ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling the Ideal Cipher in Linicrypt

Master Thesis

Frederik Semmel

August 24, 2022

Advisors: Fabio Banfi, Ueli Maurer
Institute of Theoretical Computer Science, ETH Zürich

Abstract

Todo

Contents

Contents **ii**

1 Introduction **1**

2 Preliminaries **2**
2.1 Linicrypt 2
2.1.1 Definition of a Linicrypt program 2
2.1.2 Type of Adversaries 3
2.1.3 Algebraic Representation 3
2.1.4 Characterizing Collision Resistance in Linicrypt 4

3 Revisiting Algebraic Representations **6**
3.1 Revisiting Collision Structures 12

4 Adapting Linicrypt to the ideal cipher model **16**
4.0.1 Algebraic representation for ideal cipher Linicrypt 16
4.1 Collision Structure 19
4.2 Application: Compression schemes 24
4.2.1 The PGV compression schemes in Linicrypt 25

Chapter 1

Introduction

Todo

Chapter 2

Preliminaries

2.1 Linicrypt

2.1.1 Definition of a Linicrypt program

The Linicrypt model for cryptographic constructions was introduced by Cramer & Rosulek in [**?**]. 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 allowed operations to create an intermediate variable are:

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

The program $\mathcal{P}$ can output one or more of its variables.

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.

$$\boxed{\begin{array}{l}\mathcal{P}^H(x,y)\\ \hline\quad r \leftarrow\$\, \mathbb{F}\\ \quad \textbf{return } H(x+r)+y\end{array}} \quad \rightsquigarrow \quad \boxed{\begin{array}{l}\mathcal{P}^H(x,y)\\ \hline\quad v_1 \coloneqq x\\ \quad v_2 \coloneqq y\\ \quad v_3 \leftarrow\$\, \mathbb{F}\\ \quad v_4 \coloneqq v_1 + v_3\\ \quad v_5 \coloneqq H(v_4)\\ \quad v_6 \coloneqq v_5 + v_2\\ \quad \textbf{return } (v_6)\end{array}}$$

2.1.2 Type of Adversaries

The Linicrypt model only imposes computational restrictions on the constructions, not on the adversaries. Usually one considers arbitrary adversaries $\mathcal{A}$ that are computationally unbounded, but have bounded access to the random oracle H. Therefore the behaviour of an adversary is typically described in terms of the number of queries it makes.

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. Let $\mathcal{P}$ be a linicrypt program with intermediate variables $v_1, \ldots, v_n$. These are sorted in the order in which they are created in the program.

A **base variable** is an intermediate variable which was created by retrieving an input, calling the random oracle H or sampling from $\mathbb{F}$. These are special, because they can be independent of each other. Let base be the number of base variables. We denote by $\boldsymbol{v}_{\mathsf{base}} \in \mathbb{F}^{\mathsf{base}}$ the column vector consisting of the values that the base variables take in a specific excecution of $\mathcal{P}$. Here, we order the base variables the same way as the intermediate variables. A **derived variable** is one which is created by performing a linear combination of existing itermediate variables. Note, that derived variables are therefore linear combinations of base variables. As base variables are mostly independent of each other, it makes sense to *model them as linearly independent vectors in* $\mathbb{F}^{\mathsf{base}}$. The derived variables are then modeled by linear combinations of these vectors.

Let v_i be an intermediate variable. We define the **associated vector** $\boldsymbol{v}_i$ to be the unique row vector such that for every excecution of $\mathcal{P}$ base variables taking the values $\boldsymbol{v}_{\mathsf{base}}$, the variable v_i has the value $\boldsymbol{v}_i \boldsymbol{v}_{\mathsf{base}}$. For example, if v_i is the j'th base variable, then $\boldsymbol{v}_i = \begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \end{bmatrix}$, where the 1 is in the j'th position. We follow the convention 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 $\boldsymbol{O}\boldsymbol{v}_{\mathsf{base}} = \begin{bmatrix} o_1 & \cdots & o_k \end{bmatrix}^\top$. 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 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 vector.

The input and output matrix describe the linear correlations between input and output of the program with respect to it's base variables. But the base variables are not completely independent of each other. In fact, 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 the requirement that $\boldsymbol{a}_i \boldsymbol{v}_{\mathsf{base}} = H(t_i, \boldsymbol{Q}_i \boldsymbol{v}_{\mathsf{base}})$. We denote the set of all (associated) oracle constraints of $\mathcal{P}$ by $\mathcal{C}$.

As we want the base variables to be linearly indepentent from 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 reprensentation: We assume wlog that no two constraints in $\mathcal{C}$ share the same $(t, \boldsymbol{Q})$.

Wrapping up these definitions, we define the **algebraic representation** of $\mathcal{P}$ 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 behaviour of $\mathcal{P}$ completely?

The answer is yes, because the algebraic representation does not loose 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 excecuted. 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 randomly sampled values.

The authors of the original paper on Linicrypt [**?**] establish this result for inputless programs

2.1.4 Characterizing Collision Resistance in Linicrypt

In a paper by I. McQuoid, T. Swope and M. Rosulek [**?**, Characterizing Collision and Second-Preimage Resistance in Linicrypt], the authors introduced a characterization

of collision resistance and second-preimage resistance for a class of Linicrypt program based on the algebraic representation.

They identified two reasons why a *deterministic* Linicrypt $\mathcal{P}$ program can fail to be second-preimage resistant:

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

$\mathcal{P}^H_{\text{deg}}(x, y)$

$v \coloneqq x + y$
return $H(v)$

$\mathcal{P}^H_{\text{cs}}(x, y)$

$w \coloneqq x + y$
return $H(w) + x$

The precise definition of degenerate and collision structure will be discussed in section **??**. 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 [**?**] is that they show that the converse of this is also true, for Linicrypt programs which use distinct nonces in each call to the random oracle. That is, if a Linicrypt program is not collision resistant, then it either has a collision structure, or it is degenerate.

Furthermore, checking for degeneracy and existence of a collision structure can be done efficiently.

Chapter 3

Revisiting Algebraic Representations

In the previous section we have 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 $\boldsymbol{v}_{\mathsf{base}} \in \mathbb{F}^{\mathsf{base}}$ with $\boldsymbol{I}\boldsymbol{v}_{\mathsf{base}} = \begin{bmatrix} i_1 & \cdots & i_k \end{bmatrix}^\top$ for some arbitrary input, the program can compute the rest of the base variables and output $\boldsymbol{O}\boldsymbol{v}_{\mathsf{base}} = \begin{bmatrix} o_1 & \cdots & o_l \end{bmatrix}^\top$. To simplify the notation we will sometimes write $(v_1, \ldots, v_k)$ for the column vector $\begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}^\top$.

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 $\boldsymbol{O}\boldsymbol{v}_{\mathsf{base}}$ 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 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 $\boldsymbol{a}\boldsymbol{v} = H(t, \boldsymbol{Q}\boldsymbol{v})$ 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 [**?**].

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 it's 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 $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$.

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 [**?**].

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*

$$\mathcal{C}\boldsymbol{B} \coloneqq \left\{ (t, \boldsymbol{Q}\boldsymbol{B}, \boldsymbol{a}\boldsymbol{B}) \mid (t, \boldsymbol{Q}, \boldsymbol{a}) \in \mathcal{C} \right\}.$$

*If $\boldsymbol{B}$ is invertible, we say $\mathcal{P}\boldsymbol{B} \coloneqq (\boldsymbol{I}\boldsymbol{B}, \boldsymbol{O}\boldsymbol{B}, \mathcal{C}\boldsymbol{B})$ is a **pseudo algebraic representation** of $\mathcal{P}$.*

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

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

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 which computes solutions to $\mathcal{C}$. The degeneracy and collision structure attack described in [**?**] 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. $\mathsf{rows}(\boldsymbol{Q}_i) \subseteq \mathsf{rowsp}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$
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 [**?**].

Assume there is such an ordering as in the Lemma. We will construct the program $\mathcal{P}$. First $\mathcal{P}$ will store it's 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 $v_{k+i} = H(q_1, \ldots, q_{k+i-1})$. Condition 1. from the Lemma ensures that $q_1, \ldots, q_{k+i-1}$ 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*** *fixing $\boldsymbol{I}$ if there exists an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ such that for all $i = 1, \ldots, n$:*

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

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 $\mathsf{rows}(\boldsymbol{I})$).

This is similar to the definition of collision structure in [**?**]. 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 $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$ for an arbitrary input $i_1, \ldots, i_k \in \mathbb{F}$, 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 $\mathcal{C} = \{c_1, \ldots, c_n\}$, 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 $\mathsf{rows}(I) = \{\boldsymbol{i}_1, \ldots, \boldsymbol{i}_k\}$ to the input variables are linearly independent, hence it follows that $\{\boldsymbol{i}_1, \ldots, \boldsymbol{i}_k, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ 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. This charaterization up to basis change is not a problem, if all security characterization making use of the algebraic representation are invariant under basis change.

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 $(c_1, \ldots, c_n)$ 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 $(\boldsymbol{i}_1, \ldots, \boldsymbol{i}_k, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, 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 i'th canonical row vector in $\mathbb{F}^{\mathsf{base}}$. By definition of $\boldsymbol{B}$ we have for $i = 1, \ldots, n$

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

which is equivalent to

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

Additionally, as $\mathcal{C}$ is solvable via the ordering $(c_1, \ldots, c_n)$, we know that

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

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

In order to abstract the questions about deterministic Linicrypt programs to the level of the algebraic representation, we need the following Lemma and Corollary.

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, $\mathcal{P} = (\boldsymbol{IB}, \boldsymbol{B}, \mathcal{C}\boldsymbol{B})$. 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 $\boldsymbol{Iv} = \boldsymbol{Iv}'$. 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. □

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 $(i_1, \ldots, i_k)$ 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 $\boldsymbol{IBv} = (i_1, \ldots, i_k)$. Also $\mathcal{P}$ outputs $\boldsymbol{Bv}$, so $f_{\mathcal{P}}\big((i_1, \ldots, i_k)\big) = \boldsymbol{Bv}$. Therefore

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

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. □

The benefit of this bijection is that every solution of $\mathcal{C}$ corresponds to a unique input. Therefore 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 $(i_1, \ldots, i_k)$ gives output $(o_1, \ldots, o_l)$.*
2. *There exists a $\boldsymbol{v}$ solving $\mathcal{C}$ such that $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$ and $\boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$.*

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 $\boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k)$ 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 $\boldsymbol{I}\boldsymbol{v}' = (i_1, \ldots, i_k) = \boldsymbol{I}\boldsymbol{v}$. By Lemma 3.9 $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is bijective, so we have $\boldsymbol{v} = \boldsymbol{v}'$. Running $\mathcal{P}$ on $(i_1, \ldots, i_k)$ gives output $\boldsymbol{O}\boldsymbol{v}' = \boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$. □

3.1 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 which 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 $(\boldsymbol{O}\boldsymbol{B}, \boldsymbol{I}\boldsymbol{B}, \mathcal{C}\boldsymbol{B})$. Note, that $\boldsymbol{O}$ and $\boldsymbol{I}$ have the same number of rows, because both $\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ and $\mathsf{rows}(\boldsymbol{O}) \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ 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:

$$\begin{gathered}\boldsymbol{v} \text{ solves } \mathcal{C} \text{ with } \boldsymbol{I}\boldsymbol{v} = (i_1, \ldots, i_k) \text{ and } \boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l) \\ \Updownarrow \\ \boldsymbol{B}^{-1}\boldsymbol{v} \text{ solves } \mathcal{C}\boldsymbol{B} \text{ with } \boldsymbol{O}\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{v} = (o_1, \ldots, o_l) \text{ and } \boldsymbol{I}\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{v} = (i_1, \ldots, i_k)\end{gathered}$$

Combining this equivalence with the one from Lemma 3.11 we get: Running $\mathcal{P}$ on input $i_1, \ldots, i_k$ giving output $o_1, \ldots, o_l$ is equivalent to running $\mathcal{P}^{-1}$ on input $o_1, \ldots, o_l$ with output $i_1, \ldots, i_k$. 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 $\boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$. The difference is that here we consider the easier case (from an attackers perspective): We already have a solution $\boldsymbol{v}'$ with $\boldsymbol{O}\boldsymbol{v}' = (o_1, \ldots, o_l)$ and only require a different one.

The definition of collision structure in Lemma 3.13 is sligthy more general than the one from [**?**]. 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 [**?**].

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 $\boldsymbol{I}\boldsymbol{v}' = (i_1, \ldots, i_k)$ and $\boldsymbol{O}\boldsymbol{v}' = (o_1, \ldots, o_l)$. The goal is to find a different $\boldsymbol{v}$ solving $\mathcal{C}$ with $\boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$. 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 $\mathcal{C}_{\mathit{fix}}$, hence $\boldsymbol{v}$ solves $\mathcal{C} = \mathcal{C}_{cs} \cup \mathcal{C}_{\mathit{fix}}$.

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 [**?**].

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

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

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 transformation $\boldsymbol{B}$ 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:

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

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.

4.0.1 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.

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 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 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 which $\mathcal{P}$ 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 which 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 $c = (\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$:

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

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

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

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

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 actually 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 section 3.11 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. □

4.1 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:

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

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

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

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

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

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 $[\boldsymbol{v}]_{\sim_{fix}} = [\boldsymbol{w}]_{\sim_{fix}}$. Now we show that λ is not injective. Let $\boldsymbol{w} \in \mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{fix})$ 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 $\mathsf{ker}(\boldsymbol{O}) \cap \mathsf{ker}(\mathcal{C}_{fix})$. Then

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

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 $[\boldsymbol{v}']_{\sim_{fix}} = [\boldsymbol{v}]_{\sim_{fix}}$. The latter is equivalent to $\boldsymbol{O}\boldsymbol{v}' = \boldsymbol{O}\boldsymbol{v}$ and $\boldsymbol{v}' - \boldsymbol{v} \in \mathsf{ker}(c)$ for all $c \in \mathcal{C}_{fix}$. As $\boldsymbol{v}$ is a solution to $\mathcal{C}_{fix}$ it follows that $\boldsymbol{v}'$ is a solution, too. Summing up, $\boldsymbol{v}' \in \mathsf{sol}(\mathcal{C}_{cs}) \cap \mathsf{sol}(\mathcal{C}_{fix}) = \mathsf{sol}(\mathcal{C})$ 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 it's 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. □

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

$\mathcal{P}^{\mathcal{E}}_{\mathsf{col}}(a,b,c)$
$k_1 = c$
$x_1 = b$
$y_1 = E(k_1, x_1)$
$k_2 = a$
$x_2 = y_1$
$y_2 = E(k_2, x_2)$
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. $\boldsymbol{y} \in \mathcal{F} + \mathsf{rowsp}(\{c_1, \ldots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{x}\})$ *and*
2. $\boldsymbol{x} \in \mathcal{F} + \mathsf{rowsp}(\{c_1, \ldots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{y}\})$.

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 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 $(c_1, \ldots, c_n)$ of $\mathcal{C}_{cs}$ such that for every i we have $\boldsymbol{y} \notin \mathcal{F} + \mathsf{rowsp}(\{c_1, \ldots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{x}\})$ or $\boldsymbol{x} \notin \mathcal{F} + \mathsf{rowsp}(\{c_1, \ldots, c_{i-1}\}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{y}\})$. This gives the condition *2.* of the definition of deterministically solvable. Therefore we can add some dimensions $\mathcal{F}$ 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 $\mathcal{F} = \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\mathcal{C}_{fix})$. □

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 exists 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 $\boldsymbol{x} \in \mathsf{rowsp}(\boldsymbol{O}) + \mathsf{rowsp}(\{\boldsymbol{k}, \boldsymbol{y}\})$. □

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 [**?**, Lemma 10] and of [**?**, 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 it's 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 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 vectors 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 $Z_{i,j}$, then

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

Using the union bound we have

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

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 $N(N-1)/2(|\mathbb{F}| - N)$.*

4.2 Application: Compression schemes

Outline

1. Introduction to the 64 compression schemes
2. Trick with $v = 0$ to model constants and why it is equivalent
3. Model 64 compression schemes in Linicrypt
4. Explain that the 12 secure schemes are the ones without a collision structure

The papers [**?**] and [**?**] 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 [**?**], although they go further and identify 8 compression schemes that are not collision resistant, but which become collision resistant when iterated as in the Merkle-Damgård construction.

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 $a = 0$, then we can replace the

block cipher $E(\cdot,\cdot)$ by $E(\cdot + v', \cdot)$ for some $v' \in \mathbb{F}$. From the adversaries perspective 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 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 $E(\cdot,\cdot) + v'$. Both of these modifications to an ideal cipher E yield another ideal cipher.

4.2.1 The PGV compression schemes in Linicrypt

We will give the algebraic representation 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 $a, b, c, d, e, f \in \{0, 1\}$. Each choice of these 6 binary constants corresponds to one of the compression schemes $f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$. Using Lemma 4.12 and Corollary 4.15, one can derive that $\mathcal{P}$ is collision 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 [**?**] denote these as the group-1 schemes.

$f(h,m) =$	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 (4.7)

.

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.