Chosen Ciphertext Security from Zero Knowledge Proofs

by

Camryn Steckel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

When designing encryption schemes, there are different levels of security that one can achieve. Of the two main security levels, cryptographers generally strive for the stronger notion of chosen ciphertext attack (CCA) security, which considers attackers who have the ability to obtain decryptions of their choice, over the weaker notion of chosen plaintext attack (CPA) security, which only considers attackers who have encryption abilities. However, it is much easier to find public key encryption schemes (PKEs) that satisfy CPA security. For this reason, a common technique for developing CCA-secure PKEs is to apply a CPA-to-CCA transformation to an existing CPA-secure PKE. The general idea behind such a transform is to somehow ensure that anyone who is capable of producing a valid ciphertext must already know the corresponding plaintext, which renders the additional powers that a CCA adversary has over a CPA adversary entirely useless. All existing transforms achieve this property by performing a re-encryption check in the decryption algorithm. However, this leaves the resulting PKE vulnerable to side-channel attacks, which can be used to carry out chosen ciphertext attacks on the underlying PKE.

In this thesis, we present a generic CPA-to-CCA transform that uses a zero-knowledge proof of knowledge in place of a re-encryption check. We prove security of our generic construction in the random oracle model, and we provide an instantiation of it using existing schemes. For the instantiation, we use ElGamal as our underlying PKE, and an application of Fischlin's transfomation to a variant of Schnorr's protocol for our zero-knowledge proof of knowledge, and prove that these protocols satisfy the required security definitions.

Acknowledgements

First and foremost, I would like to thank my supervisor, Douglas Stebila, for introducing me to the super fun area of applied cryptography, for his support and mentorship throughout my degree, and for putting up with (and even encouraging!) my goofy shenanigans. I would also like to thank my readers, Alfred Menezes and David Jao, for their thoughtful feedback on my thesis, and the C&O administrative staff for all of their help in the past two years. Finally, I would like to thank my family and friends for their encouragement and support of my academic, athletic, and ridiculous endeavors.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of encryption is based on a need for confidentiality: when a message is encrypted, it should not be possible for an adversary to determine the contents of the message. The ability of an adversary to do so is largely dependent on what computational powers an adversary has. For this reason, there are multiple formal definitions of cryptographic security, which differ by the strength of the adversary they are protecting against.

Two of the most common notions of security for any type of encryption schemes are *indistinguishability against chosen plaintext attacks* (IND-CPA) and *indistinguishability against chosen ciphertext attacks* (IND-CCA). In the context of public key encryption schemes (PKEs), IND-CPA security is the most basic notion of security. It guarantees security against adversaries who can encrypt messages of their choice, but since all the information required to do so is public, this is a power that all adversaries have. On the other hand, IND-CCA security is a bit more involved. It guarantees confidentiality against adversaries who can also decrypt ciphertexts of their choice. Since an adversary does not have access to the secret key, they cannot (in general) decrypt ciphertexts on their own, so they must be provided with a decryption oracle.

Given that it is possible for resourceful adversaries to have access to such oracles in the real world, it is important to have practical IND-CCA-secure PKEs. As IND-CPA-secure PKEs are much more common than IND-CCA-secure PKEs, it is desirable to have a generic transformation which converts an IND-CPA-secure PKEs into IND-CCA-secure PKEs. One approach to doing this is to modify an IND-CPA-secure PKE in such a way to ensure *plaintext awareness*; that is, to ensure that anyone who is able to produce a well-formed ciphertext must possess enough information to obtain the corresponding plaintext. This renders the presence of a decryption oracle useless, since any adversary who is able to query the oracle (without any outside help) on a valid ciphertext must already possess enough information to answer the query themself. So, the decryption oracle is not providing them with any information that they would not have as an IND-CPA-adversary, and the IND-CPA-security of the scheme therefore reduces to IND-CPA security.

Most such existing transformations achieve plaintext awareness by performing a re-encryption check before returning the plaintext in the decryption algorithm. Although they are still provably secure, re-encryption checks are known to be vulnerable to side-channel

attacks, which, similarly to Bleichenbacher's padding oracle attack[2], can be used to carry out chosen-ciphertext attacks on the underlying PKE. This is of increasing concern in recent years, as all of the finalist key encapsulation mechanism (KEM) candidates of NIST's post-quantum cryptography standardization competition rely on re-encryption checks to achieve IND-CCA security. A multitude of such attacks have already been found, and cryptographers are actively looking to reduce this risk [9, 16]. Although this work does not delve into post-quantum security, it explores a new option for a classical IND-CPA-to-IND-CCA transform that does not involve a re-encryption check. There is potential for the development of an analogous transformation that provides post-quantum security.

1.1 Related Work

There are a few such IND-CPA-to-IND-CCA transforms that can be found in the literature. One of the earliest and most widespread of these is the Fujisaki–Okamoto (FO) transform [8], which is built from an IND-CPA-secure public key encryption scheme $(\mathsf{KGen}^{pke}, \mathsf{Enc}^{pke}, \mathsf{Dec}^{pke})$ with message space $\mathcal{M}^{pke}$ and distribution Σ^{pke}, an IND-CPA-secure symmetric key encryption scheme $(\mathsf{Enc}^{sym}, \mathsf{Dec}^{sym})$ with keyspace $\mathcal{K}^{sym}$, and two hash functions, $\mathtt{G} : \{0,1\}^* \mapsto \mathcal{K}^{sym}$ and $\mathsf{H} : \{0,1\}^* \times \{0,1\}^* \mapsto \Sigma^{pke}$. The transform works as follows:

$\mathsf{KGen}(\lambda)$

1 : **return** $\mathsf{KGen}^{pke}(\lambda)$

$\mathsf{Enc}(\mathsf{pk}, m)$

1 : $\sigma \leftarrow\!\!\$\ \mathcal{M}^{pke}$
2 : $a \leftarrow \mathtt{G}(\sigma)$
3 : $c \leftarrow \mathsf{Enc}^{sym}(a, m)$
4 : $h \leftarrow \mathsf{H}(\sigma, c)$
5 : $e \leftarrow \mathsf{Enc}^{pke}(\mathsf{pk}, \sigma; h)$
6 : **return** (c, e)

$\mathsf{Dec}(\mathsf{sk}, (c, e))$

1 : $\hat{\sigma} \leftarrow \mathsf{Dec}^{pke}(\mathsf{sk}, e)$
2 : **if** $\hat{\sigma} \notin \mathcal{M}^{pke}$: **return** $\perp$
3 : $\hat{a} \leftarrow \mathtt{G}(\sigma)$
4 : $\hat{h} \leftarrow \mathsf{H}(\hat{\sigma}, c)$
5 : **if** $e \neq \mathsf{Enc}^{pke}(\mathsf{pk}, \hat{\sigma}; \hat{h})$:
6 : **return** $\perp$
7 : **return** $\mathsf{Dec}^{sym}(\hat{a}, c)$

Figure 1.1: Fujisaki-Okamoto Transform

In this case, h and $\hat{h}$ are the random coins used for encryption. Notice that the re-encryption step (line 5) of the decryption algorithm will fail if $\hat{h} \neq \mathsf{H}(\sigma, c)$. So, if the decryption algorithm doesn't fail, then this implies that the party which formed the ciphertext knows both σ and c, in which case they are able to obtain m themselves by running $\mathsf{Dec}^{sym}(\mathtt{G}(\sigma), c)$. Hence, a decryption oracle would not provide them with any useful information.

The re-encryption step is comparatively expensive, so in [12], Okamoto and Pointcheval present a new IND-CPA-to-IND-CCA transform called REACT (Rapid Enhanced-Security Asymmetric Cryptosystem Transform), which does not involve re-encryption. In a follow-up paper, [5], Coron, Handschuh, Joye, Paillier, Pointcheval, and Tymen build upon REACT

to form GEM (Generic Chosen-Ciphertext Secure Encryption Method), which also does not involve re-encryption. Similar to the FO transform, REACT and GEM are built from a weakly secure public key encryption scheme, $(\mathsf{KGen}^{pke}, \mathsf{Enc}^{pke}, \mathsf{Dec}^{pke})$, with message space $\mathcal{M}^{pke}$ and distribution Σ^{pke}. However, instead of requiring it to be IND-CPA-secure, they require a slightly stronger notion of scecurity: one-wayness against plaintext-checking attacks (OW-PCA). They also require an IND-CPA-secure symmetric key encryption scheme $(\mathsf{Enc}^{sym}, \mathsf{Dec}^{sym})$ with keyspace $\mathcal{K}^{sym}$, and two hash functions, $\mathtt{G} : \{0,1\}^* \mapsto \mathcal{K}^{sym}$ and $\mathsf{H} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^\lambda \times \{0,1\}^\lambda \mapsto \{0,1\}^{\ell(\lambda)}$. REACT is defined as follows:

$\mathsf{KGen}(\lambda)$

1 : **return** $\mathsf{KGen}^{pke}(\lambda)$

$\mathsf{Enc}(\mathsf{pk}, m)$

1 : $R \leftarrow\$ \mathcal{M}^{pke}$
2 : $r \leftarrow\$ \Sigma^{pke}$
3 : $c_1 \leftarrow \mathsf{Enc}^{pke}(\mathsf{pk}, R; r)$
4 : $k \leftarrow \mathtt{G}(R)$
5 : $c_2 \leftarrow \mathsf{Enc}^{sym}(k, m)$
6 : $c_3 \leftarrow \mathsf{H}(R, m, c_1, c_2)$
7 : **return** (c_1, c_2, c_3)

$\mathsf{Dec}(\mathsf{sk}, (c_1, c_2, c_3))$

1 : $\hat{R} \leftarrow \mathsf{Dec}^{pke}(\mathsf{sk}, c_1)$
2 : **if** $\hat{R} \notin \mathcal{M}^{pke}$: **return** $\perp$
3 : $\hat{k} \leftarrow G(\hat{R})$
4 : $m \leftarrow \mathsf{Dec}^{sym}(\hat{k}, c_2)$
5 : **if** $c_3 \neq \mathsf{H}(\hat{R}, \hat{m}, c_1, c_2)$:
6 : **return** $\perp$
7 : **return** m

Figure 1.2: Rapid Enhanced-Security Asymmetric Cryptosystem Transform

Instead of performing a re-encryption check, REACT uses the hash function H in line 4 of decryption to ensure that any party who is able to create a valid ciphertext must also have knowledge of the message that was encrypted. GEM uses the same underlying framework, but embeds this check value in c_1, thus eliminating c_3 and shortening the resulting ciphertext. However, unlike REACT, GEM does not allow for session keys to be computed in advance, since session keys in GEM are message-dependent. This decreases efficiency in some cases, specifically when using public key encryption schemes that are based on the discrete logarithm problem, such as ElGamal.

In [10], Hofheinz, Hövelmanns, and Kiltz describe various, similar transformations that can ultimately be used to convert a weakly secure PKE into a strongly secure KEM. To do this, they break up the FO transform into two parts: the T-transform and the U-transform. The T-transform takes in a PKE which satisfies either IND-CPA or OW-CPA (one-wayness against chosen plaintext attacks) security, upon which it performs a re-encryption check to convert it into one which satisfies either OW-PCA (one-wayness against plaintext checking attacks) or OW-PCVA (one-wayness against plaintext and validity checking attacks) security. The U-transform then converts the resulting scheme into an IND-CCA-secure key encapsulation mechanism (KEM). They provide a different versions of each of these transforms to cater to the different input and output security levels. The authors note that all known generic transformations to IND-CCA-security (or the KEM versions of these transformations), including the original FO transform and the REACT and GEM transforms, can be written as some combination of the various T and U transforms, which suggests that all

known generic transforms are simply variants of the FO transform. In particular, this means that all known IND-CPA-to-IND-CCA transforms require derandomization of the encryption scheme, and perform a re-encryption check. It should be noted that for the ElGamal cryptosystem, the "twinning" technique presented in [4] avoids the use of derandomization and re-encryption, but it cannot be generalized as it uses specific zero-knowledge proofs that cannot be applied to all PKEs. The authors of [10] remark that "it is an interesting open problem to come up with alternative transformations that get rid of derandomization or that dispense with re-encryption (while preserving efficiency)."

1.2 Contributions

In this thesis, we present a new approach to an IND-CPA-to-IND-CCA transformation for public key encryption schemes, which does not involve derandomization or re-encryption.

In Chapter 3, we describe a generic transfomation and provide a proof of its security in the random oracle model. Our transformation takes in two pre-existing protocols. The first is an IND-CPA-secure PKE for which there exists an algorithm to perform decryption of a ciphertext via the random coins used to produce said ciphertext. The second is a non-interactive straight-line extractable zero knowledge proof of knowledge. Suppose that `PKE` is such a public key encryption scheme and that Π is such a proof system. Then our combined PKE, `PKEZK`, encrypts a message m by first encrypting it under `PKE` to obtain a ciphertext c, and then uses Π to produce a zero knowledge proof of knowledge π of the random coins used to form c, which is appended onto c to form the `PKEZK` ciphertext. Decryption is performed by first checking that (c, π) is a valid statement/proof pair under Π, and if so, then returning the decryption of c under `PKE`. At a high level, since a decryption oracle will only decrypt ciphertexts for which the adversary knows the random coins that were used to produce them, which the adversary is able to use to decrypt the ciphertext, then access to a decryption oracle does not provide the adversary with any information that they could not obtain without one. So, security of the combined scheme reduces to security of `PKE` and of Π. We show this formally using a game-hopping proof.

In Chapter 4, we provide an instantiation of our construction. We use ElGamal [6] as our `PKE`, and apply Fischlin's transformation, which converts a Sigma protocol into a straight-line extractable non-interactive zero knowledge proof of knowledge, [7] to a modified version of Schnorr's protocol [14] to use as our proof system Π. We prove that each of the components satisfies the required security properties.

Chapter 2

Preliminaries

In this chapter, we introduce the random oracle model, and present the definitions required for our construction. All security definitions are presented as adversarial games.

2.1 Random Oracle Model

The random oracle model was first introduced by Bellare and Rogaway in [1]. It assumes the existence of a truly random function, implemented as an oracle that all parties have access to, which can be used to model hash functions in security experiments. The intent of a cryptographic proof in the random oracle model is to show that a given protocol does not have any design flaws; it does not guarantee the security of an instantiation of the protocol. Truly random functions do not have efficient representations in the real world, so in practice, the security of an instantiation is dependent on the hash function used.

In a game-based setting, the adversary lives in a world provided for them by the challenger. In particular, this means that the challenger provides the random oracle to the adversary, and therefore the challenger gets to see the queries that the adversary makes, as well as program the responses that the adversary receives back. These properties are known as *extractability* and *programmability*, respectively. The only restrictions are that, if the same value is queried to the oracle multiple times, the oracle must respond the same way to each query, and the responses provided by the challenger must appear uniformly random.

In this paper, we model the random oracle $\mathcal{O}_{\mathsf{H}}$ as an oracle which performs lazy sampling, as described below for a list `HList` (controlled by the challenger) of oracle queries.

$\mathcal{O}_{\mathsf{H}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow\!\!\$\ \{0,1\}^{\lambda}$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

Figure 2.1: Random Oracle with Lazy Sampling

The challenger's access to `HList` models extractability, while programmability allows the challenger to program $\mathcal{O}_{\mathsf{H}}$ in whichever way they like (as long as their responses appear uniformly random).

2.2 Public Key Encryption Schemes

Our first primitive is a public key encryption scheme. Definitions 1, 3, 4, and 5 are all standard definitions, which have been taken from [11]. Definitions 6 and 7 are novel. We will start by defining a public key encryption scheme.

Definition 1. *A* public key encryption scheme `PKE` *consists of a public key space* $\mathcal{K}$*, a message space* $\mathcal{M}$*, a ciphertext space* $\mathcal{C}$*, and a tuple of algorithms* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$*, where*

- $\mathsf{KGen}(\lambda) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$ *is a probabilistic key generation algorithm, which takes as input a security parameter* λ *and outputs a tuple* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$ *consisting of a public key/secret key pair.*
- $\mathsf{Enc}(\mathsf{pk}, m) \xrightarrow{\$} c$ *is a probabilistic polynomial time encryption algorithm that takes as input a public key* pk *and a message* $m \in \mathcal{M}$*, and outputs a ciphertext* $c \in \mathcal{C}$*.*
- $\mathsf{Dec}(\mathsf{sk}, m) \rightarrow m$ *is a deterministic polynomial time decryption algorithm that takes as input a secret key* sk *and a ciphertext* $c \in \mathcal{C}$*, and outputs either a message* $m \in \mathcal{M}$ *or a distinguished error symbol* $\perp$*.*

The random coins used in Enc are generated during the execution of the algorithm. We can make Enc deterministic by sending in the random coins as an additional input, rather than having Enc generate them internally. This gives the following equivalent definition of a public key encryption scheme:

Definition 2. *A* public key encryption scheme `PKE` *consists of a key space* $\mathcal{K}$*, a probability distribution* Σ*, a message space* $\mathcal{M}$*, a ciphertext space* $\mathcal{C}$*, and a tuple of algorithms* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$*, where*

- $\mathsf{KGen}(\lambda) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$ *is a probabilistic key generation algorithm, which takes as input a security parameter* λ *and outputs a tuple* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$ *consisting of a public key/secret key pair.*
- $\mathsf{Enc}(\mathsf{pk}, m; r) \to c$ *is a deterministic polynomial time encryption algorithm that takes as input a public key* pk*, a message* $m \in \mathcal{M}$ *and random coins* r *sampled from* Σ*, and outputs a ciphertext* $c \in \mathcal{C}$*, and outputs a ciphertext* $c \in \mathcal{C}$*.*
- $\mathsf{Dec}(\mathsf{sk}, m; r) \to m$ *is a deterministic polynomial time decryption algorithm that takes as input a secret key* sk*, a ciphertext* $c \in \mathcal{C}$ *and random coins* r *sampled from* Σ*, and outputs either a message* $m \in \mathcal{M}$ *or a distinguished error symbol* $\perp$*.*

In order for a public key encryption scheme to be useful, certain properties should be satisfied. Arguably, the most important of these is correctness; i.e., that decrypting the encryption of a message results in the original message. Formally, correctness is defined as follows:

Definition 3. *We say that a public key encryption scheme is* ϵ-correct *if for all* $m \in \mathcal{M}$ *and* $r \xleftarrow{\$} \Sigma$*,*

$$\Pr\left[m' \neq m : \begin{smallmatrix}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KGen}(\lambda);\\ c \xleftarrow{} \mathsf{Enc}(\mathsf{pk}, m; r);\\ m' \xleftarrow{} \mathsf{Dec}(\mathsf{sk}, c)\end{smallmatrix}\right] < \epsilon.$$

We also generally require that public key encryption schemes satisfy certain security properties. The weakest of these is what we call *indistinguishability against chosen plaintext attacks*, which guarantees that an adversary is unable to determine which of two chosen messages a given ciphertext is the encryption of, while having access only to the public key. Formally, we define this as follows:

Definition 4. *We say that a public key encryption scheme* $\mathtt{PKE}$ *is* ϵ-IND-CPA-secure *if, for all probabilistic polynomial time adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\mathtt{Exp}^{\text{IND-CPA}}_{\mathtt{PKE}}(\mathcal{A}) \implies 1\right] - \frac{1}{2}\right| < \epsilon,$$

where $\mathtt{Exp}^{\text{IND-CPA}}_{\mathtt{PKE}}$ *is defined as*

$\mathtt{Exp}^{\text{IND-CPA}}_{\mathtt{PKE}}(\mathcal{A})$

1: $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathtt{PKE}.\mathsf{KGen}()$
2: $(m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{pk})$
3: $b \xleftarrow{\$} \{0, 1\}$
4: $ctxt \xleftarrow{\$} \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m_b)$
5: $b' \xleftarrow{\$} \mathcal{A}(\mathsf{pk}, ctxt, \mathsf{st})$
6: **return** $[\![b = b']\!]$

A stronger notion of security is *indistinguishability against chosen ciphertext attacks*, which guarantees that an adversary is unable to determine which of two chosen messages a given ciphertext is the encryption of, while having access to both the public key and a decryption oracle. Formally, this is defined as:

Definition 5. *We say that a public key encryption scheme* PKE *is* ϵ-IND-CCA*-secure if, for all probabilistic polynomial time adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\mathrm{Exp}^{\text{IND-CCA}}_{\mathtt{PKE}}(\mathcal{A}) \implies 1\right] - \frac{1}{2}\right| < \epsilon,$$

where $\mathrm{Exp}^{\text{IND-CCA}}_{\mathtt{PKE}}$ *is defined as*

$\mathrm{Exp}^{\text{IND-CCA}}_{\mathtt{PKE}}(\mathcal{A})$

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathtt{PKE}.\mathsf{KGen}()$
2 : $(m_0, m_1, \mathsf{st}) \leftarrow\$ \mathcal{A}^{\mathtt{PKE}.\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{pk})$
3 : $b \leftarrow\$ \{0,1\}$
4 : $ctxt \leftarrow\$ \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m_b)$
5 : $b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{Dec}}(\mathsf{sk},\cdot)}(\mathsf{pk}, ctxt, \mathsf{st})$
6 : **return** $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{Dec}}(c)$

1 : **if** $c = ctxt$: **return** $\perp$
2 : **return** $\mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c)$

To prevent the adversary from trivially winning the experiment, we do not allow them to query the decryption oracle with the challenge ciphertext. This is achieved by programming the oracle to fail in this case.

The basis of our construction will be an IND-CPA-secure public key encryption scheme. However, in order for it to work, we need some additional constraints. Specifically, we require that the random coins used during encryption can be used in place of the secret key for decryption. We refer to such a scheme as a *decryptable with randomness*, or, *DWR* public key encryption scheme, and define it below.

Definition 6. *Let* PKE *be a public key encryption scheme. We say that* PKE *is* decryptable with randomness (DWR) *if there is an algorithm* DecRand*, where*

- $\mathtt{DecRand}(\mathsf{pk}, c, r) \to m$ *is a deterministic polynomial time decryption algorithm that takes as input the random coins* r *sampled from* Σ *as well as a ciphertext* $c \in \mathcal{C}$*, and outputs a message* $m \in \mathcal{M}$ *or a distinguished error symbol* $\perp$*.*

Similarly to the regular decryption algorithm, we also require that DecRand satisfies some notion of correctness. In our case, we need that it produces the same output as Dec whenever Dec produces a correct output. Formally, this can be stated as:

Definition 7. *Let* PKE *be a DWR public key encryption scheme with ciphertext space* $\mathcal{C}$ *and probability distribution* Σ*. Additionally, define the relation* $\mathcal{R}_{\mathsf{pk}}$ *as*

$$\mathcal{R}_{\mathsf{pk}} = \{(r, c) \in \mathtt{PKE}.\Sigma \times \mathtt{PKE}.\mathcal{C} : c = \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c); r)\}.$$

We say that PKE *is* ϵ-DWR-correct *if the underlying* PKE *scheme is correct, and for all probabilistic polynomial time adversaries* $\mathcal{A}$*,*

$$\Pr\left[\mathrm{Exp}^{DWR}_{PKE}(\mathcal{A}) \implies 1\right] < \epsilon,$$

where Exp^{DWR}_{PKE} *is defined as*

$\mathsf{Exp}^{DWR}_{\mathsf{PKE}}(\mathcal{A})$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}()$

$(c, r) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}, \mathsf{sk})$

return $[\![\mathtt{DecRand}(\mathsf{pk}, r, c) \neq \mathsf{Dec}(\mathsf{sk}, c) \wedge (r, c) \in \mathcal{R}_{\mathsf{pk}}]\!]$

IND-CPA- and IND-CCA-security of a DWR PKE are defined identically to that of a PKE.

2.3 Sigma Protocols

Before introducing our second primitive, non-interactive zero knowledge proofs of knowledge, we will introduce Sigma protocols, as many of the definitions used for non-interactive zero knowledge proofs of knowledge build off those for Sigma protocols. Definitions 8, 9, and 10 are taken from [3], while definitions 11, 12, 13, and 14 are taken from [7]. Definition 15 is adapted from the collision resistance definition for hash functions.

For the remainder of this section, let $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ be a relation where $\mathcal{W}$, $\mathcal{X}$, and $\mathcal{R}$ are efficiently recognizable finite sets. Let $\mathcal{L}$ be the language defined by $\mathcal{R}$; that is, let $\mathcal{L}$ be the set of $x \in \mathcal{X}$ such that there exists $w \in \mathcal{W}$ with $(w, x) \in \mathcal{R}$.

Definition 8. *A* Sigma protocol *for relation $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ is a collection of probabilistic polynomial time algorithms $(\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2))$. $\mathcal{P}$ is an interactive protocol called the prover, and $\mathcal{V}$ is an interactive protocol called the verifier. The protocol flow is as follows:*

- *$\mathcal{P}_1$ takes as input a witness/statement pair $(w, x) \in \mathcal{R}$, and outputs a commitment* `com`*.*
- *$\mathcal{V}_1$ takes as input the statement x and the commitment* `com` *from above, and outputs a challenge* `ch`*.*
- *$\mathcal{P}_2$ retains the state of $\mathcal{P}_1$, takes as input the challenge* `ch` *from above, and outputs a response* `rsp`*.*
- *$\mathcal{V}_2$ retains the state of $\mathcal{V}_1$, takes as input the response* `rsp` *from above, and outputs either* true *or* false*, as a deterministic function of x,* `com`*,* `ch`*, and* `rsp`*. In the case where $\mathcal{V}_2$ outputs* true*, we say that* $(\mathtt{com}, \mathtt{ch}, \mathtt{rsp})$ *is an accepting transcript for x.*

We require that for all $(w, x) \in \mathcal{R}$, when $\mathcal{P}(w, x)$ and $\mathcal{V}(x)$ interact with each other, the output of $\mathcal{V}_2$ is always true*.*

Sigma protocols are generally used when the prover wants to convince the verifier that they know the value of the witness corresponding to a given statement, without disclosing the witness to the verifier. An accepting transcript for a statement functions as a *proof of knowledge* of the witness corresponding to that statement. When used in this way, there are three main things that we want to ensure:

- A valid proof should be able to be produced for any statement in the language, for any given challenge.
- Malicious provers who do not know the witness for a given statement should not be able to produce an accepting transcript for that statement.
- Anyone who sees the transcript should not be able to deduce any information about the witness that they could not have deduced without seeing the transcript.

The first property listed is called *completeness*, and it is implicit in definition 8 by the requirement that any honest interaction between a prover and verifier must produce a valid transcript.

The second point is less straightforward. Proving that someone "knows" a witness is rather ambiguous, so instead, we show that anyone capable of producing accepting transcripts could easily compute the witness from these transcripts. We achieve this through a property called *special soundness*, defined below.

Definition 9. *Let* $\Pi = (\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2))$ *be a Sigma protocol for relation* $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$*. We say that* Π *provides* ϵ-special soundness *if there is an efficient, deterministic algorithm* $\Pi.\mathtt{Ext}$*, called a witness extractor, such that for all provers* $\mathcal{P}$*,*

$$\Pr\left[\mathrm{Exp}^{\mathtt{Ext}}_{\Pi}(\mathcal{P}) \implies 1\right] < \epsilon,$$

where

$\mathrm{Exp}^{\mathtt{Ext}}_{\Pi}(\mathcal{P}^{\mathcal{O}_H})$

$\mathtt{HList} \leftarrow \emptyset$
$(\mathtt{com}, \mathtt{st}_{P_1}) \leftarrow\!\$\ \mathcal{P}_1(w, x)$
$\mathtt{ch} \leftarrow\!\$\ \mathcal{V}_1(x, \mathtt{com})$
$\mathtt{rsp} \leftarrow \mathcal{P}_2(\mathtt{ch}, \mathtt{st}_{P_1})$
$\mathtt{ch}' \leftarrow\!\$\ \mathcal{V}_1(x, \mathtt{com})$
$\mathtt{rsp}' \leftarrow \mathcal{P}_2(\mathtt{ch}', \mathtt{st}_{P_2})$
$w \leftarrow \Pi.\mathtt{Ext}((x, \mathtt{com}, \mathtt{ch}, \mathtt{rsp}), (x, \mathtt{com}, \mathtt{ch}', \mathtt{rsp}'))$
return $[\![(w, x) \in \mathcal{R} \wedge \mathtt{ch} \neq \mathtt{ch}'$
$\quad \wedge \mathcal{V}_2(x, \mathtt{com}, \mathtt{ch}, \mathtt{rsp}) = \mathtt{true}$
$\quad \wedge \mathcal{V}_2(x, \mathtt{com}, \mathtt{ch}', \mathtt{rsp}') = \mathtt{true}]\!]$

$\mathcal{O}_H(x)$

if $\exists y$ *s.t.* $(x, y) \in \mathtt{HList}$:
$\quad$ **return** y
$y \leftarrow\!\$\ \{0,1\}^{\lambda}$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

It is worth noting that special soundness assumes that the prover is able to create multiple accepting transcripts for the same commitment, but different challenges. In practice, the same commitment and challenge should not be used (and will not be used, if the protocol is followed honestly, as these values are chosen uniformly at random), so this does not leak information about the witness to the public.

The third point is captured through a property called *special honest verifier zero knowledge*, defined below. The idea is that if an accepting transcript can be generated without knowledge of the witness, then it cannot possibly leak information about the witness.

Definition 10. *Let* Π *be a Sigma protocol. We say that* Π *is* special honest verifier zero knowledge *(or* SHVZK*) if there exists a probabilstic polynomial time algorithm* $\Pi.\texttt{Sim}$*, such that for all possible inputs* $(x, \texttt{ch})$ *where* x *is a statement and* $\texttt{ch}$ *is a challenge, it outputs an accepting transcript* $(\texttt{com}, \texttt{ch}, \texttt{rsp})$ *for* x *that has the same distribution as an honest conversation between the prover and the verifier.*

In most cases, $\Pi.\texttt{Sim}$ will write the transcript out of order (for instance, it might pick the commitment last, based off the values for $\texttt{ch}$ and $\texttt{rsp}$).

In Chapter 4, we require a Sigma protocol that also satisfies the following additional properties:

Definition 11. *Let* Π *be a Sigma protocol. We say that* Π *satisfies* commitment entropy *if, for security parameter* λ *and any* $(w, x) \in \mathcal{R}$*, the min-entropy of an honestly generated commitment is superlogarithmic in* λ*.*

Definition 12. *Let* Π *be a Sigma protocol. We say that* Π *is* public coin *if, for security parameter* λ*, any* $(w, x) \in \mathcal{R}$*, and any honestly generated commitment* $\texttt{com}$*, an honestly generated challenge* $\texttt{ch}$ *is uniform on the challenge space.*

Definition 13. *Let* Π *be a Sigma protocol. We say that* Π *has* quasi-unique responses *if for any probabilistic polynomial time algorithm* $\mathcal{A}$*, security parameter* λ*, and* $(x, \texttt{com}, \texttt{ch}, \texttt{rsp}, \texttt{rsp}') \leftarrow\!\!\$\ \mathcal{A}(k)$*, we have that*

$$\Pr[\mathcal{V}_2(x, \texttt{com}, \texttt{ch}, \texttt{rsp}) = \mathcal{V}_2(x, \texttt{com}, \texttt{ch}, \texttt{rsp}') = \texttt{true} \wedge \texttt{rsp} \neq \texttt{rsp}']$$

is negligible.

A Sigma protocol which satisfies all of these properties is called a *Fiat–Shamir proof of knowledge*, which we formally define below. Essentially, this is a Sigma protocol to which the Fiat–Shamir transform can be applied to create a non-interactive zero knowledge proof of knowledge.

Definition 14. *Let* λ *be a security parameter. A* Fiat–Shamir proof of knowledge *with* $O(\log(\lambda))$*-bit challenges for a relation* $\mathcal{R}$ *is a Sigma protocol consisting of a pair* $(\mathcal{P}, \mathcal{V})$ *of probabilistic polynomial time algorithms* $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2)$ *and* $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2)$ *which satisfies completeness, commitment entropy, public coin, quasi-unique responses, special soundness, and honest verifier zero knowledge.*

Finally, we will be making use of a property we call *collision resistance*, in order to show that the non-interactive zero-knowledge proof that we use in our instantiation satisfies an analogous property. We define collision resistance similarly to how it is defined for hash functions. The idea is that it is difficult to find two distinct statements which verify with the same proof.

Definition 15. *Let* $\Pi = (\mathcal{P}, \mathcal{V})$ *be a Sigma protocol for relation* $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ *with simulator* $\texttt{Sim}$*. We say that* Π *satisfies* ϵ-collision resistance *if, for all probabilistic polynomial time adversaries* $\mathcal{A}$*, we have that*

$$\Pr\left[\texttt{Exp}_{\Pi}^{CR}\mathcal{A} \implies 1\right] < \epsilon,$$

where $\texttt{Exp}_{\Pi}^{CR}$ *is defined as*

$\underline{\mathtt{Exp}^{CR}_{\Pi}(\mathcal{A})}$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $(x, x', (\mathtt{com}, \mathtt{ch}, \mathtt{rsp})) \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}()$
3 : **return** $[\![\Pi.\mathsf{Vf}(x, \mathtt{com}, \mathtt{ch}, \mathtt{rsp})$
4 : $\quad \wedge\ \Pi.\mathsf{Vf}(x', \mathtt{com}, \mathtt{ch}, \mathtt{rsp})$
5 : $\quad \wedge\ x \neq x']\!]$

$\underline{\mathcal{O}_H(x)}$

1 : **if** $\exists y$ *s.t.* $(x, y) \in \mathtt{HList}$:
2 : $\quad$ **return** y
3 : $y \leftarrow\$\ \{0,1\}^{\lambda}$
4 : $\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
5 : **return** y

2.4 Zero Knowledge Proofs

Our second primitive is a zero knowledge proof of knowledge. Definitions 16, 17, 18, 19, and 21 are taken from [3], definition 20 was taken from [7], definition 23 was modified from the one in [13], and definition 22 was adapted from the collision resistance definition for hash function given in [11].

To begin with, we introduce the idea of non-interactive proof systems.

Definition 16. *Let $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ be a relation, where $\mathcal{W}$, $\mathcal{X}$, and $\mathcal{R}$ are efficiently recognizable finite sets. A non-interactive proof system for $\mathcal{R}$ is a pair of algorithms $(\mathtt{PfGen}, \mathsf{Vf})$, where:*

- $\mathtt{PfGen}(w, x) \$\rightarrow \pi$ *is an efficient probabilistic algorithm that takes in $(w, x) \in \mathcal{R}$ and outputs a proof π in some proof space $\mathcal{PS}$.*
- $\mathsf{Vf}(x, \pi) \rightarrow \mathsf{true}/\mathsf{false}$ *is an efficient deterministic algorithm that takes in a statement $x \in \mathcal{X}$ and proof $\pi \in \mathcal{PS}$ as input and outputs either* true *or* false*. If $\mathsf{Vf}(x, \pi) = \mathsf{true}$, we say that π is a valid proof for x.*

We require that for all $(w, x) \in \mathcal{R}$, the output of $\mathtt{PfGen}(w, x)$ is a valid proof for x.

In our construction, we require that out proof system satisfies certain properties. To start with, we need that proofs exist for every statement in our language. We call this property *completeness*, and define it below.

Definition 17. *We say that a proof system Π for language $\mathcal{L}$ is* complete *if, for every $x \in \mathcal{L}$, there exists a proof π such that $\mathsf{Vf}(x, \pi) = \mathsf{true}$.*

Next, we require that someone who sees a valid proof of any $x \in \mathcal{L}$ is not able to determine any more about the witness than they would be able to had they not seen a valid proof. Similarly to the previous section, this is defined using the concept of a simulator. In this case, the simulator responds to hash queries in place of the random oracle, and responds to proof queries in place of the prover, without knowing the witness.

Definition 18. *Suppose that Π is a non-interactive proof system which makes use of a hash function $\mathsf{H} : \mathcal{U} \rightarrow \mathcal{C}$, and that we wish to model H as a random oracle. A* simulator *for Π is an interactive machine $\mathtt{Sim}$ that responds to a series of queries, where each query is one of the following two types:*

- $(\texttt{proofQuery}, x)$ *where* $x \in \mathcal{X}$*, to which* Sim *responds with* $\pi \in \mathcal{PS}$*.*
- $(\texttt{hashQuery}, u)$ *where* $u \in \mathcal{U}$*, to which* Sim *responds with* $c \in \mathcal{C}$*.*

If there exists such a simulator for which no adversary is able to distinguish an execution of the protocol where they are interacting with the simulator from a real execution of the protocol, then we can be confident that the proof system does not leak information about the witness: since the simulator does not know the witness, it cannot possibly leak information about it, so if the adversary is unable to tell a simulated execution apart from a real execution, then the real execution must not leak any witness information either. This property is known as *zero knowledge*, and is defined below.

Definition 19. *Let* $\Pi = (\texttt{PfGen}, \mathsf{Vf})$ *be a non-interactive proof system for* $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ *with proof space* $\mathcal{PS}$*, which makes use of a hash function* $\mathsf{H} : \mathcal{U} \to \mathcal{C}$ *modeled as a random oracle. Let* Sim *be a simulator for* Π*, as above. We say that* Π *provides* ϵ*-non-interactive zero-knowledge if*

$$\left| \Pr\left[\texttt{Exp}_{\Pi}^{niZK}(\mathcal{A}) \implies 1\right] - \frac{1}{2} \right| < \epsilon,$$

where $\texttt{Exp}_{\Pi}^{niZK}$ *is defined as*

$\texttt{Exp}_{\Pi}^{niZK}(\mathcal{A})$

$\texttt{HList} \leftarrow \emptyset$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_b}()$
return $[\![b = b']\!]$

$\mathcal{O}_0(qType,\ query)$

if $qType = \texttt{proofQuery}$:
  return $\texttt{PfGen}(query)$
if $qType = \texttt{hashQuery}$:
  if $\exists y$ *s.t.* $(x, y) \in \texttt{HList}$:
    return y
  $y \leftarrow\$ \{0,1\}^{\lambda}$
  $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
  return y

$\mathcal{O}_1(qType,\ query)$

if $qType = \texttt{proofQuery}$:
  Parse $(w, x) \leftarrow query$
  return $\texttt{Sim}(\texttt{proofQuery}, x, \texttt{HList})$
if $qType = \texttt{hashQuery}$:
  if $\exists y$ *s.t.* $(x, y) \in \texttt{HList}$:
    return y
  $y \leftarrow\$ \texttt{Sim}(\texttt{hashQuery}, query, \texttt{HList})$
  $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
  return y

In order for a proof to be a proof of knowledge, there needs to be a way of ensuring that only parties who possess knowledge of the witness are able to produce a valid proof. Again similarly to the previous section, we specify the existence of an algorithm called an extractor, whose input includes a statement and a valid proof of said statement (among other things), and whose output is the witness to that statement. The reasoning here is that any party who could come up with a valid proof of a statement could simply run the extraction algorithm to recover the witness, thus ensuring possession of knowledge.

In our case, we require that for any statement x, the extractor only takes a single proof π of x as input. To preserve the zero-knowledge property, it must also take another input: for us, this is the list of queries, $\texttt{HList}$, that the prover made to the random oracle when creating π. We call this property *straight-line extractability*, and define it below.

Definition 20. *We say that a proof system Π is ϵ-straight-line extractable if there exists a probabilistic polynomial time algorithm $\Pi.\texttt{SLE}$ such that for all probabilistic polynomial time algorithms $\mathcal{A}$,*

$$\Pr\left[\texttt{Exp}^{\texttt{SLE}}_{\Pi}(\mathcal{A}^{\mathcal{O}_H}) \implies 1\right] < \epsilon,$$

where $\texttt{Exp}^{\texttt{SLE}}_{\Pi}$ is defined as follows:

$\texttt{Exp}^{\texttt{SLE}}_{\Pi}(\mathcal{A}^{\mathcal{O}_H})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $(x, \pi) \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}()$
3 : **if** $x \notin \mathcal{L}$: **return** $\bot$
4 : $w \leftarrow \Pi.\texttt{SLE}(x, \pi, \texttt{HList})$
5 : **return** $[\![(w, x) \notin \mathcal{R} \wedge \Pi.\mathsf{Vf}(x, \pi) = \mathsf{true}]\!]$

$\mathcal{O}_H(x)$

1 : **if** $\exists y$ *s.t.* $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow\$\ \{0,1\}^{\lambda}$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

Notice that, unlike the extractor defined for Sigma protocols, the straight-line extractor is only required to output a witness for statements in $\mathcal{L}$. As such, its existence does not guarantee that any statement with a valid proof must be in $\mathcal{L}$. For this, we need some notion of soundness.

Definition 21. *Let $\Pi = (\texttt{PfGen}, \mathsf{Vf})$ be a non-interactive proof system for language $\mathcal{L}$. We say Π satisfies ϵ-non-interactive soundness if, for all probabilistic polynomial time adversaries $\mathcal{A}$,*

$$\Pr\left[\texttt{Exp}^{niSound}_{\Pi}\mathcal{A} \implies 1\right] < \epsilon,$$

where $\texttt{Exp}^{niSound}_{\Pi}$ is defined as

$\texttt{Exp}^{niSound}_{\Pi}(\mathcal{A})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $(x, \pi) \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}()$
3 : **return** $[\![\mathsf{Vf}(x, \pi) = \mathsf{true} \wedge x \notin \mathcal{L}]\!]$

$\mathcal{O}_H(x)$

1 : **if** $\exists y$ *s.t.* $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow\$\ \{0,1\}^{\lambda}$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

Finally, for our construction, we require two additional, non-standard properties. First, we need that it is *collision resistant*, which we define analogously to how we defined it for Sigma protocols.

Definition 22. *Let* $\Pi = (\mathtt{PfGen}, \mathsf{Vf})$ *be a non-interactive proof system for relation* $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ *with simulator* $\mathtt{Sim}$*. We say that* Π *satisfies* ϵ-collision resistance *if, for all probabilistic polynomial time adversaries* $\mathcal{A}$*, we have that*

$$\Pr\left[\mathtt{Exp}^{CR}_{\Pi}(\mathcal{A}) \implies 1\right] < \epsilon,$$

where $\mathtt{Exp}^{CR}_{\Pi}$ *is defined as*

$\mathtt{Exp}^{CR}_{\Pi}(\mathcal{A})$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $(x, x', \pi) \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}()$
3 : **return** $[\![\Pi.\mathsf{Vf}(x, \pi) \wedge \Pi.\mathsf{Vf}(x', \pi) \wedge x \neq x']\!]$

$\mathcal{O}_H(x)$

1 : **if** $\exists y$ *s.t.* $(x, y) \in \mathtt{HList}$:
2 : **return** y
3 : $y \leftarrow\$\ \{0,1\}^{\lambda}$
4 : $\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
5 : **return** y

The second additional property we require is a variation of what is called *non-malleability*, which states that, given a valid statement/proof pair (x, π), it is difficult to find a different valid proof π' of a related statement x'. In our case, we need that the adversary has some choice in the initial statement x that they are given: in particular, x is derived by applying a function to an input of the adversary's choice. We also only require that it is hard to find a different, valid proof π' of the statement x; in other words, the relation between x and x' is the identity. Formally, this is captured in the following definition:

Definition 23. *Let* Π *be a non-interactive zero-knowledge proof system for a language* $\mathcal{L}$*, and let* $\mathcal{F}$ *consist of probabilistic algorithms* f *and* $\mathtt{Gen}$*, where* $\mathtt{Gen}$ *generates auxillary inputs for* f*. We say that* Π *is* ϵ-$\mathcal{F}$-non-malleable *if, for all outputs* $\mathtt{aux}$ *of* $\mathcal{F}.\mathtt{Gen}$ *and all probabilistic polynomial time adversaries* $\mathcal{A}$*, there exists a probabilistic polynomial time machine* $\mathcal{M}$ *such that*

$$\left|\Pr\left[\mathtt{Exp1}^{NM}_{\Pi,\mathcal{F}}(\mathcal{M}^{\mathcal{A}}) \implies 1\right] - \Pr\left[\mathtt{Exp2}^{NM}_{\Pi,\mathcal{F}}(\mathcal{A}) \implies 1\right]\right| < \epsilon,$$

where $\mathtt{Exp1}^{NM}_{\Pi,\mathcal{F}}$ *and* $\mathtt{Exp2}^{NM}_{\Pi,\mathcal{F}}$ *are defined as*

$\mathtt{Exp1}^{NM}_{\Pi,\mathcal{F}}(\mathcal{M}^{\mathcal{A}})$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $\mathtt{aux} \leftarrow\$\ \mathcal{F}.\mathtt{Gen}()$
3 : $y \leftarrow\$\ \mathcal{M}^{\mathcal{A},\mathcal{O}_H}(\mathtt{aux})$
4 : $x^* \leftarrow\$\ \mathcal{F}.f(\mathtt{aux}, y)$
5 : $\pi \leftarrow\$\ \mathcal{M}^{\mathcal{A},\mathcal{O}_H}(\mathtt{aux}, x^*)$
6 : **return** $[\![\Pi.\mathsf{Vf}(\pi, x^*)]\!]$

$\mathtt{Exp2}^{NM}_{\Pi,\mathcal{F}}(\mathcal{A})$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $\mathtt{aux} \leftarrow\$\ \mathcal{F}.\mathtt{Gen}$
3 : $y \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}(\mathtt{aux})$
4 : $x^* \leftarrow\$\ f(y)$
5 : $\pi^* \leftarrow\$\ \Pi.\mathtt{Sim}(\mathtt{proofQuery}, x^*, \mathtt{HList})$
6 : $\pi \leftarrow\$\ \mathcal{A}^{\mathcal{O}_H}(\mathtt{aux}, x^*, \pi^*)$
7 : **return** $[\![\Pi.\mathsf{Vf}(\pi, x^*) \wedge \pi \neq \pi^*]\!]$

and $\mathcal{O}_H$ *is defined as*

$\mathcal{O}_H(x)$

1 : **if** $\exists y$ *s.t.* $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\texttt{Sim}(\texttt{hashQuery}, x, \texttt{HList})$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup (x, y)$
5 : **return** y

The presence of the machine $\mathcal{M}$ is meant to capture the idea that the performance of any adversary in $\texttt{Exp2}^{\text{NM}}_{\Pi,\mathcal{F}}$ can be matched by *some* adversary $\mathcal{A}$ for $\texttt{Exp1}^{\text{NM}}_{\Pi,\mathcal{F}}$. We cannot simply directly compare a given adversary's performance in both experiments, since the difference in inputs would allow for trivial wins. However, $\mathcal{M}$ should still have access to the algorithms used by the adversary $\mathcal{A}$.

Now that we have seen the main building blocks, we can begin describing our construction.

Chapter 3

Generic Construction

Our generic construction creates an IND-CCA-secure PKE, PKEZK, from an IND-CPA-secure PKE, PKE, and a non-interactive zero knowledge proof of knowledge, Π. To encrypt a message, we first encrypt it using PKE to get a ciphertext c, and then compute a proof of knowledge π of the random coins used during encryption, using Π. Our resulting ciphertext is then (π, c). For decryption, we decrypt c if and only if π verifies. We use a game-hopping proof to reduce the security of PKEZK to the security of PKE and Π.

We require a few properties of PKE and Π in order for our construction to be secure. The eventual goal of our proof is to show that an IND-CPA adversary for PKE can implement the PKEZK decryption oracle, without knowing the secret key. In order for decryption to be possible without the secret key, we require that PKE is DWR. This requires that, for each query, the IND-CPA adversary for PKE is able to obtain the random coins used during encryption. For this, we require that Π is straight-line extractable, since the adversary will only have one proof to work with per query. In order to use our extractor, we require that all queried ciphertexts are in the language of valid ciphertexts, which in this case, requires Π to satisfy non-interactive soundness. To allow the adversary to respond to proof queries, we need them to have access to a simulator algorithm, and hence we require that Π satisfies non-interactive zero-knowledge. Finally, to prevent trivial wins, we require that Π is collision resistant and non-malleable.

3.1 Construction

Let PKE be an $\epsilon_{\mathtt{PKE}}-$IND-CPA-secure public key encryption scheme with message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and random distribution Σ that is decryptable with randomness. For each public key pk for PKE, we require that the language

$$\mathcal{L}_{\mathsf{pk}} = \{c \in \mathtt{PKE}.\mathcal{C} : \exists r \in \mathtt{PKE}.\Sigma : c = \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c); r)\}$$

that results from the relation

$$\mathcal{R}_{\mathsf{pk}} = \{(r, c) \in \mathtt{PKE}.\Sigma \times \mathtt{PKE}.\mathcal{C} : c = \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c); r)\}$$

on the set $\texttt{PKE}.\Sigma \times \texttt{PKE}.\mathcal{C}$ is decidable.

Now, suppose that for each choice of pk, there is an ϵ_{sound}-sound, $\epsilon_{\texttt{SLE}}$-straightline extractable, ϵ_{CR}-collision resistant, ϵ_{ZK}-zero-knowledge proof of knowledge proof system Π.

Under these assumptions, we construct a new PKE, PKEZK, as follows:

$\texttt{PKEZK}.\mathsf{KGen}(\lambda)$

1 : **return** $\texttt{PKE}.\mathsf{KGen}(\lambda)$

$\texttt{PKEZK}.\mathsf{Enc}(\mathsf{pk}, m; r)$

1 : $c \leftarrow \texttt{PKE}.\mathsf{Enc}(\mathsf{pk}, m; r)$
2 : $\pi \leftarrow \Pi.\texttt{PfGen}(c, (r, c))$
3 : $return(\pi, c)$

$\texttt{PKEZK}.\mathsf{Dec}(\mathsf{sk}, c)$

1 : Parse $(\pi, c) \leftarrow ctxt$
2 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
3 : **return** $\texttt{PKE}.\mathsf{Dec}(\mathsf{sk}, c)$

Figure 3.1: PKEZK Public Key Encryption Scheme

Theorem 1. PKEZK *is an* $(2\epsilon_{sound}+2\epsilon_{niZK}+\epsilon_{CR}+2\epsilon_{\texttt{SLE}}+2\epsilon_{DWR}+\epsilon_{NM}+2\epsilon_{\texttt{PKE}})$-IND-CCA-*secure public key encryption scheme.*

3.2 Proof Strategy

We will start with the IND-CCA experiment for PKEZK, and through a series of game hops (as Shoup described in [15]), show that the probability of an adversary winning the IND-CCA experiment for PKEZK is only negligibly different from the probability of an adversary winning the IND-CPA experiment for PKE. The idea is to show that the decryption oracle in the IND-CCA-experiment does not give the adversary any information that they could not have computed themselves.

Let S_i be the event that the adversary $\mathcal{A}$ wins game i. Then, the proof is outlined as follows:

- In G_1, we begin with the IND-CCA experiment for PKEZK.
- In G_2, our PKEZK decryption oracle fails on all PKEZK ciphertext queries (c, π) such that (c, π) is a valid ciphertext/proof pair for Π, but the PKE ciphertext component c is not in the language $\mathcal{L}_{\mathsf{pk}}$. Since Π satisfies non-interactive soundness, the probability of an adversary producing such a pair is $\epsilon_{\texttt{sound}}$, and we get that
$$|\Pr[S_2] - \Pr[S_1]| < \epsilon_{\texttt{sound}}.$$
- In G_3, our PKEZK decryption oracle fails on all PKEZK ciphertext queries (c, π) such that π is equal to the proof in the PKEZK challenge ciphertext, π_b, but c is not equal to the ciphertext in the PKEZK challenge ciphertext, c_b. Since Π is collision resistant, the probability of the adversary producing such a pair is ϵ_{CR}, and we get that
$$|\Pr[S_3] - \Pr[S_2]| < \epsilon_{\text{CR}}.$$

- In G_4, we respond to all proof queries and hash queries using the simulator, rather than responding honestly. Since Π satisfies non-interactive zero-knowledge, an adversary can only distinguish between these two worlds with a probability of (a constant factor of) ϵ_{niZK}, and we get that
$$|\Pr[S_4] - \Pr[S_3]| < 2\epsilon_{\text{niZK}}.$$

- In G_5, for each query (c, π), our $\texttt{PKEZK}$ decryption oracle uses the straight-line extractor to extract the random coins used during the encryption of c, and checks that this was done properly by decrypting c using the secret key, re-encrypting using the random coins extracted by the straight-line extractor, and checking that the resulting value is equal to c. Since we have already ensured that $c \in \mathcal{L}_{\mathsf{pk}}$, $\texttt{PKE.Dec}$ is guaranteed to output the correct result, so the only way this step can fail is if the straight-line extractor does not output the correct value of r. So, we get that
$$|\Pr[S_5] - \Pr[S_4]| < \epsilon_{\texttt{SLE}}.$$

- In G_6, we replace all instances of $\texttt{PKE.Dec}$ with $\texttt{PKE.DecRand}$ in our $\texttt{PKEZK}$ decryption oracle. Since $\texttt{PKE}$ satisfies $\texttt{DecRand}$-correctness, we get that
$$|\Pr[S_6] - \Pr[S_5]| < \epsilon_{\text{DWR}}.$$

- In G_7, our $\texttt{PKEZK}$ decryption oracle rejects all $\texttt{PKEZK}$ decryption queries (c, π) in which the $\texttt{PKE}$ ciphertext c is equal to the $\texttt{PKE}$ challenge ciphertext c_b. First, since $\texttt{PKE}$ is assumed to be $\epsilon_{\texttt{PKE}}$-IND-CPA-secure and ϵ_{DWR}-correct, and Π is assumed to be $\epsilon_{\texttt{SLE}}$-straight-line extractable and ϵ_{sound}-sound, we are able to show that for any efficient machine $\mathcal{M}$,
$$\Pr\left[\texttt{Exp1}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{M})\right] < \epsilon_{\text{DWR}} + \epsilon_{\texttt{SLE}} + \epsilon_{\text{sound}} + \epsilon_{\texttt{PKE}}.$$
Then, we show how any adversary who can distinguish between G_6 and G_7 can be used to win $\texttt{Exp2}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_{\mathsf{pk}}}$, giving us that
$$|\Pr[S_6] - \Pr[S_7]| \leq \Pr\left[\texttt{Exp2}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_{\mathsf{pk}}} \implies 1\right].$$
Finally, since Π is non-malleable, we get that
$$\left|\Pr\left[\texttt{Exp1}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{M}^{\mathcal{A}}) \implies 1\right] - \Pr\left[\texttt{Exp2}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_{\mathsf{pk}}} \implies 1\right]\right| < \epsilon_{\text{NM}},$$
giving a final result of
$$|\Pr[S_7] - \Pr[S_6]| < \epsilon_{\text{DWR}} + \epsilon_{\texttt{SLE}} + \epsilon_{\text{sound}} + \epsilon_{\texttt{PKE}} + \epsilon_{\text{NM}}(\mathcal{A}).$$

- Finally, we present an algorithm, which, given access to an adversary who can win the G_7, is able to win the IND-CPA game for $\texttt{PKE}$, to complete our proof.

3.3 Game-Hopping Proof

In each game, we highlight the lines that are different from the previous game.

3.3.1 Game 1

In G_1, we start out with the usual IND-CCA experiment for public key encryption, with the addition of a list $\texttt{HList}$ to keep track of hash queries, since we are working in the random oracle model. Our adversary is provided with a $\texttt{PKEZK}$ decryption oracle and a hash oracle.

$G_1(\lambda, \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \texttt{PKE.KGen}(\lambda)$
3 : $r \leftarrow\!\$\ \texttt{PKE}.\Sigma$
4 : $(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $b \leftarrow\!\$\ \{0,1\}$
6 : $c_b \leftarrow \texttt{PKE.Enc}(\mathsf{pk}, m_b; r)$
7 : $\pi_b \leftarrow \Pi.\texttt{PfGen}(c_b, r)$
8 : $ctxt_b \leftarrow (\pi_b, c_b)$
9 : $b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
10 : **return** $[\![b = b']\!]$

$\mathcal{O}_{\texttt{PKEZK.Dec}}(ctxt)$

1 : **if** $ctxt = ctxt_b$: **return** $\perp$
2 : Parse $(\pi, c) \leftarrow ctxt$
3 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
4 : **return** $\texttt{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow\!\$\ \{0,1\}^\lambda$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

3.3.2 Game 2

In G_2, our decryption oracle rejects all queries where the queried ciphertext c is not in $\mathcal{L}_{\mathsf{pk}}$.

$G_2(\lambda, \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \texttt{PKE.KGen}(\lambda)$
3 : $r \leftarrow\!\$\ \texttt{PKE}.\Sigma$
4 : $(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $b \leftarrow\!\$\ \{0,1\}$
6 : $c_b \leftarrow \texttt{PKE.Enc}(\mathsf{pk}, m_b; r)$
7 : $\pi_b \leftarrow \Pi.\texttt{PfGen}(c_b, r)$
8 : $ctxt_b \leftarrow (\pi_b, c_b)$
9 : $b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
10 : **return** $[\![b = b']\!]$

$\mathcal{O}_{\texttt{PKEZK.Dec}}(ctxt)$

1 : **if** $ctxt = ctxt_b$: **return** $\perp$
2 : Parse $(\pi, c) \leftarrow ctxt$
3 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
4 : **if** $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
5 : **return** $\texttt{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow \{0,1\}^\lambda$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

We will write a reduction which uses the failure event introduced in G_2 to win the non-interactive soundness attack game. Let $\mathcal{A}$ be an adversary who is attempting to distinguish between G_1 and G_2, and consider the following reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_{\mathcal{O}_{\text{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$, which acts as a challenger for the indistinguishability game between G_1 and G_2 and an adversary for the non-interactive soundness game:

$\mathcal{B}_1(\lambda)$

$\texttt{HList} \leftarrow \emptyset$
$(x_{\texttt{sound}}, \pi_{\texttt{sound}}) \leftarrow (\texttt{null}, \texttt{null})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \texttt{PKE}.\mathsf{KGen}(\lambda)$
$r \leftarrow\!\$\ \texttt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{\text{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk})$
$b \leftarrow\!\$\ \{0,1\}$
$c_b \leftarrow \texttt{PKE}.\mathsf{Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\texttt{PfGen}(c_b, r)$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{\text{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $(x_{\texttt{sound}}, \pi_{\texttt{sound}})$

$\mathcal{B}_{\mathcal{O}_{\text{PKEZK.Dec}}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$:
  $(x_{\texttt{sound}}, \pi_{\texttt{sound}}) \leftarrow (c, \pi)$
  return $\perp$
return $\texttt{PKE}.\mathsf{Dec}(\mathsf{sk}, c)$

$\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}(x)$

if $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
  return y
$y \leftarrow \{0,1\}^{\lambda}$
$\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
return y

Since $\mathcal{B}$ returns a non-null pair if and only if the failure event from G_2 is triggered, and any non-null pair returned by $\mathcal{B}$ must be a valid statement/proof pair for Π by a previous failure event of $\mathcal{O}$, then

$$|\Pr[S_2] - \Pr[S_3]| = \Pr\left[\texttt{Exp}_{\Pi}^{\text{niSound}}(\mathcal{B}) \implies 1\right].$$

But, by non-interactive soundness,

$$\Pr\left[\texttt{Exp}_{\Pi}^{\text{niSound}}\mathcal{B} \implies 1\right] < \epsilon_{\text{sound}}.$$

So,

$$|\Pr[S_2] - \Pr[S_3]| < \epsilon_{\text{sound}}.$$

3.3.3 Game 3

In G_3, our decryption oracle rejects all queries in which the queried proof π is equal to the challenge proof π_b.

$G_4(\lambda, \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

$\mathtt{HList} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathtt{PKE.KGen}(\lambda)$
$r \leftarrow\!\$\ \mathtt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
$b \leftarrow\!\$\ \{0,1\}$
$c_b \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\mathtt{PfGen}(c_b, r)$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{PKEZK.Dec}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
return $\mathtt{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
  return y
$y \leftarrow \{0,1\}^\lambda$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

We will write a reduction which uses the failure event introduced in G_3 to win the collision resistance attack game. Let $\mathcal{A}$ be an adversary who is attempting to distinguish between G_2 and G_3, and consider the following reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$, which acts as a challenger for the indistinguishability game between G_2 and G_3, and an adversary for the collision resistance attack game:

$\mathcal{B}_1^{\mathcal{A}}(\lambda)$

$\mathtt{HList} \leftarrow \emptyset$
$x' \leftarrow \mathtt{null}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathtt{PKE.KGen}(\lambda)$
$r \leftarrow\!\$\ \mathtt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk})$
$b \leftarrow\!\$\ \{0,1\}$
$c_b \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\mathtt{PfGen}(c_b, r)$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return (c_b, x', π_b)

$\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$:
  $x' \leftarrow c$
  return $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
return $\mathtt{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{B}_{\mathcal{O}_{\mathsf{H}}(x)}$

if $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
  return y
$y \leftarrow \{0,1\}^\lambda$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

Since $\mathcal{B}$ returns a non-null result if and only if the failure event from G_3 is triggered, and any non-null result returned by $\mathcal{B}$ must be a valid statement/proof pair for Π by a previous

failure event of $\mathcal{O}$, then

$$|\Pr[S_3] - \Pr[S_2]| = \Pr\left[\mathtt{Exp}^{\mathrm{CR}}_{\Pi}(\mathcal{B}) \implies 1\right].$$

But, by collision resistance,

$$\Pr\left[\mathtt{Exp}^{\mathrm{CR}}_{\Pi}(\mathcal{B}) \implies 1\right] < \epsilon_{\mathrm{CR}}.$$

So,

$$|\Pr[S_3] - \Pr[S_2]| < \epsilon_{\mathrm{CR}}.$$

3.3.4 Game 4

In G_4, we respond to all proof queries and hash queries using the simulator, rather than responding honestly.

$G_3(\lambda, \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

$\mathtt{HList} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathtt{PKE.KGen}(\lambda)$
$r \leftarrow\$ \mathtt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
$b \leftarrow\$ \{0, 1\}$
$c_b \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c_b, \mathtt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{PKEZK.Dec}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
return $\mathtt{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
  return y
$y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

We will write a reduction that reduces the indistinguishability of G_3 and G_4 to the indistinguishability game for simulators. Let $\mathcal{A}$ be an adversary who is attempting to distinguish between G_3 and G_4, and consider the following reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}})$, which acts as a challenger for the indistinguishability game between G_3 and G_4 and an adversary for the indistinguishability game for simulators, where $\mathcal{O}$ is the oracle provided to $\mathcal{B}$ in the indistinguishability game for simulators:

$\mathcal{B}_1^{\mathcal{A},\mathcal{O}}(\lambda)$

$\texttt{HList} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \ \mathtt{PKE}.\mathsf{KGen}(\lambda)$
$r \leftarrow\$ \ \mathtt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\$ \ \mathcal{A}^{\mathcal{B}^{\mathcal{O}_{\mathtt{PKEZK.Dec}}}, \mathcal{B}^{\mathcal{O}}}(\mathsf{pk})$
$b \leftarrow\$ \ \{0,1\}$
$c_b \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m_b; r)$
$\pi \leftarrow \mathcal{O}(\texttt{proofQuery}, c_b, \texttt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\$ \ \mathcal{A}^{\mathcal{B}^{\mathcal{O}_{\mathtt{PKEZK.Dec}}}, \mathcal{B}^{\mathcal{O}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
if $b = b'$: **return** 1
else : **return** 0

$\mathcal{B}^{\mathcal{O}_{\mathtt{PKEZK.Dec}}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
return $\mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c)$

$\mathcal{B}^{\mathcal{O}}(x)$

if $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
  return y
$y \leftarrow \mathcal{O}(\texttt{hashQuery}, x)$
$\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
return y

Notice that when $\mathcal{O} = \mathcal{O}_0$ from $\mathtt{Exp}_{\Pi}^{\mathrm{niZK}}$, $\mathcal{B}$ is identical to G_3, and when $\mathcal{O} = \mathcal{O}_1$ from $\mathtt{Exp}_{\Pi}^{\mathrm{niZK}}$, then $\mathcal{B}$ is identical to G_4. Since Π is ϵ_{niZK}-secure, we have that

$$\left| \Pr[\mathcal{B} \text{ wins}] - \frac{1}{2} \right| < \epsilon_{\mathrm{niZK}}.$$

However,

$$\begin{aligned}
\Pr[\mathcal{B} \text{ wins}] &= \frac{1}{2}\Pr[\mathcal{B} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0] + \frac{1}{2}\Pr[\mathcal{B} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] \\
&= \frac{1}{2}\Pr[\mathcal{A} \text{ loses} \mid \mathcal{O} = \mathcal{O}_0] + \frac{1}{2}\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] \\
&= \frac{1}{2}\left(1 - \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]\right) + \frac{1}{2}\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] \\
&= \frac{1}{2} + \frac{1}{2}\left(\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] - \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]\right).
\end{aligned}$$

So, this gives us that

$$\begin{aligned}
&\left| \frac{1}{2} + \frac{1}{2}\left(\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] - \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]\right) - \frac{1}{2} \right| \\
&= \frac{1}{2}\left|\left(\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] - \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]\right)\right| \\
&< \epsilon_{\mathrm{niZK}},
\end{aligned}$$

and hence that

$$\left|\Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1] - \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]\right| < 2\epsilon_{\mathrm{niZK}}.$$

But $\Pr[S_4] = \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_1]$ and $\Pr[S_3] = \Pr[\mathcal{A} \text{ wins} \mid \mathcal{O} = \mathcal{O}_0]$, so we get that

$$\left|\Pr[S_4] - \Pr[S_3]\right| < 2\epsilon_{\mathrm{niZK}}.$$

3.3.5 Game 5

In G_5, we use the straightline extractor to extract the randomness used during encryption, and check that encrypting under that randomness results in the given ciphertext.

$G_5(\lambda, \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

$\mathtt{HList} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{PKE.KGen}(\lambda)$
$r \leftarrow\$ \mathsf{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
$b \leftarrow\$ \{0,1\}$
$c_b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c_b, \mathtt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{PKEZK.Dec}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\pi = \pi_b$: **return** $\perp$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \mathsf{SLE}(c, \pi, \mathtt{HList})$
if $c \neq \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{PKE.Dec}(\mathsf{sk}, c); r)$:
  return $\perp$
return $\mathsf{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
  return y
$y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

We will write a reduction which uses the failure event introduced in G_5 to win the SLE attack game. Let $\mathcal{A}$ be an adversary who is attempting to distinguish between G_4 and G_5, and consider the following reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$, which acts as a challenger for the indistinguishability game between G_4 and G_5, and an adversary for the SLE attack game:

$\mathcal{B}_1^{\mathcal{A}}(\lambda)$

$\texttt{HList} \leftarrow \emptyset$
$\tilde{r} \leftarrow \texttt{null}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\ \mathsf{PKE.KGen}()$
$r \leftarrow\$\ \mathsf{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
$b \leftarrow\$\ \{0,1\}$
$c_b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\texttt{Sim}(\texttt{proofQuery}, c_b, \texttt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $\tilde{r}$

$\mathcal{O}_{\mathsf{PKEZK.Dec}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \mathsf{SLE}(c, \pi, \texttt{HList})$
if $c \neq \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{PKE.Dec}(\mathsf{sk}, c); r)$:
$\tilde{r} \leftarrow r$
  return $\perp$
return $\mathsf{PKE.Dec}(\mathsf{sk}, c)$

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
  return y
$y \leftarrow \Pi.\texttt{Sim}(\texttt{hashQuery}, x, \texttt{HList})$
$\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
return y

Since $\mathcal{B}$ returns a non-null result if and only if the failure event from G_5 is triggered, any non-null result returned by $\mathcal{B}$ is a valid proof/statement pair for Π by a previous failure event of $\mathcal{O}$, and $\mathsf{PKE.Dec}(\mathsf{sk}, c)$ will be correct with probability 1 since $c \in \mathcal{L}_{\mathsf{pk}}$, then

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr\left[\mathbf{Exp}_{\Pi}^{\mathtt{SLE}}(\mathcal{B}^{\mathcal{A}}) \implies 1\right].$$

But, by straight-line extractability,

$$\Pr\left[\mathbf{Exp}_{\Pi}^{\mathtt{SLE}}(\mathcal{B}^{\mathcal{A}}) \implies 1\right] < \epsilon_{\mathtt{SLE}}.$$

So,

$$|\Pr[S_5] - \Pr[S_4]| < \epsilon_{\mathtt{SLE}}.$$

3.3.6 Game 6

In G_6, we replace all instances of $\mathsf{PKE.Dec}$ with $\mathsf{PKE.DecRand}$.

$G_6(\lambda, \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

$\mathtt{HList} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathtt{PKE.KGen}()$
$r \leftarrow\!\$\ \mathtt{PKE}.\Sigma$
$(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
$b \leftarrow\!\$\ \{0, 1\}$
$c_b \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c_b, \mathtt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{PKEZK.Dec}}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \mathtt{SLE}(c, \pi, \mathtt{HList})$
if $c \neq \mathtt{PKE.Enc}(\mathsf{pk}, \mathtt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
  return $\perp$
return $\mathtt{PKE.DecRand}(\mathsf{pk}, c, r)$

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
  return y
$y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
$\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
return y

We will write a reduction which uses the failure event introduced in G_6 to win the DecRand-correctness attack game. Let $\mathcal{A}$ be an adversary who is attempting to distinguish between G_5 and G_6, and consider the following reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_{\mathcal{O}_0}, \mathcal{B}_{\mathcal{O}_1}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$, which is a challenger for the indistinguishability game between G_6 and G_7, and an adversary for the DecRand-correctness attack game.

$\mathcal{B}_1^{\mathcal{A}}(\mathsf{pk}, \mathsf{sk})$

$\texttt{HList} \leftarrow \emptyset$
$FList \leftarrow \emptyset$
$bb \leftarrow\$ \{0,1\}$
$(m_0, m_1, \mathsf{st}) \leftarrow\$ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{bb}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk})$
$b \leftarrow\$ \{0,1\}$
$c_b \leftarrow \texttt{PKE.Enc}(\mathsf{pk}, m_b; r)$
$\pi_b \leftarrow \Pi.\texttt{Sim}(\texttt{proofQuery}, c_b, \texttt{HList})$
$ctxt_b \leftarrow (\pi_b, c_b)$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{bb}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
foreach (c, r) **in** $Flist$ **do** :
  if $\texttt{PKE.DecRand}(\mathsf{pk}, c, r) \neq \texttt{PKE.Dec}(\mathsf{sk}, c)$:
  return (c, r)
return `null`

$\mathcal{O}_{\mathsf{H}}(x)$

if $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
  return y
$y \leftarrow \Pi.\texttt{Sim}(\texttt{hashQuery}, x, \texttt{HList})$
$\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
return y

$\mathcal{B}_{\mathcal{O}_0}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \texttt{SLE}(c, \pi, \texttt{HList})$
if $c \neq \texttt{PKE.Enc}(\mathsf{pk}, \texttt{PKE.Dec}(\mathsf{sk}, c); r)$:
  $Flist \leftarrow Flist \cup \{(c, r)\}$
  return $\perp$
return $\texttt{PKE.Dec}(\mathsf{pk}, c, r)$

$\mathcal{B}_{\mathcal{O}_1}(ctxt)$

if $ctxt = ctxt_b$: **return** $\perp$
Parse $(\pi, c) \leftarrow ctxt$
if $\pi = \pi_b$: **return** $\perp$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \texttt{SLE}(c, \pi, \texttt{HList})$
if $c \neq \texttt{PKE.Enc}(\mathsf{pk}, \texttt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
  $Flist \leftarrow Flist \cup \{(c, r)\}$
  return $\perp$
return $\texttt{PKE.DecRand}(\mathsf{pk}, c, r)$

Since $\mathcal{B}_{\mathcal{O}_0}$ and $\mathcal{B}_{\mathcal{O}_1}$ behave identically unless the failure event on line 7 gets triggered, we have that

$$|\Pr[S_6] - \Pr[S_5]| \leq |\Pr[\text{Line 7 returns } \texttt{true} \text{ in } \mathcal{B}_{\mathcal{O}_0}] - \Pr[\text{Line 7 returns } \texttt{true} \text{ in } \mathcal{B}_{\mathcal{O}_1}]|.$$

Notice that in $\mathcal{B}_{\mathcal{O}_0}$, line 7 will return `true` if and only if the straight-line extractor does not return the correct value of r, or $\texttt{PKE.Dec}$ does not decrypt c properly. However, since a previous failure event ensures that $c \in \mathcal{L}_{\mathsf{pk}}$, the probability $\texttt{PKE.Dec}$ not decrypting c properly is 0. So,

$$\Pr[\text{Line 7 returns } \texttt{true} \text{ in } \mathcal{B}_{\mathcal{O}_0}] \leq \Pr[\Pi.\texttt{SLE} \text{ returns incorrect value}] < \epsilon_{\texttt{SLE}}$$

by the previous reduction. Similarly, in $\mathcal{B}_{\mathcal{O}_1}$, line 7 will return `true` if and only if the straight-line extractor does not return the correct value of r, or $\texttt{PKE.DecRand}$ does not decrypt c properly. So,

$$\begin{aligned}
&\Pr[\text{Line 7 returns } \texttt{true} \text{ in } \mathcal{B}_{\mathcal{O}_1}] \\
\leq\ &\Pr[\Pi.\texttt{SLE} \text{ returns incorrect value}] \\
&+ \Pr[\texttt{PKE.DecRand} \text{ returns incorrect value } \wedge\ \Pi.\texttt{SLE} \text{ returns correct value}].
\end{aligned}$$

If $\texttt{PKE.DecRand}$ returns an incorrect value but $\Pi.\texttt{SLE}$ returns a correct value, then $\mathcal{B}$ will be a successful adversary for $\texttt{Exp}^{\text{DWR}}_{\texttt{PKE}}$. In this case, $FList$ will contain at least one pair (c, r) such that $(c, r) \in \mathcal{R}_{\mathsf{pk}}$, since the previous failure event ensures that $c \in \mathcal{L}_{\mathsf{pk}}$ and hence that there exists r such that $(c, r) \in \mathcal{R}_{\mathsf{pk}}$, and by assumption, the value of r returned by $\Pi.\texttt{SLE}$ satisfies this relation. But $\texttt{PKE.DecRand}$ is assumed to be ϵ_{DWR}-correct, so

$$\begin{aligned}
&\Pr[\texttt{PKE.DecRand} \text{ returns incorrect value } \wedge\ \Pi.\texttt{SLE} \text{ returns correct value}] \\
&\leq \Pr\left[\texttt{Exp}^{\text{DWR}}_{\texttt{PKE}}(\mathcal{B}) \implies 1\right] \\
&< \epsilon_{\text{DWR}}.
\end{aligned}$$

Since

$$\Pr[\Pi.\texttt{SLE} \text{ returns incorrect value}] < \epsilon_{\texttt{SLE}}$$

by a previous reduction, we get that

$$\Pr[\text{Line 7 returns } \texttt{true} \text{ in } \mathcal{B}_{\mathcal{O}_1}] < \epsilon_{\texttt{SLE}} + \epsilon_{\text{DWR}}.$$

So,

$$|\Pr[S_6] - \Pr[S_5]| < |\epsilon_{\texttt{SLE}} + \epsilon_{\text{DWR}} - \epsilon\texttt{SLE}| = \epsilon_{\text{DWR}}.$$

3.3.7 Game 7

In G_7, our decryption oracle rejects all queries in which the queried ciphertext c is equal to the challenge ciphertext c_b. Notice that since we are now comparing c with c_b and π with π_b individually, the line of code comparing $ctxt$ with $ctxt_b$ is now redundant and can therefore be removed.

$G_7(\lambda, \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \texttt{PKE.KGen}(\lambda)$
3 : $r \leftarrow\!\$\ \texttt{PKE}.\Sigma$
4 : $(m_0, m_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $b \leftarrow\!\$\ \{0, 1\}$
6 : $c_b \leftarrow \texttt{PKE.Enc}(\mathsf{pk}, m_b; r)$
7 : $\pi_b \leftarrow \Pi.\texttt{Sim}(\texttt{proofQuery}, c_b, \texttt{HList})$
8 : $ctxt_b \leftarrow (\pi_b, c_b)$
9 : $b' \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}_{\texttt{PKEZK.Dec}}, \mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
10 : **return** $[\![b = b']\!]$

$\mathcal{O}_{\texttt{PKEZK.Dec}}(ctxt)$

1 : Parse $(\pi, c) \leftarrow ctxt$
2 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
3 : **if** $\pi = \pi_b$: **return** $\perp$
4 : **if** $c = c_b$: **return** $\perp$
5 : **if** $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
6 : $r \leftarrow \texttt{SLE}(c, \pi, \texttt{HList})$
7 : **if** $c \neq \texttt{PKE.Enc}(\mathsf{pk}, \texttt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
8 : **return** $\perp$
9 : **return** $\texttt{PKE.DecRand}(\mathsf{pk}, c, r)$

$\mathcal{O}_{\mathsf{H}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\texttt{Sim}(\texttt{hashQuery}, x, \texttt{HList})$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

The non-malleability experiments are defined in terms of a probabilistic algorithm f, with auxiliary inputs generated from an algorithm $\mathtt{Gen}$. In our case, we set f to be $\mathtt{PKE}.\mathsf{Enc}$, for a specific public key pk, which is generated by KGen. We also allow the random coins to be given as input, rather than having them be generated by $\mathtt{PKE}.\mathsf{Enc}$, as we have done throughout this thesis. The experiments, with these changes inlined, are defined as follows:

$\underline{\mathtt{Exp1}^{\mathrm{NM}}_{\Pi,\mathtt{PKE}.\mathsf{Enc}_{\mathsf{pk}}}(\mathcal{M})}$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\ \mathtt{PKE}.\mathsf{KGen}()$
3 : $r \leftarrow\$\ \mathtt{PKE}.\Sigma$
4 : $m \leftarrow\$\ \mathcal{M}^{\mathcal{A},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $c^* \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m; r)$
6 : $\pi \leftarrow\$\ \mathcal{M}^{\mathcal{A},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, c^*)$
7 : **return** $[\![\Pi.\mathsf{Vf}\pi, c^*]\!]$

$\underline{\mathtt{Exp2}^{\mathrm{NM}}_{\Pi,\mathtt{PKE}.\mathsf{Enc}_{\mathsf{pk}}}(\mathcal{A})}$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$\ \mathtt{PKE}.\mathsf{KGen}()$
3 : $r \leftarrow\$\ \mathtt{PKE}.\Sigma$
4 : $m \leftarrow\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $c^* \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m; r)$
6 : $\pi^* \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c^*, \mathtt{HList})$
7 : $\pi \leftarrow\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, c^*, \pi^*)$
8 : **return** $[\![\Pi.\mathsf{Vf}\pi, c^* \wedge \pi \neq \pi^*]\!]$

where $\mathcal{O}_{\mathsf{H}}$ is the same as in the non-malleability definition:

$\underline{\mathcal{O}_{\mathsf{H}}(x)}$

1 : **if** $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
4 : $\mathtt{HList} \leftarrow \mathtt{HList} \cup (x, y)$
5 : **return** y

Lemma 1. *For all probabilistic polynomial time adversaries $\mathcal{A}$ for $\mathtt{Exp2}^{NM}_{\Pi,\mathtt{PKE}.\mathsf{Enc}_{\mathsf{pk}}}$ and all probabilistic polynomial time machines $\mathcal{M}$,*

$$\Pr\left[\mathtt{Exp1}^{NM}_{\Pi,\mathtt{PKE}.\mathsf{Enc}_{\mathsf{pk}}}(\mathcal{M}^{\mathcal{A}}) \implies 1\right] < \epsilon_{DWR} + \epsilon_{\mathtt{SLE}} + \epsilon_{sound} + \epsilon_{\mathtt{PKE}}.$$

Proof. To show this, we will write a reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$ that can win the IND-CPA game for $\mathtt{PKE}$ whenever $\mathcal{M}$ is able to win $\mathtt{Exp1}^{\mathrm{NM}}_{\Pi,\mathtt{PKE}.\mathsf{Enc}_{\mathsf{pk}}}$. Consider the following algorithm:

$\mathcal{B}_1(\mathsf{pk})$

1 : $\texttt{HList} \leftarrow \emptyset$
2 : $m_0 \leftarrow\!\!\$\ \mathcal{M}^{\mathcal{A},\mathcal{B}_{\mathcal{O}_\mathsf{H}}}()$
3 : $m_1 \leftarrow\!\!\$\ \texttt{PKE}.\mathcal{M}$
4 : **return** (m_0, m_1)

$\mathcal{B}_2(c_b, m_0, m_1)$

1 : $\pi \leftarrow\!\!\$\ \mathcal{M}^{\mathcal{A},\mathcal{B}_{\mathcal{O}_\mathsf{H}}}(\mathsf{pk}, c_b)$
2 : **if** $c \notin \mathcal{L}_\mathsf{pk}$: **return** $\bot$
3 : $r \leftarrow \Pi.\texttt{SLE}(c^*, \pi, \texttt{HList})$
4 : **if** $c \neq \texttt{PKE}.\mathsf{Enc}(\mathsf{pk}, \texttt{PKE.DecRand}(\mathsf{pk}, c^*, r); r)$:
5 : **return** $\bot$
6 : $m_b \leftarrow \texttt{DecRand}(\mathsf{pk}, c_b, r)$
7 : **if** $m_b = m_0$: **return** 0
8 : **else** : **return** 1

$\mathcal{B}_{\mathcal{O}_\mathsf{H}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \texttt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\texttt{Sim}(\texttt{hashQuery}, x, \texttt{HList})$
4 : $\texttt{HList} \leftarrow \texttt{HList} \cup \{(x, y)\}$
5 : **return** y

By the same steps as were taken in earlier (and later) reductions in the proof, since DecRand is ϵ_{DWR}-correct, and Π is $\epsilon_{\texttt{SLE}}$-straight-line extractable and ϵ_{sound}-sound, we get that

$$\left|\Pr\left[\texttt{Exp}^{\text{IND-CPA}}_{\texttt{PKE}}(\mathcal{B}) \implies 1\right] - \Pr\left[\texttt{Exp1}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_\mathsf{pk}}(\mathcal{M}^{\mathcal{A}}) \implies 1\right]\right| < \epsilon_{\text{DWR}} + \epsilon_{\texttt{SLE}} + \epsilon_{\text{sound}} + \epsilon_{\texttt{PKE}}.$$

■

Now, we'll write a reduction $(\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_{\mathcal{O}_{\texttt{PKEZK.Dec}}}, \mathcal{B}_{\mathcal{O}_\mathsf{H}})$ that is able to use any adversary $\mathcal{A}$ who can distinguish between games G_6 and G_7 in order to win $\texttt{Exp2}^{\text{NM}}_{\Pi,\texttt{PKE.Enc}_\mathsf{pk}}$. Consider the following algorithm:

$\mathcal{B}_1^{\mathcal{A},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$

$\texttt{HList}^* \leftarrow \emptyset$
$\tilde{\pi} \leftarrow \texttt{null}$
$(m_0, m_1, \mathsf{st}) \leftarrow\$ \mathcal{A}_G^{\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}},\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk})$
$b \leftarrow\$ \{0,1\}$
return m_b

$\mathcal{B}_2^{\mathcal{A},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, c^*, \pi^*)$

$ctxt_b \leftarrow (\pi^*, c^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}},\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
return $\tilde{\pi}$

$\mathcal{B}_{\mathcal{O}_{\mathsf{PKEZK.Dec}}}(ctxt)$

Parse $(\pi, c) \leftarrow ctxt$
if $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
if $\pi = \pi_b$: **return** $\perp$
if $c = c_b$:
  $\tilde{\pi} \leftarrow \pi$
  return $\perp$
if $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
$r \leftarrow \mathtt{SLE}(c, \pi, \texttt{HList}^*)$
if $c \neq \mathtt{PKE.Enc}(\mathsf{pk}, \mathtt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
  return $\perp$
return $\mathtt{PKE.DecRand}(\mathsf{pk}, c, r)$

$\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}(x)$

if $\exists y$ s.t. $\{(x, y)\} \in \texttt{HList}^*$:
  return y
$y \leftarrow \mathcal{O}_{\mathsf{H}}(x)$
$\texttt{HList}^* \leftarrow \texttt{HList}^* \cup \{(x, y)\}$
return y

Since $\mathcal{B}_2$ returns a non-null result if and only if the failure event from G_7 is triggered, we get that

$$|\Pr[S_6] - \Pr[S_7]| \leq \Pr\left[\mathtt{Exp2}^{\mathrm{NM}}_{\Pi,\mathtt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{B}^{\mathcal{A}}) \implies 1\right].$$

Since Π is non-malleable, we have that

$$\left|\Pr\left[\mathtt{Exp1}^{\mathrm{NM}}_{\Pi,\mathtt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{M}^{\mathcal{B}^{\mathcal{A}}}) \implies 1\right] - \Pr\left[\mathtt{Exp2}^{\mathrm{NM}}_{\Pi,\mathtt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{B}^{\mathcal{A}}) \implies 1\right]\right| < \epsilon_{\mathrm{NM}}$$

and by Lemma 1, we have that

$$\Pr\left[\mathtt{Exp1}^{\mathrm{NM}}_{\Pi,\mathtt{PKE.Enc}_{\mathsf{pk}}}(\mathcal{M}) \implies 1\right] < \epsilon_{\mathrm{DWR}} + \epsilon_{\mathtt{SLE}} + \epsilon_{\mathrm{sound}} + \epsilon_{\mathtt{PKE}}.$$

Combining these probabilities gives us that

$$|\Pr[S_6] - \Pr[S_7]| < \epsilon_{\mathrm{NM}} + \epsilon_{\mathrm{DWR}} + \epsilon_{\mathtt{SLE}} + \epsilon_{\mathrm{sound}} + \epsilon_{\mathtt{PKE}}.$$

3.3.8 Reduction to IND-CPA-security

Suppose $\mathcal{A}$ is an adversary for G_7, and consider the reduction $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_{\mathcal{O}_{\mathsf{Dec}}}, \mathcal{B}_{\mathcal{O}_{\mathsf{H}}})$ which acts as an IND-CPA adversary for PKE:

$\mathcal{B}_1(\mathsf{pk})$

1 : **return** $\mathcal{A}^{\mathcal{B}\mathcal{O}_{\mathsf{Dec}},\mathcal{B}\mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$

$\mathcal{B}_2(\mathsf{pk}, c_b, \mathsf{st})$

1 : $\pi_b \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c_b, \mathtt{HList})$
2 : $ctxt_b \leftarrow (\pi_b, c_b)$
3 : **return** $\mathcal{A}^{\mathcal{B}\mathcal{O}_{\mathsf{Dec}},\mathcal{B}\mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$

$\mathcal{B}_{\mathcal{O}_{\mathsf{Dec}}}(ctxt)$

1 : Parse $(\pi, c) \leftarrow ctxt$
2 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
3 : **if** $\pi = \pi_b$: **return** $\perp$
4 : **if** $c = c_b$: **return** $\perp$
5 : **if** $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
6 : $r \leftarrow \mathtt{SLE}(c, \pi, \mathtt{HList})$
7 : **if** $c \neq \mathtt{PKE.Enc}(\mathsf{pk}, \mathtt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
8 : **return** $\perp$
9 : **return** $\mathtt{PKE.DecRand}(\mathsf{pk}, c, r)$

$\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}(x)$

1 : **if** $\exists y$ s.t. $\{(x, y)\} \in \mathtt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
4 : $\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
5 : **return** y

Let S_8 be the event that $\mathcal{B}$ wins the IND-CPA experiment for PKE. Then we have that

$$\Pr[S_8] = \Pr[S_7],$$

since inlining $\mathcal{B}$ into the IND-CPA experiment for PKE gives us

$\mathrm{Exp}^{\text{IND-CPA}}_{\mathtt{PKE}}(\lambda, (\mathcal{B}_1, \mathcal{B}_2))$

1 : $\mathtt{HList} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathtt{PKE.KGen}(\lambda)$
3 : $r \leftarrow\!\!\$\ \mathtt{PKE}.\Sigma$
4 : $(m_0, m_1, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathtt{PKEZK.Dec}},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk})$
5 : $b \leftarrow\!\!\$\ \{0, 1\}$
6 : $c_b \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b; r)$
7 : $\pi_b \leftarrow \Pi.\mathtt{Sim}(\mathtt{proofQuery}, c_b, \mathtt{HList})$
8 : $ctxt_b \leftarrow (\pi_b, c_b)$
9 : $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathtt{PKEZK.Dec}},\mathcal{O}_{\mathsf{H}}}(\mathsf{pk}, ctxt_b, \mathsf{st})$
10 : **return** $[\![b = b']\!]$

$\mathcal{B}_{\mathcal{O}_{\mathsf{Dec}}}(ctxt)$

1 : Parse $(\pi, c) \leftarrow ctxt$
2 : **if** $\Pi.\mathsf{Vf}(c, \pi) \neq \mathsf{true}$: **return** $\perp$
3 : **if** $\pi = \pi_b$: **return** $\perp$
4 : **if** $c = c_b$: **return** $\perp$
5 : **if** $c \notin \mathcal{L}_{\mathsf{pk}}$: **return** $\perp$
6 : $r \leftarrow \mathtt{SLE}(c, \pi, \mathtt{HList})$
7 : **if** $c \neq \mathtt{PKE.Enc}(\mathsf{pk}, \mathtt{PKE.DecRand}(\mathsf{pk}, c, r); r)$:
8 : **return** $\perp$
9 : **return** $\mathtt{PKE.DecRand}(\mathsf{pk}, c, r)$

$\mathcal{B}_{\mathcal{O}_{\mathsf{H}}}(x)$

1 : **if** $\exists y$ s.t. $(x, y) \in \mathtt{HList}$:
2 : **return** y
3 : $y \leftarrow \Pi.\mathtt{Sim}(\mathtt{hashQuery}, x, \mathtt{HList})$
4 : $\mathtt{HList} \leftarrow \mathtt{HList} \cup \{(x, y)\}$
5 : **return** y

which is exactly G_7.

3.3.9 Final Result

Combining all of the above gives us that

$$|\Pr[S_1] - \Pr[S_8]| < 2\epsilon_{\text{sound}} + 2\epsilon_{\text{niZK}} + \epsilon_{\text{CR}} + 2\epsilon_{\texttt{SLE}} + 2\epsilon_{\text{DWR}} + \epsilon_{\text{NM}} + \epsilon_{\texttt{PKE}}.$$

But PKE is assumed to be $\epsilon_{\texttt{PKE}}$-IND-CPA-secure, so

$$\Pr[S_8] < \epsilon_{\texttt{PKE}}.$$

Thus,

$$\Pr[S_1] < 2\epsilon_{\text{sound}} + 2\epsilon_{\text{niZK}} + \epsilon_{\text{CR}} + 2\epsilon_{\texttt{SLE}} + 2\epsilon_{\text{DWR}} + \epsilon_{\text{NM}} + 2\epsilon_{\texttt{PKE}},$$

and hence PKEZK is an $(\epsilon_{\texttt{PKE}} + 2\epsilon_{\text{sound}} + 2\epsilon_{\text{niZK}} + \epsilon_{\text{CR}} + 2\epsilon_{\texttt{SLE}} + 2\epsilon_{\text{DWR}} + \epsilon_{\text{NM}} + 2\epsilon_{\texttt{PKE}})$-IND-CCA-secure public key encryption scheme. This concludes the proof of the theorem.

Chapter 4

Instantiation

To instantiate our construction, we use ElGamal [6] as our base public key encryption scheme. For the zero knowledge proof of knowledge, we start with a modified version of Schnorr's protocol [14], and then apply Fischlin's transformation [7] to convert this into a straight-line extractable zero knowledge proof of knowledge in the random oracle model. In this section, we describe each of these compenents in detail.

4.1 The ElGamal Public Key Encryption Scheme

Let $\mathcal{G}$ be a polynomial-time algorithm that, on input λ, outputs (a description of) a cyclic group $\mathbb{G}$, the prime order q of $\mathbb{G}$, and a generator g of $\mathbb{G}$. The Elgamal encryption scheme consists of the tuple of algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, defined as follows:

$\mathsf{KGen}(\lambda)$

1 : $(\mathbb{G}, q, g) \leftarrow\!\$\ \mathcal{G}(\lambda)$
2 : $x \leftarrow\!\$\ \mathbb{Z}_q$
3 : $h \leftarrow g^x$
4 : **return** $(\mathsf{pk} = (\mathbb{G}, q, g, h), \mathsf{sk} = (\mathbb{G}, q, g, x))$

$\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{G}, r \in \mathbb{Z}_q)$

1 : $c_1 \leftarrow g^r$
2 : $c_2 \leftarrow m \cdot h^r$
3 : **return** $ctxt = (c_1, c_2)$

$\mathsf{Dec}(\mathsf{sk}, ctxt)$

1 : **return** $\dfrac{c_2}{c_1^x}$

Figure 4.1: ElGamal Public Key Encryption Scheme

It is shown in [6] that this encryption scheme satisfies perfect correctness and IND-CPA security, under the assumption that computing discrete logarithms is hard. It remains for us to show that it is decryptable with randomness, and satisfies DWR-correctness.

Lemma 2. *The ElGamal public key encryption scheme satisfies DWR-correctness.*

Proof. Suppose c is a well-formed ElGamal ciphertext and consider the following algorithm:

$$\frac{\mathtt{DecRand}(\mathsf{pk}, c = (c_1, c_2), r)}{1: \quad \textbf{return } \frac{c_2}{h^r}}$$

For any $r \in \mathbb{Z}_q$ and $m \in \mathbb{G}$, we have that

$$\frac{c_2}{h^r} = \frac{m \cdot h^r}{h^r} = m,$$

as desired, and hence the algorithm described above is correct. ■

4.2 Schnorr's Protocol

The base of the non-interactive zero knowledge proof used in our instantiation is a modified version of Schnorr's Sigma protocol. We will begin by defining Schnorr's protocol [14].

Let $\mathbb{G}$ be a cyclic group of prime order q. Schnorr's protocol is a Sigma protocol which proves knowledge of a discrete logarithm; that is, it is a proof that one knows x such that $g^x = y$, for a known generator g of $\mathbb{G}$ and (public) commitment y. The protocol proceeds as follows:

$\mathcal{P}_1(\mathbb{G}, q, x, y)$

1: $k \leftarrow\$ \mathbb{Z}_q$
2: $\mathtt{cmt} \leftarrow g^k$
3: **return** $(g^k, \mathtt{st}_P)$

$\mathcal{P}_2(\mathtt{ch}, \mathtt{st}_P)$

1: $\mathtt{rsp} \leftarrow k + \mathtt{ch} \cdot x$
2: **return** $\mathtt{rsp}$

$\mathcal{V}_1(\mathbb{G}, q, y, \mathtt{cmt})$

1: $\mathtt{ch} \leftarrow\$ \mathbb{Z}_q$
2: **return** $(\mathtt{ch}, \mathtt{st}_V)$

$\mathcal{V}_2(\mathtt{rsp}, \mathtt{st}_V)$

1: **return** $[\![g^{\mathtt{rsp}} = \mathtt{cmt} \cdot y^{\mathtt{ch}}]\!]$

Figure 4.2: Schnorr's Protocol

Schnorr's protocol is known to satisfy completeness, soundness, and SHVZK [3].

Recall that for our construction, we require that the statement for our zero knowledge proof of knowledge is a ciphertext, and the witness is the random coins that were used to produce that ciphertext. With no further conditions on the proof, an adversary could "cheat" by choosing a random ciphertext as the statement, and proving knowledge of any valid set of random coins. For PKE's such as ElGamal, where for any choice of ciphertext and random coins, there exists a valid message which encrypts under those random coins into the given ciphertext, this makes the addition of the proof of knowledge completely useless, since anyone could come up with such a proof.

In our construction, we eliminate this issue by requiring our proofs to be collision resistant. Notice that any proof system which does not "bind" the proof of knowledge of the random coins to the message that gets encrypted (as is described above) does not satisfy collision resistance, since the random coins can (and should) be chosen independently of the message, so the proof does not necessarily need to be dependent on the statement.

Example 1. *Using Schnorr's protocol to prove knowledge of random coins used in ElGamal encryption falls victim to the problem described above. Let* $\mathsf{pk} = (\mathbb{G}, q, g, h)$ *be an ElGamal public key and* $\mathsf{sk} = (\mathbb{G}, q, g, x)$ *be an ElGamal secret key, as defined in the previous section. Let* (c_1, c_2) *be an ElGamal ciphertext, where* $c_1 = g^r$ *and* $c_2 = m \cdot h^r$ *for message* m *and random coins* r*. Let* π *be a Schnorr proof of knowledge of* r *such that* $g^r = h$*, i.e., a proof of knowledge of the random coins used during encryption. Then, for any choice of* $\alpha \in \mathbb{Z}_q$*,* π *is also a valid Schnorr proof of knowledge for the ciphertext* $(c_1, \alpha \cdot c_2)$*, which is the encryption of* $\alpha \cdot m$ *using random coins* r*.*

For our instantiation, it only matters that the final non-interactive zero knowledge proof system formed through Fischlin's transformation is collision resistant. However, proving the collision resistance of that reduces to proving the collision resistance of the Sigma protocol used. For this reason, we present a modification of Schnorr's protocol that satisfies collision resistance.

Let (c_1, c_2) be the ElGamal encryption of message m using random coins r. The modified Schnorr protocol is as follows:

$\mathcal{P}_1(\mathbb{G}, q, r, (c_1, c_2))$

1 : $k \leftarrow\$ \mathbb{Z}_q$
2 : $\mathtt{cmt} \leftarrow g^k$
3 : **return** (g^k, st_P)

$\mathcal{P}_2(\mathtt{ch}, \mathsf{st}_P)$

1 : $\mathtt{rsp} \leftarrow k + \mathsf{H}(\mathtt{ch}, c_2) \cdot r$
2 : **return** $\mathtt{rsp}$

$\mathcal{V}_1(\mathbb{G}, q, c_1, \mathtt{cmt})$

1 : $\mathtt{ch} \leftarrow\$ \mathbb{Z}_q$
2 : **return** $(\mathtt{ch}, \mathsf{st}_V)$

$\mathcal{V}_2(\mathtt{rsp}, \mathsf{st}_V)$

1 : **return** $[\![g^{\mathtt{rsp}} = \mathtt{cmt} \cdot c_1^{\mathsf{H}(\mathtt{ch}, c_2)}]\!]$

Figure 4.3: Modified Version of Schnorr's Protocol

Lemma 3. *The modified version of Schnorr's protocol is collision resistant.*

Proof. Let $\mathbb{G}$ be a cyclic group with prime order q and generator g. Let x be an ElGamal secret key and let $h = g^x$ be the corresponding ElGamal public key. Suppose by contradiction that there exists a probabilistic polynomial time adversary $\mathcal{A}$ which can find two distinct ElGamal ciphertexts $(c_1, c_2) = (g^r, m \cdot h^r)$ and $(\overline{c_1}, \overline{c_2}) = (g^{\overline{r}}, \overline{m} \cdot h^{\overline{r}})$ for messages m, $\overline{m}$ and random coins r, $\overline{r}$, which verify with the same accepting transcript $(\mathtt{com}, \mathtt{ch}, \mathtt{rsp})$. Then, we get that

$$\mathtt{cmt} \cdot c_1^{\mathsf{H}(\mathtt{ch}, c_2)} = \mathtt{cmt} \cdot \overline{c_1}^{\mathsf{H}(\mathtt{ch}, \overline{c_2})},$$

and hence that

$$c_1^{\mathsf{H}(\mathtt{ch},c_2)} = \overline{c_1}^{\mathsf{H}(\mathtt{ch},\overline{c_2})}.$$

If $c_1 = \overline{c_1}$, then $\mathcal{A}$ has found a collision for H (since H outputs a value in $\mathbb{Z}_q$), so this case is only negligibly probable and hence we can assume that $c_1 \neq \overline{c_1}$. So, we have that $c_1 = g^r$ and $\overline{c_1} = g^{\overline{r}}$ for some $r \neq \overline{r}$, which implies that

$$g^{r \cdot \mathsf{H}(\mathtt{ch},c_2)} = g^{\overline{r} \cdot \mathsf{H}(\mathtt{ch},\overline{c_2})},$$

and hence that

$$r \cdot \mathsf{H}(\mathtt{ch}, c_2) = \overline{r} \cdot \mathsf{H}(\mathtt{ch}, \overline{c_2}) \mod q.$$

At this point, $\mathcal{A}$ can perform a birthday attack to find a collision on the output given by the random oracle. This decreases its bit security by a factor of 2, which contradicts the assumption on the random oracle's bit security. So, no such adversary exists, and hence the modified version of Schnorr's protocol is collision resistant. ■

Completeness, special soundness, and SHVZK of this modified protocol follow immediately from the completeness, special soundness, and SHVZK of the original version of Schnorr's protocol. Furthermore, our modified protocol satisfies commitment entropy, as honestly generated commitments are uniformly random over $\mathbb{Z}_q$, and it is public coin, since the challenges are generated uniformly at random. Lastly, it has quasi-unique responses, by the collision resistance of the hash function.

4.3 Fischlin's Transformation

In [7], Fischlin presents a way to convert Sigma protocols into non-interactive proofs of knowledge in the random oracle model, which support a straight-line extractor. The transformation is described as a pair of algorithms $(\mathcal{P}, \mathcal{V})$ for which we can build a corresponding straight-line extractor $\mathtt{SLE}$. We start by assuming we have access to a Fiat–Shamir proof of knowledge $(\mathcal{P}_{\mathtt{FS}}, \mathcal{V}_{\mathtt{FS}})$ with logarithmic challenge length[1] and corresponding extractor $\mathtt{Ext}$. The idea is to force the prover $\mathcal{P}$ to make multiple hash queries of accepting transcripts, which have the same statement and commitment, but a different challenge and response. This way, since $\mathtt{SLE}$ (i.e., the challenger) has access to all of $\mathcal{P}$'s random oracle queries, it can find a set of valid inputs for $\mathtt{Ext}$, which it can then use to extract the witness. To decrease the soundness error, multiple parallel executions are run for the same statement x, but different commitments. The final proof produced is then the set of transcripts which result from each execution.

In order to force the prover $\mathcal{P}$ to make multiple random oracle queries, the verifier $\mathcal{V}$ imposes the condition that the hash of the final accepting transcript must also be sufficiently

[1] Fischlin notes that one can vary the challenge length of a Fiat–Shamir proof of knowledge to make it logarithmic by running parallel repetitions if the challenge is too short, or by restricting the challenge length to the required length, and that in both these cases, the desired properties are conserved. In our case, the challenge length of the modified Schnorr's protocol needs to be restricted.

small. Since a hash output of this size is unlikely to occur for the first input used, the prover computes consecutive transcripts for a predetermined list of challenge values, and chooses the first transcript that results in the smallest hash output.

We will now describe the transformation. Let λ be a security parameter and let $(\mathcal{P}_{\mathtt{FS}}, \mathcal{V}_{\mathtt{FS}})$ be a Fiat–Shamir proof of knowledge protocol for the relation $\mathcal{R} \subseteq \mathcal{W} \times \mathcal{X}$ with challenges of length $\ell = O(\log(\lambda))$ and distribution Σ. Let b, r, S, and t be functions of λ, where b is the length of the output of the hash function, r is the number of parallel repetitions performed, S is the maximum allowed sum of the hashes of the transcripts resulting from each repetition, and t is the length of the challenges used. We require that $br = \omega(\log(\lambda))$, $2^{t-b} = \omega(\log(\lambda))$, $b, r, t = O(\log(\lambda))$, $S = O(r)$, and $b \leq t \leq \ell$. The Fischlin transformation then defines a non-interactive proof system $(\mathcal{P}, \mathcal{V})$ for relation $\mathcal{L}$, written in the random oracle model.

In all constructions below, the random oracle $\mathcal{O}_{\mathsf{H}}$ can be modelled as an oracle which performs lazy sampling. First, we define the prover as follows:

Prover $\mathcal{P}^{\mathcal{O}_{\mathsf{H}}}(x, w)$

1 : $\mathtt{coins} \leftarrow \emptyset, \mathtt{com} \leftarrow \emptyset, \mathtt{ch} \leftarrow \emptyset, \mathtt{rsp} \leftarrow \emptyset$
2 : **for** $i = 1, \ldots, r$:
3 : $\quad \mathtt{coins}[i] \leftarrow\!\!\$\ \Sigma$
4 : $\quad \mathtt{com}[i] \leftarrow \mathcal{P}_{\mathtt{FS},1}(x, w, \mathtt{coins}[i])$
5 : **for** $i = 1, \ldots, r$:
6 : $\quad \mathtt{ch}[i] \leftarrow 0$
7 : $\quad \mathtt{rsp}[i] \leftarrow 0 \quad /\!/$ this is a dummy value
8 : $\quad \mathtt{minHash} \leftarrow 1^b$
9 : $\quad$ **for** $j = 0, \ldots, 2^t - 1$:
10 : $\quad\quad \mathtt{rsp\_j} \leftarrow \mathcal{P}_{\mathtt{FS},2}(x, w, \mathtt{coins}[i], \mathtt{com}[i], j)$
11 : $\quad\quad$ **if** $\mathcal{O}_{\mathsf{H}}(x, \mathtt{com}, i, j, \mathtt{rsp\_j}) = 0^b$:
12 : $\quad\quad\quad \mathtt{ch}[i] \leftarrow j$
13 : $\quad\quad\quad \mathtt{rsp}[i] \leftarrow \mathtt{rsp\_j}$
14 : $\quad\quad$ **elseif** $\mathcal{O}_{\mathsf{H}}(x, \mathtt{com}, i, \mathtt{ch}[i], \mathtt{rsp}[i]) < \mathtt{hashValue}$:
15 : $\quad\quad\quad \mathtt{ch}[i] \leftarrow j$
16 : $\quad\quad\quad \mathtt{rsp}[i] \leftarrow \mathtt{rsp\_j}$
17 : $\quad\quad\quad \mathtt{hashValue} \leftarrow \mathcal{O}_{\mathsf{H}}(x, \mathtt{com}, i, \mathtt{ch}[i], \mathtt{rsp\_j})$
18 : **return** $(\mathtt{com}[i], \mathtt{ch}[i], \mathtt{rsp}[i])_{i=1,\ldots,r}$

Figure 4.4: Fischlin Prover

It is worth noting that the purpose of the dummy value on line 7 in $\mathcal{P}$ is to simplify the algorithm and prevent the prover from trying to return a null value in the case that all hash

queries evaluate to 1^b. This will only happen with negligible probability, and in this case, the verification algorithm will fail regardless of whether the transcript is accepting, since the hash value of this transcript will exceed the value S.

Next, we define the verifier as follows:

Verifier $\mathcal{V}^{\mathcal{O}_{\mathsf{H}}}(x, (\mathtt{com}[i], \mathtt{ch}[i], \mathtt{rsp}[i])_{i=1,\ldots,r})$

1 : **for** $i = 1, \ldots, r$:
2 : **if** $\mathcal{V}_{\mathsf{FS},2}(x, (\mathtt{com}[i], \mathtt{ch}[i], \mathtt{rsp}[i])) = \mathsf{false}$:
3 : **return** false
4 : **if** $\sum_{i=1}^{r} \mathcal{O}_{\mathsf{H}}(x, \mathtt{com}, i, \mathtt{ch}[i], \mathtt{rsp}[i]) > S$:
5 : **return** false
6 : **return** true

Figure 4.5: Fischlin Verifier

The straight-line extractor for the above construction can then be defined as follows:

Straight-line extractor $\mathtt{SLE}(x, (\mathtt{com}[i], \mathtt{ch}[i], \mathtt{rsp}[i])_{i=1,\ldots,r})$

1 : $\pi_1 \leftarrow$ **null**
2 : $\pi_2 \leftarrow$ **null**
3 : **for** $i = 1, \ldots, r$:
4 : $\pi_1 \leftarrow (x, \mathtt{com}[i], \mathtt{ch}[i], \mathtt{rsp}[\mathtt{i}])$
5 : **if** $\exists (x, \mathtt{com}, i, \mathtt{ch}_j, \mathtt{rsp}_j) \in \mathtt{HList}$ s.t. $\mathtt{ch}[j] \neq \mathtt{ch}[i]$:
6 : $\pi_2 \leftarrow (x, \mathtt{com}[i], \mathtt{ch}_j, \mathtt{rsp}_j)$
7 : `break`
8 : **return** $\mathtt{Ext}(x, \pi_1, \pi_2)$

Figure 4.6: Fischlin Straight-line Extractor

Fischlin proves that his construction satisfies a stronger version of straight-line extractability than what we require here; namely, he shows that his staight-line extractor can produce a witness for any valid statement proof pair (x, π), not just those where $x \in \mathcal{L}$. In addition to implying our definition of straight-line extractability, Fischlin's definition also implies soundness. Furthermore, Fischlin proves that his construction satisfies completeness and zero knowledge.

It remains to show that Fischlin's transformation satisfies collision resistance and non-malleability.

Lemma 4. *An instantiation of Fischlin's transformation is collision resistant if the underlying Fiat–Shamir proof of knowledge is collision resistant.*

Proof. Suppose by contradiction that there exists a probabilistic polynomial time adversary $\mathcal{A}$ who is able to find two statements x and x' that verify with the same proof $(\texttt{com}[i], \texttt{ch}[i], \texttt{rsp}[i])_{i=1,\ldots,r}$. Then, for each i, $(\texttt{com}[i], \texttt{ch}[i], \texttt{rsp}[i])$ is an accepting transcript for both x and x' for the underlying Fiat–Shamir proof of knowledge. But this contradicts the assumption that the underlying Fiat–Shamir proof of knowledge is collision resistant. ■

We proved in the previous section that the modified version of Schnorr's protocol is collision resistant, so by lemma 4, the proof system we obtain by applying Fischlin's transformation to this protocol is collision resistant.

Lemma 5. *The proof system that results from applying Fischlin's transformation to a Fiat–Shamir proof of knowledge is non-malleable.*

Proof. Suppose that there exists a probabilistic polynomial time adversary $\mathcal{A}$ who, given a valid proof $(\texttt{com}[i], \texttt{ch}[i], \texttt{rsp}[i])_{i=1,\ldots,r}$ for a statement $x = f(y)$ for a value y of their choosing, can produce a different valid proof $(\texttt{com}'[i], \texttt{ch}'[i], \texttt{rsp}'[i])_{i=1,\ldots,r}$ for the same statement x.

First, suppose that there exist indices i and j such that $\texttt{com}[i] = \texttt{com}'[j]$, $\texttt{ch}[i] = \texttt{ch}'[j]$, but $\texttt{rsp}[i] \neq \texttt{rsp}'[j]$. This is impossible by the assumption that the underlying Sigma protocol has quasi-unique responses, so we may assume that this is not the case.

Now, suppose that there exist indices i and j such that $\texttt{com}[i] = \texttt{com}'[j]$, but $\texttt{ch}[i] \neq \texttt{ch}'[j]$. Then $\mathcal{A}$ can use the extractor for the underlying Sigma protocol on these two proofs in order to compute the witness. But this contradicts the assumption that Fischlin's transformation is zero knowledge, so we may assume that this is not the case.

So, we must have that, for all indices i and j, $\texttt{com}[i] \neq \texttt{com}[j]$. As our hash function is modeled as a random oracle whose output is required to verify the proof, this means that the proof $(\texttt{com}'[i], \texttt{ch}'[i], \texttt{rsp}'[i])_{i=1,\ldots,r}$ is entirely independent of the original proof $(\texttt{com}[i], \texttt{ch}[i], \texttt{rsp}[i])_{i=1,\ldots,r}$. So, $\mathcal{A}$ contains a subroutine $\mathcal{M}$ which, without being given a valid proof of x, can produce a valid proof of x. Thus, the non-malleability definition has been satisfied. ■

So, applying Fischlin's transformation to our modified version of Schnorr's protocol results in a proof system satisfying the requirements of the PKEZK construction.

Chapter 5

Conclusion

In this work, we presented a new approach for transforming an IND-CPA-secure PKE into an IND-CCA-secure PKE, which has a tight security reduction and does not involve derandomization or re-encryption. We provided a security proof in the random oracle model for the generic construction, and provided an instantiation that builds upon existing protocols.

5.1 Comparison with Existing Work

While our transformation does not involve a re-encryption step and provides tight security, it is less efficient and results in larger ciphertexts than pre-existing constructions. The table below provides a comparison between our transform and some of the more popular pre-existing transforms: FO and REACT [8, 12]. We do not compare directly to the modularized transforms in [10] as they result in KEMs instead of PKEs. The size and efficiency measurements are done assuming ElGamal is used as the underlying PKE. Ciphertext size is given assuming a λ-bit modulus, and run time is given in terms exponentiations that need to be performed, as this is the dominating operation. For the `PKEZK` row, the number of repetitions, r and the challenge bitlength, t are as defined in Chapter 4.

	PKEZK	FO	REACT	$FO^{\not\perp}$
Underlying Enc security	IND-CPA	OW-CPA	OW-PCA	IND-CPA
Underlying Enc correctness	ϵ	Perfect	Perfect	ϵ
Uses re-encryption	No	Yes	No	Yes
Uses derandomization	No	Yes	Yes	Yes
Tight?	Yes	No	Yes	Yes
Resulting scheme	PKE	PKE	PKE	KEM
Ciphertext size	$3r + 1$	3	≈ 3	≈ 2
Run time (Encryption)	$r(2^t - 1) + 2$	2	2	2
Run time (Decryption)	$r + 1$	2	1	2

Table 5.1: Comparision of various IND-CPA-to-IND-CCA transforms, where ciphertext size is given in terms of a λ-bit modulus and run time is given in terms of the number of modular exponentiations

Of all the transformations described in [10], we have chosen to include $FO^{\not\perp}$ in our comparison, as it is an IND-CPA-to-IND-CCA transform and does not require the underlying PKE to be γ-spread. It is worth noting that the second module of this transform, $U^{\not\perp}$, is the KEM version of REACT.

The ciphertext sizes of REACT and $FO^{\not\perp}$ are shown as $\approx$3 and $\approx$2, respectively, because they are dependent on the output length of the hash function H described in Chapter 1. It is reasonable to assume that this is approximately equal to λ. Also, it is worth noting that in practice, the values of s and t would be in the ballpark of 10 and 12, respectively [7].

The size and efficiency drawbacks of our instantiation come from the zero knowledge proof system used. To the best of our knowledge, there do not currently exist straight-line extractable zero knowledge proof systems that are smaller and/or more efficient, however, if one is constructed in the future, then this could improve our instantiation.

5.2 Future Work

Our construction arose from a few observations:

1. The random coins used during ElGamal encryption can be used for decryption purposes.
2. Attaching a zero knowledge proof of knowledge of the random coins used for ElGamal encryption to an ElGamal ciphertext forces plaintext awareness from any party who is capable of creating a well-formed ciphertext.
3. Schnorr's protocol is compatible with ElGamal as a zero knowledge proof of knowledge of the random coins used during ElGamal encryption.

With this instantiation in mind, we came up with the generic transform which forces plaintext awareness by proving knowledge of the random coins used. However, a similar generic transform could be obtained by proving knowledge of the message instead of proving knowledge of the radom coins. This transform would be more widely applicable, since it would not require the the underlying PKE to be decryptable with randomness. It would also admit a simpler security proof, for the same reason. However, to our knowledge, there are no obvious ways to instantiate such a transformation, so its interest is more theoretical than practical. However, this provides motivation to explore efficient non-interactive straight-line extractable zero knowledge proof protocols that can be used for this purpose. Similarly, development of smaller and faster straight-line extractable zero-knowledge proofs of knowledge for discrete logarithms are of interest, to improve the size and efficiency of our instantiation.

Additionally, with the looming possibility of quantum computers and the resulting interest in post-quantum secure cryptosystems, it could be useful to have an IND-CPA-to-IND-CCA transformation that is secure in the quantum random oracle model. The instantiation we provided is not, as it relies on the hardness of the discrete logarithm problem, but we have not yet analyzed our generic transformation under this lens. This could also be an avenue for future work.

References

[1] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[2] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[3] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2023.

[4] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 127–145, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[5] Jean-Sébastien Coron, Helena Handschuh, Marc Joye, Pascal Paillier, David Pointcheval, and Christophe Tymen. GEM: A generic chosen-ciphertext secure encryption method. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 263–276, San Jose, CA, USA, February 18–22, 2002. Springer, Heidelberg, Germany.

[6] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[7] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[8] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[9] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 359–386, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[10] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[11] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[12] Tatsuaki Okamoto and David Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany.

[13] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 543–553, 1999.

[14] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[15] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332, 2004. https://eprint.iacr.org/2004/332.

[16] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):296–322, Nov. 2021.