ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling the Ideal Cipher in Linicrypt

Master Thesis

Frederik Semmel

July 31, 2022

Advisors: Fabio Banfi, Ueli Maurer

Institute of Theoretical Computer Science, ETH Zürich

Abstract

Todo

Contents

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 $\boldsymbol{a}_i \boldsymbol{v}_{\mathsf{base}} = H\left(t_i, \boldsymbol{Q}_i \boldsymbol{v}_{\mathsf{base}}\right)$. 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 such 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

In chapter 3 we will discuss the

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 chapter **??**, in a variation that is adapted to the goals of this thesis. 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

Linicrypt with Ideal Ciphers

3.1 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 also write $(i_1, \ldots, i_k)$ for the column vector $\begin{bmatrix} i_1 & \cdots & i_k \end{bmatrix}^\top$.

Our goal for this section is to find a better understanding of 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 the structure 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 $\boldsymbol{a} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$. If the nonce is the empty string, then we just write $(\boldsymbol{Q}, \boldsymbol{a})$ instead of $(t, \boldsymbol{Q}, \boldsymbol{a})$. We call t the nonce and refer to $\boldsymbol{Q}$ as the query matrix and to $\boldsymbol{a}$ as answer matrix. 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 plays well 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 equivalences due to 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 $\boldsymbol{B} : \mathsf{sol}(\mathcal{C}\boldsymbol{B}) \to \mathsf{sol}(\mathcal{C})$ 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 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. $\mathsf{rows}(\boldsymbol{Q}_i) \in \mathsf{span}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{k+i-1})$
2. $\boldsymbol{a}_i = \boldsymbol{e}_{k+i}$

Proof. This is only a sketch 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 garantuees that the variable v_{k+i} hasn't been used previously, so the command is a valid Linicrypt commmand.

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

We now generalizing the condition from the 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 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 $i = 1, \ldots, n$:*

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

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 (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, for some i, $\boldsymbol{a}$ is the i'th canonical basis row vector, and $\boldsymbol{Q}$ only has only zeros in the columns 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 equivalence up to basis change is not a problem, because

any security characterization we make based off of the algebraic representation will always be 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 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

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

This shows that $(\boldsymbol{IB}, \boldsymbol{OB}, \mathcal{C}\boldsymbol{B})$ is a straight-line algebraic representation, 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 $\boldsymbol{B}$. The associated function $f_{\mathcal{P}} : \mathbb{F}^k \to \mathbb{F}^{\mathsf{base}}$ is injective, as the input variables are base variables. Note, that by definition of $\mathcal{P}$ we know that $f_{\mathcal{P}}$ actually maps to $\mathsf{sol}(\mathcal{C}\boldsymbol{B})$. Hence $|\mathbb{F}^k| \leq |\mathsf{sol}(\mathcal{C}\boldsymbol{B})|$. Because $\boldsymbol{v}$ solves $\mathcal{C}\boldsymbol{B}$ is equivalent to $\boldsymbol{B}\boldsymbol{v}$ solves $\mathcal{C}$, we also know that $|\mathsf{sol}(\mathcal{C}\boldsymbol{B})| = |\mathsf{sol}(\mathcal{C})|$. Putting this together we find that $|\mathbb{F}^k| \leq |\mathsf{sol}(\mathcal{C})|$.

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 we $\mathcal{C}$ is deterministically solvable fixing $\boldsymbol{I}$, so we have an 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 $\mathsf{rows}(\boldsymbol{Q}_{i+1}) \in \mathsf{span}\{\mathsf{rows}(\boldsymbol{I}) \cup \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_i\}\}$. 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

$$\boldsymbol{B} = \begin{bmatrix}\boldsymbol{I} \\ \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_n\end{bmatrix}$$

is invertible. Using the assumptions and the results from the induction we have shown that $\boldsymbol{B}\boldsymbol{v}' = \boldsymbol{B}\boldsymbol{v}$ and therefore $\boldsymbol{v}' = \boldsymbol{v}$. This shows that $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is injective and that $|\mathbb{F}^k| \geq |\mathsf{sol}(\mathcal{C})|$. By the previous argument this proves that $|\mathbb{F}^k| = |\mathsf{sol}(\mathcal{C})|$ and hence $\boldsymbol{I}|_{\mathsf{sol}(\mathcal{C})}$ is a bijection. □

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 algebaric representation and solutions of $\mathcal{C}$. Explicitly, we can formulate the following corollary.

Corollary 3.10. *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 $\boldsymbol{O}\boldsymbol{v}' = \boldsymbol{O}\boldsymbol{v} = (o_1, \ldots, o_l)$. □

3.2 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 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 3.11. *$\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ 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 3.10 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 following statement about second preimages we introduce a shorthand notation for all the row vectors contained in a set of constraints $\mathcal{C}$. We define

$$\mathsf{rows}(\mathcal{C}) = \bigcup_{(t,\boldsymbol{Q},\boldsymbol{a})\in\mathcal{C}} \big\{\mathsf{rows}(\boldsymbol{Q}) \cup \{\boldsymbol{a}\}\big\}.$$

Lemma 3.12. *Let $\mathcal{P} = (\boldsymbol{I}, \boldsymbol{O}, \mathcal{C})$ be a deterministic Linicrypt program. Assume we can write $\mathcal{C} = \mathcal{C}_{fix} \cup \mathcal{C}_{cs}$ and $\mathcal{C}_{cs}$ is deterministically solvable fixing some $\boldsymbol{I}_{cs}$ such that*

$$\mathsf{span}(\mathsf{rows}(\boldsymbol{I}_{cs})) \supset \mathsf{span}\big(\mathsf{rows}(\mathcal{C}_{fix}) \cup \mathsf{rows}(\boldsymbol{O})\big). \tag{3.1}$$

Then an attacker can find second preimages with $\leq n$ queries.

This Lemma is similar to Lemma 3.11 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 case where we already have a solution $\boldsymbol{v}'$ with $\boldsymbol{O}\boldsymbol{v}' = (o_1, \ldots, o_l)$, and only require a different one.

Note, that it is crucial that the space on the left is $\supset$ 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 [**?**]. This characterization includes the case of degeneracy from [**?**], namely it corresponds to choosing $\mathcal{C}_{cs} = \{\}$ and $\boldsymbol{I}_{\text{cs}} = \mathbb{1}_{\mathsf{base}}$. Degeneracy means precisely that $\mathbb{F}^{1\times\mathsf{base}} \supset \mathsf{span}(\mathsf{rows}(\mathcal{C}) \cup \mathsf{rows}(\boldsymbol{O}))$.

Proof. 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 (3.1) 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 [**?**].

$\mathcal{P}(x, y, z)$
$w = H(x) + H(z) + y$
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 $\mathcal{C}_{fix} = \{c_2\}$. Then (c_3, c_1) is a solution ordering for $\mathcal{C}_{cs}$ fixing

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

Note, that for this matrix we have fulfilled the required condition (3.1), because

$$\mathsf{span}(\mathsf{rows}(\boldsymbol{I}_{cs})) \supseteq \mathsf{rows}(\boldsymbol{O}) \cup \mathsf{rows}(\mathcal{C}_{fix}) \cup \{\boldsymbol{q}_3\},$$

and $\boldsymbol{q}_3$ is linearly independent from the rest on the left hand side. 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:

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

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.

3.3 Adapting the Linicrypt model to use block ciphers

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.

3.3.1 Algebraic representation for ideal cipher Linicrypt

Paralell to standard Linicrypt, we will define what the algebraic representation of an ideal cipher Linicrypt program is. Towards this goal we will make the definitions as in section 3.1.

Definition 3.13 (Ideal cipher oracle constraint)**.** *An **ideal cipher oracle constraint** of dimension* base *taking m inputs 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 3.14 (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 just as with standard Linicrypt. The only difference is that base variables can now be created 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 $\boldsymbol{O}$ is defined just as with 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 in the standard case.

Lemma 3.15. *Let $\mathcal{P}$ be an indeal 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 $(\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{C}$. It came either from the command $y = E(k, x)$ or from 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 $\boldsymbol{v}$ solves all constraints in $\mathcal{C}$ and is hence a solution to $\mathcal{C}$. □

TODO

Remove and change everything.

As with standard Linicrypt, we want to exclude programs that make unnecessary queries to the block cipher. This way the base variables are linearly independent, except for the dependencies the adversary might introduce by carefully choosing the input. Hence we assume wlog that no two constraints have the same $(E, \boldsymbol{k}, \boldsymbol{x})$ or $(D, \boldsymbol{k}, \boldsymbol{y})$.

With ideal ciphers there is a second way to make an unnecessary query. That is by first computing $y = E(k, x)$ and then $x' = D(k, y)$. As D_k is the inverse of E_k we have $x = x'$ although $\boldsymbol{x}$ and $\boldsymbol{x}'$ are linearly independent.

Therefore for all $\boldsymbol{k}, \boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y}, \boldsymbol{y}' \in \mathbb{F}^{\mathsf{base}}$ we can assume there are no two constraints $(E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$ and $(D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x}')$ in $\mathcal{C}$ for $\boldsymbol{x} \neq \boldsymbol{x}'$. Neither can there be $(D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x})$ and $(E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}')$ in $\mathcal{C}$ for $\boldsymbol{y} \neq \boldsymbol{y}'$.

TODO: Maybe it is simpler with equivalence relation called always colliding queries

$$
\begin{aligned}
(E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) &\sim (E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}') \\
(E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}) &\sim (D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x}') \\
(D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x}) &\sim (D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x}') \\
(D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x}) &\sim (E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y}')
\end{aligned}
$$

And saying that no two constraints in $\mathcal{C}$ are in the same equivalence class. This might be cleaner, if the equivalence relation used later to analyze repeated nonces case is defined similarly.

3.4 Collision Structure

To explain the concept of a collision structure, we will make use of an example. Consider the following Linicrypt program:

$\mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a, b, c)$

$w = E(c, b + c) + a$

return $c + E(w, b)$

A second preimage to (a, b, c) can be found by the following procedure: Choose some $w' \neq w$. It will turn out, that even choosing w' at random, one can calculate what the other base variables need to be such that the output stays the same. As we want $c + E(w, b) = c' + E(w', b')$ we can again choose any b' and compute c'. Finally, we can compute a' from the equation $w' = E(c', c' + b') + a'$

One can more easily see that such a procedure is possible by looking at the algebraic representation of $\mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}$. In order to highlight which are the base variables, we rewrite the program a bit more explicitly.

$\mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a, b, c)$

$k_1 = c$

$x_1 = b + c$

$y_1 = E(k_1, x_1)$

$k_2 = y_1 + a$

$x_2 = b$

$y_2 = E(k_2, x_2)$

return $c + y_2$

The base variables are (a, b, c, y_1, y_2) and the algebraic representation is

$$\boldsymbol{O} = \big[0, 0, 1, 0, 1\big] \qquad \mathcal{C} = \{(E, \boldsymbol{k}_1, \boldsymbol{x}_1, \boldsymbol{y}_1), (E, \boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2)\}$$

where

$$\begin{aligned}
\boldsymbol{k}_1 &= \big[0, 0, 1, 0, 0\big] \\
\boldsymbol{x}_1 &= \big[0, 1, 1, 0, 0\big] \\
\boldsymbol{y}_1 &= \big[0, 0, 0, 1, 0\big] \\
\boldsymbol{k}_2 &= \big[1, 0, 0, 1, 0\big] \\
\boldsymbol{x}_2 &= \big[0, 1, 0, 0, 0\big] \\
\boldsymbol{y}_2 &= \big[0, 0, 0, 0, 1\big] .
\end{aligned}$$

We can formulate the previous attack from in terms of the algebraic representation. The task is to find a $\boldsymbol{v}'_{\mathsf{base}} = [a', b', c', y'_1, y'_2] \neq \boldsymbol{v}_{\mathsf{base}}$ such that:

$$\mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a', b', c') = \boldsymbol{O}\boldsymbol{v}'_{\mathsf{base}} = \boldsymbol{O}\boldsymbol{v}_{\mathsf{base}} = \mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a, b, c) \tag{3.2}$$

$$y'_1 = \boldsymbol{y}_i\boldsymbol{v}'_{\mathsf{base}} = E(\boldsymbol{k}_1\boldsymbol{v}'_{\mathsf{base}}\,,\ \boldsymbol{x}_1\boldsymbol{v}'_{\mathsf{base}}) \tag{3.3}$$

$$y'_2 = \boldsymbol{y}_i\boldsymbol{v}'_{\mathsf{base}} = E(\boldsymbol{k}_2\boldsymbol{v}'_{\mathsf{base}}\,,\ \boldsymbol{x}_2\boldsymbol{v}'_{\mathsf{base}}) \tag{3.4}$$

We can fulfill these requirements step by step. First, we constrain $\boldsymbol{v}'_{\mathsf{base}}$ by requiring

$$\boldsymbol{O}\boldsymbol{v}'_{\mathsf{base}} = \boldsymbol{O}\boldsymbol{v}_{\mathsf{base}}.$$

This reduces the dimension of the space of possible solutions for $\boldsymbol{v}'_{\mathsf{base}}$ to 4, as $\boldsymbol{O}\boldsymbol{v}_{\mathsf{base}}$ is in the range of $\boldsymbol{O}$ by definition. Now note, that neither $\boldsymbol{k}_2$ nor $\boldsymbol{x}_2$ are in the span of $\mathsf{rows}(\boldsymbol{O})$. Therefore we can choose k'_2 and x'_2 at random such that $(k'_2, x'_2) \neq (k_2, x_2)$, and constrain $\boldsymbol{v}'_{\mathsf{base}}$ by requiring

$$\boldsymbol{k}_2\boldsymbol{v}'_{\mathsf{base}} = k'_2 \quad \text{and} \quad \boldsymbol{x}_2\boldsymbol{v}'_{\mathsf{base}} = x'_2.$$

Now we can compute $y'_2 = E(k'_2, x'_2)$ and add constraint (3.4). This constraint is compatible with the previous constraints because $\boldsymbol{y}_2$ is not in the span of $\mathsf{rows}(\boldsymbol{O}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2\}$. The dimension of the subspace of solutions is now 1, as we have added 4 one-dimensional constraints.

Now one only needs to fulfill constraint (3.3). As $\boldsymbol{k}_1$ and $\boldsymbol{x}_1$ are in the span of $\mathsf{rows}(\boldsymbol{O}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2\}$ the intermediate variables k'_1 and x'_1 are uniquely determined. E.g. $\boldsymbol{k}_1 = \boldsymbol{O} - \boldsymbol{y}_2$ and therefore $k'_1 = \mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a, b, c) - y'_2$.

Finally, note that $\boldsymbol{y}_1 \notin \mathsf{span}\big(\mathsf{rows}(\boldsymbol{O}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2\} \cup \{\boldsymbol{k}_1, \boldsymbol{x}_1\}\big)$. Therefore, adding the constraint

$$\boldsymbol{y}_1\boldsymbol{v}'_{\mathsf{base}} = y'_1 = E(k'_1, x'_1)$$

reduces the solution space of possible $\boldsymbol{v}'_{\mathsf{base}}$ to a single point in $\mathbb{F}^{\mathsf{base}}$. We know that $\boldsymbol{v}'_{\mathsf{base}} \neq \boldsymbol{v}_{\mathsf{base}}$ because $(k_2, x'_2) \neq (k_2, x_2)$. The only way to produce different intermediate variables in a deterministic program is to choose different input, hence $(a', b', c') \neq (a, b, c)$.

This example would have worked exactly the same if we replaced E with H. What follows is an example where the invertibility property of E_k plays a role.

$\mathcal{P}^{\mathcal{E}}_{\mathsf{col},2}(a,b,c)$
$k_1 = c$
$x_1 = b$
$y_1 = E(k_1, x_1)$
$k_2 = a$
$x_2 = y_1$
$y_2 = E(k_2, x_2)$
return $y_1 + y_2$

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

For this program there is a similar procedure to find second preimages. As before, the first constraint is $\boldsymbol{O}$ In this case we can fix $\boldsymbol{k}_2\boldsymbol{v}'_{\mathsf{base}} = k'_2 = k_2$, $\boldsymbol{x}_2\boldsymbol{v}'_{\mathsf{base}} = x'_2 = x_2$ and $\boldsymbol{y}_2\boldsymbol{v}'_{\mathsf{base}} = y'_2 = y_2$. Therefore condition (3.4) is fulfilled trivially. After adding these 4 constraints the solutions space is still 1-dimensional. Note, that $\boldsymbol{y}_1 = \boldsymbol{x}_2$ and it is therefore already fixed at this point, hence to fulfill (3.3) we have to make use of the invertibility property of E_k. Because

$$
\begin{aligned}
& \boldsymbol{y}_1\boldsymbol{v}'_{\mathsf{base}} = E(\boldsymbol{k}_1\boldsymbol{v}'_{\mathsf{base}}, \boldsymbol{x}_1\boldsymbol{v}'_{\mathsf{base}}) \\
\iff\ & \boldsymbol{x}_1\boldsymbol{v}'_{\mathsf{base}} = D(\boldsymbol{k}_1\boldsymbol{v}'_{\mathsf{base}}, \boldsymbol{y}_1\boldsymbol{v}'_{\mathsf{base}}),
\end{aligned}
$$

we can choose a random $k'_1 \neq k_1$ and compute $x'_1 = D(k'_1, y'_1)$ in order to fulfill (3.1). With this fifth constraint we have found a single $\boldsymbol{v}'_{\mathsf{base}} \neq \boldsymbol{v}_{\mathsf{base}}$.

We want to briefly summarize the conditions for this kind of second preimage attack. Let $\mathcal{P}'$ denote the execution of

- Some ideal cipher constraints are fulfilled by setting the intermediate variables in $\mathcal{P}(a', b', c')$ to the same value as in $\mathcal{P}(a, b, c)$
- There is some constraint for which either the input or the output is undetermined by the previously fixed intermediate variables
- For this constraint and all following constraint one can iteratively call the ideal cipher $\mathcal{E}$ to set the intermediate variables such that the constraints are fulfilled. This is only possible, if the either the output or the input is undetermined by previously fixed variables.

Definition 3.16 (Collision structure)**.** *Let $\mathcal{P} = (\boldsymbol{O}, \mathcal{C})$ be a Linicrypt program with $|\mathcal{C}| = n$. A **collision structure** for $\mathcal{P}$ is an index $1 \leq i^* \leq n$, an ordering $(c_1, \ldots, c_n)$*

of $\mathcal{C}$ for $c_i = (Op_i, \boldsymbol{k}_i, \boldsymbol{q}_i, \boldsymbol{a}_i)$ and a tuple $(dir_{i^*}, \ldots, dir_n)$ for $dir_i \in \{\mathsf{Forward}, \mathsf{Backward}\}$, such that the following two conditions hold:

1. *Let $\mathcal{F}_{i^*} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{i^*-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{i^*-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i^*-1}\}$. One of the following is true:*
   a) $dir_{i^*} = \mathsf{Forward}$ *and* $\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{q}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F}_{i^*} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$
   b) $dir_{i^*} = \mathsf{Backward}$ *and* $\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{a}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F}_{i^*} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$
2. *For all $j \geq i^*$ let $\mathcal{F}_j = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1}\}$. One of the following is true:*
   a) $dir_j = \mathsf{Forward}$ *and* $\boldsymbol{a}_j \notin \mathsf{span}\big(\mathcal{F}_j \cup \{\boldsymbol{k}_j, \boldsymbol{q}_j\} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$
   b) $dir_j = \mathsf{Backward}$ *and* $\boldsymbol{q}_j \notin \mathsf{span}\big(\mathcal{F}_j \cup \{\boldsymbol{k}_j, \boldsymbol{a}_j\} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$

TODO: Remove this wordy definition. All the info from here should be integrated into the example

Definition 3.17 (Collision structure)**.** *Let $\mathcal{P} = (\boldsymbol{O}, \mathcal{C})$ be a Linicrypt program with $|\mathcal{C}| = n$. A **collision structure** for $\mathcal{P}$ is an index $1 \leq i^* \leq n$, an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ for $c_i = (Op_i, \boldsymbol{k}_i, \boldsymbol{q}_i, \boldsymbol{a}_i)$ and a tuple $(dir_{i^*}, \ldots, dir_n)$ for $dir_i \in \{\mathsf{Forward}, \mathsf{Backward}\}$, such that:*

1. *The i^*'th constraint is unconstrained by the output of $\mathcal{P}$ and previous fixed constraints. Let $\mathcal{F} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{i^*-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{i^*-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i^*-1}\}$ denote the vectors fixed by previous constraints in the ordering.*
   a) *if $dir_{i^*} = \mathsf{Forward}$, the input of the query associated to c_{i^*} is unconstrained:*
   $$\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{q}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$$
   b) *if $dir_{i^*} = \mathsf{Backward}$, the output of the query associated to c_{i^*} is unconstrained:*
   $$\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{a}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$$
2. *For all $j \geq i^*$ the constraint c_j does not contradict previous constraints. Let $\mathcal{F} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1}\}$ denote the vectors fixed by previous constraints in the ordering.*
   a) *if $dir_j = \mathsf{Forward}$*
   $$\boldsymbol{a}_j \notin \mathsf{span}\big(\{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1}, \} \cup \{\boldsymbol{k}_j, \boldsymbol{q}_j\} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$$
   b) *if $dir_j = \mathsf{Backward}$*
   $$\boldsymbol{q}_j \notin \mathsf{span}\big(\{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1}, \} \cup \{\boldsymbol{k}_j, \boldsymbol{a}_j\} \cup \mathsf{rows}\,(\boldsymbol{O})\big)$$

Lemma 3.18 (Collision structure gives second preimages)**.** *If collision structure blabla exists for $\mathcal{P} = (\boldsymbol{O}, \mathcal{C})$ then blabla with probability 1.*

$\mathsf{FindSecondPreimage}\Big(\Big)$

Compute $\boldsymbol{v}_{\mathsf{base}}$ by executing $\mathcal{P}^{\mathcal{E}}(\boldsymbol{x})$
... very similar

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.