Tools for analysing the security of symmetric primitives

Kristjan Krips
University of Tartu, Estonia

Abstract. There is a lack of tools for analysing the security of symmetric primitives. Such tools can help researchers to prove the correctness and security of protocols by using reductions. The proof is done by applying reductions to the protocol one step at a time. ProveIt is one of these tools that is currently being developed. For adding reductions to ProveIt it is necessary to analyse different symmetric primitives to find out which reductions are needed.

1 Introduction

There is a lack of tools for analysing the security of symmetric primitives. Such tools can help researchers to verify the correctness and security of protocols that use symmetric primitives. The probability of breaking the protocol can be found by doing a finite number of reductions. However, it is not convenient to do such proofs on paper and therefore a tool would help the researcher. ProveIt is a tool that is currently being developed for doing the reduction based proofs. ProveIt lets the user to choose which reductions to do on a given protocol. As a first step the user has to import the protocol into ProveIt or write the protocol in ProveIt. After completing the first step the user can start to transform the initial protocol one step at a time in order to find the probability of a successful attack against it. After a finite number of reductions the proof is complete and the probability of a successful attack is found. However, currently ProveIt is able to do only a few reductions and this limits the usability of the tool. In order to add reductions to ProveIt it is necessary to analyse different symmetric primitives to find out which reductions are required. For that it is necessary to find enough protocols that contain symmetric primitives and to do manually the reductions on these protocols. In this seminar paper we are analysing a primitive in order to show how such proofs are done. In the process we find the necessary reductions and describe how to implement them in ProveIt.

2 ProveIt

ProveIt is a tool for proving the security of symmetric primitives. ProveIt has its own specific syntax and the analysable protocols have to written in this language. After inserting the protocol ProveIt uses a parser for generating a tree of the
protocol. The reductions are done on this syntax tree. All reductions have to be done manually, i.e. the user has to interact with ProveIt. Performed reductions are displayed in ProveIt.

TODO: add citation to a paper about ProveIt

3 Reductions

Statement Switching

A protocol might contain a statement that should be moved in the protocol tree. For that we implement a reduction called Statement Switching. It works by choosing a statement which will be moved and a statement after which it will be placed in the protocol tree. Then the initial statement is inserted after the chosen node and the initial node is removed from the protocol tree. For example, if the adversary chooses the output value too early then the statement that generates the output could be switched with a statement that should be seen by the adversary before the adversary generates the output.

Dead Code Elimination

A protocol might contain redundant statements that are not needed in order to produce the output. These statements can be removed without changing the probability of the outcome. In order to remove the redundant nodes from the protocol tree a data flow analysis has to made. For that we have to generate a control flow graph and check which variables are required to produce the output. The remaining variables/statements can be removed from the protocol tree. We start by traversing the protocol tree using depth first search in order to create the control flow graph. By traversing the protocol tree we label the nodes and mark for each node which are the parent nodes and which are the child nodes. With this information we can create the control flow graph. In the control flow graph we know for each node where the control can go and from where it could have come from. On the control flow graph we can perform the required analysis for dead code elimination. For more information about dead code elimination see [NNH99].

An example of a control flow graph and the dead code elimination is shown on Figure 1.

Remove condition

A condition for a variable can be removed if we know the probability for the condition to succeed. Therefore after removing a condition on a variable two consequent proof steps differ by the probability of the condition to succeed.

TODO: implement + add details
4 Protocol to Analyse

In order to show what are reduction based proofs we are going to do proofs on the security of CPA and CCA on schemes from [BR93]. The article is about protocols with random oracles, i.e. all parties in the protocol have access to a public random oracle. If a protocol can be proved to be correct in a random oracle model then it might be possible to replace the random oracle by an existing function in order to get a correct and realistic protocol. This is important because some results are only possible in the random oracle setting and are impractical in the standard setting. But from our viewpoint the article is good as many security proofs finally reduce to settings with a random oracle.

We view a public key encryption scheme in the random oracle model:

\[ E_G(x) = (f(r), G(r) + x), \]

where \( r \) is a random value from the domain \( f \). For the encryption scheme to function addition has to be defined between the input set and the set of the random values. Thus, encryption function maps a value from message space to \( \mathbb{R} \times M \),

\[ E_G : M \rightarrow \mathbb{R} \times M. \]

They define \( G : \mathbb{R} \rightarrow \mathbb{R} \) as a random generator(random oracle), \( f \) as a trapdoor permutation and \( f^{-1} \) as its inverse. The random oracle \( G \) is defined so that \( G : \mathbb{R} \rightarrow Y \), the trapdoor permutation and its inverse are defined so that \( f : \mathbb{R} \rightarrow \mathbb{R} \) and \( f^{-1} : \mathbb{R} \rightarrow \mathbb{R} \).

The encryption and decryption algorithms are generated by a probabilistic polynomial time(PPT) generator \( GEN \), which outputs a pair of probabilistic algorithms \( (E,D) \). The algorithm \( E \) is made public while \( D \) is kept secret. In our case
the generator $\text{GEN}$ generates $(f, f^{-1})$ so that

$$E^G \leftarrow \langle x : r \leftarrow R : (f(r), G(r) \oplus x) \rangle$$

$$D^G(y, s) = s \oplus G(f^{-1}(y)).$$

The decryption function $D^G$ is defined on $D^G : R \times M \rightarrow M \times M$. Therefore, in this encryption scheme the random oracle, trapdoor permutation and its inverse are defined on the following sets: $G : M \rightarrow M$, $f : M \rightarrow M$ and $f^{-1} : M \rightarrow M$.

Now we will define security in the random oracle model. A chosen plaintext adversary $A$ runs in two independent stages, in the first stage it chooses two plaintexts and in the second stage it is given an encryption of one of these plaintexts and the adversary outputs a guess about which plaintext was encrypted. For an encryption scheme to be secure in the random oracle model we require that no chosen plaintext adversary is able to distinguish an encryption when given two plaintexts and an encryption of one of these plaintexts with greater than negligible probability. I.e. when the adversary is given two uniformly chosen plaintexts, an encryption of one of these plaintexts and the encryption algorithm, then he is not able to distinguish which of the two plaintexts was encrypted.

We will do the proofs of security against CPA and CCA in two phases, for both we will first prove the security in a general model and then in the full model using the realistic setting. For doing the reduction based proofs we start by defining two games $G_0$ and $G_1$. In the game $G_0$ the adversary always gets an encryption of the first message and in game $G_1$ the adversary always gets the encryption of the second message. After these games are defined we perform reductions on both of them until we see that the games $G_0$ and $G_1$ are equal.

We start by defining IND-CPA using games $G_0$ and $G_1$.

$$G^A_0 \leftarrow \{ f : R \rightarrow M \}$$
$$E, D \leftarrow \text{Gen}$$
$$m_0, m_1 \leftarrow A^G(E)$$
$$c \leftarrow E^G(m_0)$$
$$\text{return } A^G(c)$$

The advantage of an IND-CPA adversary $A$ in distinguishing games $G_0$ and $G_1$ is defined as

$$\text{Adv}_{G_0, G_1}^{\text{Ind}}(A) = |\Pr[G^A_0 = 1] - \Pr[G^A_1 = 1]|.$$
If an encryption scheme is secure against IND-CPA adversary then the advantage should be negligible.

Now we substitute our encryption scheme into the IND-CPA games $G_0$ and $G_1$.

$$
G^A_0 \begin{cases} 
G \leftarrow \{ f : R \rightarrow M \} \\
f \leftarrow F_{\text{trapdoor}} \\
m_0, m_1 \leftarrow A^G(f) \\
r \leftarrow R \\
c_1 \leftarrow f(r) \\
c_2 \leftarrow G(r) + m_0 \\
\text{return } A^G(c_1, c_2)
\end{cases}
$$

$$
G^A_1 \begin{cases} 
G \leftarrow \{ f : R \rightarrow M \} \\
f \leftarrow F_{\text{trapdoor}} \\
m_0, m_1 \leftarrow A^G(f) \\
r \leftarrow R \\
c_1 \leftarrow f(r) \\
c_2 \leftarrow G(r) + m_1 \\
\text{return } A^G(c_1, c_2)
\end{cases}
$$

In order to simplify the proof we consider a class of adversaries $A$ that do not query $G$. With that assumption we can remove $G$ from the games. There are two reductions in the following games, we can randomly pick a value $x \in M$ and substitute it in the statement that generates $c_2$ in order to simulate $G$. This can be done because it is not possible to distinguish a randomly picked value from a value returned by the random oracle if they are from the same set $M$.

$$
G^A_0 \begin{cases} 
f \leftarrow F_{\text{trapdoor}} \\
m_0, m_1 \leftarrow A(f) \\
r \leftarrow R \\
c_1 \leftarrow f(r) \\
x \leftarrow M \\
c_2 \leftarrow x + m_0 \\
\text{return } A(c_1, c_2)
\end{cases}
$$

$$
G^A_1 \begin{cases} 
f \leftarrow F_{\text{trapdoor}} \\
m_0, m_1 \leftarrow A(f) \\
r \leftarrow R \\
c_1 \leftarrow f(r) \\
x \leftarrow M \\
c_2 \leftarrow x + m_1 \\
\text{return } A(c_1, c_2)
\end{cases}
$$

Now we notice that $m_0, m_1$ are from the same message space as $x$, thus we can replace $c_2$ by a uniformly chosen element from $M$. After this reduction games $G_0$ and $G_1$ are equal and thus they are indistinguishable by the adversary.

$$
G^A_0 \begin{cases} 
f \leftarrow F_{\text{trapdoor}} \\
m_0, m_1 \leftarrow A(f) \\
r \leftarrow R \\
c_1 \leftarrow f(r) \\
c_2 \leftarrow M \\
\text{return } A(c_1, c_2)
\end{cases}
$$
Proof of security against CPA

Now we will view the realistic setting of the encryption scheme where the adversary $A$ is able to query the random oracle. We define the adversary $A$ that it is able to make $q_1$ queries in the first stage and $q_2$ queries in the second stage. Thus, we will have to modify the initial games $G_0$ and $G_1$ so that the adversary can query the random oracle before $m_0, m_1$ are chosen and after $c_1, c_2$ have been fixed. Additionally we assume that each $x_i, x'_i$ is queried only once as querying the same value twice does not give additional information to the adversary. Besides that, we add extra conditions for the adversary in the case $x_i = r$ or $x'_i = r$.

The first condition is important because if it holds then the adversary knows the corresponding response of the random oracle $G$ and will therefore know which plaintext was encrypted. In order to model this situation we will create a collision set $CS$ that will contain all the queries of the adversary. If the uniformly chosen random value $r$ is in the collision set then the adversary knows which plaintext is encrypted and will output the result. This could be described by a game with an if condition that checks whether there exists a collision or not. We will add the if condition of the hypothetical game into our games to output the response when there is a collision. However, the probability of collision $\Pr[\text{collision}] = \frac{q_1 + q_2}{m}$ is negligible because the number of queries is limited and $r$ is uniformly taken from the set $R$. Thus, adding this condition into the games changes the outcome only by a negligible probability.

We can also bound the adversary in the the second stage after he has knowledge of the encryption, i.e. $(c_1, c_2)$. Namely, if the adversary would query a value $x'_i$ equal to the random value $r$ that was used for encrypting, then he would know which plaintext was encrypted. Therefore, we could model a condition to checks if $x'_i = r$ and output the results if the condition evaluates to true. The probability of this equality evaluating to true is negligible, as this would mean that the adversary has found an inverse of a value generated by the trapdoor permutation. Therefore, adding this condition into the game changes the outcome only by a negligible probability.
On the next pages we will show how to prove the security against CPA in the realistic setting.

\[ G_0^A \]

\[
\begin{align*}
G & \leftarrow \{ f : R \to M \} \\
f & \leftarrow F_{\text{trapdoor}} \\
CS & \leftarrow 0 \\
y_0 & \leftarrow 0 \\
& \quad \text{For } i \in \{1, \ldots, q_1\} \text{ do} \\
& \quad \quad x_i \leftarrow A(y_{i-1}) \\
& \quad \quad CS \leftarrow CS \cup x_i \\
& \quad \quad y_i \leftarrow G(x_i) \\
m_0, m_1 & \leftarrow A^G(f) \\
r & \leftarrow R \\
& \quad \text{if } (r \in CS) \{ \text{return 1} \} \\
c_1 & \leftarrow f(r) \\
c_2 & \leftarrow x + m_0 \\
A(c_1, c_2) & \leftarrow 0 \\
\overline{y}_0 & \leftarrow 0 \\
& \quad \text{For } i \in \{1, \ldots, q_2\} \text{ do} \\
& \quad \quad \pi_i \leftarrow A(\overline{y}_{i-1}) \\
& \quad \quad \text{if } (\pi_i = r) \{ \text{return 1} \} \\
& \quad \quad \overline{y}_i \leftarrow G(\pi_i) \\
& \quad \text{return } A^G(c_1, c_2)
\end{align*}
\]

\[ G_1^A \]

\[
\begin{align*}
G & \leftarrow \{ f : R \to M \} \\
f & \leftarrow F_{\text{trapdoor}} \\
CS & \leftarrow 0 \\
y_0 & \leftarrow 0 \\
& \quad \text{For } i \in \{1, \ldots, q_1\} \text{ do} \\
& \quad \quad x_i \leftarrow A(y_{i-1}) \\
& \quad \quad CS \leftarrow CS \cup x_i \\
& \quad \quad y_i \leftarrow G(x_i) \\
m_0, m_1 & \leftarrow A^G(f) \\
r & \leftarrow R \\
& \quad \text{if } (r \in CS) \{ \text{return 1} \} \\
c_1 & \leftarrow f(r) \\
c_2 & \leftarrow G(r) + m_1 \\
A(c_1, c_2) & \leftarrow 0 \\
\overline{y}_0 & \leftarrow 0 \\
& \quad \text{For } i \in \{1, \ldots, q_2\} \text{ do} \\
& \quad \quad \pi_i \leftarrow A(\overline{y}_{i-1}) \\
& \quad \quad \text{if } (\pi_i = r) \{ \text{return 1} \} \\
& \quad \quad \overline{y}_i \leftarrow G(\pi_i) \\
& \quad \text{return } A^G(c_1, c_2)
\end{align*}
\]
Now, for the reduction step we have to replace the random oracle by a uniformly chosen value from the message space $M$. In order to simulate the random oracle we have to answer the queries that the adversary makes. We can substitute the answers of the random oracle by uniformly chosen values from the same range $M$. This substitution is not noticeable by the adversary as the distribution of the responses stays the same. Besides answering the queries of the adversary we will have to simulate the random oracle in generating the ciphertext. In order to do that we can uniformly take a value $x$ from $M$ and replace the output of $G(r)$ by $x$. As $x$ is uniformly taken from $M$ it is not possible to distinguish if $x$ was generated by the random oracle. Now we can remove the random oracle from the remaining games.

\[
\begin{align*}
G_0^A & \quad f \leftarrow F_{\text{trapdoor}} \\
CS & \leftarrow \emptyset \\
y_0 & \leftarrow \emptyset \\
For i \in \{1, ..., q_1\} do & \\
x_i & \leftarrow A(y_{i-1}) \\
CS & \leftarrow CS \cup x_i \\
y_i & \leftarrow M \\
m_0, m_1 & \leftarrow A(f) \\
r & \leftarrow R \\
if(r \in CS) \{ \text{return 1} \} & \\
c_1 & \leftarrow f(r) \\
x & \leftarrow M \\
c_2 & \leftarrow x + m_0 \\
A(c_1, c_2) & \\
\overline{y}_0 & \leftarrow \emptyset \\
For i \in \{1, ..., q_2\} do & \\
\overline{x}_i & \leftarrow A(\overline{y}_{i-1}) \\
if(\overline{x}_i = r) \{ \text{return 1} \} & \\
\overline{y}_i & \leftarrow M \\
return A(c_1, c_2)
\end{align*}
\]

\[
\begin{align*}
G_1^A & \quad f \leftarrow F_{\text{trapdoor}} \\
CS & \leftarrow \emptyset \\
y_0 & \leftarrow M \\
For i \in \{1, ..., q_1\} do & \\
x_i & \leftarrow A(y_{i-1}) \\
CS & \leftarrow CS \cup x_i \\
y_i & \leftarrow M \\
m_0, m_1 & \leftarrow A(f) \\
r & \leftarrow R \\
if(r \in CS) \{ \text{return 1} \} & \\
c_1 & \leftarrow f(r) \\
x & \leftarrow M \\
c_2 & \leftarrow x + m_1 \\
A(c_1, c_2) & \\
\overline{y}_0 & \leftarrow \emptyset \\
For i \in \{1, ..., q_2\} do & \\
\overline{x}_i & \leftarrow A(\overline{y}_{i-1}) \\
if(\overline{x}_i = r) \{ \text{return 1} \} & \\
\overline{y}_i & \leftarrow M \\
return A(c_1, c_2)
\end{align*}
\]
Now we notice that \( m_0, m_1 \) are from the same message space as \( x \), thus we can replace \( c_2 \) by a uniformly chosen element from \( M \). After this reduction games \( G_0 \) and \( G_1 \) are equal and therefore not distinguishable by the adversary.

**Proof of security against CCA-1** In the CCA the adversary has access to a decryption oracle. At first we will do a proof of CCA-1, i.e. the non-adaptive case. As in the CPA, the adversary runs in two independent stages, in the first stage it can make encryption and decryption queries and in the second case it can make only encryption queries. The encryption scheme that we will view against CCA is slightly different from the previous scheme as the previous scheme is not secure against CCA in the random oracle model. Thus, in this case the generator \( \text{GEN} \) generates \((f, f^{-1})\) so that

\[
E^{G,H} \leftarrow \{x : r \leftarrow R : (f(r), G(r) + x, H(r, x))\}
\]
and $D^{G,H}$, where $H$ is a random hash function derived from the random oracle. In order to decrypt a string $y$ it has to be parsed to the form $(a,w,b)$ where $|a| = |b|$ and $D^{G,H} = w + G(f^{-1}(a))$ if $H(f^{-1}(a), w + G(f^{-1}(a))) = b$ and 0 otherwise. We will define a $q_1, q_2, q_3$ adversary who will make $q_1$ encryption queries, $q_2$ decryption queries in the first stage and $q_3$ encryption queries in the second stage.

We will begin by writing out games that define IND-CCA-1. Games $G_0$, $G_1$ are general without a specific encryption scheme. In these games the CCA-1 adversary queries the decryption oracle $q_1$ times and the encryption oracle $q_2$ times before generating the pair of plaintexts $(m_0, m_1)$. In the second stage the CCA-1 adversary is not allowed to query the decryption oracle but is allowed to make $q_3$ queries to the encryption oracle.

\[ G_0^A \]
\[
\begin{align*}
G &\leftarrow \{ f : R \rightarrow M \} \\
G_2 &\leftarrow f^{-1} : M \rightarrow M \\
E, D &\leftarrow \text{Gen} \\
z_0 &\leftarrow \emptyset \\
y_0 &\leftarrow \emptyset \\
\text{For } i \in \{1, \ldots, q_1\} \text{ do} & \\
& z_i \leftarrow A(x_{i-1}) \\
& x_{2i} \leftarrow G_2(z_i) \\
\text{For } i \in \{1, \ldots, q_2\} \text{ do} & \\
& x_i \leftarrow A(y_{i-1}) \\
& y_i \leftarrow G(x_i) \\
m_0, m_1 &\leftarrow A^G(E) \\
c &\leftarrow E^G(m_0) \\
A(c) & \\
y_0 &\leftarrow \emptyset \\
\text{For } i \in \{1, \ldots, q_3\} \text{ do} & \\
& \overline{x_i} \leftarrow A(\overline{y_{i-1}}) \\
& \overline{y_i} \leftarrow G(\overline{x_i}) \\
\text{return } & A^G(c)
\end{align*}
\]

\[ G_1^A \]
\[
\begin{align*}
G &\leftarrow \{ f : R \rightarrow M \} \\
G_2 &\leftarrow f^{-1} : M \rightarrow M \\
E, D &\leftarrow \text{Gen} \\
z_0 &\leftarrow \emptyset \\
y_0 &\leftarrow \emptyset \\
\text{For } i \in \{1, \ldots, q_1\} \text{ do} & \\
& z_i \leftarrow A(x_{i-1}) \\
& x_{2i} \leftarrow G_2(z_i) \\
\text{For } i \in \{1, \ldots, q_2\} \text{ do} & \\
& x_i \leftarrow A(y_{i-1}) \\
& y_i \leftarrow G(x_i) \\
m_0, m_1 &\leftarrow A^G(E) \\
c &\leftarrow E^G(m_1) \\
A(c) & \\
y_0 &\leftarrow \emptyset \\
\text{For } i \in \{1, \ldots, q_3\} \text{ do} & \\
& \overline{x_i} \leftarrow A(\overline{y_{i-1}}) \\
& \overline{y_i} \leftarrow G(\overline{x_i}) \\
\text{return } & A^G(c)
\end{align*}
\]
The advantage of an IND-CCA-1 adversary $A$ in distinguishing games $G_0$ and $G_1$ is defined as

$$\text{Adv}^{\text{Ind}}_{G_0, G_1}(A) = |\Pr_{G_0} [G_0 = 1] - \Pr_{G_1} [G_1 = 1]|.$$

If an encryption scheme is secure against IND-CCA-1 adversary then the advantage should be negligible.

Now we will substitute the previously described encryption scheme into games $G_0$ and $G_1$. As in the proof of security against CPA we will have to use collision sets in order to detect collisions. Additionally, we will not allow the adversary to choose the plaintexts which he has already queried.
$G_0^A$

$G \leftarrow \{f : R \rightarrow M\}$

$G^2 \leftarrow f^{-1} : M \rightarrow M$

$H : 0, 1^* \rightarrow K$

$f, f^{-1} \leftarrow F_{\text{trapdoor}}$

$CS \leftarrow \emptyset$

$y_{20} \leftarrow 0$, $y_0 \leftarrow 0$

For $i \in \{1, ..., q_1\}$ do

\[ y_{2i} \leftarrow A(y_{2i-1}) \]
\[ (a_i, w_i, b_i) \leftarrow A(y_{2i}) \]
\[ x_{2i} \leftarrow G^2(a_i) \]

For $i \in \{1, ..., q_2\}$ do

\[ x_i \leftarrow A(y_{i-1}}) \]
\[ CS \leftarrow CS \cup x_i \]
\[ y_i \leftarrow G(x_i) \]
\[ (x_{2i}, x_i) \leftarrow A(x_i) \]
\[ z_i \leftarrow H((x_i, x_{2i})) \]

$m_0, m_1 \leftarrow A^G(f)$

$r \leftarrow R$

\begin{itemize}
\item if ($r \in CS$) \{return 1\}
\item $c_1 \leftarrow f(r)$
\item $c_2 \leftarrow G(r) + m_1$
\item $c_3 \leftarrow H(r, m_1)$
\item $A(c_1, c_2, c_3)$
\item $y_0 \leftarrow \emptyset$
\end{itemize}

For $i \in \{1, ..., q_3\}$ do

\begin{itemize}
\item $\pi_i \leftarrow A(y_{i-1})$
\item if ($\pi_i = r$) \{return 1\}
\item $y_i \leftarrow G(\pi_i)$
\item $(x_{2i}, x_i) \leftarrow A(\pi_i)$
\item $z_i \leftarrow H((x_i, x_{2i}))$
\item return $A^G(c_1, c_2, c_3)$
\end{itemize}
Now we will remove the random oracle and for that we will have to simulate random oracle in the encryption and decryption queries and also in generating the ciphertext for the challenge. The simulations are done in a similar way as in the proof of security against CPA. The answers from the encryption, decryption oracle are replaced by a uniformly chosen value from the same range. In order to simulate the encrypting of the challenge a value $x$ will be uniformly taken from $M$ and used to replace the call to the encryption oracle. Besides that we will have to simulate the hash function and for that we will uniformly take a value from the range of the hash function and use it to replace the call to the hash function. With these substitutions we can remove the random oracle and the hash function. The collision probabilities are the same as in the proof of security against CPA and thus adding the checks for collision and inversing the encryption changes the outcome of the game only by a negligible probability.
$\mathcal{G}^A_0$

\begin{align*}
& f, f^{-1} \leftarrow \text{Trapdoor} \\
& CS \leftarrow \emptyset \\
& y_{20} \leftarrow 0, y_0 \leftarrow 0
\end{align*}

For $i \in \{1, \ldots, q_1\}$ do

\begin{align*}
& y_{2i} \leftarrow A(y_{2i-1}) \\
& (a_i, w_i, b_i) \leftarrow A(y_{2i}) \\
& x_{2i} \leftarrow x_{v_i}^	op
\end{align*}

For $i \in \{1, \ldots, q_2\}$ do

\begin{align*}
& x_i \leftarrow A(y_{i-1}) \\
& CS \leftarrow CS \cup x_i \\
& x_{\pi_i} \leftarrow M \\
& y_i \leftarrow x_{\pi_i} \\
& (x_{2i}, x_i) \leftarrow A(x_i) \\
& z_i \leftarrow K
\end{align*}

$m_0, m_1 \leftarrow A(f)$

$r \leftarrow R$

$\text{if}(r \in CS) \{ \text{return 1} \}$

$c_1 \leftarrow f(r)$

$x \leftarrow M$

$c_2 \leftarrow x + m_0$

$c_3 \leftarrow K$

$A(c_1, c_2, c_3)$

$y_0 \leftarrow \emptyset$

For $i \in \{1, \ldots, q_3\}$ do

\begin{align*}
& x_{\pi_i} \leftarrow A(y_{\pi_i-1}) \\
& \text{if}(x_{\pi_i} = r) \{ \text{return 1} \}$

$y_i \leftarrow M$

$(x_{2i}, x_i) \leftarrow A(x_{\pi_i})$

$z_i \leftarrow K$

\text{return } A(c_1, c_2, c_3)

$\mathcal{G}^d_0$

\begin{align*}
& f, f^{-1} \leftarrow \text{Trapdoor} \\
& CS \leftarrow \emptyset \\
& y_{20} \leftarrow 0, y_0 \leftarrow 0
\end{align*}

For $i \in \{1, \ldots, q_1\}$ do

\begin{align*}
& y_{2i} \leftarrow A(y_{2i-1}) \\
& (a_i, w_i, b_i) \leftarrow A(y_{2i}) \\
& x_{2i} \leftarrow x_{v_i}^	op
\end{align*}

For $i \in \{1, \ldots, q_2\}$ do

\begin{align*}
& x_i \leftarrow A(y_{i-1}) \\
& CS \leftarrow CS \cup x_i \\
& x_{\pi_i} \leftarrow M \\
& y_i \leftarrow x_{\pi_i} \\
& (x_{2i}, x_i) \leftarrow A(x_i) \\
& z_i \leftarrow K
\end{align*}

$m_0, m_1 \leftarrow A(f)$

$r \leftarrow R$

$\text{if}(r \in CS) \{ \text{return 1} \}$

$c_1 \leftarrow f(r)$

$x \leftarrow M$

$c_2 \leftarrow x + m_1$

$c_3 \leftarrow K$

$A(c_1, c_2, c_3)$

$y_0 \leftarrow \emptyset$

For $i \in \{1, \ldots, q_3\}$ do

\begin{align*}
& x_{\pi_i} \leftarrow A(y_{\pi_i-1}) \\
& \text{if}(x_{\pi_i} = r) \{ \text{return 1} \}$

$y_i \leftarrow M$

$(x_{2i}, x_i) \leftarrow A(x_{\pi_i})$

$z_i \leftarrow K$

\text{return } A(c_1, c_2, c_3)$
Now we can see that in both games $x$ and $m_0, m_1$ are from the same distribution and thus the sums $x + m_0$ and $x + m_1$ can be replaced by a uniformly chosen value from $M$. After these substitutions the games are equal and therefore not distinguishable by the adversary.

\[
\mathcal{G}_0^A \quad \begin{align*}
  & f, f^{-1} \leftarrow F_{\text{trapdoor}} \\
  & CS \leftarrow \emptyset \\
  & y_0, y_0 \leftarrow \emptyset \\
  & \text{For } i \in \{1, ..., q_1\} \text{ do} \\
  & \quad y_{2i} \leftarrow A(y_{2i-1}) \\
  & \quad (a_i, w_i, b_i) \leftarrow A(y_{2i}) \\
  & \quad x_{2i} \leftarrow \frac{y_{2i}}{x_{2i}} \\
  & \text{For } i \in \{1, ..., q_2\} \text{ do} \\
  & \quad x_i \leftarrow A(y_{i-1}) \\
  & \quad CS \leftarrow CS \cup x_i \\
  & \quad y_i \leftarrow \frac{y_{i}}{y_{i}} \\
  & \quad (x_{2i}, x_{i}) \leftarrow A(x_{i}) \\
  & \quad z_i \leftarrow K \\
  & m_0, m_1 \leftarrow A(f) \\
  & r \leftarrow R \\
  & \text{if } (r \in CS) \{ \text{return 1} \} \\
  & c_1 \leftarrow f(r) \\
  & c_2 \leftarrow M \\
  & c_3 \leftarrow K \\
  & A(c_1, c_2, c_3) \\
  & \bar{y}_0 \leftarrow \emptyset \\
  & \text{For } i \in \{1, ..., q_3\} \text{ do} \\
  & \quad \bar{x}_i \leftarrow A(y_i-1) \\
  & \quad \text{if } (\bar{x}_i = r) \{ \text{return 1} \} \\
  & \quad \bar{y}_i \leftarrow M \\
  & \quad (x_{2i}, x_i) \leftarrow A(x_i) \\
  & \quad z_i \leftarrow K \\
  & \text{return } A(c_1, c_2, c_3)
\end{align*}
\]

\[
\mathcal{G}_0^A \quad \begin{align*}
  & f, f^{-1} \leftarrow F_{\text{trapdoor}} \\
  & CS \leftarrow \emptyset \\
  & y_0, y_0 \leftarrow \emptyset \\
  & \text{For } i \in \{1, ..., q_1\} \text{ do} \\
  & \quad y_{2i} \leftarrow A(y_{2i-1}) \\
  & \quad (a_i, w_i, b_i) \leftarrow A(y_{2i}) \\
  & \quad x_{2i} \leftarrow \frac{y_{2i}}{x_{2i}} \\
  & \text{For } i \in \{1, ..., q_2\} \text{ do} \\
  & \quad x_i \leftarrow A(y_{i-1}) \\
  & \quad CS \leftarrow CS \cup x_i \\
  & \quad y_i \leftarrow \frac{y_{i}}{y_{i}} \\
  & \quad (x_{2i}, x_{i}) \leftarrow A(x_{i}) \\
  & \quad z_i \leftarrow K \\
  & m_0, m_1 \leftarrow A(f) \\
  & r \leftarrow R \\
  & \text{if } (r \in CS) \{ \text{return 1} \} \\
  & c_1 \leftarrow f(r) \\
  & c_2 \leftarrow M \\
  & c_3 \leftarrow K \\
  & A(c_1, c_2, c_3) \\
  & \bar{y}_0 \leftarrow \emptyset \\
  & \text{For } i \in \{1, ..., q_3\} \text{ do} \\
  & \quad \bar{x}_i \leftarrow A(y_i-1) \\
  & \quad \text{if } (\bar{x}_i = r) \{ \text{return 1} \} \\
  & \quad \bar{y}_i \leftarrow M \\
  & \quad (x_{2i}, x_i) \leftarrow A(x_i) \\
  & \quad z_i \leftarrow K \\
  & \text{return } A(c_1, c_2, c_3)
\end{align*}
\]
5 Conclusion

This seminar paper shows how to prove the security of symmetric primitives by using reductions. For that we construct example proofs of CPA and CCA in [BR93]. We also show that it is possible to do simple reduction based proofs using ProveIt. Additionally we describe which reductions are necessary for ProveIt and show how they are implemented.

References
