



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling the Ideal Cipher in Linicrypt

Master Thesis
Frederik Semmel
August 14, 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 the Linicrypt model to the ideal cipher model	16
4.0.1 Algebraic representation for ideal cipher Linicrypt	16
4.1 Collision Structure	19
4.2 Collision Structure old	21

Chapter 1

Introduction

Todo

Preliminaries

2.1 Linicrypt

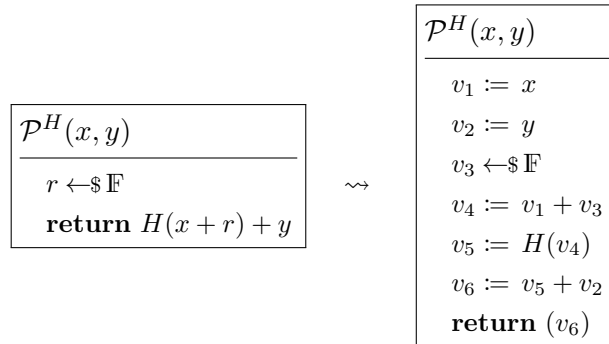
2.1.1 Definition of a Linicrypt program

The Linicrypt model for cryptographic constructions was introduced by Cramer & Rouseluke 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 $H : \{0, 1\}^* \times \mathbb{F}^* \rightarrow \mathbb{F}^*$
- 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.



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, \dots, 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 $\mathbf{v}_{\text{base}} \in \mathbb{F}^{\text{base}}$ the column vector consisting of the values that the base variables take in a specific execution 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 intermediate 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}^{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** \mathbf{v}_i to be the unique row vector such that for every execution of \mathcal{P} base variables taking the values \mathbf{v}_{base} , the variable v_i has the value $\mathbf{v}_i \mathbf{v}_{\text{base}}$. For example, if v_i is the j 'th base variable, then $\mathbf{v}_i = [0 \ \cdots \ 1 \ \cdots \ 0]$, 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 $o_1, \dots, o_l \in \{v_1, \dots, v_n\}$ be the output variables of \mathcal{P} . Then the **output matrix** \mathbf{O} of \mathcal{P} is defined by

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

By the definition of the associated vectors \mathbf{o}_i we have $\mathbf{O} \mathbf{v}_{\text{base}} = [o_1 \ \cdots \ o_k]^\top$. 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 $i_1, \dots, i_k \in \{v_1, \dots, v_n\}$ are

the intermediate variables created by retrieving an input, then we write

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

As i_1, \dots, i_k are base variables, the rows of \mathbf{I} are canonical basis vectors. If the Linicrypt program is written such that it first retrieves all its inputs, then \mathbf{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 its 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 $a_i = H(t_i, (q_1, \dots, q_n))$ be an operation in \mathcal{P} . The **associated oracle constraint** c of this operation is the tuple

$$c = \left(t_i, \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}, \mathbf{a}_i \right) = (t_i, \mathbf{Q}_i, \mathbf{a}_i).$$

This should be interpreted as the requirement that $\mathbf{a}_i \mathbf{v}_{\text{base}} = H(t_i, \mathbf{Q}_i \mathbf{v}_{\text{base}})$. We denote the set of all (associated) oracle constraints of \mathcal{P} by \mathcal{C} .

As we want the base variables to be linearly independent 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 representation: We assume wlog that no two constraints in \mathcal{C} share the same (t, \mathbf{Q}) .

Wrapping up these definitions, we define the **algebraic representation** of \mathcal{P} to be the tuple $(\mathbf{I}, \mathbf{O}, \mathcal{C})$. 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 lose any relevant information about the operations executed in \mathcal{P} . Informally, this is because the constraints in \mathcal{C} have a particular form, which makes it clear in which order the oracles calls have to be executed. Given the order, and using the input matrix, one can determine which variables used in the calls are retrieved from the input and which have to be sampled. Finally the output matrix completely describes how to construct the output from the input, the results of the queries and 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, $\mathcal{P}_{\text{deg}}^H$ is degenerate and $\mathcal{P}_{\text{cs}}^H$ 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 $\mathcal{P}_{\text{cs}}^H$ stays the same, and finally find y' according to $w' = x' + y'$.

$\mathcal{P}_{\text{deg}}^H(x, y)$ <hr style="width: 80%; margin: 5px auto;"/> $v := x + y$ return $H(v)$	$\mathcal{P}_{\text{cs}}^H(x, y)$ <hr style="width: 80%; margin: 5px auto;"/> $w := x + y$ 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 $\mathbf{I} \in \mathbb{F}^{k \times \text{base}}$, the output matrix $\mathbf{O} \in \mathbb{F}^{l \times \text{base}}$ and the set of constraints $\mathcal{C} = \{(t_i, \mathbf{Q}_i, \mathbf{a}_i) \mid i = 1, \dots, n\}$, with $\mathbf{Q}_i \in \mathbb{F}^{m_i \times \text{base}}$ and $\mathbf{a}_i \in \mathbb{F}^{1 \times \text{base}}$. These matrices are such that, if one constrains $\mathbf{v}_{\text{base}} \in \mathbb{F}^{\text{base}}$ with $\mathbf{I}\mathbf{v}_{\text{base}} = [i_1 \ \dots \ i_k]^\top$ for some arbitrary input, the program can compute the rest of the base variables and output $\mathbf{O}\mathbf{v}_{\text{base}} = [o_1 \ \dots \ o_l]^\top$. To simplify the notation we will sometimes write (v_1, \dots, v_k) for the column vector $[v_1 \ \dots \ v_k]^\top$.

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 $\mathbf{O}\mathbf{v}_{\text{base}}$ 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 $(t, \mathbf{Q}, \mathbf{a})$ for $t \in \{0, 1\}^*$, $\mathbf{Q} \in \mathbb{F}^{m \times \text{base}}$ and $\mathbf{a} \in \mathbb{F}^{1 \times \text{base}}$.

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 $\mathbf{a} = [0 \ \dots \ 0 \ 1 \ 0 \ \dots \ 0]$. We call t the nonce and refer to \mathbf{Q} as the query matrix and to \mathbf{a} as answer matrix. If the nonce is the empty string, then we just write (\mathbf{Q}, \mathbf{a}) instead of $(t, \mathbf{Q}, \mathbf{a})$. Usually we just say “a constraint” when the other variables are clear from the context.

The constraint $(t, \mathbf{Q}, \mathbf{a})$ 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 $\mathbf{v} \in \mathbb{F}^{\text{base}}$ **solves** \mathcal{C} if $\mathbf{a}\mathbf{v} = H(t, \mathbf{Q}\mathbf{v})$ for all $(t, \mathbf{Q}, \mathbf{a}) \in \mathcal{C}$. Such a \mathbf{v} is also called a **solution** of \mathcal{C} . The set of all solutions to \mathcal{C} is called $\text{sol}(\mathcal{C})$.*

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 $(t, \mathbf{Q}) = (t', \mathbf{Q}') \implies \mathbf{a} = \mathbf{a}'$.*

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 $(t, \mathbf{Q}, \mathbf{a})$, we will also use the shorthand $(t, \mathbf{Q}) \mapsto \mathbf{a}$, or $\mathbf{Q} \mapsto \mathbf{a}$ if 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, \dots, 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 $\mathbf{v} \in \mathbb{F}^{\text{base}}$. Then it is a solution of \mathcal{C} satisfying $\mathbf{I}\mathbf{v} = (i_1, \dots, i_k)$.

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

$$\mathcal{C} = \{[1 \ 0] \mapsto [0 \ 1], [0 \ 1] \mapsto [1 \ 0]\}.$$

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 \mathbf{B} be any matrix in $\mathbb{F}^{\text{base} \times \text{base}'}$ and $\mathcal{P} = (\mathbf{I}, \mathbf{O}, \mathcal{C})$ be a Linicrypt program. We define*

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

*If \mathbf{B} is invertible, we say $\mathcal{P}\mathbf{B} := (\mathbf{I}\mathbf{B}, \mathbf{O}\mathbf{B}, \mathcal{C}\mathbf{B})$ 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 \mathbf{B} be a basis change matrix. Note the following equivalence which follows from associativity of the matrix product:

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

This implies that $\mathbf{B}|_{\text{sol}(\mathcal{C}\mathbf{B})} : \text{sol}(\mathcal{C}\mathbf{B}) \rightarrow \text{sol}(\mathcal{C})$ is a well-defined bijection.

We restrict our goal to characterizing which matrices \mathbf{I} and \mathbf{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 $(\mathbf{I}, \mathbf{O}, \mathcal{C})$ 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 $\mathbf{I} = [\mathbf{1}_k \ 0] \in \mathbb{F}^{k \times \text{base}}$ for $k = \text{base} - n$ and let $\mathbf{O} \in \mathbb{F}^{l \times \text{base}}$ for some $l \in \mathbb{N}$.*

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

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

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 its k inputs into intermediate variables, thus the input matrix is $\mathbf{I} = [\mathbf{1}_k \ 0]$. Then each of the random oracle constraints is converted into Linicrypt commands of the form $v_{k+i} = H(q_1, \dots, q_{k+i-1})$. Condition 1. from the Lemma ensures that q_1, \dots, q_{k+i-1} are linear combinations only of previously defined intermediate variables. Condition 2. from the Lemma guarantees that the variable v_{k+i} has not been used previously, so the command is a valid Linicrypt command.

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

We will now generalize the condition from this Lemma in order to make it independent of the chosen basis of \mathbb{F}^{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 $\mathbf{I} \in \mathbb{F}^{k \times \text{base}}$ for some $k \in \mathbb{N}$. \mathcal{C} is **deterministically solvable fixing \mathbf{I}** if there exists an ordering (c_1, \dots, c_n) of \mathcal{C} such that for all $i = 1, \dots, n$:*

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

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

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

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

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

Proof. Consider a constraint $(t, \mathbf{Q}, \mathbf{a})$ in \mathcal{C} . By definition $\mathbf{a} = \mathbf{e}_i$ for some i . and \mathbf{Q} has only zeros in the columns including and to the right of column i . We can sort $\mathcal{C} = \{c_1, \dots, c_n\}$, such that the \mathbf{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 $\text{base} = k + n$. The associated vectors $\text{rows}(\mathbf{I}) = \{\mathbf{i}_1, \dots, \mathbf{i}_k\}$ to the input variables are linearly independent, hence it follows that $\{\mathbf{i}_1, \dots, \mathbf{i}_k, \mathbf{a}_1, \dots, \mathbf{a}_n\}$ is a basis of $\mathbb{F}^{1 \times \text{base}}$. \square

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 characterization 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 $\mathbf{I} \in \mathbb{F}^{k \times \text{base}}$ for some $k \in \mathbb{N}$. Let $\mathbf{O} \in \mathbb{F}^{\text{out} \times \text{base}}$ be an arbitrary output matrix for some $\text{out} \in \mathbb{N}$.*

Then there is a basis change $\mathbf{B} \in \mathbb{F}^{\text{base} \times \text{base}}$ and a deterministic Linicrypt program \mathcal{P} , such that $(\mathbf{I}\mathbf{B}, \mathbf{O}\mathbf{B}, \mathcal{C}\mathbf{B})$ is its algebraic representation.

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

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

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

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

which is equivalent to

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

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

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

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

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 $\mathcal{P} = (\mathbf{I}, \mathbf{O}, \mathcal{C})$ 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 $f_{\mathcal{P}} : \mathbb{F}^k \rightarrow \mathbb{F}^l$ which maps (i_1, \dots, i_k) onto the output of $\mathcal{P}(i_1, \dots, i_k)$.

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

Proof. If we set $\mathbf{O} = \mathbf{1}_{\text{base}}$ then we can apply Lemma 3.8 to get a deterministic program \mathcal{P} and basis change \mathbf{B} such that, $\mathcal{P} = (\mathbf{IB}, \mathbf{B}, \mathcal{CB})$. The associated function $f_{\mathcal{P}} : \mathbb{F}^k \rightarrow \mathbb{F}^{\text{base}}$ 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 $\text{sol}(\mathcal{CB})$. Note that the output matrix of \mathcal{P} is \mathbf{B} and $\mathbf{B}|_{\text{sol}(\mathcal{CB})} : \text{sol}(\mathcal{CB}) \rightarrow \text{sol}(\mathcal{C})$ is a bijection. Therefore $f_{\mathcal{P}}$ is an injective map from \mathbb{F}^k to $\text{sol}(\mathcal{C})$.

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

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

Assume this is true for all $j \leq i$. Because of the solution ordering of \mathcal{C} we know that $\text{rows}(\mathbf{Q}_{i+1}) \subseteq \text{rowsp}(\mathbf{I}) + \text{rowsp}(\{\mathbf{a}_1, \dots, \mathbf{a}_i\})$. This means there is a λ such that $\mathbf{Q}_{i+1} = \lambda [\mathbf{I}^\top \quad \mathbf{a}_1^\top \quad \dots \quad \mathbf{a}_i^\top]^\top =: \lambda \mathbf{A}_i$. Therefore we have $\mathbf{Q}_{i+1}\mathbf{v}' = \lambda \mathbf{A}_i\mathbf{v}' = \lambda \mathbf{A}_i\mathbf{v} = \mathbf{Q}_{i+1}\mathbf{v}$. Because \mathbf{v} and \mathbf{v}' are solutions to \mathcal{C} we know that $\mathbf{a}_{i+1}\mathbf{v}' = H(\mathbf{Q}_{i+1}\mathbf{v}') = H(\mathbf{Q}_{i+1}\mathbf{v}) = \mathbf{a}_{i+1}\mathbf{v}$.

Because $\text{rows}(\mathbf{I}) \cup \{\mathbf{a}_1, \dots, \mathbf{a}_n\}$ is a basis, the matrix

$$\mathbf{C} = \begin{bmatrix} \mathbf{I} \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{bmatrix}$$

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

Corollary 3.10. *Let \mathcal{C} be deterministically solvable fixing some \mathbf{I} . For each element in \mathbb{F}^k its inverse under $\mathbf{I}|_{\text{sol}(\mathcal{C})}$ 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 $(\mathbf{IB}, \mathbf{B}, \mathcal{CB})$ is an algebraic representation, we have $\mathbf{IB} = \mathbf{1}_k \mid 0$. If we run \mathcal{P} on (i_1, \dots, i_k) we compute the values of the remaining base variables. Let us call the vector containing the values of the base variables $\mathbf{v} \in \mathbb{F}^{\text{base}}$. Clearly $\mathbf{IB}\mathbf{v} = (i_1, \dots, i_k)$. Also \mathcal{P} outputs $\mathbf{B}\mathbf{v}$, so $f_{\mathcal{P}}((i_1, \dots, i_k)) = \mathbf{B}\mathbf{v}$. Therefore

$$\mathbf{IB}\mathbf{v} = (\mathbf{I} \circ f_{\mathcal{P}})((i_1, \dots, i_k)) = (i_1, \dots, i_k). \quad (3.1)$$

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

The benefit of this bijection is that every solution of \mathcal{C} corresponds to a unique input. 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 $\mathcal{P} = (\mathbf{I}, \mathbf{O}, \mathcal{C})$ be a deterministic Linicrypt program. The following two statements are equivalent:*

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

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

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

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

Because of the bijection $\mathbf{B} : \text{sol}(\mathcal{C}\mathbf{B}) \rightarrow \text{sol}(\mathcal{C})$ we have the following equivalence:

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

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

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

$\frac{\mathcal{P}(x, y)}{\text{return } (y + H(x), x)}$	$\frac{\mathcal{P}^{-1}(w, z)}{\text{return } (z, w - H(z))}$
--	---

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 $c = (t, \mathbf{Q}, \mathbf{a})$:

$$\begin{aligned} \text{rowsp}(c) &= \text{rowsp}(\mathbf{Q}) + \text{rowsp}(\mathbf{a}) \\ \text{ker}(c) &= \text{ker}(\mathbf{Q}) \cap \text{ker}(\mathbf{a}) \end{aligned}$$

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

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

Lemma 3.13. *Let $\mathcal{P} = (\mathbf{I}, \mathbf{O}, \mathcal{C})$ 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 \mathbf{I}_{cs} such that*

$$\text{rowsp}(\mathbf{I}_{cs}) \supsetneq \text{rowsp}(\mathcal{C}_{fix}) + \text{rowsp}(\mathbf{O}). \quad (3.2)$$

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 \mathbf{v} solving \mathcal{C} while fixing $\mathbf{O}\mathbf{v} = (o_1, \dots, o_l)$. The difference is that here we consider the easier case (from an attackers perspective): We already have a solution \mathbf{v}' with $\mathbf{O}\mathbf{v}' = (o_1, \dots, o_l)$ and only require a different one.

The definition of collision structure in Lemma 3.13 is slightly more general than the one from [?]. It also includes the case of degeneracy, namely it corresponds to choosing $\mathcal{C}_{cs} = \{\}$ and $\mathbf{I}_{cs} = \mathbb{1}_{\text{base}}$. Degeneracy means precisely that $\mathbb{F}^{1 \times \text{base}} \supset \text{rowsp}(\mathcal{C}) + \text{rowsp}(\mathbf{O})$. 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 \mathbf{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 \mathbf{v}' solving \mathcal{C} such that $\mathbf{I}\mathbf{v}' = (i_1, \dots, i_k)$ and $\mathbf{O}\mathbf{v}' = (o_1, \dots, o_l)$. The goal is to find a different \mathbf{v} solving \mathcal{C} with $\mathbf{O}\mathbf{v} = (o_1, \dots, o_l)$.

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

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

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 \mathbf{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 $(H(w) + x, H(z))$

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

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

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

Condition (3.2) is fulfilled because

$$\text{rowsp}(\mathbf{I}_{cs}) = (\text{rowsp}(\mathbf{O}) + \text{rowsp}(\mathcal{C}_{fix})) \oplus \text{rowsp}(\mathbf{q}_3).$$

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

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

which leads to the representation:

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

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 the Linicrypt model 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 $\mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$ instead of the hash function $H : \{0, 1\}^* \times \mathbb{F}^* \rightarrow \mathbb{F}$. 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.

$\frac{\mathcal{P}^H(k, x)}{\text{return } H(k, x)}$	$\frac{\mathcal{P}^{\mathcal{E}}(k, x)}{\text{return } E(k, x)}$
--	--

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 $(\mathbf{k}, \mathbf{x}, \mathbf{y})$ for \mathbf{k}, \mathbf{x} and \mathbf{y} in $\mathbb{F}^{1 \times \text{base}}$.*

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 $\mathbf{v} \in \mathbb{F}^{\text{base}}$ **solves** \mathcal{C} if $\mathbf{y}\mathbf{v} = E(\mathbf{k}\mathbf{v}, \mathbf{x}\mathbf{v})$ for all $(\mathbf{k}, \mathbf{x}, \mathbf{y}) \in \mathcal{C}$. Such a \mathbf{v} is also called a **solution** of \mathcal{C} . The set of all solutions to \mathcal{C} is called $\text{sol}(\mathcal{C})$.*

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** $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$. 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 $(\mathbf{v}_1, \mathbf{v}_3, \mathbf{v}_2)$. We call the set of all associated constraints \mathcal{C} . The input matrix \mathbf{I} and output matrix \mathbf{O} are defined in the same way as in standard Linicrypt. We call $(\mathbf{I}, \mathbf{O}, \mathcal{C})$ 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 $(\mathbf{I}, \mathbf{O}, \mathcal{C})$. Let \mathbf{v} denote the values of the base variables in an execution of \mathcal{P} with input (i_1, \dots, i_k) and output (o_1, \dots, o_l) . Then \mathbf{v} is a solution to \mathcal{C} with $\mathbf{I}\mathbf{v} = (i_1, \dots, i_k)$ and $\mathbf{O}\mathbf{v} = (o_1, \dots, o_l)$*

Proof. The statements $\mathbf{I}\mathbf{v} = (i_1, \dots, i_k)$ and $\mathbf{O}\mathbf{v} = (o_1, \dots, o_l)$ 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 $(\mathbf{k}, \mathbf{x}, \mathbf{y}) \in \mathcal{C}$. 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 $\mathbf{y}\mathbf{v} = y = E(k, x) = E(\mathbf{k}\mathbf{v}, \mathbf{x}\mathbf{v})$. In the second case we also have $y = E(k, x)$ by the properties of the ideal cipher $\mathcal{E} = (E, D)$. Therefore \mathbf{v} is a solution to \mathcal{C} . \square

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 $c = (\mathbf{k}, \mathbf{x}, \mathbf{y})$:

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

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

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

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

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

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

such that:

1. $\text{rows}(\mathbf{Q}_i) \subseteq \text{rowsp}(\mathbf{I}) + \text{rowsp}(\{c_1, \dots, c_{i-1}\})$
2. $\mathbf{a}_i \notin \text{rowsp}(\mathbf{I}) + \text{rowsp}(\{c_1, \dots, c_{i-1}\})$

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

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

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

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

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

- (a) $\mathbf{y}_i = \mathbf{e}_{k+i}$ and $\{\mathbf{k}_i, \mathbf{x}_i\} \subseteq \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_{k+i-1})$ or
- (b) $\mathbf{x}_i = \mathbf{e}_{k+i}$ and $\{\mathbf{k}_i, \mathbf{y}_i\} \subseteq \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_{k+i-1})$

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 $\{\mathbf{k}_i, \mathbf{x}_i\} \subseteq \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_{k+i-1})$ ensures that these linear combinations are well defined. As $\text{base} = k + n$, all base variables have been set after n query commands. We can use the rows of \mathbf{O} to define the Linicrypt commands for the output variables. \square

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

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

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

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

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

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

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.

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

$$\text{rowsp}(\mathbf{I}_{cs}) \supsetneq \text{rowsp}(\mathbf{O}) + \text{rowsp}(\mathcal{C}_{fix}). \quad (4.1)$$

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, \dots, i_k) to \mathcal{P} . By executing \mathcal{P} on (i_1, \dots, i_k) we compute the values of the base variables $\mathbf{v} \in \text{sol}(\mathcal{C})$. Recall the bijection $\mathbf{I}_{cs}|_{\text{sol}(\mathcal{C}_{cs})} : \text{sol}(\mathcal{C}_{cs}) \rightarrow \mathbb{F}^{k'}$ where k' is the number of rows of \mathbf{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 $\mathbb{F}^{\text{base}}/\ker(\mathbf{I}_{cs})$. This quotient space is defined by the equivalence relation $\mathbf{v} \sim_{cs} \mathbf{w} \iff \mathbf{v} - \mathbf{w} \in \ker(\mathbf{I}_{cs})$. We denote it by:

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

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

$$(4.1) \iff \ker(\mathbf{I}_{cs})^\top \supsetneq \ker(\mathbf{O})^\top + \ker(\mathcal{C}_{fix})^\top \quad (4.2)$$

$$\iff \ker(\mathbf{I}_{cs})^\top \supsetneq (\ker(\mathbf{O}) \cap \ker(\mathcal{C}_{fix}))^\top \quad (4.3)$$

$$\iff \ker(\mathbf{I}_{cs}) \subsetneq \ker(\mathbf{O}) \cap \ker(\mathcal{C}_{fix}) \quad (4.4)$$

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

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

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

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

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

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

We argue why the second preimage is computable with $|\mathcal{C}_{cs}|$ queries. To compute such a \mathbf{v}' we need to compute preimages of $\lambda \circ \widetilde{\mathbf{I}_{cs}}$. 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 $[\mathbf{v}']_{\sim_{cs}}$ to $[\mathbf{v}]_{\sim_{fix}}$ from its space of preimages arbitrarily while making sure that $[\mathbf{v}']_{\sim_{cs}} \neq [\mathbf{v}]_{\sim_{cs}}$. The inverse of the bijection $\widetilde{\mathbf{I}}_{cs}$ is computable with $|\mathcal{C}_{cs}|$ queries by Corollary 4.9. \square

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

$\mathcal{P}_{col}^{\mathcal{E}}(a, b, c)$	
$k_1 = c$	$\mathbf{O} = [0 \ 0 \ 0 \ 1 \ 1]$
$x_1 = b$	$\mathbf{k}_1 = [0 \ 0 \ 1 \ 0 \ 0]$
$y_1 = E(k_1, x_1)$	$\mathbf{x}_1 = [0 \ 1 \ 0 \ 0 \ 0]$
$k_2 = a$	$\mathbf{y}_1 = [0 \ 0 \ 0 \ 1 \ 0]$
$x_2 = y_1$	$\mathbf{k}_2 = [1 \ 0 \ 0 \ 0 \ 0]$
$y_2 = E(k_2, x_2)$	$\mathbf{x}_2 = [0 \ 0 \ 0 \ 1 \ 0]$
return $y_1 + y_2$	$\mathbf{y}_2 = [0 \ 0 \ 0 \ 0 \ 1]$

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

Then $\text{rowsp}(\mathbf{I}_{cs}) = (\text{rowsp}(\mathbf{O}) + \text{rowsp}(\mathcal{C}_{fix})) \oplus \text{rowsp}(\mathbf{k}_1)$, so condition (4.1) is fulfilled.

\mathcal{C}_{cs} is deterministically solvable fixing \mathbf{I}_{cs} . We can set $\mathbf{Q}_1 = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{y}_1 \end{bmatrix}$ and $\mathbf{a}_1 = \mathbf{x}_1$ and then we have:

1. $\text{rowsp}(\mathbf{Q}_1) \subseteq \text{rowsp}(\mathbf{I}_{cs})$
2. $\mathbf{a}_1 \notin \text{rowsp}(\mathbf{I}_{cs})$

In other words, 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 and answer y_1 to the first query to E , but not the query itself. By using D we get the correct query x_1 . Then (k_2, x_1, k_1) are a second preimage.

4.2 Collision Structure old

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

$\mathcal{P}_{col,1}^{\mathcal{E}}(a, b, c)$
$w = E(c, b + c) + a$
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 $\mathcal{P}_{\text{col},1}^{\mathcal{E}}$. In order to highlight which are the base variables, we rewrite the program a bit more explicitly.

$\mathcal{P}_{\text{col},1}^{\mathcal{E}}(a, b, c)$
$k_1 = c$
$x_1 = b + c$
$y_1 = E(k_1, x_1)$
$k_2 = y_1 + a$
$x_2 = b$
$y_2 = E(k_2, x_2)$
return $c + y_2$

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

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

where

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

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

$$\mathcal{P}_{\text{col},1}^{\mathcal{E}}(a', b', c') = \mathbf{O}\mathbf{v}'_{\text{base}} = \mathbf{O}\mathbf{v}_{\text{base}} = \mathcal{P}_{\text{col},1}^{\mathcal{E}}(a, b, c) \quad (4.5)$$

$$y'_1 = \mathbf{y}_i \mathbf{v}'_{\text{base}} = E(\mathbf{k}_1 \mathbf{v}'_{\text{base}}, \mathbf{x}_1 \mathbf{v}'_{\text{base}}) \quad (4.6)$$

$$y'_2 = \mathbf{y}_i \mathbf{v}'_{\text{base}} = E(\mathbf{k}_2 \mathbf{v}'_{\text{base}}, \mathbf{x}_2 \mathbf{v}'_{\text{base}}) \quad (4.7)$$

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

$$\mathbf{O}\mathbf{v}'_{\text{base}} = \mathbf{O}\mathbf{v}_{\text{base}}.$$

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

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

Now we can compute $y'_2 = E(k'_2, x'_2)$ and add constraint (4.7). This constraint is compatible with the previous constraints because \mathbf{y}_2 is not in the span of $\text{rows}(\mathbf{O}) \cup \{\mathbf{k}_2, \mathbf{x}_2\}$. 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 (4.6). As \mathbf{k}_1 and \mathbf{x}_1 are in the span of $\text{rows}(\mathbf{O}) \cup \{\mathbf{k}_2, \mathbf{x}_2, \mathbf{y}_2\}$ the intermediate variables k'_1 and x'_1 are uniquely determined. E.g. $\mathbf{k}_1 = \mathbf{O} - \mathbf{y}_2$ and therefore $k'_1 = \mathcal{P}_{\text{col},1}^{\mathcal{E}}(a, b, c) - y'_2$.

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

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

reduces the solution space of possible $\mathbf{v}'_{\text{base}}$ to a single point in \mathbb{F}^{base} . We know that $\mathbf{v}'_{\text{base}} \neq \mathbf{v}_{\text{base}}$ 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.

$\mathcal{P}_{\text{col},2}^{\mathcal{E}}(a, b, c)$	
$k_1 = c$	$\mathbf{O} = [0, 0, 0, 1, 1]$
$x_1 = b$	$\mathbf{k}_1 = [0, 0, 1, 0, 0]$
$y_1 = E(k_1, x_1)$	$\mathbf{x}_1 = [0, 1, 0, 0, 0]$
$k_2 = a$	$\mathbf{y}_1 = [0, 0, 0, 1, 0]$
$x_2 = y_1$	$\mathbf{k}_2 = [1, 0, 0, 0, 0]$
$y_2 = E(k_2, x_2)$	$\mathbf{x}_2 = [0, 0, 0, 1, 0]$
return $y_1 + y_2$	$\mathbf{y}_2 = [0, 0, 0, 0, 1]$

For this program there is a similar procedure to find second preimages. As before, the first constraint is \mathbf{O} . In this case we can fix $\mathbf{k}_2 \mathbf{v}'_{\text{base}} = k'_2 = k_2$, $\mathbf{x}_2 \mathbf{v}'_{\text{base}} = x'_2 = x_2$ and $\mathbf{y}_2 \mathbf{v}'_{\text{base}} = y'_2 = y_2$. Therefore condition (4.7) is fulfilled trivially. After adding these 4 constraints the solutions space is still 1-dimensional. Note, that $\mathbf{y}_1 = \mathbf{x}_2$ and it is therefore already fixed at this point, hence to fulfill (4.6) we have to make use of the invertibility property of E_k . Because

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

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 $\mathbf{v}'_{\text{base}} \neq \mathbf{v}_{\text{base}}$.

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 4.11 (Collision structure). *Let $\mathcal{P} = (\mathbf{O}, \mathcal{C})$ 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, \dots, c_n) of \mathcal{C} for $c_i = (Op_i, \mathbf{k}_i, \mathbf{q}_i, \mathbf{a}_i)$ and a tuple $(dir_{i^*}, \dots, dir_n)$ for $dir_i \in \{\text{Forward}, \text{Backward}\}$, such that the following two conditions hold:*

1. *Let $\mathcal{F}_{i^*} = \{\mathbf{k}_1, \dots, \mathbf{k}_{i^*-1}, \mathbf{q}_1, \dots, \mathbf{q}_{i^*-1}, \mathbf{a}_1, \dots, \mathbf{a}_{i^*-1}\}$. One of the following is true:*
 - a) $dir_{i^*} = \text{Forward}$ and $\text{span}(\{\mathbf{k}_{i^*}, \mathbf{q}_{i^*}\}) \not\subseteq \text{span}(\mathcal{F}_{i^*} \cup \text{rows}(\mathbf{O}))$
 - b) $dir_{i^*} = \text{Backward}$ and $\text{span}(\{\mathbf{k}_{i^*}, \mathbf{a}_{i^*}\}) \not\subseteq \text{span}(\mathcal{F}_{i^*} \cup \text{rows}(\mathbf{O}))$
2. *For all $j \geq i^*$ let $\mathcal{F}_j = \{\mathbf{k}_1, \dots, \mathbf{k}_{j-1}, \mathbf{q}_1, \dots, \mathbf{q}_{j-1}, \mathbf{a}_1, \dots, \mathbf{a}_{j-1}\}$. One of the following is true:*
 - a) $dir_j = \text{Forward}$ and $\mathbf{a}_j \notin \text{span}(\mathcal{F}_j \cup \{\mathbf{k}_j, \mathbf{q}_j\} \cup \text{rows}(\mathbf{O}))$
 - b) $dir_j = \text{Backward}$ and $\mathbf{q}_j \notin \text{span}(\mathcal{F}_j \cup \{\mathbf{k}_j, \mathbf{a}_j\} \cup \text{rows}(\mathbf{O}))$

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

Definition 4.12 (Collision structure). *Let $\mathcal{P} = (\mathbf{O}, \mathcal{C})$ 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, \dots, c_n) of \mathcal{C} for $c_i = (Op_i, \mathbf{k}_i, \mathbf{q}_i, \mathbf{a}_i)$ and a tuple $(dir_{i^*}, \dots, dir_n)$ for $dir_i \in \{\text{Forward}, \text{Backward}\}$, such that:*

1. *The i^* 'th constraint is unconstrained by the output of \mathcal{P} and previous fixed constraints. Let $\mathcal{F} = \{\mathbf{k}_1, \dots, \mathbf{k}_{i^*-1}, \mathbf{q}_1, \dots, \mathbf{q}_{i^*-1}, \mathbf{a}_1, \dots, \mathbf{a}_{i^*-1}\}$ denote the vectors fixed by previous constraints in the ordering.*
 - a) *if $dir_{i^*} = \text{Forward}$, the input of the query associated to c_{i^*} is unconstrained:*

$$\text{span}(\{\mathbf{k}_{i^*}, \mathbf{q}_{i^*}\}) \not\subseteq \text{span}(\mathcal{F} \cup \text{rows}(\mathbf{O}))$$

b) if $\text{dir}_{i^*} = \text{Backward}$, the output of the query associated to c_{i^*} is unconstrained:

$$\text{span}(\{\mathbf{k}_{i^*}, \mathbf{a}_{i^*}\}) \not\subseteq \text{span}(\mathcal{F} \cup \text{rows}(\mathbf{O}))$$

2. For all $j \geq i^*$ the constraint c_j does not contradict previous constraints. Let $\mathcal{F} = \{\mathbf{k}_1, \dots, \mathbf{k}_{j-1}, \mathbf{q}_1, \dots, \mathbf{q}_{j-1}, \mathbf{a}_1, \dots, \mathbf{a}_{j-1}\}$ denote the vectors fixed by previous constraints in the ordering.

a) if $\text{dir}_j = \text{Forward}$

$$\mathbf{a}_j \notin \text{span}(\{\mathbf{k}_1, \dots, \mathbf{k}_{j-1}, \mathbf{q}_1, \dots, \mathbf{q}_{j-1}, \mathbf{a}_1, \dots, \mathbf{a}_{j-1}\} \cup \{\mathbf{k}_j, \mathbf{q}_j\} \cup \text{rows}(\mathbf{O}))$$

b) if $\text{dir}_j = \text{Backward}$

$$\mathbf{q}_j \notin \text{span}(\{\mathbf{k}_1, \dots, \mathbf{k}_{j-1}, \mathbf{q}_1, \dots, \mathbf{q}_{j-1}, \mathbf{a}_1, \dots, \mathbf{a}_{j-1}\} \cup \{\mathbf{k}_j, \mathbf{a}_j\} \cup \text{rows}(\mathbf{O}))$$

Lemma 4.13 (Collision structure gives second preimages). *If collision structure blabla exists for $\mathcal{P} = (\mathbf{O}, \mathcal{C})$ then blabla with probability 1.*

FindSecondPreimage()
<hr/> Compute \mathbf{v}_{base} by executing $\mathcal{P}^{\mathcal{E}}(\mathbf{x})$... 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.