call $(c_1, \ldots, c_n)$ a solution ordering of $\mathcal{C}$ fixing $\boldsymbol{I}$ (or fixing $\mathsf{rows}(\boldsymbol{I})$).

Note that the case distinction (Case a and Case b) is what models the invertibility of the ideal cipher. It is indeed an exclusive “or” because condition 1. of Case a implies that condition 2. of Case b is false. With these definitions in place, the Lemmas from Chapter 3.1 can be formulated in the same way.

Lemma 4.5. *Let $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be the algebraic representation of a deterministic ideal cipher Linicrypt program $\mathcal{P}$ taking k inputs. Then $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}$.*

Proof Sketch. For each constraint $c = (\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$ in $\mathcal{C}$ we set $\boldsymbol{Q} = \begin{bmatrix}\boldsymbol{k}^\top & \boldsymbol{x}^\top\end{bmatrix}^\top$ and $\boldsymbol{a} = \boldsymbol{y}$ if c is associated to a call to E. If c is associated to a call to D, we set $\boldsymbol{Q} = \begin{bmatrix}\boldsymbol{k}^\top & \boldsymbol{y}^\top\end{bmatrix}^\top$ and $\boldsymbol{a} = \boldsymbol{x}$. The rest of the proof is identical to the proof of Lemma 3.7. □

Lemma 4.6. *Let $\mathcal{C}$ be a finite well-defined set of ideal cipher constraints of dimension* base *with $|\mathcal{C}| = n$, let $\boldsymbol{I} = \begin{bmatrix}\mathbb{1}_k & 0\end{bmatrix} \in \mathbb{F}^{k\times\mathsf{base}}$ for $k = \mathsf{base} - n$ and let $\boldsymbol{O} \in \mathbb{F}^{l\times\mathsf{base}}$ for some $l \in \mathbb{N}$.*

$(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ is the algebraic representation of a deterministic ideal cipher Linicrypt program $\mathcal{P}$ if there exists an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ such that for all $i = 1, \ldots, n$ one of the following cases hold for $(\boldsymbol{k}_i, \boldsymbol{x}_i, \boldsymbol{y}_i) = c_i$:

(a) $\boldsymbol{y}_i = \boldsymbol{e}_{k+i}$ and $\{\boldsymbol{k}_i, \boldsymbol{x}_i\} \subseteq \mathsf{span}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$ or

(b) $\boldsymbol{x}_i = \boldsymbol{e}_{k+i}$ and $\{\boldsymbol{k}_i, \boldsymbol{y}_i\} \subseteq \mathsf{span}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$

Proof Sketch. The proof idea is the same as for Lemma 3.5. The k inputs are handled as in standard Linicrypt. Consider the constraint c_i. If case (a) holds, then we convert this constraint into a command of the form $v_{k+i} = E(q_1, q_2)$, for q_1 and q_2 being intermediate variables created by a linear combination. Then condition $\{\boldsymbol{k}_i, \boldsymbol{x}_i\} \subseteq \mathsf{span}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$ ensures that these linear combinations are well-defined. As $\mathsf{base} = k + n$, all base variables have been set after n query commands. We can use the rows of $\boldsymbol{O}$ to define the Linicrypt commands for the output variables. □

Lemma 4.7. *Let $\mathcal{C}$ be a set of deterministically solvable constraints fixing $\boldsymbol{I} \in \mathbb{F}^{k \times \mathsf{base}}$ for some $k \in \mathbb{N}$. Let $\boldsymbol{O} \in \mathbb{F}^{\mathsf{out} \times \mathsf{base}}$ be an arbitrary output matrix for some $\mathsf{out} \in \mathbb{N}$. Then there is a basis change $\boldsymbol{B} \in \mathbb{F}^{\mathsf{base} \times \mathsf{base}}$ and a Linicrypt program $\mathcal{P}$, such that $(\boldsymbol{IB}, \boldsymbol{OB}, \mathcal{C}\boldsymbol{B})$ is its algebraic representation.*

Proof Sketch. By setting $\boldsymbol{Q}_i$ and $\boldsymbol{a}_i$ as in the definition 4.4 we can copy the proof of Lemma 3.8. □

Lemma 4.8. *Let $\mathcal{C}$ be deterministically solvable fixing some $\boldsymbol{I}$. If we view $\boldsymbol{I}$ as a function $\mathbb{F}^{\mathsf{base}} \to \mathbb{F}^k$, then $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is a bijection.*

Proof Sketch. By setting $\boldsymbol{Q}_i$ and $\boldsymbol{a}_i$ as in the definition 4.4 we can copy the proof of Lemma 3.9. □

Corollary 4.9. *Let $\mathcal{C}$ be deterministically solvable fixing some $\boldsymbol{I}$. For each element in $\mathbb{F}^k$ its inverse under $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ can be computed with $|\mathcal{C}|$ queries to the ideal cipher.*

Proof Sketch. The proof is identical to the proof of Corollary 3.10. □

4.2 Collision Structure

We have all the prerequisites to prove the analog of Lemma 3.13. That is, we can give a sufficient condition for a program to be susceptible to a constant time second preimage attack.

Proposition 4.10. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic ideal cipher Linicrypt program. Assume we can write $\mathcal{C} = \mathcal{C}_{cs} \sqcup \mathcal{C}_{fix}$ and $\mathcal{C}_{cs}$ is deterministically solvable fixing some $\boldsymbol{I}_{cs}$ such that*

$$\mathsf{rowsp}(\boldsymbol{I}_{cs}) \supsetneq \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix}). \tag{4.1}$$

Then an attacker can find second preimages with $|\mathcal{C}_{cs}|$ queries. We say a program has a collision structure if it fulfills condition (4.1).

Here we will give a more formal proof of this statement.

Proof. The proof describes how to find a second preimage to some input $(i_1, \ldots, i_k)$ to $\mathcal{P}$. By executing $\mathcal{P}$ on $(i_1, \ldots, i_k)$ we compute the values of the base variables $\boldsymbol{v} \in \mathsf{sol}(\mathcal{C})$. Recall the bijection $\boldsymbol{I}_{cs}|_{\mathsf{sol}(\mathcal{C}_{cs})} : \mathsf{sol}(\mathcal{C}_{cs}) \to \mathbb{F}^{k'}$ where k' is the number of rows of $\boldsymbol{I}_{cs}$. Instead of seeing this bijection as a map into the input space of $\mathcal{P}$, we will see it as a map into the quotient space $\mathbb{F}^{\mathsf{base}}/\mathsf{ker}(\boldsymbol{I}_{cs})$. This quotient space is defined by the equivalence relation $\boldsymbol{v} \sim_{cs} \boldsymbol{w} \iff \boldsymbol{v} - \boldsymbol{w} \in \mathsf{ker}(\boldsymbol{I}_{cs})$. We denote it by:

$$\widetilde{\boldsymbol{I}_{cs}} : \mathsf{sol}(\mathcal{C}_{cs}) \to \mathbb{F}^{\mathsf{base}} \Big/ \mathsf{ker}(\boldsymbol{I}_{cs})$$
$$\boldsymbol{v} \mapsto [\boldsymbol{v}]_{\sim_{cs}}$$

Using condition (4.1) we will map $\mathbb{F}^{\mathsf{base}}/\mathsf{ker}(\boldsymbol{I}_{cs})$ to $\mathbb{F}^{\mathsf{base}}/(\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}}))$ non injectively. First we rewrite the condition (4.1):

$$(4.1) \iff \mathsf{ker}(\boldsymbol{I}_{cs})^\top \supsetneq \mathsf{ker}(\boldsymbol{O})^\top + \mathsf{ker}(\mathcal{C}_{\mathit{fix}})^\top \tag{4.2}$$
$$\iff \mathsf{ker}(\boldsymbol{I}_{cs})^\top \supsetneq \big(\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}})\big)^\top \tag{4.3}$$
$$\iff \mathsf{ker}(\boldsymbol{I}_{cs}) \subsetneq \mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}}) \tag{4.4}$$

Let $\sim_{\mathit{fix}}$ denote the equivalence relation defining the quotient $\mathbb{F}^{\mathsf{base}}/(\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}}))$. Consider the map

$$\lambda : \mathbb{F}^{\mathsf{base}} \Big/ \mathsf{ker}(\boldsymbol{I}_{cs}) \to \mathbb{F}^{\mathsf{base}} \Big/ \big(\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}})\big)$$
$$[\boldsymbol{v}]_{\sim_{cs}} \mapsto [\boldsymbol{v}]_{\sim_{fix}}.$$

To show that it is well-defined, we need to show that it is independent of the chosen representative. Let $[\boldsymbol{v}]_{\sim_{cs}} = [\boldsymbol{w}]_{\sim_{cs}}$ for arbitrary $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}^{\mathsf{base}}$. By definition $\boldsymbol{v} - \boldsymbol{w} \in \mathsf{ker}(\boldsymbol{I}_{cs})$. Using (4.4) we have $[\boldsymbol{v}]_{\sim_{\mathit{fix}}} = [\boldsymbol{w}]_{\sim_{\mathit{fix}}}$. Now we show that λ is not injective. Let $\boldsymbol{w} \in \mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}})$ but $\boldsymbol{w} \notin \mathsf{ker}(\boldsymbol{I}_{cs})$, so $[\boldsymbol{v}+\boldsymbol{w}]_{\sim_{cs}} \neq [\boldsymbol{v}]_{\sim_{cs}}$. This is possible because by (4.4) $\mathsf{ker}(\boldsymbol{I}_{cs})$ is strictly smaller than $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{\mathit{fix}})$. Then

$$\lambda([\boldsymbol{v}+\boldsymbol{w}]_{\sim_{cs}}) = [\boldsymbol{v}+\boldsymbol{w}]_{\sim_{\mathit{fix}}} = [\boldsymbol{v}]_{\sim_{\mathit{fix}}} = \lambda([\boldsymbol{v}]_{\sim_{cs}}).$$

As a consequence the concatenation $\lambda \circ \widetilde{\boldsymbol{I}_{cs}}$ is not injective. Therefore, we can find a $\boldsymbol{v}' \in \mathsf{sol}(\mathcal{C}_{cs})$ with $\boldsymbol{v}' \neq \boldsymbol{v}$ such that $[\boldsymbol{v}']_{\sim_{\mathit{fix}}} = [\boldsymbol{v}]_{\sim_{\mathit{fix}}}$. The latter is equivalent to $\boldsymbol{O}\boldsymbol{v}' = \boldsymbol{O}\boldsymbol{v}$ and $\boldsymbol{v}' - \boldsymbol{v} \in \mathsf{ker}(c)$ for all $c \in \mathcal{C}_{\mathit{fix}}$. As $\boldsymbol{v}$ is a solution to $\mathcal{C}_{\mathit{fix}}$ it follows that $\boldsymbol{v}'$ is a solution, too. Summing up, $\boldsymbol{v}' \in \mathsf{sol}(\mathcal{C}_{cs}) \cap \mathsf{sol}(\mathcal{C}_{\mathit{fix}}) = \mathsf{sol}(\mathcal{C})$ and $\boldsymbol{v}' \neq \boldsymbol{v}$.

By the bijection argument, we know that $\boldsymbol{I}\boldsymbol{v}' \neq \boldsymbol{I}\boldsymbol{v}$ and hence $\boldsymbol{I}\boldsymbol{v}'$ is a second preimage to $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$.

We argue why the second preimage is computable with $|\mathcal{C}_{cs}|$ queries. To compute such a $\boldsymbol{v}'$ we need to compute preimages of $\lambda \circ \widetilde{\boldsymbol{I}_{cs}}$. Because λ is just a linear map, the space of preimages to any element in its image can be computed without any queries to H.

We can choose a preimage $[\boldsymbol{v}']_{\sim_{cs}}$ to $[\boldsymbol{v}]_{\sim_{fix}}$ from its space of preimages arbitrarily while making sure that $[\boldsymbol{v}']_{\sim_{cs}} \neq [\boldsymbol{v}]_{\sim_{cs}}$. The inverse of the bijection $\widetilde{\boldsymbol{I}_{cs}}$ is computable with $|\mathcal{C}_{cs}|$ queries by Corollary 4.9. $\square$

We give an example of a program that has a collision structure due to the invertibility of E.

$\mathcal{P}^{\mathcal{E}}_{col}(a, b, c)$
$k_1 = c$
$x_1 = b$
$y_1 = E(k_1, x_1)$
$k_2 = a$
$x_2 = y_1$
$y_2 = E(k_2, x_2)$
return $y_1 + y_2$

Algebraic Representation

$$\begin{aligned} \boldsymbol{O} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\ \boldsymbol{k}_1 &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\ \boldsymbol{x}_1 &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{y}_1 &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \\ \boldsymbol{k}_2 &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{x}_2 &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \\ \boldsymbol{y}_2 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

We can set $\mathcal{C}_{cs} = \{c_1\}$, $\mathcal{C}_{fix} = \{c_2\}$ and $\boldsymbol{I}_{cs} = \begin{bmatrix} \boldsymbol{O} \\ \boldsymbol{k}_2 \\ \boldsymbol{x}_2 \\ \boldsymbol{k}_1 \end{bmatrix}$.

Then $\mathsf{rowsp}(\boldsymbol{I}_{cs}) = \big(\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})\big) \oplus \mathsf{rowsp}(\boldsymbol{k}_1)$, so condition (4.1) is fulfilled. $\mathcal{C}_{cs}$ is deterministically solvable fixing $\boldsymbol{I}_{cs}$. We can set $\boldsymbol{Q}_1 = \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{y}_1 \end{bmatrix}$ and $\boldsymbol{a}_1 = \boldsymbol{x}_1$, and then we have:

1. $\mathsf{rows}(\boldsymbol{Q}_1) \subseteq \mathsf{rowsp}(\boldsymbol{I}_{cs})$
2. $\boldsymbol{a}_1 \notin \mathsf{rowsp}(\boldsymbol{I}_{cs})$

This makes Lemma 4.10 applicable and the Lemma describes how to construct an attack in the form of a Linicrypt program.

In plain language, we can fix the output, set the second query and answer to the same values as for the given preimage and choose k_1 arbitrarily. This determines the key k_1 and answer y_1 of the first query to E, but not the query itself. By using D we get the unique corresponding query x_1. Then (k_2, x_1, k_1) is a second preimage.

Now will work towards the converse of this statement. That is, we are trying to answer the question: If $\mathcal{P}$ doesn't have a collision structure, is it collision resistant? We will confirm this for the case of a Linicrypt program which makes only a single query to the ideal cipher.

Definition 4.11. *A set of ideal cipher constraints $\mathcal{C}$ is called unsolvable fixing a space $\mathcal{F} \subseteq \mathbb{F}^{1\times\mathsf{base}}$ if for every ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$, we have some $c_i = (\boldsymbol{k}_i, \boldsymbol{x}_i, \boldsymbol{y}_i)$ such that both*

1. $\boldsymbol{y} \in \mathcal{F} + \mathsf{rowsp}(\{c_1, \dots, c_{i-1}\}) + \mathsf{rowsp}(\boldsymbol{k}, \boldsymbol{x})$ *and*
2. $\boldsymbol{x} \in \mathcal{F} + \mathsf{rowsp}(\{c_1, \dots, c_{i-1}\}) + \mathsf{rowsp}(\boldsymbol{k}, \boldsymbol{y})$.

Lemma 4.12. *If $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ does not have a collision structure, then for every split $\mathcal{C} = \mathcal{C}_{cs} \sqcup \mathcal{C}_{fix}$ we have one of the following:*

1. *$\mathcal{C}_{cs}$ is deterministically solvable fixing $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$*
2. *$\mathcal{C}_{cs}$ is unsolvable fixing $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$*

Proof. Assume $\mathcal{P}$ does not have a collision structure. Let $\mathcal{C} = \mathcal{C}_{cs} \sqcup \mathcal{C}_{fix}$ be an arbitrary split of the set of constraints. Then for any matrix $\boldsymbol{I}_{cs}$ with $\mathsf{rowsp}(\boldsymbol{I}_{cs}) \supsetneq \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$ we have $\mathcal{C}_{cs}$ is not deterministically solvable fixing $\boldsymbol{I}_{cs}$. We now assume condition *2.* from Lemma 4.12 not true and show that this implies *1.* from Lemma 4.12 is true. So we assume $\mathcal{C}_{cs}$ is not unsolvable fixing $\mathcal{F} = \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$. Then there is an ordering $(c_1, \dots, c_n)$ of $\mathcal{C}_{cs}$ such that for every i we have $\boldsymbol{y} \notin \mathcal{F} + \mathsf{rowsp}(\{c_1, \dots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{x}\})$ or $\boldsymbol{x} \notin \mathcal{F} + \mathsf{rowsp}(\{c_1, \dots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{y}\})$. This gives the condition *2.* of the definition of deterministically solvable. Hence, we can add some dimensions to $\mathcal{F}$ and call it $\mathcal{F}' \supset \mathcal{F}$ such that $\mathcal{C}_{cs}$ is deterministically solvable fixing $\mathcal{F}'$. But if $\mathcal{F}' \supsetneq \mathcal{F}$ this would contradict the assumption that $\mathcal{P}$ does not have a collision structure. Therefore, we know that $\mathcal{C}_{cs}$ is deterministically solvable fixing $\mathcal{F} = \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$. $\square$

Corollary 4.13. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a program making only a single call to the ideal cipher. Then $\mathcal{C} = \{(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})\}$ for some $\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^{1 \times \mathsf{base}}$. Assume $\mathcal{P}$ does not have a collision structure but there exists $\boldsymbol{v} \neq \boldsymbol{v}'$ in $\mathsf{sol}(\mathcal{C})$ with $\boldsymbol{O}\boldsymbol{v} = \boldsymbol{O}\boldsymbol{v}'$. Then the following hold:*

1. $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \{0\}$
2. $\boldsymbol{y} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\boldsymbol{k}, \boldsymbol{x})$
3. $\boldsymbol{x} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\boldsymbol{k}, \boldsymbol{y})$

Proof. If we set $\mathcal{C}_{cs} = \{\}$ then case *2.* from Lemma 4.12 cannot be true. Because $\{\}$ is deterministically solvable fixing $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C})$ we have a bijection between $\mathsf{sol}(\{\}) = \mathbb{F}^{\mathsf{base}}$ and $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C})$. Because $\mathsf{rowsp}(\boldsymbol{O}) = \mathsf{ker}(\boldsymbol{O})^{\perp}$ and $\mathsf{rowsp}(\mathcal{C}) = \mathsf{ker}(\mathcal{C})^{\perp}$ this implies $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \{0\}$.

If we set $\mathcal{C}_{cs} = \mathcal{C}$ on the other hand, then case *1.* from Lemma 4.12 cannot be true. This is because if it was, then there wouldn't exist vectors $\boldsymbol{v} \neq \boldsymbol{v}'$ in $\mathsf{sol}(\mathcal{C})$ with $\boldsymbol{O}\boldsymbol{v} = \boldsymbol{O}\boldsymbol{v}'$. Therefore, $\mathcal{C}$ is unsolvable fixing $\mathsf{rowsp}(\boldsymbol{O})$ which gives exactly $\boldsymbol{y} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{x}\})$ and $\boldsymbol{x} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{y}\})$. $\square$

Proposition 4.14. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a program making only a single call to the ideal cipher, i.e. $\mathcal{C} = \{(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})\}$ for some $\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^{1 \times \mathsf{base}}$. Assume there is an adversary $\mathcal{A}$*

making N queries to the ideal cipher winning the collision resistance security game with

$$\Pr[\mathsf{ColGame}(\mathcal{P}, \mathcal{A}, \lambda) = 1] > \frac{N(N-1)}{2(|\mathbb{F}| - N)}.$$

Then $\mathcal{P}$ has a collision structure.

This proof is based on the proofs of McQuoid, Swope and Rosulek in [MSR19, Lemma 10] and of Boneh and Shoup in [BS20, Theorem 8.4 (Davies-Meyer)].

Proof. Assume towards a contradiction that there is such an adversary $\mathcal{A}$ making N queries and that $\mathcal{P}$ does not have a collision structure. We will work with the transcript between the adversary and the ideal cipher. Let $\mathcal{T} : \{1, \ldots, N\} \to \mathbb{F}^3$ be the function defined by $\mathcal{T}(i) = (k, x, y)$, if the ith query of $\mathcal{A}$ is $E(k, x)$ with response y or if it is $D(k, y)$ with response x.

Let $\boldsymbol{i}$ and $\boldsymbol{i}'$ be the output of $\mathcal{A}$ in the event that it successfully found a collision. As $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}$, they correspond to unique vectors $\boldsymbol{v} \neq \boldsymbol{v}'$ in $\mathsf{sol}(\mathcal{C})$. We can assume the adversary actually makes the queries corresponding to these inputs, i.e. $(\boldsymbol{kv}, \boldsymbol{xv}, \boldsymbol{yv})$ and $(\boldsymbol{kv}', \boldsymbol{xv}', \boldsymbol{yv}')$ are contained in $\mathsf{im}(\mathcal{T})$. Otherwise, we can force $\mathcal{A}$ to make these queries by modifying it, so that it runs $\mathcal{P}(\boldsymbol{i})$ and $\mathcal{P}(\boldsymbol{i}')$ as its last action. Let i and j be defined by $\mathcal{T}(i) = (\boldsymbol{kv}, \boldsymbol{xv}, \boldsymbol{yv})$ and $\mathcal{T}(j) = (\boldsymbol{kv}', \boldsymbol{xv}', \boldsymbol{yv}')$. That is, at query i the adversary has determined the values involved in the query for $\mathcal{P}(\boldsymbol{i})$ and at query j it has determined the values involved in the query for $\mathcal{P}(\boldsymbol{i}')$.

Consider the case $i = j$. This means $\boldsymbol{v} - \boldsymbol{v} \in \mathsf{ker}(\mathcal{C})$. By assumption $\boldsymbol{i}$ and $\boldsymbol{i}'$ are collisions, so $\boldsymbol{v} - \boldsymbol{v}' \in \mathsf{ker}(\boldsymbol{O})$. Corollary 4.13 states $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \{0\}$, so $\boldsymbol{v} = \boldsymbol{v}'$. This is a contradiction to $\boldsymbol{i} \neq \boldsymbol{i}'$.

Therefore, we have $i \neq j$. By modifying $\mathcal{A}$ again, we can assume it outputs the value it fixes first as the first output, then $i < j$. Consider the point where $\mathcal{A}$ sent its jth query and is waiting to receive the answer from the ideal cipher. If this query was to E we define $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{k} \\ \boldsymbol{x} \end{bmatrix}$ and $\boldsymbol{a} = \boldsymbol{y}$, otherwise we define $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{k} \\ \boldsymbol{y} \end{bmatrix}$ and $\boldsymbol{a} = \boldsymbol{x}$.

Then we know from Corollary 4.13 that

$$\boldsymbol{a} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\boldsymbol{Q}), \quad \text{which means} \quad \boldsymbol{a} = \boldsymbol{\lambda O} + \boldsymbol{\gamma Q}. \tag{4.5}$$

At query j the vectors $\boldsymbol{Qv}, \boldsymbol{av}, \boldsymbol{Qv}'$ and $\boldsymbol{O}(\boldsymbol{v}' - \boldsymbol{v})$ are determined. The latter is 0 by the assumption that $\boldsymbol{i}$ and $\boldsymbol{i}'$ are a collision. Therefore, we can use (4.5) to show

$$\boldsymbol{a}(\boldsymbol{v}' - \boldsymbol{v}) = \boldsymbol{\lambda O}(\boldsymbol{v}' - \boldsymbol{v}) + \boldsymbol{\gamma Q}(\boldsymbol{v}' - \boldsymbol{v}),$$

which is equivalent to

$$\boldsymbol{av}' = \boldsymbol{av} + \boldsymbol{\gamma Qv}' - \boldsymbol{\gamma Qv}. \tag{4.6}$$

All the values on the right of (4.6) have been determined, while $\boldsymbol{av'}$ is sampled uniformly from a set of size at least $|\mathbb{F}| - j + 1$. We call the event that this equation holds Z_{ij}, then

$$\Pr[Z_{ij}] = \frac{1}{|\mathbb{F}| - j + 1}.$$

Using the union bound we have

$$\Pr[\mathsf{ColGame}(\mathcal{P}, \mathcal{A}, \lambda) = 1] \leq \sum_{j=1}^{N} \sum_{i=1}^{j-1} \Pr[Z_{ij}] \leq \sum_{j=1}^{N} \frac{j-1}{|\mathbb{F}| - j + 1} \leq \frac{N(N-1)}{2(|\mathbb{F}| - N)},$$

which gives the contradiction. □

Corollary 4.15. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a program making only a single call to the ideal cipher. Then the following are equivalent:*

1. *$\mathcal{P}$ has a collision structure.*
2. *There is an adversary making 2 queries that always finds second preimages.*
3. *There is an adversary making N queries that finds collisions with probability $N(N-1)/2(|\mathbb{F}| - N)$.*

4.3 Application: Compression schemes

The papers [BRS02] and [PGV94] have analyzed the 64 most basic constructions for a compression function based on a single call to a block cipher $\mathcal{E} = (E, D)$. These compression schemes $f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ are of the form $f(h, m) = E(a, b) + c$ for $a, b, c \in \{h, m, h+m, v\}$. Here $v \in \mathbb{F}$ is a fixed constant. As these compression schemes are often used in the Merkle-Damgård construction, one should think about h as the chaining value and m as the message block. If not for the constant v, these constructions would be Linicrypt programs. But as we will show, one can set $v = 0$ without loss of generality. Then Corollary 4.15 applies, and it determines that 12 of these constructions are collision resistant. This agrees with the results achieved by [BRS02], although they go further and identify 8 compression schemes that are not collision resistant, but for which the Merkle-Damgård construction is collision resistant.

First, we argue why we can set $v = 0$. Assume that an adversary $\mathcal{A}$ can find collisions for some f as described above in the case that $v = 0$. If $a = v = 0$, then we can replace the block cipher $E(\cdot, \cdot)$ by $E(\cdot + v', \cdot)$ for some $v' \in \mathbb{F}$. From the perspective of the adversary, this is still an ideal cipher, so its success probability could not have been affected. But this is equivalent to $\mathcal{A}$ being able to find collisions for the same compression scheme f but with the constant set to v'. If $b = 0$ we replace E with $E(\cdot, \cdot + v')$ and if $c = 0$ we replace it with $E(\cdot, \cdot) + v'$. All of these modifications to an ideal cipher E yield another ideal cipher, even when applied at the same time.

4.3.1 The PGV compression schemes in Linicrypt

We will give the algebraic representation of the compression schemes described above. First, let us write them as an ideal cipher Linicrypt program.

$\mathcal{P}(h, m)$
$y = E(ch + dm, eh + fm)$
return $ah + bm + y$

Algebraic Representation

$$\begin{aligned} \boldsymbol{O} &= \begin{bmatrix} a & b & 1 \end{bmatrix} \\ \boldsymbol{k} &= \begin{bmatrix} c & d & 0 \end{bmatrix} \\ \boldsymbol{x} &= \begin{bmatrix} e & f & 0 \end{bmatrix} \\ \boldsymbol{y} &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The base variables are $\{h, m, y\}$, and for the algebraic representation we have ordered them as (h, m, y). The algebraic representation is $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ for $\boldsymbol{I} = \begin{bmatrix} \mathbb{1}_2 & 0 \end{bmatrix}$ and $\mathcal{C} = \{(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})\}$. $\mathcal{P}$ depends on the constants $a, b, c, d, e, f \in \{0, 1\}$ (we redefine the parameters a, b, c which were used above). Each choice of these 6 binary constants corresponds to one of the compression schemes. The is, $f_{\mathcal{P}} = f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ for some compression scheme f as described above. Using Lemma 4.12 and Corollary 4.15, one can derive that $\mathcal{P}$ is collision resistant and second preimage resistant if and only if

$$\begin{bmatrix} a & b \end{bmatrix}, \begin{bmatrix} c & d \end{bmatrix}, \begin{bmatrix} e & f \end{bmatrix} \neq \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} a & b \end{bmatrix} \neq \begin{bmatrix} c & d \end{bmatrix} \neq \begin{bmatrix} e & f \end{bmatrix}. \tag{4.7}$$

There are exactly 12 schemes fulfilling this condition, which we list in Table 4.1. The authors of [BRS02] denote these as the group-1 schemes. If condition (4.7) is not met, then the compression function is susceptible to a second preimage attack needing only a single query to the ideal cipher.

4.3.2 Merkle-Damgård in Linicrypt

In the context of the Merkle-Damgård construction, the classification of the compression schemes is more nuanced. Some schemes that are broken as a compression function can be used to form collision resistant Merkle-Damgård constructions. This is because the initialization vector (IV) is fixed and cannot be chosen by the attacker. Therefore, a second preimage attack on the compression scheme is useless if it gives no control over the chaining value. In order to formalize this idea, we define the Merkle-Damgård construction in the Linicrypt model.

Definition 4.16 (Merkle-Damgård construction). *Let $\mathcal{P}$ be an ideal cipher Linicrypt program taking 2 inputs and returning one output, i.e. $f_{\mathcal{P}} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$. We define the Linicrypt program $H^n_{f_{\mathcal{P}}}$ by*

$f(h,m) =$	a	b	c	d	e	f	Name
$E(h,m)+m$	0	1	1	0	0	1	Matyas-Meyer-Oseas
$E(h,h+m)+m$	0	1	1	0	1	1	
$E(h+m,m)+m$	0	1	1	1	0	1	
$E(h+m,h)+m$	0	1	1	1	1	0	
$E(m,h)+h$	1	0	0	1	1	0	Davies-Meyer
$E(m,h+m)+h$	1	0	0	1	1	1	
$E(h+m,m)+h$	1	0	1	1	0	1	
$E(h+m,h)+h$	1	0	1	1	1	0	
$E(m,h)+h+m$	1	1	0	1	1	0	
$E(m,h+m)+h+m$	1	1	0	1	1	1	
$E(h,m)+h+m$	1	1	1	0	0	1	Miyaguchi-Preneel
$E(h,h+m)+h+m$	1	1	1	0	1	1	

Table 4.1: Parameters for the 12 secure compression schemes according to equation (4.7). These are the group-1 schemes in [BRS02]

$$
\begin{array}{|l|}
\hline
H^n_{f_{\mathcal{P}}}(m_1,\ldots,m_n) \\
\hline
\quad h_0 = 0 \\
\quad h_1 = \mathcal{P}(h_0, m_1) \\
\quad \vdots \\
\quad h_n = \mathcal{P}(h_{n-1}, m_n) \\
\quad \textbf{return } h_n \\
\hline
\end{array}
$$

We call the program $H^n_{f_{\mathcal{P}}}$ the Linicrypt Merkle-Damgård construction using $f_{\mathcal{P}}$ as its compression function.

The $H^n_{f_{\mathcal{P}}}$ in the definition is clearly a Linicrypt program as the calls to the Linicrypt program $\mathcal{P}$ could be expanded into regular Linicrypt commands. In the standard Merkle-Damgård construction, the IV is chosen to be any fixed constant. Linicrypt does not allow for fixed constants. But as 0 is a linear combination of the input variables, we can use 0 as the IV.

Another way in which the definition varies from the usual definition, is that $H^n_{f_{\mathcal{P}}}$ takes n inputs for some fixed n. Usually, the Merkle-Damgård Hash function is allowed to take any number of inputs. The Linicrypt model does not allow for variable input length. One has to consider H^n_f to be a different Linicrypt program from $H^{n'}_f$ for $n \neq n'$. Therefore, H^n_f is actually a compression function with a compression ratio of n-to-1.

4.3.3 Deriving a taxonomy of attacks from Linicrypt

The authors of [PGV94] and [BRS02] classify the 64 compression schemes by the attacks they allow, when used in the Merkle-Damgård construction. The attack types are stated in a table, but no derivation or intuition for why this structure makes sense is given. We will briefly present their categorizations of the attacks. This serves as a comparison to the classification which we will later derive from the Linicrypt model. Our classification agrees for the most part with theirs, although a few interesting differences remain.

In [BRS02], the authors classify the schemes into 3 groups. There are 12 group-1 schemes that are secure compression functions and give secure Merkle-Damgård constructions. Then they define 8 additional group-2 schemes. Although these are insecure compression functions, the Merkle-Damgård construction is still collision resistant. Finally, the remaining 44 schemes are defined as group-3, and it is stated that a collision resistance attack using at most 3 queries to the ideal cipher exists. In Figure 1 of said paper, each scheme is assigned to an attack type ranging from (a) to (g). The group-1 schemes correspond to attack types (d) and (e), while group-2 schemes are susceptible to attack type (c).

Their attack categorization agrees almost completely with the one created by [PGV94]. The capital letters, (✓) and (-) are the attack types from [PGV94].

- **Trivially weak (-) ↔ (a):** f only depends on one of its inputs.
- **Direct Attack (D) ↔ (b):** Given $o \in \mathbb{F}$ and $h \in \mathbb{F}$ one can find the corresponding $m \in \mathbb{F}$ such that $f(h, m) = o$.
- **Permutation Attack (P) ↔ (f):** The chaining value h is not used in the query to E, hence the order of inputs for H_f^n does not matter.
- **Forward Attack (F) ↔ 5 schemes from (g):** Given $o, h, m, h' \in \mathbb{F}$ one can find an $m' \in \mathbb{F}$ such that $f(h', m') = o = f(h, m)$.
- **Backward Attack (B) ↔ (c) and 5 schemes from (g):** Given $o \in \mathbb{F}$ one can find (h, m) such that $f(h, m) = o$.
- **Fixed Point Attack (FP) ↔ (e):** One can find $h, m \in \mathbb{F}$ such that $f(h, m) = h$.
- **Secure (✓) ↔ (d):** f has none of the weaknesses above.

The most interesting schemes are the ones only susceptible to a backward attack (B). 13 schemes are in this category. According to [BRS02], 8 of them are secure in the Merkle Damgard construction. For the other 5, they claim a collision can be found by at most 2 ideal cipher queries. It turns out that for these 5 schemes the collisions that can be found are only for different length messages. Some of them also rely on the fact that they only consider the field $\mathbb{F} = \{0, 1\}^n$, for which $x + x = 0$ for any $x \in \{0, 1\}^n$. This will be analyzed in more detail below.

Both papers [PGV94] and [BRS02] do not give a derivation for which schemes correspond to which attack types. In [BRS02] they proof the security of group-1 (type (d) and (e)) and group-2 schemes (type (c)). This is done by choosing a representative scheme of the attack type and performing the proof for that scheme. But, in order to convince yourself that the proof works for the other compression schemes with the same attack type, one needs to check the proof again. Ideal cipher Linicrypt has enabled us to prove collision resistance for all 12 group-1 compression functions (type (✓) and (FP) in [PGV94]) in an abstract manner. The sufficient and necessary condition (4.7) for a scheme to belong to group-1 can be easily derived from the more general Corollary 4.15. Intuitively, using the language we introduced in the previous chapters, a compression function is secure if the associated ideal cipher constraint is unsolvable fixing the output.

The ideal cipher Linicrypt model can also be used to derive the conditions for a scheme f to be of type -, D, F, or P. These attack types have in common that they allow for an easy second-preimage attack on H_f^n for $n \geq 2$. We will define 3 Linicrypt attack types:

1. Degenerate
2. Deterministically solvable fixing $\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{h})$
3. Permutation attack

We will define category 1) and 2) by deriving them from Lemma 4.12. The permutation attack, on the other hand, can only be explained from a Linicrypt perspective once you take into account interactions that can happen between different ideal cipher queries.

This categorization is guided by visual intuition. As defined above, the parameters (a, b, c, d, e, f) determine the vectors $\boldsymbol{O}, \boldsymbol{k}$ and $\boldsymbol{x}$. Each attack type corresponds can be derived from the relationship these three vectors have with each other and with the basis vectors $\boldsymbol{h}, \boldsymbol{m}, \boldsymbol{y}$. The graphical representation introduced in Figure 4.1 of a compression function might assist the visual intuition.

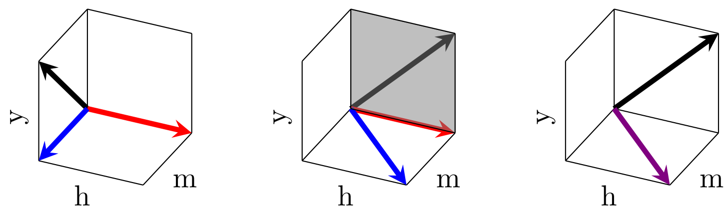


Figure 4.1: Visual representation of the Matyas-Meyer-Oseas compression function (left) and two insecure compression functions (middle, right). The arrows represent $\boldsymbol{O}$ (black), $\boldsymbol{k}$ (red) and $\boldsymbol{x}$ (blue). A violet arrow means $\boldsymbol{k} = \boldsymbol{x}$. One should keep in mind that $\mathsf{rowsp}(\mathcal{C})$ includes the vector $\boldsymbol{y}$, which is omitted as it always points up. A gray plane is sometimes drawn to highlight the fact that $\mathcal{C}$ is deterministically solvable fixing $\mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(k)$.

Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be the Linicrypt program corresponding to a scheme f which insecure as a compression function (condition (4.7) does not hold). This is equivalent to the statement: $\mathcal{P}$ has a collision structure. By definition of collision structure, there is a split $\mathcal{C} = \mathcal{C}_{cs} \sqcup \mathcal{C}_{\mathit{fix}}$ such that $\mathcal{C}_{cs}$ is deterministically solvable fixing a space

$$\mathcal{F} \supsetneq \mathsf{rowsp}(\mathcal{C}_{\mathit{fix}}) + \mathsf{rowsp}(\boldsymbol{O}). \tag{4.8}$$

As $\mathcal{C} = \{(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})\}$ contains only a single element, there are only two cases for this split.

Case 1: $\mathcal{C}_{cs} = \{\}$ and $\mathcal{C}_{\mathit{fix}} = \mathcal{C}$.

In this case, we know by definition that $\mathcal{F}$ has to be the whole space $\mathbb{F}^{1\times\mathsf{base}}$. Therefore, equation 4.8 translates to $\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) \neq \mathcal{F} = \mathbb{F}^{1\times\mathsf{base}}$. This is exactly the condition by which the authors of [MSR19] call $\mathcal{P}$ degenerate.

Definition 4.17. *Let f be one of the 64 PGV compression functions. We assign it to the Linicrypt attack category "Degenerate" if* $\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) \neq \mathbb{F}^{1\times\mathsf{base}}$.

Lemma 4.18. *(Degenerate Attack) Let f be a compression scheme from the Degenerate category. Then there is a second preimage attack on H_f^n if $n > 1$. There are 22 such compression schemes.*

Proof. Let $(m_1, \ldots, m_n)$ be an input for H_f^n. We will describe how to find a second-preimage for it. Note, that

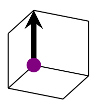
$$\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})^\top = \mathsf{ker}(\boldsymbol{O})^\perp + \mathsf{ker}(\mathcal{C})^\perp = \big(\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C})\big)^\perp.$$

Recall, that the space $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C})$ carries the following meaning: Let $\boldsymbol{w} \in \mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C})$ be arbitrary. If $\boldsymbol{v}$ is in $\mathsf{sol}(\mathcal{C})$ then $\boldsymbol{v} + \boldsymbol{w}$ is in $\mathsf{sol}(\mathcal{C})$ with $\boldsymbol{O}\boldsymbol{v} = \boldsymbol{O}(\boldsymbol{v} + \boldsymbol{w})$. In other words: $\mathcal{P}\big(\boldsymbol{I}(\boldsymbol{v} + \boldsymbol{w})\big) = \boldsymbol{O}(\boldsymbol{v} + \boldsymbol{w}) = \boldsymbol{O}\boldsymbol{v} = \mathcal{P}(\boldsymbol{I}\boldsymbol{v})$.

As f is degenerate, $\mathsf{dim}\big(\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})\big) \leq 2$. Also, $\boldsymbol{y} = \boldsymbol{e}^3 \in \mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$. Therefore, $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C})$ can only be one of 4 subspaces. We do a case separation for these 4 subspaces.

Case 1.1: $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \mathsf{rowsp}(\boldsymbol{h}, \boldsymbol{m})^\top = \mathsf{span}(\boldsymbol{e}_1, \boldsymbol{e}_2)$.

It follows that $\mathsf{rowsp}(\mathcal{C}) = \mathsf{rowsp}(\boldsymbol{O}) = \mathsf{rowsp}(\boldsymbol{e}_3)$, i.e. $\boldsymbol{k} = \boldsymbol{x} = 0$ and $\boldsymbol{O} = \boldsymbol{e}^3$. This is the scheme $f(h, m) = E(0, 0)$. The function H_f^n is constant, so every input is a collision with every other input. This is the graphical representation of its algebraic representation.

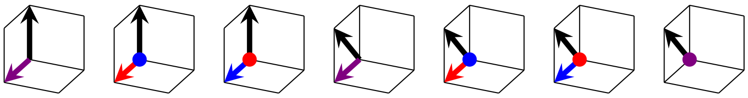


Case 1.2: $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \mathsf{rowsp}(\boldsymbol{m})^\top = \mathsf{span}(\boldsymbol{e}_2)$.

Let $\begin{bmatrix} 0 & \delta & 0 \end{bmatrix} \in \mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C})$ be arbitrary. By the argument above, we have $f(h, m) = f(h, m + \delta)$ for any $h, m \in \mathbb{F}$. We set $m_i' = m_i$ for $i < n$ and choose an $m_n' \neq m_n$. Then

$$H_f^n(m_1', \ldots, m_n') = h_n' = f(h_{n-1}, m_n') = f(h_{n-1}, m_n) = h_n = H_f^n(m_1, \ldots, m_n).$$

There are $2 \times 3 + 1 = 7$ compression schemes that fulfill the condition from his case.

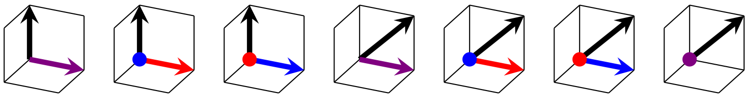


Case 1.3: $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \mathsf{rowsp}(\boldsymbol{h})^\top = \mathsf{span}(\boldsymbol{e}_1)$**.**

By the same argument as in the previous case, we get $f(h, m) = f(h', m)$ for any $h' \in \mathbb{F}$. We set $m'_i = m_i$ for $i < n-1$, $m'_{n-1} \neq m_{n-1}$ and $m'_n = m_n$. Then

$$H_f^n(m'_1, \ldots, m'_n) = h'_n = f(h'_{n-1}, m_n) = f(h_{n-1}, m_n) = h_n = H_f^n(m_1, \ldots, m_n).$$

There are $2 \times 3 + 1 = 7$ compression schemes that fulfill the condition from his case.

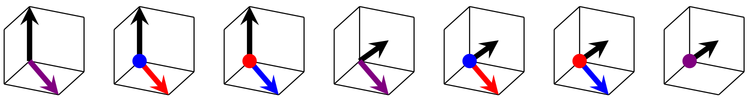


Case 1.4: $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}) = \mathsf{rowsp}(\boldsymbol{h} - \boldsymbol{m})^\top = \mathsf{span}(\boldsymbol{e}_1 - \boldsymbol{e}_2)$**.**

This implies that $f(h, m) = f(h + \delta, h - \delta)$ for any $\delta \in \mathbb{F}$. We set $m'_i = m_i$ for $i < n-1$, $m'_{n-1} \neq m_{n-1}$, $\delta = h_{n-1} - h'_{n-1}$ and $m'_n = m_n + \delta$. Then

$$h'_n = f(h'_{n-1}, m'_n) = f(h'_{n-1} + \delta, m'_n - \delta) = f(h_{n-1}, m_n) = h_n.$$

There are $2 \times 3 + 1 = 7$ compression schemes that fulfill the condition from his case.

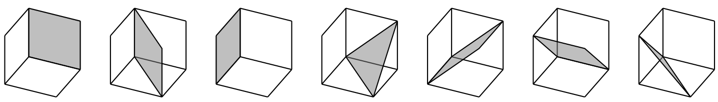


In total found $1 + 3 \times 7 = 22$ degenerate compression schemes. This completes the proof. $\square$

This completes the analysis of case 1. Now consider the other split of $\mathcal{C}$ into $\mathcal{C}_{cs} \sqcup \mathcal{C}_{fix}$.

Case 2: $\mathcal{C}_{cs} = \mathcal{C}$ **and** $\mathcal{C}_{fix} = \{\}$**.**

By the assumption that f has a collision structure, we get: $\mathcal{C}$ is deterministically solvable fixing $\mathcal{F} \supsetneq \mathsf{rowsp}(\boldsymbol{O})$. Recall, that this means one can fix $\boldsymbol{Ov}$ and some other base variables value arbitrarily and still find a $\boldsymbol{v} \in \mathsf{sol}(\mathcal{C})$. $\mathcal{F}$ has to be 2 dimensional, because of the bijection from the inputs of a program to $\mathsf{sol}(\mathcal{C})$. The 2-dimensional subspace $\mathcal{F}$ can only be one of these 7.



In the case that $\boldsymbol{h} \in \mathcal{F}$ a second preimage attack is possible. Only the first and the 5th subspace pictured above fulfill this condition.

Case 2.1: $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{O}$ and $\boldsymbol{h}$.

Definition 4.19. *Let f be one of the 64 PGV compression functions with algebraic representation $(\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$. We assign it to the Linicrypt attack category "$\langle \boldsymbol{O}, \boldsymbol{h} \rangle$-Collision Structure" if $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{O}$ and $\boldsymbol{h}$.*

Lemma 4.20. *Let f be a compression scheme from the $\langle \boldsymbol{O}, \boldsymbol{h} \rangle$-Collision Structure category. Then there is a second-preimage attack on H_f^n making a single query to the ideal cipher. Also, its attack type is (b) in [BRS02] and (D) in [PGV94]. There are 12 such compression schemes.*

Proof. Assume we are given any $o, h \in \mathbb{F}$. We show that a corresponding $m \in \mathbb{F}$ can be found such that $f(h, m) = o$. This is the definition of the direct attack (D) from [PGV94] which corresponds to attack (b) from [BRS02]. Assume as stated in the Lemma: $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}_{cs} = \begin{bmatrix} \boldsymbol{O} \\ \boldsymbol{h} \end{bmatrix}$.

By Lemma 4.8, $\boldsymbol{I}_{cs}|_{\mathsf{sol}(\mathcal{C})} : \mathsf{sol}(\mathcal{C}) \to \mathbb{F}^2$ is a bijection. Corollary 4.9 states that the preimage to (o, h) can be computed using $|\mathcal{C}| = 1$ queries. We call this preimage $\boldsymbol{v} \in \mathsf{sol}(\mathcal{C})$. By definition of $\boldsymbol{I}_{cs}$ we have $\boldsymbol{O}\boldsymbol{v} = o$ and $\boldsymbol{h}\boldsymbol{v} = h$. We set $m = \boldsymbol{m}\boldsymbol{v}$. Then $f(h, m) = f(\boldsymbol{h}\boldsymbol{v}, \boldsymbol{m}\boldsymbol{v}) = \boldsymbol{O}\boldsymbol{v} = o$ as required.

This allows for an easy second-preimage attack on H_f^n. Let $(m_1, \ldots, m_n)$ be an input to H_f^n. We set $m_i' = m_i$ for $i < n - 1$ and $m_{n-1}' \neq m_{n-1}$. According to the argument above, we can compute an m_n', such that

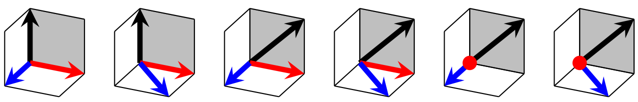
$$H_f^n(m_1', \ldots, m_n') = h_n' = f(h_{n-1}', m_n') = h_n = H_f^n(m_1, \ldots, m_n).$$

We will now list and count the schemes described by the Lemma. First, note that $\boldsymbol{k} \in \mathsf{span}(\boldsymbol{O}, \boldsymbol{h})$ by definition of deterministically solvable. Therefore, $\boldsymbol{k} = 0$ or $\boldsymbol{k} = \boldsymbol{h}$. We can derive these two implications:

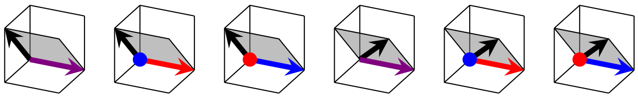
$$\boldsymbol{k} \neq 0 \implies \mathcal{C} \text{ deterministically solvable fixing } \boldsymbol{O}, \boldsymbol{k} \tag{4.9}$$

$$\boldsymbol{k} = 0 \implies \mathcal{C} \text{ deterministically solvable fixing } \boldsymbol{O}, \boldsymbol{y} \text{ or fixing } \boldsymbol{O}, \boldsymbol{x} \tag{4.10}$$

Assume $\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{h}) = \mathsf{rowsp}(\boldsymbol{y}, \boldsymbol{h})$. Then these 6 schemes correspond to this subcase:



Now assume $\mathsf{rowsp}(\boldsymbol{O}, \boldsymbol{h}) = \mathsf{rowsp}(\boldsymbol{y} + \boldsymbol{m}, \boldsymbol{h})$. Then these 6 schemes correspond to this subcase:



Altogether these are 12 compression schemes. □

At this point, out of the 42 schemes with a collision structure, 18 are left. Unfortunately, the categorization of these does not follow directly from the Linicrypt model as the first two categories do. This is because the attacks on them rely on causing interactions between different queries. Neither our security proof, which is restricted to single query constructions nor the proof from [MSR19, Theorem 1], which is restricted to constructions using distinct nonces, can explain these behaviors.

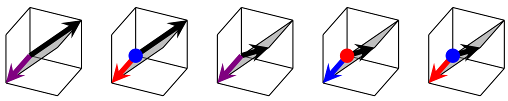
Consider the Permutation Attack from [PGV94]. It can be described by the following condition on the algebraic representation of f.

Case 2.2: $\boldsymbol{h}^\top \in \mathsf{ker}(\mathcal{C})$.

This case overlaps with both Case 1 and Case 2.1. $\boldsymbol{h}^\top \in \mathsf{ker}(\mathcal{C})$ equivalent to setting the parameters c and e to 0. That is, $f(h, m) = ah + f_2(m)$ for some function f_2 and $a \in \{0, 1\}$. Then

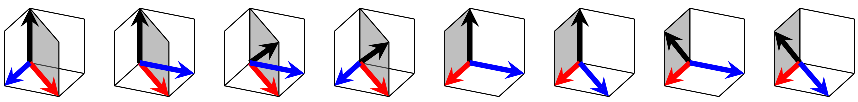
$$H_f^n(m_1, \dots, m_n) = f(h_{n-1}, m_n) = ah_{n-1} + f_2(m_n) = \cdots = a^n h_0 + \sum_{i=1}^{n} a^{n-i} f_2(m_i).$$

Therefore, $H_f^n(m_0, \dots, m_n) = H_f^n(m_{\pi(0)}, \dots, m_{\pi(0)})$ for a permutation π of $\{1, \dots, n\}$. If $n \geq 1$ this allows for a second preimage attack which does not need any queries to the ideal cipher. One can identify $2^{6-2} = 16$ schemes f such that H_f^n is susceptible to this permutation attack. The 5 schemes that meet this requirement and have not yet been categorized are:



Case 2.3: None of the above cases.

Finally, we are left with 13 schemes. These are exactly the 13 schemes of the type Backward Attack (B) from [PGV94]. 8 of those schemes have been shown to be secure if composed in the Merkle-Damgård construction by [BRS02]. They call them the group-2 schemes with attack type (c). This is the visual representation corresponding to them:



The 5 remaining schemes are marked with attack type (g). That means, that the authors [BRS02] identified a collision attack making 2 or fewer queries. These collisions are special, as they can be produced only for different input lengths. Take for example the compression scheme $f(h, m) = E(0, h+m)+m$. Let n be fixed. For $i \le n$ we define the message blocks:

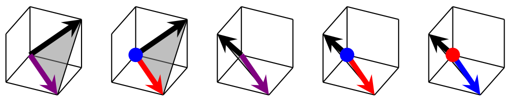
$$m_i = \begin{cases} 0 & \text{for odd } i \\ -E(0,0) & \text{for even } i \end{cases}$$

Then the Linicrypt Merkle-Damgård construction computes to:

$$H_f^n(m_1, \dots, m_n) = \begin{cases} E(0,0) & \text{for odd } n \\ 0 & \text{for even } n \end{cases}$$

All the collisions we found are then: $H_f^n(m_1, \dots, m_n) = H_f^{n-2}(m_1, \dots, m_{n-2})$.

For the two schemes where $\boldsymbol{O} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ the collisions are very similar in structure, if one assumes the field $\mathbb{F}$ has characteristic 2, i.e. $x + x = 0$ for any $x \in \mathbb{F}$.



To conclude this section we have summarized the comparison of the Linicrypt attack taxonomy and the previous taxonomies in Table 4.2. From the derivation, it is clear that the subspaces for which $\mathcal{C}$ is deterministically solvable contain a lot of information about the security properties of the scheme.

Linicrypt Attack	BRS	PGV	#	$\mathrm{SPR}\mathsf{adv}^{\mathrm{ic}}(\mathcal{A}, H_f^n)$
Degenerate	a	-	15	1
	b	D	2	1
	g	F	5	1
$\langle \boldsymbol{O}, \boldsymbol{h} \rangle$-Collision Structure	b	D	12	1
Permutation attack	f	P	5	1
Secure	d	✓	4	$\leq N/\lvert\mathbb{F}\rvert$
	e	FP	8	$\leq N/\lvert\mathbb{F}\rvert$
Other Collision Structure	c	B	8	
	g	B	5	

Table 4.2: Comparison of the classification derived by the Linicrypt model and the ones found by [PGV94] and [BRS02]. The f in H_f^n stands for the associated function to one of the compression schemes in that row. The $\mathcal{A}$ denotes the best possible adversary. This table, including the Linicrypt attack type, has been generated automatically. The code is accessible in [Sem22].

Chapter 5

Limitations and Future Work

5.1 On the permutation attack and the 5 remaining schemes

The attacks on the collision resistance of the schemes from Case 2.2 and the schemes marked with both attack types (g) and (B) are different in nature than the attacks on those in the categories Degenerate or $\langle \boldsymbol{O}, \boldsymbol{h} \rangle$-Collision Structure. They make use of the fact that the block cipher takes no nonces, and hence independent oracle calls can be switched around or made to collapse.

Consider a compression function f such that the Permutation Attack applies and some message $\boldsymbol{m} = (m_0, \ldots, m_n)$. Let $\pi\boldsymbol{m} := (m_{\pi(1)}, \ldots, m_{\pi(n)})$ for some permutation π of $\{1, \ldots, n\}$. The queries made in the execution of $H(\boldsymbol{m})$ are the same as those for $H(\pi\boldsymbol{m})$, just in a different order. This attack can be described in Linicrypt.

Lemma 5.1 (Permutation Attack)**.** *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a (standard or ideal cipher) Linicrypt program. Assume there exists a basis change $\boldsymbol{B} \neq 1$ such that $\boldsymbol{O} = \boldsymbol{O}\boldsymbol{B}$ and $\mathcal{C} = \mathcal{C}\boldsymbol{B}$. Let $\boldsymbol{v} \in \mathsf{sol}(\mathcal{C})$ with $\boldsymbol{B}\boldsymbol{v} \neq \boldsymbol{v}$. Then $\boldsymbol{I}\boldsymbol{B}\boldsymbol{v}$ is a second preimage to $\boldsymbol{I}\boldsymbol{v}$.*

Proof. The basis change $\boldsymbol{B}$ is a bijection from $\mathsf{sol}(\mathcal{C}\boldsymbol{B}) = \mathsf{sol}(\mathcal{C})$ to $\mathsf{sol}(\mathcal{C})$. Using the assumptions from the Lemma this means that $\boldsymbol{B}\boldsymbol{v}$ is in $\mathsf{sol}(\mathcal{C})$. Also, $\boldsymbol{O}\boldsymbol{B}\boldsymbol{v} = \boldsymbol{O}\boldsymbol{v}$. By the definition of the algebraic representation we have $\mathcal{P}(\boldsymbol{I}\boldsymbol{v}) = \boldsymbol{O}\boldsymbol{v} = \boldsymbol{O}\boldsymbol{B}\boldsymbol{v} = \mathcal{P}(\boldsymbol{I}\boldsymbol{B}\boldsymbol{v})$. Lemma 3.9 states that $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is a bijection. Therefore $\boldsymbol{I}\boldsymbol{B}\boldsymbol{v} \neq \boldsymbol{I}\boldsymbol{v}$ follows from $\boldsymbol{B}\boldsymbol{v} \neq \boldsymbol{v}$. $\square$

In the context of second preimage resistance and collision resistance, the extra condition in the Lemma that $\boldsymbol{B}\boldsymbol{v} \neq \boldsymbol{v}$ is not a strong restriction. One can expect $\mathsf{span}\big(\mathsf{sol}(\mathcal{C})\big) = \mathbb{F}^{\mathsf{base}}$. If $\boldsymbol{B} \neq 1$ then its eigenspace has dimension $\leq \mathsf{base} - 1$. Therefore choosing a random $\boldsymbol{v} \in \mathsf{sol}(\mathcal{C})$, which equates to choosing a random input, has a very low probability of landing in the eigenspace of $\boldsymbol{B}$, i.e. $\boldsymbol{B}\boldsymbol{v} \neq \boldsymbol{v}$.

Now let f be one of the 5 compression functions for which we found collisions only for different input lengths. In order to be concrete, let us consider the same example as before: $f(h,m) = E(0, h+m) + m$. Let $\boldsymbol{m} = (m_1, \ldots, m_n)$ be the message we have defined before that causes the collisions. In the execution of $H_f^n(\boldsymbol{m})$ only a single independent query is made, that is $E(0,0)$. By carefully choosing an input, we have caused the n different queries to collapse to a single one. The authors of [MSR19] discuss this issue in the section "Why the Restriction to Distinct Nonces". They give the example program $\mathcal{P}(x,y) = H(H(x)) - H(y)$. For this example, setting $x = y$ causes the program to "collapse" to $\mathcal{P}(x) = 0$, clearly a program that is degenerate. In our case, setting $\boldsymbol{m}$ in that specific way, causes the program to collapse to $H_f^n() = 0$ for even n and $H_f^n() = E(0,0)$ for odd n. We put the phrase "collapse a program" in quotes here, as this needs to be rigorously defined. Such a collapse attack can probably be defined similarly to the permutation attack.

5.2 Collision Resistance for Linicrypt with repeated nonces

The missing pieces in the derivation of the attack taxonomy for the 64 PGV compression functions hints towards further attack types apart from the collision structure attack. Maybe a further generalization of the collision structure attack could contain the attacks described above as a special case. We believe that this is an interesting area for future research towards solving collision resistance for Linicrypt programs that use repeated nonces.

We identify these two simple programs which are not collision resistant, but which exemplify the attacks described above.

$\mathcal{P}_{\text{collapse}}(x,y)$
return $H(x) - H(y)$

Algebraic Representation

$$\begin{aligned} \boldsymbol{O} &= \begin{bmatrix} 0 & 0 & 1 & -1 \end{bmatrix} \\ \boldsymbol{q}_1 &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{a}_1 &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \\ \boldsymbol{q}_2 &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \\ \boldsymbol{a}_2 &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$\mathcal{P}_{\text{permutation}}(x,y)$
return $H(x) + H(y)$

Algebraic Representation

$$\begin{aligned} \boldsymbol{O} &= \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \\ \boldsymbol{q}_1 &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{a}_1 &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \\ \boldsymbol{q}_2 &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \\ \boldsymbol{a}_2 &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

All inputs of the form (x,x) collide under $\mathcal{P}_{\text{collapse}}$. In contrast to that, every (x,y) is a collision with (y,x) under $\mathcal{P}_{\text{permutation}}$ if $x \neq y$. A general characterization of collision resistance would have to take these two examples into account.

Must do:

1. Intro for chapter 5
2. Ideal cipher security proposition needs SPR statement.
3. Summary table should contain both SPR advantage and CR advantage?
4. State MD security of Secure schemes comes from standard argument

Want to do:

1. Properly define the collapse attack
2. Prove that the special compression functions (type (f) and the 5 real (g)) fall into permutation attack and collapse attack
3. Conjecture for no nonces theorem
4. Use Linicrypt to get bounds on compression ratio, how many ideal cipher calls per message block are necessary. At a certain point, a collision structure can be shown to exist. Collapse attacks and permutation attacks could cause this bound to not be tight. Compare this with Stams bound.

Bibliography

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. "Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV". In: 2002, pp. 320–335. DOI: `10.1007/3-540-45708-9_21`.

[BS20] Dan Boneh and Victor Shoup. "Course in Applied Cryptography". In: 2020.

[CR16] Brent Carmer and Mike Rosulek. "Linicrypt: A Model for Practical Cryptography". In: 2016, pp. 416–445. DOI: `10.1007/978-3-662-53015-3_15`.

[HRR22] Tommy Hollenberg, Mike Rosulek, and Lawrence Roy. *A Complete Characterization of Security for Linicrypt Block Cipher Modes*. Cryptology ePrint Archive, Report 2022/1033. `https://eprint.iacr.org/2022/1033`. 2022.

[MSR19] Ian McQuoid, Trevor Swope, and Mike Rosulek. "Characterizing Collision and Second-Preimage Resistance in Linicrypt". In: 2019, pp. 451–470. DOI: `10.1007/978-3-030-36030-6_18`.

[PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. "Hash Functions Based on Block Ciphers: A Synthetic Approach". In: 1994, pp. 368–378. DOI: `10.1007/3-540-48329-2_31`.

[Sem22] Frederik Semmel. *Code for Modeling the Ideal Cipher in Linicrypt*. Version 0.1. Sept. 2022. URL: `https://github.com/frederiksemmel/linicrypt_ideal_cipher_experiments`.

[Sha49] Claude E. Shannon. "Communication theory of secrecy systems". In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.