Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling the Ideal Cipher in Linicrypt

Master Thesis

Frederik Semmel

May 26, 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.

```
$\mathcal{P}^H(x, y)$
    $r \leftarrow\$ \mathbb{F}$
    return $H(x + r) + y$
```

$\rightsquigarrow$

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

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 $v_1, \dots, v_n$.

A **base variable** is an intermediate variable which was created by retrieving an input, calling the random oracle H or sampling from $\mathbb{F}$. Let base be the number of base variables and let $\boldsymbol{v}_{\mathsf{base}} \in \mathbb{F}^{\mathsf{base}}$ denote the vector of the base variables for an excecution of $\mathcal{P}$. 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 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 $v_i = \boldsymbol{v}_i \times \boldsymbol{v}_{\mathsf{base}}$ for every excecution of $\mathcal{P}$. For example, if v_i is the j'th base variable, then $\boldsymbol{v}_i = \begin{bmatrix}0, \dots, 1, \dots, 0\end{bmatrix}$, where the 1 is in the j'th position. We follow the convention to write vectors in $\mathbb{F}^{\mathsf{base}}$ using a bold font.

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

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{o}_1 \\ \vdots \\ \boldsymbol{o}_k \end{bmatrix}.$$

By the definition of the associated vectors $\boldsymbol{o}_i$ we have $\boldsymbol{M} \times \boldsymbol{v}_{\mathsf{base}} = \begin{bmatrix}o_1, \dots, o_k\end{bmatrix}^\top$. The output matrix describes the linear correlations in the output of $\mathcal{P}$.

But the output matrix doesn't describe all correlations in $\boldsymbol{v}_{\mathsf{base}}$. Namely, the relationship between the queries and answers to the random oracle H need to be captured algebraically. Let $v_i = H(t_i, (q_1, \dots, q_n))$ be an operation in $\mathcal{P}$. The **associated oracle constraint** c to this operation is

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

This should be interpreted as the requirement that $\boldsymbol{v}_i \times \boldsymbol{v}_{\mathsf{base}} = H\left(t_i, \boldsymbol{Q}_i \times \boldsymbol{v}_{\mathsf{base}}\right)$. 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 $(t, \boldsymbol{Q})$.

TODO: Base variables which are created by a call of the same vector have to be removed. This is what is done wlog in the first two papers. This case corresponds to calling the random orcacle twice on the same input. But all this seems to be closely related to the ability of the adversary to make queries collapse, therefore I leave this open until I have thought more about that.

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

2.1.4 Normalization and Indistinguishability

Todo: This leads to indistinguishability, normalization and theorem from first paper

2.1.5 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 $\mathcal{P}^H_{\text{deg}}$ 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 $w' = x' + y'$.

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

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

We have constructed the algebraic representation of a Linicrypt program. A question that arises is: Which combination of matrices of the structure $(\boldsymbol{M}, \mathcal{C})$ correspond to some valid Linicrypt program?

To answer this question, we need to define the terminology more carefully.

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

We call t the nonce and refer to $\boldsymbol{Q}$ and $\boldsymbol{a}$ as the query and answer to the random oracle. Usually we just say constraint when the other variables are clear from the context. The constraints $(t, \boldsymbol{Q}, \boldsymbol{a})$ encodes a relationship between the base variables $\boldsymbol{v}_{\mathsf{base}} \in \mathbb{F}^{\mathsf{base}}$ of a program. Namely $H(t, \boldsymbol{Q}\boldsymbol{v}_{\mathsf{base}}) = \boldsymbol{a}\boldsymbol{v}_{\mathsf{base}}$. Because H is a well-defined function, and not just any relation, these requirements extend to the constraints.

Definition 3.2. *A set of (random oracle) constraints $\mathcal{C}$ is well-defined if for any pair of constraints $c_i, c_j \in \mathcal{C}$ we have $(t_i, \boldsymbol{Q}_i) = (t_j, \boldsymbol{Q}_j) \implies \boldsymbol{a}_i = \boldsymbol{a}_j$.*

When we use a set of constraints, we will implicitly also require that it is well-defined. Now we can characterize which sets of constraints correspond to a Linicrypt program.

Definition 3.3 (Solvable)**.** *Let $\mathcal{C}$ be a finite set of valid constraints. $\mathcal{C}$ is (deterministically) solvable if there exists an ordering $(c_1, \ldots, c_n)$ of $\mathcal{C}$ and a subspace $\mathcal{F}$ of $\mathbb{F}^{\mathsf{base}}$ such that for all $i = 1, \ldots n$:*

1. $\mathsf{span}(\boldsymbol{Q}_i) \subset \mathcal{F} + \mathsf{span}(c_1, \ldots, c_{i-1})$
2. $\boldsymbol{a}_i \notin \mathcal{F} + \mathsf{span}(c_1, \ldots, c_{i-1}) + \mathsf{span}(\boldsymbol{Q}_i)$
3. *TODO or this notation*

4. $\mathsf{rows}(\boldsymbol{Q}_i) \subset \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}(c_1) \cup \cdots \cup \mathsf{rows}(c_{i-1})\big)$

5. $\boldsymbol{a}_i \notin \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}(c_1) \cup \cdots \cup \mathsf{rows}(c_{i-1}) \cup \mathsf{rows}(\boldsymbol{Q}_i)\big)$

We call $\mathcal{F}$ the solvable space (TODO or free space or fixable space or fixed space) of $\mathcal{C}$ and write $\mathsf{sol}(\mathcal{C}) = \mathcal{F}$. We call $(c_1, \ldots, c_n)$ the (solution) ordering of $\mathcal{C}$.

If we construct the algebraic representation of a Linicrypt program $\mathcal{P}$, we get a solvable set of constraints. Indeed, the ordering of the constraints in the definition can be exactly the order of the corresponding queries in the execution of $\mathcal{P}$. In this case, the solvable space would be the space spanned by the corresponding vectors to the input variables and the randomly sampled variables.

The other direction is also true.

Lemma 3.4 (Solvable constraints)**.** *Let $\mathcal{C}$ be a set of solvable constraints and $k = \dim(\mathsf{sol}(\mathcal{C}))$. Let $\mathsf{out} \in \mathbb{N}$ be arbitrary and $\mathsf{in} \leq k$. Let $\boldsymbol{M} \in \mathbb{F}^{\mathsf{out} \times \mathsf{base}}$ be an arbitrary output matrix. Then there is a basis change $\boldsymbol{B} \in \mathbb{F}^{\mathsf{base} \times \mathsf{base}}$ and a Linicrypt program $\mathcal{P}$ taking in inputs such that $(\boldsymbol{M}, \mathcal{C}\boldsymbol{B}^{-1})$ is it's algebraic representation.*

Proof. Let $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ be a basis for $\mathsf{sol}(\mathcal{C})$ and let $(c_1, \ldots, c_n)$ be the ordering. The new basis for $\mathbb{F}^{\mathsf{base}}$ is $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$.

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{x}_1 & \ldots & \boldsymbol{x}_k & \boldsymbol{a}_1 & \ldots & \boldsymbol{a}_n \end{bmatrix}^\top$$

TODO or

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

Then we define the constraints $\mathcal{C}'$ via $t'_i = t_i$, $\boldsymbol{Q}'_i B = \boldsymbol{Q}_i$ and $\boldsymbol{a}'_i B = \boldsymbol{a}_i$. Note, that, as $\mathcal{C}$ is solvable via the ordering $(c_1, \ldots, c_n)$, these constraints have the form

$$\begin{aligned} \boldsymbol{Q}'_i &= \begin{bmatrix} \lambda^i_1 & \cdots & \lambda^i_{i-1} & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad \text{for} \quad \lambda^i_j \in \mathbb{F} \\ \boldsymbol{a}'_i &= \begin{bmatrix} 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \end{bmatrix} \end{aligned}$$

This is the correct form for an algebraic representation of a program $\mathcal{P}$ taking $\mathsf{in} \leq k$ inputs, randomly sampling the next $k - \mathsf{in} \geq 0$ base variables, and outputting according to $\boldsymbol{M}$. □

TODO maybe switch to using input matrices, this would maybe clean things up later

Lemma 3.5 (Solvable constraints)**.** *Let $\mathcal{C}$ be a set of solvable constraints and $k = \dim(\mathsf{sol}(\mathcal{C}))$. Let* $\mathsf{out} \in \mathbb{N}$ *be arbitrary and* $\mathsf{in} \leq k$. *Let $\boldsymbol{M} \in \mathbb{F}^{\mathsf{out} \times \mathsf{base}}$ be arbitrary and $\boldsymbol{I} \in \mathbb{F}^{\mathsf{in} \times \mathsf{base}}$ such that* $\mathsf{span}(\boldsymbol{I}) \subseteq \mathsf{sol}(\mathcal{C})$. *Then there is a basis change $\boldsymbol{B} \in \mathbb{F}^{\mathsf{base} \times \mathsf{base}}$ and a Linicrypt program $\mathcal{P}$ such that $(\boldsymbol{I}\boldsymbol{B}^{-1}, \boldsymbol{M}\boldsymbol{B}^{-1}, \mathcal{C}\boldsymbol{B}^{-1})$ is it's algebraic representation.*

3.2 Revisiting Collision Structures

Using this language we can argue about the invertibility of a Linicrypt program and about the possibility to directly calculate second preimages. Let $\mathcal{P} = (\boldsymbol{M}, \mathcal{C})$ be a Linicrypt program.

Lemma 3.6. *$\mathcal{P}$ is invertible if* $\mathsf{rows}(\boldsymbol{M}) \subseteq \mathsf{sol}(\mathcal{C})$.

Proof. TODO □

Lemma 3.7. *$\mathcal{P}$ has a collision structure if $\mathcal{C} = \mathcal{C}_{\mathit{fixed}} \sqcup \mathcal{C}_{cs}$ such that*

$$\mathsf{sol}(\mathcal{C}_{cs}) \supset \mathsf{span}\big(\mathsf{rows}(\mathcal{C}_{\mathit{fixed}}) \cup \mathsf{rows}(\boldsymbol{M})\big). \tag{3.1}$$

Proof. TODO □

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. This is the same role that w' plays in the example from [**?**].

This characterization directly includes the case of degeneracy, because then $\mathcal{C}_{\text{cs}} = \{\}$ and $\mathsf{sol}(\mathcal{C}_{\text{cs}}) = \mathbb{F}^{\mathsf{base}}$, while degeneracy means precisely $\mathbb{F}^{\mathsf{base}} \supset \mathsf{span}(\mathsf{rows}(\mathcal{C}) \cup \mathsf{rows}(\boldsymbol{M}))$.

The following example was slightly adapted so that it is invertible.

```
$\mathcal{P}^{\mathcal{E}}_{\mathsf{inv},1}(x, y, z)$
    $w = H(x) + H(z) + y$
    **return** $(H(w) + x, z)$
```

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{q}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{a}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{q}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{q}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{q}_3 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{q}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note, that (c_3, c_2, c_1) is an ordering solving $\mathcal{C}$ and $\mathsf{rows}(M) \subset \mathsf{sol}(\mathcal{C})$.

TODO finish typing this example

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.

The command $y = E(k, x)$ in a 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

Let $\mathcal{P}$ be a ideal cipher Linicrypt program. For each query to E of the form $y = E(k, x)$ we define the **associated ideal cipher constraint** $(E, \boldsymbol{k}, \boldsymbol{x}, \boldsymbol{y})$. Each query to D of the form $x = D(k, y)$, is associated with the constraint $(D, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{x})$.

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

$$\boldsymbol{M} = \begin{bmatrix}0, 0, 1, 0, 1\end{bmatrix} \qquad \mathcal{C} = \{(E, \boldsymbol{k}_1, \boldsymbol{x}_1, \boldsymbol{y}_1), (E, \boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2)\}$$

where

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

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:

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

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

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

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

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

This reduces the dimension of the space of possible solutions for $\boldsymbol{v}'_{\mathsf{base}}$ to 4, as $\boldsymbol{M}\boldsymbol{v}_{\mathsf{base}}$ is in the range of $\boldsymbol{M}$ by definition. Now note, that neither $\boldsymbol{k}_2$ nor $\boldsymbol{x}_2$ are in the span of $\mathsf{rows}(\boldsymbol{M})$. 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 $\mathsf{rows}(\boldsymbol{M}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2\}$. 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 $\mathsf{rows}(\boldsymbol{M}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2\}$ the intermediate variables k'_1 and x'_1 are uniquely determined. E.g. $\boldsymbol{k}_1 = \boldsymbol{M} - \boldsymbol{y}_2$ and therefore $k'_1 = \mathcal{P}^{\mathcal{E}}_{\mathsf{col},1}(a,b,c) - y'_2$.

Finally, note that $\boldsymbol{y}_1 \notin \mathsf{span}\big(\mathsf{rows}(\boldsymbol{M}) \cup \{\boldsymbol{k}_2, \boldsymbol{x}_2, \boldsymbol{y}_2\} \cup \{\boldsymbol{k}_1, \boldsymbol{x}_1\}\big)$. 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 $y_1 + y_2$
```

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

For this program there is a similar procedure to find second preimages. As before, the first constraint is $\boldsymbol{M}$ 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 3.8 (Collision structure)**.** *Let $\mathcal{P} = (\boldsymbol{M}, \mathcal{C})$ 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* $\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{q}_{i^*}\}\big) \not\subseteq \mathsf{span}\big(\mathcal{F}_{i^*} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$

   b) $dir_{i^*} = \mathsf{Backward}$ *and* $\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{a}_{i^*}\}\big) \not\subseteq \mathsf{span}\big(\mathcal{F}_{i^*} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$

2. *For all $j \geq i^*$ let $\mathcal{F}_j = \{\boldsymbol{k}_1, \dots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \dots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{j-1}\}$. One of the following is true:*

   a) $dir_j = \mathsf{Forward}$ *and* $\boldsymbol{a}_j \notin \mathsf{span}\big(\mathcal{F}_j \cup \{\boldsymbol{k}_j, \boldsymbol{q}_j\} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$

   b) $dir_j = \mathsf{Backward}$ *and* $\boldsymbol{q}_j \notin \mathsf{span}\big(\mathcal{F}_j \cup \{\boldsymbol{k}_j, \boldsymbol{a}_j\} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$

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

Definition 3.9 (Collision structure)**.** *Let $\mathcal{P} = (\boldsymbol{M}, \mathcal{C})$ 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 $c_i = (Op_i, \boldsymbol{k}_i, \boldsymbol{q}_i, \boldsymbol{a}_i)$ and a tuple $(dir_{i^*}, \dots, dir_n)$ 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 $\mathcal{F} = \{\boldsymbol{k}_1, \dots, \boldsymbol{k}_{i^*-1}, \boldsymbol{q}_1, \dots, \boldsymbol{q}_{i^*-1}, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{i^*-1}\}$ 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:*

   $$\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{q}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$$

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

   $$\mathsf{span}\big(\{\boldsymbol{k}_{i^*}, \boldsymbol{a}_{i^*}\}\big) \nsubseteq \mathsf{span}\big(\mathcal{F} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$$

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

   a) *if $dir_j = \mathsf{Forward}$*

   $$\boldsymbol{a}_j \notin \mathsf{span}\big(\{\boldsymbol{k}_1, \dots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \dots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{j-1}, \} \cup \{\boldsymbol{k}_j, \boldsymbol{q}_j\} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$$

   b) *if $dir_j = \mathsf{Backward}$*

   $$\boldsymbol{q}_j \notin \mathsf{span}\big(\{\boldsymbol{k}_1, \dots, \boldsymbol{k}_{j-1}, \boldsymbol{q}_1, \dots, \boldsymbol{q}_{j-1}, \boldsymbol{a}_1, \dots, \boldsymbol{a}_{j-1}, \} \cup \{\boldsymbol{k}_j, \boldsymbol{a}_j\} \cup \mathsf{rows}\,(\boldsymbol{M})\,\big)$$

Lemma 3.10 (Collision structure gives second preimages)**.** *If collision structure blabla exists for $\mathcal{P} = (\boldsymbol{M}, \mathcal{C})$ then blabla with probability 1.*

```
FindSecondPreimage()
    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.