# On a Canonical and Systematic Approach to Cryptographic Subversion

Master of Science

in

Mathematics

Torben Brandt Hansen, 20091714

August 29, 2015

Supervisors: Claudio Orlandi and Johan P. Hansen

AARHUS UNIVERSITY
DEPARTMENT OF MATHEMATICS

English title: On a canonical and systematic approach to cryptographic subversion
Danish title: En kanonisk og systematisk tilgang til konstruktion af kryptografiske bagdøre

Typeset using LaTeX and the memoir document class.

## Abstract

We present a novel approach to the topic of cryptographic subversion by introducing the technique of *canonical* cryptographic subversion. We abstract the idea of a cryptographic game in which a notion of subvertability is defined. In the Random Oracle model, we present two canonical cryptographic subversions in the shape of the CDH and DDH hardness assumptions. The CDH subversion is then used to construct a subversion, in the Random Oracle Model, of the El-Gamal public-key encryption scheme. In connection with introducing the technique of canonical cryptographic subversion, we also define a definitional framework that unify the theory of cryptographic subversion in a common body of language. A result by Bellare, Paterson and Rogaway is recovered in our new framework that affirms this.

## Resumé

Vi introducerer en ny tilgang til konstruktionen af kryptografiske bagdøre ved at indføre *kanoniske* kryptografiske bagdøre. Vi generaliserer ideen om et kryptografisk spil hvortil vi vil indføre et kryptografisk bagdørsbegreb. I Random Oracle Modellen konstruerer vi to kanoniske kryptografiske bagdøre i form af to hårdhedsantagelser: CDH og DDH. CDH bagdøren bliver efterfølgende brugt til at konstruere, i Random Oracle Modellen, en bagdør for El-Gamal offentlig nøgle kryptering. I forbindelse med definitionen af kanoniske kryptografiske bagdøre definerer vi også en definitions-ramme, der indkapsler teorien om kryptografiske bagdøre i et fælles sprog. Som et eksempel konstruerer vi en tidligere beskrevet bagdør i vores nye mængde af definitioner.

**Acknowledgements**

I would like to use this opportunity to thank my two supervisors. First, Claudio Orlandi for his guidance, many fruitful discussions and a lot of fun. His patience and knowledge has been indispensable. Second, also many thanks to Johan P. Hansen who stepped in as formal supervisor even though the topic of this thesis is not his speciality.

I am in debt to Anders and Elpiniki who both proofread a draft of this thesis with $\varepsilon$-notice. They managed to cryptographically hide my many silly grammar mistakes.

# Contents

Chapter 1

# Introduction

The use of cryptographic algorithms as black box devices/software is a common practice. Internet browsers, mobile phones and computers in general all use cryptography in a way that is hidden to the user. That is, the implementation is trusted to behave as specified. For tamper-proof devices such as HSMs it is even impossible to scrutinise the code and the inner workings of the device. This prevents the discovery of potential malicious content embedded in the black box device. The norm among cryptographers is to assume that such malicious behaviour never happens. This assumption is potentially dangerous. Specifically because it is often the case that e.g. crucial cryptographic key management functions are implemented in hardware and companies produce commercial software implementations of cryptographic systems with strictly proprietary source code. In our modern era, billions of people implicitly trust cryptographic implementations: when browsing on secure websites or paying with a credit card in the local store. Nation wide PKI systems has been set up in many countries (that the population must trust in their day to day life). Systems such as NemID [1], the National Danish PKI, require users to trust software from a third party provider and works as a trust anchor for the population towards trusting digital communication with public institutions, banks etc. Only very IT inclined users has the possibility to inspect this implementation, which is heavily obfuscated [23] due to proprietary concerns. For less inclined users this software is effectively a black-box.

In light of the recent revelations by Edward Snowden [20, 24] it is clear that powerful actors will go to remarkable lengths in terms of both morale and finance, to obtain secret information. The National Security Agency (NSA) has for example (allegedly) tampered with public standards [19], a tampering that allows practical exploitation [13]. NSA even went so far as intercepting hardware deliveries to insert back doors before they reached the intended customer [19]. These kind of adversaries with almost unlimited resources and determination has lately obtained their own name in the security industry: Advanced Persistent Threats (APTs). They are believed to impose significant threats against financial and physical infrastructures as well as human rights.

The sheer complexity of modern cryptographic implementations today pose a serious risk, leaving many users vulnerable to simple but devastating attacks [2, 4, 5, 6]. The extreme complexity makes it a very difficult task for experts, and in turn an impossible task for users, to detect anomalies and vulnerabilities in hardware and software. Therefore, deliberately inserted back doors in software or hardware is extremely hard to detect. Competitions have been created [3], which paints a clear picture on this particular matter.

The current state of research into cryptographic subversions is sparse and has only recently been kick-started [8]. In light of this and facing the reality of real world subversion attacks, we ask the following important question: *Which schemes are susceptible to subversion attacks?* Reading this thesis gives the incomplete answer: *Many!*

**Prior work**

The study of cryptographic subversion was initiated by Simmons [26, 25] who defined the notation of a subliminal channel. Simmons phrase his work as a story involving two prisoners, Alice and Bob, and a warden. It is the goal of Alice and Bob to communicate in a secret way while it is the wardens job to prevent this communication. A long series of work followed this general theme [15, 11, 10, 12]. In the late 90s Young and Yung elaborated on this work and began a line of research involving among other topics kleptography [27, 28, 30, 29]. Young and Yung considered cryptographic algorithms in which they embed a public key but kept the private key for themselves. Using this model they were able to construct schemes that leaks information subliminally. Their model of using asymmetric keys is similar to ours: the compromised implementation carries the attacker's public key, but a private key is needed to read the subliminal channel.

In the realm of NSA, new work has been surfacing that tries to address the challenge of mass surveillance in a cryptographic way. In 2014 Bellare, Paterson and Rogaway [8] presented a new and rigorous approach to cryptographic subversion. They define the notion of an algorithm-substitution attack (ASA) and observe that modern standards for symmetric encryption crucially rely on sender-chosen randomness to attain acceptable security levels. Since implementations do not have a way to verify the sender-chosen randomness it enables a communication channel, which can be used to leak secret information. Our subversion-definitions mainly arise from ideas of this work. Recent papers has copied the ideas of Bellare, Paterson and Rogaway, and define similar notions for other areas of cryptography [17, 7, 16].

**Cryptographic game**

The notion of security for a scheme is often modelled by a game. Familiar notions such as CPA security of a symmetric (or asymmetric) encryption scheme and DDH hardness relative to a group are fundamental in cryptography. They can all be formulated as a cryptographic game in which their notion of security is quantified. There are interesting connections between CPA and DDH: to prove CPA security for certain schemes e.g. the El-Gamal public-key encryption scheme, one needs to do a reduction to DDH hardness of the underlying group instance. In other words, the game defining CPA security relates to the game defining DDH hardness. In this sense the DDH hardness game is canonical: a basic game/assumption that can be used to construct lower-level schemes.

The CPA security game and DDH hardness game are different in the sense that the former is an interactive game while the latter is a non-interactive game. To capture both games, we therefore need to produce two set of definitions: one for the non-interactive cryptographic games and one for the interactive cryptographic games. In Section 3.3 we abstract the idea of a cryptographic game and build a definitional framework where both interactive and non-interactive cryptographic game can be formulated.

**Subversion**

Cryptographic subversion is the study of subverting cryptographic algorithms. In a cryptographic subversion attack, an attacker aims to subvert an algorithm in such a way that the subversion remains unnoticed but the subverted algorithm simultaneously leaks secret information through its output. In such an attack, an attacker is in some way using cryptography against cryptography. In this respect, a cryptographic subversion is the proof that even though we can do great things with crypto, it can also prevent us from foiling certain attacks. While some cryptographers seemed to had dismissed real world subversion attacks as far-fetched, recent revelations suggest this attitude to be naive [20]. Real world cryptographic subversion may well be going on today, possibly on a massive scale.

We wish to define cryptographic subversion with respect to a cryptographic game. This gives us some advantages over definitions appearing in prior work. On the conceptual plane it will allow us to construct what we will call *canonical cryptographic subversions* (see next paragraph). On the technical level it gives us the possibility of defining the properties of a subversion in a coherent and compact manner. Hopefully this will help emphasise the actual properties of a subversion and the properties we really want.

First task is to define big brother $\mathsf{B}$. Big brother is the modifier and thus the entity responsible of constructing the subversion. A big brother is defined with respect to a specific cryptographic game and to be successful in constructing the subversion $\mathsf{B}$ must satisfy three requirements: recoverability, undetectability and preservability. First requirement secures that $\mathsf{B}$ can recover the leaked information; second requirement makes sure that $\mathsf{B}$ can not be detected; last requirement ensures that the subverted scheme satisfy at least the same notion of security as the non-subverted scheme. In Section 4.2 it is proven that the last requirement is actually redundant.

Previous work has only defined cryptographic subversions of one strength. We expand this and define both a weak and a strong form of cryptographic subversion. This makes it possible to deeper explore the state of possible subversions.

**Canonical subversion**

In this thesis we will develop and explore a new approach to cryptographic subversion. Recent work [7, 16, 8, 17] has dealt with cryptographic subversions on a low-level by building their own separate definitional frameworks. As was mentioned above, cryptographic schemes are often proven secure with respect to some security game by constructing a reduction to an underlying (canonical) hardness assumption. We want to carry the idea of canonical assumptions for proving security of schemes to the topic of cryptographic subversion. Our new approach is to define and investigate a new technique, canonical cryptographic subversion, which can be used to easily construct (and formally prove success of) subversions of low-level schemes e.g. construct a subversion for $\mathsf{CDH}$ that can be used to subvert $\mathsf{El\text{-}Gamal}$.

There is a hierarchy in the usual hardness assumptions in the sense that $\mathsf{DDH}$ hardness implies $\mathsf{CDH}$ hardness, which in turn implies $\mathsf{DL}$ hardness. It turns out that there also exists a hierarchy in canonical cryptographic subversions but turned up-side-down: a subversion of $\mathsf{CDH}$ implies a subversion of $\mathsf{DDH}$. Therefore, it is a stronger requirement to assume that $\mathsf{CDH}$ can be subverted. In contrast it is a stronger assumption to assume $\mathsf{DDH}$ is hard than $\mathsf{CDH}$ is hard. We present (in the Random Oracle Model) two canonical cryptographic subversions in Chapter 5

**Systematic approach**

A secondary aim of this thesis is to propose a systematic approach to the study of cryptographic subversion. In particular, we propose a common body of language to be used when describing subversions, defined in the context of cryptographic games. In addition this provides a common set of definitions that completely characterise the properties that are desired for a successful subversion (these were presented above). In Chapter 7 it is shown that our framework indeed enables the recovery of previous results. Namely, we recover a result from [8] that shows that all IV-surfacing, probabilistic and stateless symmetric encryption schemes are subversion prone. In Chapter 6 a result from [27] is recovered that highlight the subvertability of the El-Gamal public-key encryption scheme but constructed by making use of a canonical cryptographic subversion.

Furthermore our approach also provides a systematic method to analyse cryptographic subversions. Out definitions include the concept of a parameter function together, which together with the definition of two different subversions strength gives a degree of agility that can be used to analyse different scenarios and (hopefully) yield greater insight into the state of cryptographic subversion.

**Contributions**

We propose a new framework to the study of cryptographic subversion that encapsulate previous work into a combined set of definitions. Moreover, we construct, in the Random Oracle Model, canonical subversions, that are subversions of hardness assumptions. These subversions are used to construct a subversion of the El-Gamal public-key encryption scheme. Figure 1.1 below contains a schematic view of the subversions presented in this thesis.

| | weak subversion | strong subversion | ROM |
|---|---|---|---|
| **Hardness assumption** | | | |
| CDH | ✓ | ✓ | ✓ |
| DDH | ✓ | ✓ | ✓ |
| **Public-key encryption** | | | |
| El-Gamal | ✓ | ✓ | ✓ |
| **Symmetric encryption** | | | |
| IV-surfacing & probabilistic & stateless | ✓ | ✓ | ✗ |

**Figure 1.1:** Overview of hardness assumptions and schemes that are subverted in this thesis.

The table shows positive results with respect to big brother. A conclusion on the basis of this is that a big brother can achieve mass surveillance if there are any green check marks. ROM stands for *Random Oracle Model*, which means that the construction of the subversion utilises the Random Oracle Model.

By establishing a common definitional framework we provide a set of definitions to unify the theory of cryptographic subversion where previous subversions can also be formulated. As a side effect, it brings old work [27] into a rigorous condition that is on par with nowadays cryptography standards.

We take an overly negative approach, in the sense that we are not much interested in constructing schemes that are subversion resistant but only to show that a subset of *known* schemes are vulnerable to subversion attacks. Even though we do not provide constructions of schemes secure against subversion attacks, we believe that mapping the field of possible subversions in a consistent and rigorous manner using the same body of language, aids the awareness of the dangers that such attacks impose and thereby paying the way for further work on subversion resistant schemes.

**Summary**

*Chapter 2* covers a number of preliminaries, which will be needed for later chapters. We fix the common notation and conventions that is used throughout. The notion of pseudo-random functions and block ciphers are defined. We define and discuss indistinguishability of probability ensembles. The different hardness assumptions needed in the sequel are presented and their connections are discussed and proved. A general definition of public-key encryption and symmetric encryption is provided together with their security definitions. Last, we discuss the Random Oracle Model.

*Chapter 3* begins by discussing the requirements of a cryptographic game as well as motivations. We then introduce the definition of a non-interactive cryptographic game and an interactive cryptographic game.

*Chapter 4* first presents the definition of a big brother, which leads to the introduction of the definition of a subversion of a non-interactive cryptographic game and a subversion of an interactive cryptographic game. For each notion, results on general properties are stated and proved. In addition, several examples are presented that elaborate on the definitions.

*Chapter 5* presents two interactive cryptographic games in the shape of the CDH and DDH hardness assumption. A strong subversion of both games are constructed in the Random Oracle Model. In addition, we discuss the relation between the two subversions.

*Chapter 6* use the subversions presented in Chapter 5 to subvert the El-Gamal public-key encryption scheme.

*Chapter 7* first defines a symmetric big brother. The chapter then continues to state and prove a result from [8] in our new framework.

Torben Hansen, August 29, 2015.

*I do not want to live in a world where everything I do and say is recorded.*
*That is not something I am willing to support or live under.*
— Edward Snowden(1983–)

CHAPTER 2

# Preliminaries

In this chapter we will treat tools used in the following chapters. Many of these tools should be familiar to the reader. We describe them anyway to fix formal notation and thereby have a common body of language. The chapter is partially based on [21] and partially on [18].

The chapter is organised as follows: in Section 2.1, we state the conventions and fix notation. In the next section, we discuss basic tools used in cryptography. Pseudo-random functions are discussed in Section 2.3. Section 2.4 defines block ciphers. In Section 2.5 we define DL flavoured hardness assumptions, and state and prove the connection between these as well. In the following two sections, we define respectively, public-key and symmetric encryption schemes. We end with a discussion of the Random Oracle Model.

## 2.1 Conventions and Notation

Here we describe common notation used throughout. Conventions that are used implicitly are also mentioned.

### 2.1.1 Conventions

We use the following conventions, often without explicitly mentioning them.

- A probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ is a probabilistic algorithm which always (i.e. independent of the outcome of its internal coin tosses) halts after a polynomial (in the length of the input) number of steps. We write $a \leftarrow \mathcal{A}(x)$ to denote running $\mathcal{A}$ on input $x \in \{0,1\}^*$, producing output $a$.
- A $l$-bit number is an integer in the range $\left[2^{l-1}, 2^l\right)$.
- All parties considered have maximal knowledge of each others algorithms. That is, we enforce Kerckhoffs's principle for all parties involved.

### 2.1.2 Notation

We fix the following notation.

- $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Z}_q$ denotes respectively the natural numbers, the integers and the set of residue classes modulo $q$. As usual we will associate $\mathbb{Z}_q = \{0, 1, \dots, q-1\}$.

- $n \in \mathbb{N}$ denotes the security parameter.
- $|q|$ denotes the bit-length for a bit-number $q$ i.e. $|q| := 1 + \lfloor \log_2(q) \rfloor$.
- $\perp$ denotes the empty string.
- $\mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\ldots}$ indicates that $\mathcal{A}$ has oracle access to $\mathcal{O}_1, \mathcal{O}_2, \ldots$.
- $\mathcal{R}$ denotes a random oracle (see Section 2.8).
- $x \leftarrow_\chi S$ denotes sampling of an element from a (finite) set $S$ from distribution $\chi$. In the case where $\chi$ is the uniform distribution we write $x \leftarrow_R S$.

## 2.2 Negligible Functions, Probability Ensembles and Indistinguishability

In this thesis we are concerned with computational security and the most basic tools required are defined in this section. These represent absolute basic tools in modern cryptography.

In our work, we will consider PPT adversaries and their success probability in completing specific tasks. Most often this probability should be very small. More precisely the probability should be smaller than all inverse polynomials, also called *negligible*. We precisely define the term below.

**Definition 2.2.1.** (**Negligible function**) We call a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ negligible if for every *positive* polynomial poly there exists an $N \in \mathbb{N}$ such that

$$\mathsf{negl}(n) < \frac{1}{\mathrm{poly}(n)}, \quad \forall n > N.$$

Standard examples of negligible functions are $2^n$, $2^{-\sqrt{n}}$ and $n^{-\log_2(n)}$. The perk of working with negligible functions is that they satisfy desirable closure properties: the sum of two negligible functions is negligible and the product of a negligible function and a polynomial is again negligible. This will become handy later.

When defining a game, we often need the notion of computational indistinguishability. More precisely, we need to define when two distributions are computational indistinguishable. In doing this, we follow the standard way: two distributions are called computational indistinguishable if no PPT distinguisher $\mathcal{D}$ can tell them apart. The formulation of that requires two infinite sequences of distributions. Such sequences are called probability ensembles.

**Definition 2.2.2.** (**Probability ensemble**) A probability ensemble indexed by $\mathbb{N}$ is a sequence of random variables indexed by $\mathbb{N}$.

The index parameter will always be the security parameter $n$ but we will never write the index parameter because it will be clear from the context. Below, we define computational indistinguishability of two probability ensembles.

**Definition 2.2.3.** (**Computational indistinguishability**) Two probability ensembles $\{X_n\}_n$ and $\{Y_n\}_n$ are computational indistinguishable if there for every PPT distinguisher $\mathcal{D}$ exists a negligible function negl such that

$$|\Pr[\mathsf{true} \leftarrow \mathcal{D}(X_n, n)] - \Pr[\mathsf{true} \leftarrow \mathcal{D}(Y_n, n)]| \leq \mathsf{negl}(n),$$

where the probability is taken over the random variables $X_i$ or $Y_i$ and the internal randomness used by $\mathcal{D}$.

In this thesis, the probability ensembles will in general be represented by random experiments, which we will call games.

## 2.3 Pseudo-Random Function

Consider a keyed function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, where the first input is called the key and denoted by $\mathsf{k}$. For a specific key $\mathsf{k}$, we write $F_{\mathsf{k}}(x) = F(\mathsf{k}, x)$ for any $x \in \{0,1\}^*$. If there exists a polynomial-time algorithm that computes $F_{\mathsf{k}}(x)$ for any key $\mathsf{k}$ and input $x$, $F$ is called efficiently computable.

The security parameter $n$ determines the key length, input length and output length. We therefore associate three functions $l_{in}(n)$, $l_{out}(n)$ and $l_{key}(n)$. For any key $\mathsf{k} \in \{0,1\}^{l_{key}(n)}$ the function $F_{\mathsf{k}}$ is only defined for input $x \in \{0,1\}^{l_{in}(n)}$ and output $F_{\mathsf{k}} \in \{0,1\}^{l_{out}(n)}$. We will however only deal with *lengh-preserving* functions, which means that $l_{in}(n) = l_{out}(n) = l_{key}(n) = n$. Hence, for each key $\mathsf{k} \in \{0,1\}^n$, we obtain a function $F_{\mathsf{k}}$ mapping $n$-bit strings to $n$-bit strings. Let $\mathsf{Func}_n$ be that set of all functions that maps $n$-bit strings to $n$-bit strings. Note $|\mathsf{Func}_n| = 2^{2^n \cdot n}$. Below we formally define a pseudo-random function.

**Definition 2.3.1.** (**Pseudo-random function**) Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficiently computable, length-preserving and keyed function. $F$ is a pseudo-random function if there for every PPT distingusher $\mathcal{D}$ exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_F^{\mathsf{PRF}}(\mathcal{D}) := \left| \Pr\left[\mathsf{true} \leftarrow \mathcal{D}^{F_k(\cdot)}(n)\right] - \Pr\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n)\right] \right| \le \mathsf{negl}(n),$$

where the first probability is over $k \leftarrow_R \{0,1\}^n$ and randomness of $\mathcal{D}$, and the second probability is over $f \leftarrow_R \mathsf{Func}_n$ and randomness of $\mathcal{D}$.

Note that the distinguisher can interact freely with its oracle i.e. ask queries adaptively. Also note that the distinguisher must not know the key $\mathsf{k}$ otherwise it would be trivial to distinguish $F$ and $f$. Requiring $F$ to be length-preserving is not an inherent part of PRFs but it simplifies applications.

## 2.4 Block Cipher

Block ciphers are essential tools for shared-key cryptography and are used in a plethora of schemes. There are two very well known real-world instantiations of a block cipher: DES and AES. The former is the oldest and is being gradually phased out. The latter is the newest and most modern block cipher, which should be used for new practical applications.

To define a block cipher, we must first introduce a little notation. Given a length-preserving, keyed function $F$. We call $F$ a keyed permutation if $F_{\mathsf{k}}$ is bijective for every key $\mathsf{k} \in \{0,1\}^n$. We call $n$ the block length. We call a keyed function $F$ efficient if it is both efficiently computable and efficiently invertible for every key $\mathsf{k}$.

**Definition 2.4.1.** (**Blockcipher**) A block cipher is an efficient, length-preserving, keyed permutation denoted by $\mathsf{Enc}^{\mathsf{block}} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.

In practice, block ciphers are constructed as secure instantiations of (strong) pseudo-random functions with some fixed key length. Strong pseudo-random is a stronger assumption than pseudo-random Def. 2.3.1 (but that kinda follows obviously from the suffix strong...). Let $\mathsf{Perm}_n$ be the set of all permutations on $\{0,1\}^n$. Note $|\mathsf{Perm}_n| = (2^n)!$.

**Definition 2.4.2.** (**Strong pseudo-random function**) An efficient, length-preserving and keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is as strong pseudo-random function if there for every PPT distinguisher $\mathcal{D}$ exists a negligible function $\mathsf{negl}$ such that

$$\left| \Pr\left[\mathsf{true} \leftarrow \mathcal{D}^{F_k(\cdot),F_k^{-1}(\cdot)}(n)\right] - \Pr\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot),f^{-1}(\cdot)}(n)\right] \right| \leq \mathsf{negl}(n),$$

where the first probability is also taken over the randomness used by $\mathcal{D}$ and $k \leftarrow_R \{0,1\}^n$, and the second probability is taken over the randomness used by $\mathcal{D}$ and $f \leftarrow_R \mathsf{Perm}_n$.

## 2.5 Hardness Assumption

In this section, we introduce a number of hardness assumptions that can be defined in any class of cyclic groups. To not be concerned with how the group is generated (especially the group description), we let $\mathcal{G}$ denote a generic, polynomial-time group-generation algorithm. This algorithm takes input $1^n$ and outputs a description[1] of a cyclic group $\mathbb{G}$, its order $q$ with $|q| = n$, and a generator $g \in \mathbb{G}$. It is implicitly assumed that there exists polynomial-time algorithms (in $n$) for computing the group operation in $\mathbb{G}$ and testing whether a given bit-string represents an element in $\mathbb{G}$.

We now proceed to define three different hardness assumptions: DL, CDH and DDH. Afterwards it is discussed how they related to each other. First the DL hardness game is presented.

**Definition 2.5.1.** (**Discrete logarithm problem**) Let $\mathcal{G}(1^n)$ be a polynomial-time algorithm that on input $1^n$ outputs a cyclic group $\mathbb{G}$ of prime order $q$, $|q| = n$ and generator $g$. The DL problem is (informally) to compute $x$ given $g^x$ with randomly chosen $x$ from $\mathbb{Z}_q$. Formally, the DL *problem is hard relative to* $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$ there exist a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DL}}(\mathcal{A}) := \Pr\left[\mathsf{true} \leftarrow \mathsf{DL}_{\mathcal{G}}^{\mathcal{A}}(n)\right] \leq \mathsf{negl}(n),$$

where the probability is taken over the randomness used the game $\mathsf{DL}_{\mathcal{G}}^{\mathcal{A}}(n)$ defined in Figure 2.1.

```
Game $\mathsf{DL}_{\mathcal{G}}^{\mathcal{A}}(n)$
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    $x \leftarrow_R \mathbb{Z}_q$
    $h := g^x$
    $h' \leftarrow \mathcal{A}(\mathbb{G}, q, g, h)$
    Return $(h \overset{?}{=} h')$
```

**Figure 2.1:** DL hardness game.

The above assumption simply states that there exists a polynomial-time algorithm $\mathcal{G}$ for which computing the discrete logarithm is hard.

Next we will define two different Diffie-Hellman problems. These also relate to the same kind of group-generation algorithm as before but are stronger assumptions than the DL assumption. We first present the CDH hardness game.

[1]Even though all cyclic groups of a given order are isomorphic, the description of the group determines the computational complexity of doing operations.

**Definition 2.5.2.** **(Computational Diffie-Hellman problem)** Let $\mathcal{G}(1^n)$ be a polynomial-time algorithm that on input $1^n$ outputs a cyclic group $\mathbb{G}$ of prime order $q$, $|q| = n$ and generator $g$. The CDH problem is (informally) to compute $g^{xy}$ given $g^x$ and $g^y$ with randomly chosen $x, y$ from $\mathbb{Z}_q$. Formally, the CDH *problem is hard relative to* $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$ there exist a negligible function negl such that

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{A}) := \Pr\left[\mathsf{true} \leftarrow \mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)\right] \leq \mathsf{negl}(n),$$

where the probability is taken over the randomness used in the game $\mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ defined in Figure 2.2.

In Propositon 2.5.4 we prove that if CDH problem is hard relative to $\mathcal{G}$ then DL is also hard relative to $\mathcal{G}$. It is not known whether the two assumptions are equivalent. The last hardness assumption we present is the DDH hardness game.

**Definition 2.5.3.** **(Decisional Diffie-Hellman problem)** Let $\mathcal{G}(1^n)$ be a polynomial-time algorithm that on $1^n$ outputs a cyclic group $\mathsf{G}$ of prime order $q$, $|q| = n$ and generator $g$. The DDH problem is (informally) to distinguish $g^{xy}$ from $g^z$ with randomly chosen $x, y, z$ from $\mathbb{Z}_q$. Formally, the DDH *problem is hard relative to* $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DDH}}(\mathcal{A}) := 2 \cdot \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{DDH}_{\mathcal{G}}^{\mathcal{A}}(n)\right] - \frac{1}{2}\right| \leq \mathsf{negl}(n),$$

where the probability is taken over the randomness used in the game $\mathsf{DDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ defined in Figure 2.2.

The DDH assumption is equivalent to the following: for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$|\Pr[\mathsf{true} \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z)] - \Pr[\mathsf{true} \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy})]| \leq \mathsf{negl}(n),$$

where we on the left takes the probability over $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$, $x, y, z \leftarrow_R \mathbb{Z}_q$ and the randomness used by $\mathcal{A}$, and on the right takes the probability over $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$, $x, y \leftarrow_R \mathbb{Z}_q$ and the randomness used by $\mathcal{A}$

```
Game $\mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)$
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    $x, y \leftarrow_R \mathbb{Z}_q$
    $h_1 := g^x$
    $h_2 := g^y$
    $\hat{h} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2)$
    Return $(\hat{h} \overset{?}{=} g^{xy})$
```

```
Game $\mathsf{DDH}_{\mathcal{G}}^{\mathcal{A}}(n)$
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    $x, y, z \leftarrow_R \mathbb{Z}_q$
    $b \leftarrow_R \{0, 1\}$
    $h_1 := g^x$
    $h_2 := g^y$
    $h_3 := g^{xy+bz}$
    $\hat{b} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2, h_3)$
    Return $(\hat{b} \overset{?}{=} b)$
```

**Figure 2.2:** CDH (left) and DDH (right) hardness games.

In Propositon 2.5.5 we prove that if the DDH problem is hard relative to $\mathcal{G}$ then CDH is also hard relative to $\mathcal{G}$. The converse statement is unknown but appears not to be true: the DDH problem is not hard in cyclic groups of the form $\mathbb{Z}_q^*$ with $q$ prime. The reason is that if $g^x$ and $g^y$ are quadratic residues in $\mathbb{Z}_q^*$ then so is $g^{xy}$, whereas $g^z$ is a non-quadratic residue in $\mathbb{Z}_q^*$ with probability $1/2$. The solution is to work in subgroups of $\mathbb{Z}_q^*$. One option is the subgroup of all quadratic residues in $\mathbb{Z}_q^*$. Another example where DDH is not hard is in bilinear groups over elliptic curves (for certain instances).

We next present the two basic (known) connections between DL, CDH and DDH formulated in two propositions. We provide (rather) informal proofs of the two statements.

**Proposition 2.5.4.** *If the* CDH *problem is hard relative to $\mathcal{G}$ then the* DL *problem is hard relative to $\mathcal{G}$.*

*Proof.* To prove this statement, we must show that given a PPT algorithm $\mathcal{A}$ that can solve the DL problem, we can construct a PPT algorithm that solves the CDH problem. Let $\mathcal{A}$ be such an algorithm i.e. $x \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x)$ with $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$ and $x \leftarrow_R \mathbb{Z}_q$. Given an instance of the CDH problem $(\mathbb{G}, q, g, g^x, g^y)$ as input let $\mathcal{A}'$ be defined as follows: compute $x \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x)$ and then return $(g^y)^x$ as the result. If $\mathcal{A}$ succeeds with non-negligible probability then clearly $\mathcal{A}'$ would too. Additionally, since $\mathcal{A}$ runs in polynomial-time $\mathcal{A}'$ also runs in polynomial-time. $\square$

**Proposition 2.5.5.** *If the* DDH *problem is hard relative to $\mathcal{G}$ then the* CDH *problem is hard relative to $\mathcal{G}$.*

*Proof.* To prove this statement, we must show that given a PPT algorithm $\mathcal{A}$ that solves the CDH problem, we can construct a PPT algorithm $\mathcal{A}'$ that solves the DDH problem. That is, the algorithm $\mathcal{A}$ computes $g^{xy} \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x, g^y)$ with $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$ and $x, y \leftarrow_R \mathbb{Z}_q$. Given a DDH instance $(\mathbb{G}, q, g, g^x, g^y, g^z)$ as input define $\mathcal{A}'$ in the following way: compute $g^{xy} \leftarrow \mathcal{A}(\mathbb{G}, q, g, g^x, g^y)$ and compare $g^{xy}$ with $g^z$; if they are equal output 0 and output 1 if they are not. Obviously, if $\mathcal{A}'$ succeeds with non-negligible probability then $\mathcal{A}$ also succeeds with non-negligible probability. Also, since $\mathcal{A}$ is a polynomial-time algorithm $\mathcal{A}'$ is also a polynomial-time algorithm. $\square$

## 2.6 Public-Key Encryption

This section will contain the syntax for public-key encryption. In addition we will present two different notions of security, EAV and CPA, and a concrete example of a public-key encryption scheme. Namely, we will present the El-Gamal public-key encryption scheme and prove that this scheme is CPA-secure. Fist we define a public-key encryption scheme.

**Definition 2.6.1.** (**Public-key encryption scheme**) We denote a public-key encryption scheme with $\Pi$, which consists of 3 algorithms: key generation algorithm, encryption algorithm and decryption algorithm. We specify each algorithm below:

- $\mathsf{Gen}(1^n)$ (key generation algorithm) is a PPT algorithm, which on input a security parameter $n$ outputs a secret key $\mathsf{sk}$ and public key $\mathsf{pk}$, both with length at least $n$. We assume that $n$ can be determined by $\mathsf{pk}$ and $\mathsf{sk}$. We write $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$.
- $\mathsf{Enc}(\mathsf{pk}, m)$ (encryption algorithm) is a PPT algorithm, which on input a public key $\mathsf{pk}$ and message $m$, where the message space may depend on the public key $\mathsf{pk}$, outputs a ciphertext $c$. We write $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$.
- $\mathsf{Dec}(\mathsf{sk}, c)$ (decryption algorithm) is a PPT algorithm, which on input a secret key $\mathsf{sk}$ and ciphertext $c$ outputs a message $\hat{m}$ or $\perp$ if decryption fails. We assume WLOG that $\mathsf{Dec}$ is deterministic. We write $\hat{m} = \mathsf{Dec}_{\mathsf{sk}}(c)$.

We reguire correctness except for some negligible probability. That is, there exists a negligible function $\mathsf{negl}$ such that

$$\Pr\left[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m\right] \geq 1 - \mathsf{negl}(n),$$

where the probability is over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ and internal randomness of $\mathsf{Enc}$.

We next present two different notions of security for a public-key encryption scheme: EAV and CPA. The notions are equivalent though, which is discussed after the presentation of the definitions.

**Definition 2.6.2.** (**Indistinguishable encryptions in the presence of an eavesdropper**) A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dev})$ has indistinguishable encryptions in the presence of an eavesdropper (EAV) if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\Pi}^{\mathsf{EAV}}(n) := 2\left|\Pr\left[\mathsf{true} \leftarrow \mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)\right] - \frac{1}{2}\right| \leq \mathsf{negl}(n),$$

where the probability is over the randomness used in Game $\mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)$ defined in Figure 2.3.

```
$\underline{\mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)}$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$
    $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$ with $|m_0| = |m_1|$
    $b \leftarrow_R \{0, 1\}$
    $c \leftarrow \mathsf{Enc}_{\mathsf{k}}(m_b)$
    $\hat{b} \leftarrow \mathcal{A}(c)$
    **Return** $(\hat{b} \stackrel{?}{=} b)$
```

**Figure 2.3:** EAV security game.

Compared to the EAV security game, the CPA security game gives more power to the adversary by allowing oracle access to the encryption function both before and after the challenge ciphertext is sent. It turns out that this gives no advantage to the adversary over the capabilities of an adversary in the EAV security game, see Propositon 2.6.4.

**Definition 2.6.3.** (CPA **security**) A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dev})$ has indistinguishable encryptions under a chosen-plaintext attack (CPA) if for every PPT adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\Pi}^{\mathsf{CPA}}(n) := 2\left|\Pr\left[\mathsf{true} \leftarrow \mathsf{CPA}_{\Pi}^{\mathcal{A}}(n)\right] - \frac{1}{2}\right| \leq \mathsf{negl}(n),$$

where the probability is over the randomness used in Game $\mathsf{CPA}_{\Pi}^{\mathcal{A}}(n)$ defined in Figure 2.4. The adversary is allowed to interact with $\mathcal{O}_{\mathsf{Enc}_{\mathsf{pk}}}$ adaptively, and as many times as it likes (polynomial many in $n$).

The intuition of the next proposition is that knowing the public key effectively allows an adversary to implement an encryption oracle. Therefore CPA provides no extra power to an adversary compared to EAV. We skip the proof.

**Proposition 2.6.4.** *Let $\Pi$ be a public-key encryption scheme. If $\Pi$ is* EAV*-secure, it is also* CPA*-secure.*

```
$\underline{\mathsf{CPA}^{\mathcal{A}}_{\Pi}(n)}$
        $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$
        $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}_{\mathsf{pk}}}(m)}(\mathsf{pk})$ with $|m_0| = |m_1|$
        $b \leftarrow_R \{0, 1\}$
        $c \leftarrow \mathsf{Enc}_{\mathsf{k}}(m_b)$
        $\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}_{\mathsf{pk}}}(m)}(c)$
        **Return** $(\hat{b} \overset{?}{=} b)$

The oracle $\mathcal{O}_{\mathsf{Enc}_{\mathsf{pk}}}$ returns the encryption of $m$ under the public key $\mathsf{pk}$.
```

**Figure 2.4:** CPA security game.

As a final act in this section, we present a concrete scheme and prove that it is CPA secure.

**Definition 2.6.5.** (El-Gamal) Let $\mathcal{G}(1^n)$ be a polynomial-time algorithm that takes as input $1^n$ and outputs, except possibly with negligible probability, a cyclic group $\mathbb{G}$ of order $q$, $|q| = n$, and generator $g$ (of $\mathbb{G}$). We specify the key generation, encryption algorithm and decryption algorithm in Figure 2.5.

```
$\underline{\mathsf{Gen}(1^n)}$
        $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
        $x \leftarrow_R \mathbb{Z}_q$
        $h := g^x$
        $\mathsf{pk} := (\mathbb{G}, q, g, h)$
        $\mathsf{sk} := (\mathbb{G}, q, g, x)$
        **Return** $(\mathsf{pk}, \mathsf{sk})$
$\underline{\mathsf{Enc}(\mathsf{pk}, m)}$
        Parse: $\mathsf{pk} = (\mathbb{G}, q, g, h)$
        $y \leftarrow_R \mathbb{Z}_q$
        $c := (g^y, h^y \cdot m)$
        **Return** $c$
$\underline{\mathsf{Dec}(\mathsf{sk}, c)}$
        Parse: $\mathsf{sk} = (\mathbb{G}, q, g, x)$
        $\hat{m} := c_2 \cdot (c_1^x)^{-1}$
        **Return** $\hat{m}$
```

**Figure 2.5:** El-Gamal public-key encryption scheme.

The El-Gamal scheme presented above can be proven CPA-secure. The proof is a reduction to the DDH hardness game.

**Proposition 2.6.6.** *The* El-Gamal *public-key encryption scheme is* CPA*-secure.*

*Proof.* To show CPA security it is enough to show cipher-text indistinguishable in the presence of an eavesdropper by Propositon 2.6.4.

Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathsf{EAV}^{\mathcal{A}}_{\Pi}(n)$. To prove the proposition, we construct a reduction to the DDH problem relative to $\mathcal{G}$. In the DDH game a distinguisher $\mathcal{D}$ receives a tuple $(\mathbb{G}, q, g, g^x, g^y, g^z)$ where $x, y$ are uniform and $z$ is either a uniform element or $xy$. Below we construct $\mathcal{D}$ using $\mathcal{A}$ as a subroutine. Note that we use the alternative (but equivalent) definition of DDH being hard.

Because DDH is hard relative to $\mathcal{G}$ there exists a negligible function such that

$$|\Pr[\mathsf{true} \leftarrow \mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^z)] - \Pr[\mathsf{true} \leftarrow \mathcal{D}(\mathbb{G}, q, g, g^x, g^y, g^{xy})]| \leq \mathsf{negl}(n),$$

**Distinguisher $\mathcal{D}$**

```
Input: $(\mathbb{G}, q, g, g^x, g^y, g^z)$
1: $\mathsf{pk} := (\mathbb{G}, q, g, g^x)$
2: $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$
3: $b' \leftarrow_R \{0,1\}$
4: $c_1 := g^y$
5: $c_2 := g^z \cdot m_{b'}$
6: $c := (c_1, c_2)$
7: $\hat{b}' \leftarrow \mathcal{A}(c)$
8: Return $(\hat{b}' \overset{?}{=} b')$
```

where $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$, $x, y \leftarrow_R \mathbb{Z}_q$ and either $z \leftarrow \mathbb{Z}_q$ or $z = xy$. On the right-hand side $\mathcal{D}$ receives the public key $(\mathbb{G}, q, g, g^x)$ and ciphertext $(g^y, g^{xy} \cdot m_{b'})$. Therefore the view of $\mathcal{D}$ when run as a subroutine by $\mathcal{D}$ is distributed identically to $\mathcal{D}$'s view in the game $\mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)$. Since $\mathcal{D}$ outputs true if and only if $\mathcal{A}$'s output satisfy $\hat{b}' = b'$, we have that

$$\Pr\left[\mathsf{true} \leftarrow \mathcal{D}\left(\mathbb{G}, q, g, g^x, g^y, g^{xy}\right)\right] = \Pr\left[\mathsf{true} \leftarrow \mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)\right].$$

On the left-hand side $\mathcal{D}$ receives the public key $(\mathbb{G}, q, g, g^x)$ and ciphertext $(g^y, g^z \cdot m_{b'})$. Since $z$ is a uniform element of $\mathbb{Z}_q$ the second component of the ciphertext is a uniformly distributed element of $\mathbb{G}$ and, in particular, is independent of the message $m_{b'}$. The first component is also independent of $m_{b'}$. Therefore, $\mathcal{A}$ learns no information on the ciphertext $m_{b'}$ and only has the option of guessing the bit $b'$. This implies that the probability of the event $\hat{b}' = b'$ is equal to $1/2$. Since $\mathcal{D}$ outputs true if and only if $\hat{b}' = b$, we have that

$$\Pr\left[\mathsf{true} \leftarrow \mathcal{D}\left(\mathbb{G}, q, g, g^x, g^y, g^z\right)\right] = \frac{1}{2}.$$

Combining the above identities, we get (where 2 is multiplied on both sides)

$$2 \cdot \mathsf{negl}(n) \geq 2\left|\frac{1}{2} - \Pr\left[\mathsf{true} \leftarrow \mathsf{EAV}_{\Pi}^{\mathcal{A}}(n)\right]\right|.$$

$2 \cdot \mathsf{negl}(n)$ is clearly negligible which implies the result. □

It is possible to modify the El-Gamal scheme to produce a scheme that is CCA secure [21].

## 2.7 Symmetric Encryption

Below we define the syntax of a symmetric encryption scheme. Note that the notation is the same as with public-key encryption. It will be clear from the context, which kind of encryption scheme that is referred to.

**Definition 2.7.1.** (**Symmetric Encryption Scheme**) We denote a symmetric encryption scheme with $\Pi$, which consists of 3 algorithms: key generation algorithm, encryption algorithm and decryption algorithm. We specify each algorithm below.

- $\mathsf{Gen}(1^n)$ (key generation algorithm) is a PPT algorithm, which on input a security parameter $n$ outputs a symmetric key $\mathsf{k}$. It is required that $|\mathsf{k}| \geq n$. We write $\mathsf{k} \leftarrow \mathsf{Gen}(1^n)$.

- $\mathsf{Enc}(\mathsf{k}, m)$ (encryption algorithm) is a PPT algorithm, which on input a public key $\mathsf{k}$ and message $m \in \{0,1\}^*$ outputs a ciphertext $c$. We write $c \leftarrow \mathsf{Enc}_\mathsf{k}(m)$.
- $\mathsf{Dec}(\mathsf{k}, c)$ (decryption algorithm) is a PPT algorithm, which on input a symmetric key $\mathsf{k}$ and ciphertext $c$ outputs a message $\hat{m}$ or $\perp$ if decryption fails. We assume WLOG that Dec is deterministic. We write $\hat{m} = \mathsf{Dec}_\mathsf{k}(c)$.

We require perfect correctness: for all $n$, all keys $\mathsf{k}$ output by $\mathsf{Gen}(1^n)$, and all messages $m$ the following is true with probability 1:

$$\mathsf{Dec}_\mathsf{k}(\mathsf{Enc}_\mathsf{k}(m)) = m.$$

We only define one notion of security for symmetric encryption schemes. Namely, a symmetric encryption scheme $\Pi$ is secure if an adversary can not distinguish the encryption of two messages that the adversary choose. Details are as follows

**Definition 2.7.2.** (**Indistinguishable encryptions in the presence of an eavesdropper**) A symmetric encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dev})$ has indistinguishable encryptions in the presence of an eavesdropper (EAV) if there for every PPT adversary $\mathcal{A}$ exists a negligible function negl such that

$$\mathrm{Adv}_\Pi^{\mathsf{EAV}}(\mathcal{A}) := 2 \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{EAV}_\Pi^\mathcal{A}(n)\right] - \frac{1}{2} \right| \leq \mathsf{negl}(n),$$

where the probability is taken over the randomness in the game $\mathsf{EAV}_\Pi^\mathcal{A}(n)$ defined in Figure 2.6.

```
Game $\mathsf{EAV}_\Pi^\mathcal{A}(n)$
    $(m_0, m_1) \leftarrow \mathcal{A}(1^n)$
    $\mathsf{k} \leftarrow \mathsf{Gen}(1^n)$
    $b \leftarrow_R \{0,1\}$
    $c \leftarrow \mathsf{Enc}_\mathsf{k}(m_b)$
    $\hat{b} \leftarrow \mathcal{A}(c)$
    Return $(\hat{b} \overset{?}{=} b)$
```

**Figure 2.6:** EAV-security game of a symmetric encryption scheme $\Pi$.

EAV-security is one of the the weakest forms of security one can wish for in a symmetric encryption scheme. A stronger notion is the notion of a CPA secure symmetric encryption scheme. In Section 2.6 we saw that EAV-security is equivalent to CPA-security of a public-key encryption scheme. That is not the case when dealing with symmetric encryption: CPA-security implies EAV-security, but the converse does not hold, see Example 2.7.3.

*Example* 2.7.3. Consider the scheme where key generation $\mathsf{Gen}(1^n)$ is defined as $\mathsf{k} \leftarrow_R \{0,1\}^n$, encryption as $\mathsf{Enc}_\mathsf{k}(m) := \mathsf{k} \oplus m$ for $m \in \{0,1\}^n$ and decryption $\mathsf{Dec}_\mathsf{k}(c) := \mathsf{k} \oplus c$. Obviously this is the one-time-pad encryption scheme, which is a perfect secure scheme and thus also EAV-secure. It is well known that doing multiple one-time-pad-encryptions using the same key $\mathsf{k}$ allows an adversary to infer information on the message $m$. Therefore, the one-time-pad encryption scheme is not CPA-secure (it is called **one-time**-pad for a reason).

## 2.8 Random Oracle Model

In the Random Oracle Model, we assume the existence of a random function $\mathcal{R}$, and give all parties oracle access to this function. A good way to think about the Random Oracle Model is to think about it as a black box that takes a binary string as input and returns a binary string as output. Everyone, including the adversary, can ask private queries to this box. The output to each query is assumed to be uniformly random such that if a party queries $x$ then no one else learns $x$, or even learns that this party queried the oracle in the first place.

The random oracle $\mathcal{R}$ must be consistent. That is, if $y$ is the output of a query with $x$ then $y$ is also the output if $x$ is ever queried again. This property emulates the behaviour of a hash function. Replacing hash functions with random oracles (random functions) in proofs is one of the usual use-cases of the Random Oracle Model.

The probability space changes slightly when games are considered in the Random Oracle Model instead of the Standard Model. Namely, in addition to what the probability is taken over in the Standard Model, we must also take the probability over the random choices of $\mathcal{R}$.

Using the Random Oracle Model gives powerful tools. We will mainly use two properties:

- If $x$ has not been queried to $\mathcal{R}$, then the value of $\mathcal{R}(x)$ is uniform.
- If adversary $\mathcal{A}$ queries $x$ to $\mathcal{R}$, the reduction can see this query and learn $x$.

The first property says in particular that the output $\mathcal{R}(x)$ is completely uniform to an observer until the observer queries $x$ to $\mathcal{R}$ itself. The last property relate to proofs of reduction, which will be used throughout this thesis.

CHAPTER 3

# Cryptographic Game

In this chapter we present definitions of a non-interactive cryptographic game and an interactive cryptographic game. A cryptographic game will, in general, encompass a definition of a *cryptographic device* to which a notion of security, modelled via a *predicate function*, is attached. The game can in addition have parameters defined by a parameter function.

The chapter is organised as follows: in Section 3.1 we analyse (selected) previous works on cryptographic subversion and highlight their main ideas and results. Section 3.2 contains motivations for the definitions presented in the upcoming sections. Moving on to Section 3.3, we rigorously define what we mean by a cryptographic game. Two different definitions are presented: one for non-interactive cryptographic games and one for interactive cryptographic games.

## 3.1 Previous Work

Before stating our own definitions we take a detour to some previous works on cryptographic subversion. We mainly focus on two works: the fundamental paper [27] by Young and Yung, and a more recent paper [8] by Bellare, Paterson and Rogaway. From the analysis we draw ideas for the definitions presented in subsequent sections. Especially, Young and Yung present a subversion that we elaborate heavily on in Chapter 5.

**The Dark Side of Black-Box Cryptography, or: Should We Trust Capstone? [27]**

In this 1996 paper, Young and Yung present the notion of a SETUP (Secretly Embedded Trapdoor with Universal Protection) mechanism. A SETUP mechanism is an algorithm, which can be embedded within a cryptographic system in such a way that the cryptographic system leaks information and at the same time protects the attacker from detection. They called this class of subversions for Kleptographic attacks.

The main line of work presented is the construction of SETUPs for RSA, El-Gamal, DSA and Kerberos cryptosystems. The main idea is to embed a public key inside a cryptosystem and use this key to construct a subliminal channel that can only be read with the corresponding private key. The embedding of a public key in the target device is the preferred method used in this thesis to give an attacker (big brother) an advantage.

In particular Young and Yung present a technique for subverting El-Gamal. In modern terms, they make use of the Random Oracle Model and a second instance of an El-Gamal public-key encryption scheme. By a clever construction and over two separate encryptions, they 'prove' that it is possible to learn the ephemeral key used in the second encryption and thereby possible to completely recover the second encrypted message. We explore their technique by further elaborating it to construct subversions of DL flavoured hardness assumptions in Chapter 5.

Unfortunately the definition of a SETUP is vague and very ambiguous which limits the usability and extendibility of their work. We fix this shortcoming in this thesis.

**Security of Symmetric Encryption against Mass Surveillance [8]**

In light of the infamous facts uncovered by Snowden, the 2014 paper by Bellare, Paterson and Rogaway presents the first rigorous approach to cryptographic subversion. They specifically restrict their scope to symmetric encryption and prove that all probabilistic, state-less symmetric encryptions schemes are susceptible to a subversion attack if a (low) amount of randomness is employed.

Bellare et al. does not use the same technique as Young et al. Instead they embed a symmetric key from which they construct indistinguishable symmetric schemes that leak secret key material. In this work, the notion of a subversion is clearly defined by looking at the output of a specific scheme and the output of a subverted variant of the same scheme. If the output is (computationally) indistinguishable the subversion is said to be undetectable. We use this idea of defining undetectability in our own definitions but modified to fit our asymmetric approach. This is the first time in literature that this has been done in a formal and rigorous way.

Besides constructing subversions, Bellera et al. also define surveillance security i.e. quantify when a symmetric encryption scheme is secure against a subversion attack. They construct a family of deterministic, state-full symmetric encryption schemes that are surveillance secure. Surveillance security is, however, out of scope for this thesis.

## 3.2 Setting the Stage

On the quest for defining cryptographic games, we first take a drone perspective. Apart from serving as a light introduction to the terminology and ideas used subsequently, this paragraph also serves as a motivational chapter highlighting thoughts that support the more formal definitions coming later.

Recall that we want to model general cryptographic games. For example, the game that models the DL hardness assumption, see Figure 2.1. The game starts out by having a generator function $\mathcal{G}$ generate a cyclic group $\mathbb{G}$ of order $q$ and a generator $g$. The game proceeds by performing some operations that is supposed to hide the discrete logarithm $x$ in a cryptographic way. More precisely, it should be computationally hard to compute the discrete logarithm $x$ given $g^x$. The advantage $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DL}}(\mathcal{A})$ quantify the probability of computing $x$ for an (computationally bounded) adversary $\mathcal{A}$. Most cryptographic games contains the same elements as in the game for the DL hardness assumption. First parameters are generated. Such parameters may come from a standard so we have to deal with that in our definition of a cryptographic game (further discussed below). Next, (cryptographic) computations must be carried out to generate an output that hides some information. These computations can be either non-interactive meaning that the game carries out all of them without any external input (except possible parameters) or it can be interactive, which means that the game interacts with another party, affecting the computations. We deal with both situations. Last, to quantify what the computations must hide an advantage is defined with respect to an (computationally bounded) adversary. The definition of the advantage differs from each game but is generally split between two classes: 'compute something' or 'distinguish something'. In the compute case, such as in the DL hardness game, the adversary must compute a value. The advantage $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DL}}(\mathcal{A})$ catches just that: the probability of $\mathcal{A}$ computing the discrete logarithm. In the distinguish case, such as in the DDH hardness game (see Def. 2.5.3), the adversary is asked to distinguish between two distributions: $(g^x, g^y, g^{xy})$ and $(g^x, g^y, g^z)$ with $x, y, z$ uniform random elements. The trivial strategy to use, is just to guess at random, producing a probability of $\frac{1}{2}$ of guessing the correct distribution. To adjust for that strategy the definition of the advantage of the DDH hardness game must be changed compared to the advantage for the DL game (simply guessing in the DL game is also a strategy but would produce a negligible probability if $q$ is exponential in $n$). The adjustment needed is reflected in the definition of $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DDH}}(\mathcal{A})$.

In total, we need to define what it means to generate parameters, define what it means to do computations, both non-interactively and interactively, and finally define a general way to quantify the security property. These three tasks are discussed below and formalised in Section 3.3.

**Cryptographic device**

A cryptographic device will model the computations done in a cryptographic game. We will consider a device Dev taking a certain set of inputs and producing an output. The goal of Dev is to hide some piece of information in a cryptographic way. A device must therefore have a notion of security quantifying 'cryptographic way' and it is, more precisely, the goal of Dev to be secure in the sense of this notion of security. We follow the standard cryptographic literature in defining notions of security. That is, we quantify the security of a device Dev by relating a security game to Dev and imagine an adversary $\mathcal{A}$ participating in the game. The security game is constructed in such a way that winning it, breaks the security property of Dev we want to achieve. Thus, security of Dev is quantified by requiring that the advantage of winning the security game for *any* (computationally bounded) adversary $\mathcal{A}$ is at most negligible.

The output of a device Dev is split between two different transcripts: a public and a secret transcript. The reason for splitting the output into two groups is that it allows us to consider two degrees of undetectability: weak and strong undetectability (see Chapter 4). It is natural to consider two different outputs from a cryptographic operation. In the example of the DL hardness game, the operation carried out produces a public output $g^x$ and secret output $x$. A user of the device should be able to see both, while a person just observing the device should only see the first.

**Parameter function**

To a device Dev we associate a parameter function Par. In practice it is common for schemes and protocols to have parameters defined from a standard or specification. For example, elliptic curve cryptography deployed in practice use fixed curves. The area of elliptic curve cryptography is in general covered by a plethora of different standards covering selection of such curves

[9]. To maintain security of a cryptographic scheme it is very important to restrict the choice of parameters because there are often instantiations that are not secure, when certain parameters are used.

From our perspective there is a difference between parameters coming from a standard and parameters generated internally by the device, because it changes the power of a subversion attacker. The intuition behind this observation is the following: consider a scheme that can be modelled as a cryptographic game. For concreteness say this scheme uses elliptic curves. If these elliptic curves are fixed by a standard, an attacker that wants to subvert the scheme has extra knowledge[1] before generating the key that is going to be embedded inside the scheme (see Section 4.1). If the device generates the elliptic curves internally, the adversary has no a priori knowledge of these curves and may have trouble generating a suitable key to embed. It follows that having a subversion *independent* of the parameter function is a stronger and more agile subversion than a subversion that rely on parameter knowledge. Therefore, allowing the attacker access to parameters used by a device gives a more realistic model.

Below we highlight 2 different models, we consider throughout this thesis: known and unknown parameters.

*Known parameters:* In a situation where a scheme is governed by a public standard or specification, we deal with known parameters. Such parameters can be public keys for public-key encryption or Diffie-Hellman key exchange groups that satisfy certain requirements.

We may also choose to change the way parameters are generated to explore how this affects the likelihood of doing a subversion attack. For example, it might be that the subversion attacker only needs some subset of parameters fixed from the standard. In such situations we only let $\mathsf{Par}$ generate the parameters required by the adversary. We might also go the other way around allowing $\mathsf{Par}$ to generate more values than the standard advocates. This gives us more freedom in analysing when schemes fall for subversion attacks and can thereby provide more insight. In general, the addition of a parameter function provides agility to the definitional framework.

*Unknown parameters:* We also consider devices that are completely independent of any parameter function. E.g. a key generation function operate mostly with only the security parameter and some configuration of what primes it is allowed to generate, which are not parameters. Instead it is implicit in the design of the device. Such a device has a parameter function that is identical $\perp$ i.e. the parameter function provides no information.

## 3.3 Cryptographic Game

We end our quest for a definition of a cryptographic game in this section. We will formally define the notion of a cryptographic game for the non-interactive and interactive case. Both definitions will contain the following 5 elements

- A parameter algorithm that outputs parameters and constitutes a priori knowledge of the cryptographic device.

[1]Assuming that whoever is trying to construct a subversion, can read and Google the way to the standard/specification - which is probably a fair assumption to make!

- A device that performs a cryptographic operation. When executed, the device gets as input a parameter set and produces a public and secret transcript. In the case of an interactive cryptographic game, the device also interacts with another party. The secret transcript contains output from the device that is supposed to be secret for anyone except the user of the device. The public transcript contains the output that is going on the wire and thus visible for everyone.
- A predicate algorithm that together with the base probability and base factor, quantifies the security guarantee with respect to the device. This should conceptually formalise what an adversary should accomplish when trying to recover the information that the device is hiding.

The base probability and base factor are two polynomials $p_0(n)$ and $\alpha_0(n)$, which are technical tools to define the notion of security of a game.

### 3.3.1 Non-Interactive Cryptographic Game

With the definition of a non-interactive cryptographic game we specifically aim to capture games defining hardness assumptions. Several such games can be viewed in Chapter 2. Typically, hardness games involves a trapdoor of some sort or the game defines two different distributions and outputs one of them depending on a bit. For the former the goal of an adversary is to compute the inverse of the trapdoor while in the latter case, it is the goal of the adversary to guess the correct bit depending on which distribution is observed, i.e. the adversary has to distinguish between two distributions. Below we precisely formalise a non-interactive cryptographic game ($\mathsf{NICG}$).

**Definition 3.3.1.** A **non-interactive cryptographic game** is a 5-tuple $(\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$, denoted by $\mathsf{G}$, where

- $\mathsf{Par}$ is the parameter function and is a $\mathsf{PPT}$ algorithm, which on input the security parameter $n$ outputs a list of parameters $\mathsf{params} \in \{0,1\}^*$. We write $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$. In case the list is empty, we write $\perp \leftarrow \mathsf{Par}(1^n)$ and $\mathsf{params} = \perp$.
- $\mathsf{Dev}$ is the cryptographic device and is a $\mathsf{PPT}$ algorithm, which on input the security parameter $n$ and $\mathsf{params} \in \{0,1\}^*$ outputs a public transcript $\tau_p \in \{0,1\}^*$ and a secret transcript $\tau_s \in \{0,1\}^*$. We write $(\tau_p, \tau_s) \leftarrow \mathsf{Dev}(1^n, \mathsf{params})$.
- $\mathsf{Pre}$ is the predicate function and is a polynomial-time algorithm, which takes as input a public transcript $\tau_p \in \{0,1\}^*$, secret transcript $\tau_s \in \{0,1\}^*$ and subject $\delta \in \{0,1\}^*$ and outputs a predicate $\mathsf{bool} \in \{\mathsf{true}, \mathsf{false}\}$. We write $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$.
- $p_0(n)$ is the base probability, which is a polynomial in $n$ with range $[0,1]$.
- $\alpha_0(n)$ is the base factor, which is a polynomial in $n$ with range $\mathbb{R}_{>0}$.

If $\mathsf{params}$ is equal to $\perp$ for at least one $n$, the parameter function $\mathsf{Par}$ must be identical $\perp$. In this case we might ignore $\mathsf{Par}$ from the notation (as well as from $\mathsf{Dev}$) and call $\mathsf{G}$ *standardless*.

When constructing $\mathsf{Dev}$ we clearly state whether $\mathsf{G}$ is standardless or not. Therefore in the description of a specific device $\mathsf{Dev}$, we will not handle both cases i.e. we only deal with $\mathsf{params} = \perp$ or not, and no exceptional case handling is performed when describing algorithms.

In Figure 3.1 we describe a typical NICG. The group we operate in is fixed. Therefore, the group, its order and a generator is computed by the parameter function Par. The device Dev models a trapdoor scenario where the discrete logarithm is used as trapdoor. Obviously, the information Dev is trying to hide is the discrete logarithm $x$. Therefore, $x$ is the secret output while the group instance and the computed exponentiation is the public output.

```
$\mathsf{Par}(1^n)$
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    $\mathsf{params} := (\mathbb{G}, q, g)$
    Return params
$\mathsf{Dev}(1^n, \mathsf{params})$
    Parse: $\mathsf{params} = (\mathbb{G}, q, g)$
    $x \leftarrow_R \mathbb{Z}_q$
    $h := g^x$
    $\tau_p := (\mathbb{G}, q, g, h)$
    $\tau_s := (\mathbb{G}, q, g, x)$
    Return $(\tau_p, \tau_s)$
$\mathsf{Pre}(\tau_p, \tau_s, \sigma)$
    Parse: $\tau_s = (\mathbb{G}, q, g, x)$
    Return $(x \overset{?}{=} \sigma)$
$(p_0(n), \alpha_0(n)) := (0, 1)$
$\mathcal{G}(1^n)$ outputs a description of a cyclic group of order
$q$ and a generator $g$.
```

**Figure 3.1:** Cryptographic game modelling the DL hardness game.

Assume that an adversary $\mathcal{A}$ attacks Dev. How do we then measure the likelihood of $\mathcal{A}$ recovering the discrete logarithm? Pre, $p_0(n)$ and $\alpha_0(n)$ to the rescue! We set the predicate function Pre to return true if and only if the discrete logarithm is given as input and $(p_0(n), \alpha_0(n)) = (0, 1)$. The likelihood of $\mathcal{A}$ computing the discrete logarithm, given the public output, is then precisely the likelihood of $\mathcal{A}$ making Pre return true. This is formalised below in Def. 3.3.2 and Def. 3.3.3.

The predicate function Pre models the security of the device Dev. Together with the base probability $p_0(n)$ and base factor $\alpha_0(n)$ we can quantify the advance an adversary has with respect to a specific device.

**Definition 3.3.2.** For a PPT adversary $\mathcal{A}$ we define its **advantage** with respect to a non-interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ as

$$\mathrm{Adv}_{\mathsf{G}}^{\mathsf{SEC}}(\mathcal{A}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}}(n)\right] - p_0(n) \right|,$$

where the probability is taken over randomness used in game $\mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$.

```
Game $\mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$
    $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Dev}(1^n, \mathsf{params})$
    $\delta \leftarrow \mathcal{A}(\mathsf{params}, \tau_p)$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 3.2:** Security game for a non-interactive cryptographic game G.

Given parameters params and transcript $(\tau_p, \tau_s)$ the adversary $\mathcal{A}$ precisely wins the security game if $\delta$ is the correct information required by the predicate function Pre. We are now able to define what it means for a non-interactive cryptographic game to be secure.

**Definition 3.3.3.** A non-interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is **secure** if there for every PPT adversary $\mathcal{A}$ exists a negligible function negl such that

$$\mathrm{Adv}_{\mathsf{G}}^{\mathsf{SEC}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

The non-interactive cryptographic game in Figure 3.1 is actually a secure non-interactive cryptographic game (the reader hopefully already spotted the clear connection too Def. 2.5.1). The proof is a trivial reduction to the discrete logarithm problem but we omit the proof here.

### 3.3.2 Interactive Cryptographic Game

In a conventional cryptographic scheme one starts with a definition that describes the properties of the scheme. Having quantified the properties of the scheme, the normal procedure is then to define the security notion based on a game and corresponding adversarial advantage. Definitions of public-key encryption schemes and symmetric encryption schemes follows this approach. In particular, they all employ interactive security definitions.

To capture a large class of interactive cryptographic games we are first required to define interaction between two parties: the cryptographic device and a second party. The aim is to do this in a very general way. Our approach is highly inspired by [22] but with some modifications that better fit our purpose. When we describe concrete schemes later, we do it in simpler terms compared to the level of generality used in the definition below.

We will denote an interactive cryptographic game using the same notation as for the non-interactive cryptographic games. This creates some ambiguity between the two concepts but we will clearly state which type of game we are referring to in the text and in general trust the reader in telling the difference depending on the context. We provide the details of an interactive cryptographic game (ICG) below.

**Definition 3.3.4.** An **interactive cryptographic game** is a 5-tuple $(\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$, denoted by G that defines an interaction between two stateful parties: a device Dev and a party [2] P with state $\sigma_{\mathsf{Dev}}$ and $\sigma_{\mathsf{P}}$, respectively. $\sigma_{\mathsf{Dev}}$ and $\sigma_{\mathsf{P}}$ will always refer to the current state of each respective party. In addition, G defines the notion of security Dev must satisfy.

First a setup precedure is run. It returns the starting state for both parties. Next the two parties engage in an interaction, taking turn to send and receive messages. At the end, they both run a return algorithm. Details are as follows:

- Par is the setup algorithm that is a PPT algorithm, which on input the security parameter $n$ outputs parameters params $\in \{0,1\}^*$ and a message schedule. We omit explicit mentioning of the message schedule in the sequel as it will always be clear. params is the initial state for both Dev and P.

[2]The party P may be a an adversary, a big brother recovery algorithm (which we define later) or some other party.

- Dev is the interactive cryptographic device, which is a party that interacts with another party P. After the setup algorithm Par has been run, Dev and P are given params and proceed to send messages according to the message schedule. Each party has an associated next message algorithm $\mathsf{Next}_{\mathsf{P}}(\sigma_{\mathsf{P}})$ and $\mathsf{Next}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$ that is called when a message must be sent. Additionally, each party has a message receive algorithm $\mathsf{Receive}_{\mathsf{P}}(\sigma_{\mathsf{P}}, m)$ and $\mathsf{Receive}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}}, m)$, which is called when a message $m \in \{0,1\}^*$ must be received. After the interaction is finished, Dev and P run their return algorithms $\mathsf{Return}_{\mathsf{P}}(\sigma_{\mathsf{P}})$ and $\mathsf{Return}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$, and return their results: $\mathsf{Return}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$ will return two transcripts $\tau_p \in \{0,1\}^*$ and $\tau_s \in \{0,1\}^*$ containing respectively the public messages sent during the execution and the private output from Dev, while $\mathsf{Return}_{\mathsf{P}}(\sigma_{\mathsf{P}})$ will return some information $\delta \in \{0,1\}^*$. The interaction between Dev and P must end after a polynomial (in $n$) number of steps. We identify $\mathsf{Dev} = (\mathsf{Receive}_{\mathsf{Dev}}, \mathsf{Next}_{\mathsf{Dev}}, \mathsf{Return}_{\mathsf{Dev}})$ and $\mathsf{P} = (\mathsf{Receive}_{\mathsf{P}}, \mathsf{Next}_{\mathsf{P}}, \mathsf{Return}_{\mathsf{P}})$.
- Pre is the predicate function, which takes as input a public transcript $\tau_p \in \{0,1\}^*$, secret transcript $\tau_s \in \{0,1\}^*$ and subject $\delta \in \{0,1\}^*$ and outputs a predicate $\mathsf{bool} \in \{\mathsf{true}, \mathsf{false}\}$. We write $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$.
- $p_0(n)$ is the base probability, which is a polynomial in $n$ with range $[0,1]$.
- $\alpha_0(n)$ is the base factor, which is a polynomial in $n$ with range $\mathbb{R}_{>0}$.

Both Dev and P must be PPT parties meaning that each step must take at most polynomial time (in $n$) to finish i.e. algorithms Receive, Next and Return all run in polynomial time (in $n$) for all states and messages. If any of the two parties sent the special symbol # the interaction terminates[3] immediately .

By the terminology *run* G *with* P, we mean that we run the protocol defined by Par, Dev and P. If params and the message schedule is given, we also use the terminology *run* Dev *with* P *given* params (or just *run* Dev *and* P when params is clear), to mean that we run the protocol defined by Dev, P and message schedule, where Dev and P are given initial state params as input before sending messages.

When we in the sequel choose a party P and let P interact with an interactive cryptographic device Dev we implicitly assume that P respect the message schedule. For the concrete interactive cryptographic games we describe later, it will be clear which parties that can interact with a concrete interactive cryptographic device Dev.

Finally, we only consider honest parties i.e. all parties must adhere to the message schedule and are not allowed to send bogus messages, e.g. if P is required to send two elements from a group $\mathbb{G}$, P must do that and not send elements from another group[4].

To quantify security of an interactive cryptographic game, we make some modifications to Def. 3.3.2 and Def. 3.3.3. The latter stays the same while in the former, we make the game $\mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$ interactive. Adversary $\mathcal{A}$ might be adaptive: during the execution of the interaction between $\mathcal{A}$ and Dev, the adversary is allowed to adapt its strategy based on the messages received from Dev. Hence, it is not required for $\mathcal{A}$ to fix how to choose each message beforehand. $\mathcal{A}$ must still obey the message schedule though. We give details of the security quantification below.

[3]This might happen if one of the two parties are performing a task that can fail such as generating a RSA modulus.
[4]That is, for both parties we assume that the the messages received satisfy some appropriate validity checks

**Definition 3.3.5.** For a PPT adversary $\mathcal{A}$ we define its **advantage** with respect to an interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ as

$$\mathrm{Adv}_{\mathsf{G}}^{\mathsf{IN\text{-}SEC}}(\mathcal{A}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}_{\mathsf{G}}^{\mathcal{A}}(n)\right] - p_0(n) \right|,$$

where the probability is taken over randomness used in the game $\mathsf{IN\text{-}SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$.

```
Game $\mathsf{IN\text{-}SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$
    Run $\mathsf{G}$ with $\mathcal{A}$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return $\mathsf{bool}$
```

**Figure 3.3:** Security game for an interactive cryptographic game $\mathsf{G}$.

The adversary $\mathcal{A}$ precisely wins the security game if $\delta$ is the information required by $\mathsf{Pre}$. We are now able to define what it means for an interactive cryptographic game to be secure.

**Definition 3.3.6.** An interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is **secure** if there for every PPT adversary $\mathcal{A}$ exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathsf{G}}^{\mathsf{IN\text{-}SEC}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

A secure interactive cryptographic game is presented in Chapter 6.

CHAPTER 4

# Defining Subversion

Inspired from previous work [27, 8], we will in this chapter define what it means to subvert a cryptographic game. Namely, a successful subversion can leak information to the subverter, and only the subverter, and the output of a subverted device is indistinguishable from the output of the original non-subverted device for everyone except the subverter. In addition to defining subversion of cryptographic games, we also provide results on general properties of our new framework.

We are interested in investigating *subversions* of a device Dev. A subversion replaces the executable code of Dev with a modified version B.Dev authored by *big brother* B. How B manages to replace the code is out of scope. We just assume that it is somehow possible[1]. Big brother aims to construct a subversion B.Dev in such a way that it leaks information. Achieving leakage from a subverted device is basically trivial: B simply modify Dev to publicly output whatever B wants leaked. It is clear that such subversion is easy to do for B but it also allows anyone else to snatch the leaked information. Such subversions are useless from the perspective of big brother. To exclude them, we impose further restrictions on B.Dev: B should be the only one knowledgeable about the replacement of Dev with B.Dev. That is, no one should be able to *detect* that the device Dev have been subverted, except B. If B construct a subversion with this property, we call the subversion *undetectable*. Formally, undetectability will be formalised by an undetectability game, see Section 4.2.2. We call an entity trying to detect the presence of a subversion for a *distinguisher*. The strength of a distinguisher varies depending on what output from the device Dev is being made available to it. Two different strengths will be defined: weak and strong undetectability. For weak undetectability the distinguisher will only see the public output from Dev while for strong undetectability the distinguisher will have access to (as the reader probably have guessed) both the secret and public output. In addition to undetectability, we also require that B.Dev satisfy the same notion of security as Dev. Because the use of B.Dev is assumed to be the same as Dev, it makes most sense to restrict attention to subversions that maintain the same level of security. In fact, we will see later that preservability of security is implied automatically (see Propositon 4.2.2 and Propositon 4.2.7). Finally, we must also (obviously) require that B is able to obtain some secret knowledge when Dev is replaced by B.Dev. This latter restriction is elaborated in Section 4.1 where we further discuss the role and characteristics of B. Essentially, we say that big brother B

[1]Considering the negligent Internet behaviour by many people, this should be an easy task!

is successful if the corresponding subversion B.Dev can not be detected, preserves security and B is able to recover secret information.

This chapter is organised as follows: in Section 4.1, we discuss big brother. Subversion of non-interactive cryptographic games and interactive cryptographic games is defined in Section 4.2.2 and Section 4.2.3, respectively, and the scope of the definitions is also discussed. After each definition, we present some general properties of the framework.

## 4.1 Big Brother

Big brother, denoted by B is the manufacture of the subversion. It is the goal of B to replace the code for Dev by B.Dev in such a way that when the user runs B.Dev it leaks information to B. Instead of a priori defining that B should obtain (e.g.) the secret key used in Dev, we use the security notion attached to Dev. This is a new approach, in the sense that previous work has used ah hoc quantifications of the recovery abilities of big brother and generally lacks rigour when defining the recovery capabilities.

To simulate leak of information, we take a very liberal approach in which we say that B needs only break the security of Dev to 'extract information'. The reasoning behind this is that it is the least restrictive and most general approach. Since the concept of a cryptographic device is very general, it likewise does not make sense to fix on some certain information, for example B.Dev should leak the secret key, if such a key is not being generated by Dev in the first place! Instead we include a very broad range of possible subversions by defining that recovery of secret information is possible if only B can break the original security of Dev. For every B, we therefore associate a recovery algorithm B.Rec. The goal for B.Rec is to break the security of B.Dev. We formalise this by testing whether the recovery algorithm breaks the (slightly modified) security game for G.

Of course B should have some advantage over normal adversaries. This advantage is knowledge of the big brother secret key $\mathsf{sk}_\mathsf{B}$, which is an input to the recovery algorithm. To take advantage of the secret key $\mathsf{sk}_\mathsf{B}$, the corresponding public key $\mathsf{pk}_\mathsf{B}$ is given as input to B.Dev. In practice, this amounts to a situation where $\mathsf{pk}_\mathsf{B}$ has been embedded in the code for B.Dev. The big brother public key $\mathsf{pk}_\mathsf{B}$ is therefore located on the subverted device. We make the assumption that this does not allow any distinguisher to discover that the device has been compromised. Since $\mathsf{pk}_\mathsf{B}$ is a public key and there is a chance of finding it in the code, we let the key be public i.e. any distinguisher trying to distinguish Dev and B.Dev has access to $\mathsf{pk}_\mathsf{B}$.

Allowing a distinguisher access to the big brother public key is a different approach compared to recent works [8, 7, 16, 17] where knowledge of the big brother symmetric key completely compromise big brother. In our model, knowledge of the big brother public key gives no extra advantage to a distinguisher even if all owners of a device, that has been subverted with the same big brother public key, cooperate. In particular, successful reverse engineering of one device does not lead to detection of other subverted devices.

Since the subverted device B.Dev is supposed to replace device Dev *as is*, we will implicitly assume that B.Dev behaves functionally equivalent[2] to Dev. That is, whenever Dev is usable, B.Dev must also be usable and able to function in the same way without needing any modifications. It

[2]For example if Dev is modelling an encryption with perfect correctness, then B.Dev should also satisfy perfect correctness

would be a trivial task to distinguish two devices that are not functionally equivalent. We will make sure to state precisely how each device should be used and what input/output is allowed. Concretely, the functionality requirements will be clear from the description of the cryptographic device.

Recall from the previous paragraph that we require $\mathsf{B.Dev}$ to be undetectable. $\mathsf{B.Dev}$ being functionally equivalent to $\mathsf{Dev}$ is the most basic form of undetectability. However we expect that big brother will aim to evade more sophisticated forms of detection. We formalise detection security as requiring that output from the subverted device and the original (non-subverted) device is indistinguishable also after (possible) multiple invocations. As discussed above, $\mathsf{Dev}$ and $\mathsf{B.Dev}$ should be indistinguishable even to a test that has knowledge of the big brother public key.

Since a device $\mathsf{Dev}$ has two outputs, one public and one private output, we can consider two different distinguishers. The direct user $\mathcal{U}$ of a device will be the most powerful distinguisher. $\mathcal{U}$ have knowledge of both the public and private output. A distinguisher that is only observing a device can at most learn the public output. We call such a distingusher an eavesdropper, which will be denoted by $\mathcal{E}$. Even though $\mathsf{B}$ essentially is an adversary, when formulating undetectability we actually consider $\mathcal{U}$ and $\mathcal{E}$ as adversaries who are trying to distinguish two constructions through a 'security game'. This is a funny example of when one wants to describe a property in cryptography the threat models sometimes needs to be turned upside-down.

As mentioned in previous paragraphs, $\mathsf{B}$ will embed a public key $\mathsf{pk_B}$ into $\mathsf{B.Dev}$, which can be used to construct the subversion. This public key together with the corresponding private key must therefore be generated by $\mathsf{B}$ before constructing $\mathsf{B.Dev}$. More importantly the generating of the big brother keys happens *before* any user uses $\mathsf{B.Dev}$. We therefore give big brother access to parameters belonging to the device. In a real-world scenario, big brother would also have access do these parameters. Hence this choice allows for a more realistic model.

We now formally define big brother. In summary, we want big brother to learn some information that a priori should remain secret. To achieve this, big brother is constructing a similar device $\mathsf{B.Dev}$ that is functionally equivalent to $\mathsf{Dev}$ but contains a secret backdoor that only big brother can use. The backdoor is constructed by allowing big brother to embed a big brother public key $\mathsf{pk_B}$ inside $\mathsf{B.Dev}$. Big brother can use the backdoor and extract information by using the corresponding big brother secret key $\mathsf{sk_B}$.

In the sequel, we will often refer to a big brother without mentioning a non-interactive/interactive suffix. It will be clear from the context what kind of big brother that is referred to.

### 4.1.1 Non-interactive big brother

We first present the definition of a big brother for the non-interactive case.

**Definition 4.1.1.** A **Non-interactive big brother** for a non-interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is a 3-tuple $(\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ of PPT algorithms denoted by $\mathsf{B}$, where

- $\mathsf{B.Gen}$ is a PPT algorithm that we call the *big brother key generation* algorithm, which on input the security parameter $n$ and parameter $\mathsf{params}$ outputs a big brother public key $\mathsf{pk_B} \in \{0,1\}^*$ and a big brother secret key $\mathsf{sk_B} \in \{0,1\}^*$ both with length at least $n$. We write $(pk, sk) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$.

- $\mathsf{B.Dev}$ is a PPT algorithm that we call the *subversion of* $\mathsf{Dev}$, which takes as input the security parameter $n$, parameters $\mathsf{params}$ and big brother public key $\mathsf{pk_B}$ and outputs two transcripts, public and secret, in the same range as the corresponding transcripts from $\mathsf{Dev}$. We write $(\tau_p, \tau_s) \leftarrow \mathsf{B.Dev}(1^n, \mathsf{params}, \mathsf{pk_B})$.
- $\mathsf{B.Rec}$ is a PPT algorithm that we call the *big brother recovery algorithm*, which on input a big brother secret key $\mathsf{sk_B}$ and public transcript $\tau_p$ outputs information $\delta \in \{0,1\}^*$. We write $\delta \leftarrow \mathsf{B.Rec}(\mathsf{sk_B}, \tau_p)$.

Notice that the recovery algorithm $\mathsf{B.Rec}$ effectively works as an adversary for the security game of $\mathsf{G}$ with the extra knowledge of the big brother secret key $\mathsf{sk_B}$.

### 4.1.2 Interactive big brother

The interactive big brother is different from the non-interactive counterpart in the sense that we are subverting a device that interacts with another party. The subverted device $\mathsf{B.Dev}$ and the big brother recovery algorithm $\mathsf{B.Rec}$ therefore needs to be changed accordingly.

**Definition 4.1.2.** An **interactive big brother** for an interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is a 3-tuple $(\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ denoted by $\mathsf{B}$, where

- $\mathsf{B.Gen}$ is a PPT algorithm that we call the *big brother key generation* algorithm, which on input the security parameter $n$ and parameter $\mathsf{params}$ outputs a big brother public key $\mathsf{pk_B} \in \{0,1\}^*$ and a big brother secret key $\mathsf{sk_B} \in \{0,1\}^*$ both with length at least $n$. We write $(pk, sk) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$.
- $\mathsf{B.Dev}$ is a PPT party that we call the subversion of $\mathsf{Dev}$. $\mathsf{B.Dev}$ functions the same way as $\mathsf{Dev}$ in its interaction with another party (i.e it must adhere to the same message schedules as $\mathsf{Dev}$ and return a public and secret transcript) but in addition to the input given to $\mathsf{Dev}$ it is also given the big brother public key $\mathsf{pk_B} \in \{0,1\}^*$ as input.
- $\mathsf{B.Rec}$ is a PPT party that we call the *big brother recovery algorithm*. $\mathsf{B.Rec}$ functions like a PPT party $\mathsf{P}$ but in addition to be given $\mathsf{params}$ as input, it also receives the big brother secret key $\mathsf{sk_B}$ as input to its return algorithm $\mathsf{Return_{B.Rec}}$. The return function of $\mathsf{B.Rec}$ outputs information $\delta \in \{0,1\}^*$. We write $\delta \leftarrow \mathsf{Return_{B.Rec}}(\sigma_{\mathsf{B.Rec}}, \mathsf{sk_B})$.

The main difference between Def. 4.1.1 and Def. 4.1.2 is that in the latter, $\mathsf{B.Dev}$ and $\mathsf{B.Rec}$ are two parties that can interact with another party. In fact, $\mathsf{B.Rec}$ is interacting with $\mathsf{B.Dev}$ in its attempt to abuse its knowledge of $\mathsf{sk_B}$. This implies that $\mathsf{B.Rec}$ is a rather powerful recovery algorithm. For example in Chapter 6, we present a subversion of the $\mathsf{El\text{-}Gamal}$ scheme in the CPA security game. When $\mathsf{B.Rec}$ tries to recover information from $\mathsf{B.Dev}$, it can do so in an adaptive manner i.e. choose which messages to be send to $\mathsf{B.Dev}$ and act accordingly. In the case of the particular subversion of $\mathsf{El\text{-}Gamal}$ this makes no difference ($\mathsf{B.Rec}$ can choose any messages and still be able to recover information) but in other constructions it might make a difference that $\mathsf{B.Rec}$ is able to decide its own messages. We quantify how good $\mathsf{B.Rec}$ must be in the subsequent section.

## 4.2 Subversion

Consider a device Dev and its subverted counterpart B.Dev. Both devices output transcripts (in the case of an interactive subversion the transcripts are a concatenation of several messages): a public transcript $\tau_p$ and a private transcript $\tau_s$. As mentioned in the previous sections, these transcripts will, respectively, contain whatever output is considered public, thereby visible to an eavesdropper $\mathcal{E}$, and what output is regarded as private for everyone except a user $\mathcal{U}$. Subversions that are undetectable when considering eavesdroppers are weak while subversions that are also successful when considering users are strong. A distinguisher trying to distinguish Dev and B.Dev gets either $\tau_p$ or both transcripts depending on the type of distinguisher and must then guess from which device the transcript(s) is(are) generated.

The only possible mean (at least in our model) for users and eavesdroppers to detect a subversion is to consider the output of the device they are using, or observing in the case of an eavesdropper. We aim for subversions that avoids detection in a powerful way. This means we will require that B is able to fend off very refined tests. In Section 4.2.2 and Section 4.2.3, we define what it means to do so in respectively the non-interactive and the interactive case. But first we discuss the scope of our definitions.

### 4.2.1 Scope

There are some restrictions on the scope of the framework (included on purpose that is). In our definition of a subversion, we do not consider attempts that include side-channel analysis or other *physical* means of distinguishing devices. It may well be possible to construct a subliminal channel by using careful timing of a device Dev. E.g. if Dev represent an encryption algorithm then big brother could stutter the encryption of a message depending on a bit of the secret key at a specific position and thereby leaking information in the bit[3].

As mentioned in Section 4.1, we do not consider how big brother is going to replace the code. We therefore assume that by some magic divine power, big brother has a way of replacing Dev with B.Dev. This also implies that we make absolutely no conclusions on the practical exploitability level of our work - such work has been carried out in e.g. [13].

Finally, if a subversion fails to adhere to our definition it does not necessarily mean it can be detected. When a subversion fails to satisfy our undetectable condition, it implies that a distinguisher has non-negligible probability of distinguishing between a subverted device and the non-subverted device. What this means in a theoretical context is clear (non-negligible advantage) but it does not necessarily change the distinguishers ability to actually detect a subversion in practice (as such, a subversion satisfying our undetectability condition is very strong): the fact that a probability is non-negligible does not mean it is big, it simply means it does not decrease (insanely) fast. Specifically, a detection probability of $\frac{1}{2^{256}}$ would be non-negligible but totally useless in practice.

[3]This simplistic way of using side-channel does actually not represent a successful subversion in our model because a distinguisher could do the same side-channel analysis. Hence in deploying a side-channel subversion, big brother needs to be a bit more clever e.g. try to obfuscate the side-channel depending on the big brother public/secret key

### 4.2.2 Non-interactive subversion

Subversion of a non-interactive cryptographic game is the easiest to define. Details are as follows.

**Definition 4.2.1.** Given a secure non-interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$. A big brother $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ for $\mathsf{G}$ is a weak/strong $(t, K)$-subversion of $\mathsf{G}$ if

1. Big brother can recover information: define recovery advantage as

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{REC}}(\mathsf{B.Rec}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{REC}_{\mathsf{B}}^{\mathsf{B.Rec}}(n)\right] - p_0(n) \right|,$$

where the probability is over randomness used in the game $\mathsf{REC}_{\mathsf{B}}^{\mathcal{A}}(n)$, which is defined in Figure 4.2. $\mathsf{B}$ recovers information if there exists $N \in \mathbb{N}$ such that

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{REC}}(\mathsf{B.Rec}) > K$$

for all $n > N$ and $\mathsf{B.Rec}$ is a PPT algorithm.

2. Detection is impossible: for a PPT distinguisher $\mathcal{D}$, we define

$$\mathrm{Adv}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathsf{type\text{-}DETECT}}(\mathcal{D}) := 2 \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{type\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{D}}(n)\right] - \frac{1}{2} \right|$$

for type $\in \{\mathsf{weak}, \mathsf{strong}\}$ and probability is over randomness used in the game $\mathsf{type\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{A}}(n)$ defined in Figure 4.1. A PPT distinguisher $\mathcal{D}$ can not detect the subversion $\mathsf{B}$ if there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathsf{type\text{-}DETECT}}(\mathcal{D}) \leq \mathsf{negl}(n),$$

where $\mathcal{D}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$.

| **Game** $\mathsf{weak\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{E}}(n)$ | **Game** $\mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)$ |
|---|---|
| $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$ | $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$ |
| $b \leftarrow_R \{0,1\}$ | $b \leftarrow_R \{0,1\}$ |
| $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$ | $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$ |
| $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}}(\mathsf{pk_B}, \mathsf{params})$ | $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}}(\mathsf{pk_B}, \mathsf{params})$ |
| **Return** $(\hat{b} \overset{?}{=} b)$ | **Return** $(\hat{b} \overset{?}{=} b)$ |
| $\underline{\mathcal{O}}$ | $\underline{\mathcal{O}}$ |
| If $b = 0$: $(\tau_p, \tau_s) \leftarrow \mathsf{Dev}(1^n, \mathsf{params})$ | If $b = 0$: $(\tau_p, \tau_s) \leftarrow \mathsf{Dev}(1^n, \mathsf{params})$ |
| If $b = 1$: $(\tau_p, \tau_s) \leftarrow \mathsf{B.Dev}(1^n, \mathsf{params}, \mathsf{pk_B})$ | If $b = 1$: $(\tau_p, \tau_s) \leftarrow \mathsf{B.Dev}(1^n, \mathsf{params}, \mathsf{pk_B})$ |
| **Return** $\tau_p$ | **Return** $(\tau_p, \tau_s)$ |
| $\mathcal{E}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$. | $\mathcal{U}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$ |

**Figure 4.1:** Distinguish game for weak/strong subversion.

3. Preserve security: for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SEC}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

The left-hand side is defined as

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SEC}}(\mathcal{A}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] - p_0(n) \right|,$$

where the probability is taken over the randomness in Game $\mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)$ defined in Figure 4.2.

```
Game $\mathsf{REC}^{\mathcal{A}}_{\mathsf{B}}(n)$
    $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{B.Dev}(1^n, \mathsf{params}, \mathsf{pk_B})$
    $\delta \leftarrow \mathcal{A}(\mathsf{sk_B}, \mathsf{params}, \tau_p)$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

```
Game $\mathsf{SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$
    $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{B.Dev}(1^n, \mathsf{params}, \mathsf{pk_B})$
    $\delta \leftarrow \mathcal{A}(\mathsf{pk_B}, \mathsf{params}, \tau_p)$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 4.2:** Recovery (left) and preservability (right) game for big brother B

$t$ is a polynomial in $n$ and $K \in (0, 1]$. If (1) is true for arbitrary $t > 0$ we call B a weak/strong $K$-subversion of G and we call B a weak/strong subversion of G if there exist some constant $K$ such that B is a weak/strong $K$-subversion. If properties 1, 2 and 3 are satisfied we call B *recoverable, undetectable* and *security preserving*. Property 1, 2 and 3 above are referred to as *recoverability, undetectability* and *preservability*, respectively.

We give some comments on each game below:

**Recovery game:** The recovery game first generates parameters and then generates the big brother keys. B.Dev is then run with input the big brother public key and parameters, and the output is given to $\mathcal{A}$. Apart from being given the public transcript, $\mathcal{A}$ is also given the big brother secret key and parameters, and then outputs some information $\delta$ that is fed into the predicate function.

**Detection game:** To begin with $\mathcal{E}$ and $\mathcal{U}$ are given the big brother public key $\mathsf{pk_B}$. They can then query the oracle $\mathcal{O}$ at most $t$ times. The output comes either from Dev or B.Dev depending on the bit $b$ drawn in the game. In the weak undetectability game the answer from the oracle $\mathcal{O}$ is the public transcript $\tau_p$ while in the strong undetectability game the answer from the oracle $\mathcal{O}$ is both transcripts $\tau_p$ and $\tau_s$.

**Security game:** The only difference between the two games $\mathsf{REC}^{\mathcal{A}}_{\mathsf{B}}(n)$ and $\mathsf{SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$ is that in the former the adversary $\mathcal{A}$ is given the big brother secret key while in the latter this is not the case. That is, the games completely describe that obtaining the big brother secret key is enough to take advantage of the subversion. However, not knowing the secret key leaves an attacker with no chances of taking advantage of the subversion.

We emphasise that Def. 4.2.1 captures the user's or eavesdropper's inability to know which device is being used. Even if the detection advantage is rather large it is not immediately clear that detection is possible. A distingusher would simple not know what to look for. When detection advantage is small, big brother has essentially forced distingushers to rely on 'physical' means e.g. reverse-engineering or timing-analysis.

Since we are not much interested in the practical performance of big brother, we say that any big brother who satisfy the above definition with respected to a non-interactive game G implies that G is susceptible to a subversion attack.

Below we prove a consistency result: a non-interactive cryptographic game that is strongly subvertable is also weakly subvertable.

**Proposition 4.2.2.** *Let* $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be a non-interactive cryptographic game and assume* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *is a strong* $(t, K)$*-subversion of* G. *Then* B *is also a weak* $(t, K)$*-subversion of* G.

*Proof.* Since we have not changed either Dev or B, properties (1) and (3) in Def. 4.2.1 are still satisfied. We therefore only have to prove that property (2) is satisfied in the weak case.

Assume $\mathcal{E}$ is a PPT eavesdropper and consider the game weak-DETECT$^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. We then define a PPT user $\mathcal{U}$ such that

$$\mathrm{Adv}^{\text{weak-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{E}) = \mathrm{Adv}^{\text{strong-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}),$$

where $\mathrm{Adv}^{\text{strong-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U})$ is the advantage of $\mathcal{U}$ in the game strong-DETECT$^{\mathcal{U}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. We define $\mathcal{U}$ below: in the description $\mathcal{O}$ denotes the oracle from the game strong-DETECT$^{\mathcal{U}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$.

```
User $\mathcal{U}$
Input: $(\mathsf{pk_B}, \mathsf{params})$
1: Initialise list $L_{\tau_p}$
2: Make $t$ queries to $\mathcal{O}$
3: Let $L_{\tau_p}$ contain the public transcript answers from the queries to $\mathcal{O}$
4: $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}_\mathcal{U}}(\mathsf{pk_B}, \mathsf{params})$
5: Return $\hat{b}$

    Device oracle: $\mathcal{O}_\mathcal{U}$ $i$'th query
1: Return $L_{\tau_p}[i]$
```

Because $\mathcal{E}$ runs in polynomial-time, $\mathcal{U}$ also runs in polynomial-time. It is clear that $\mathcal{U}$ wins exactly when $\mathcal{E}$ guesses the correct bit. The probability of $\mathcal{E}$ guessing correctly when used as a subroutine by $\mathcal{U}$ is $\Pr\left[\mathsf{true} \leftarrow \text{weak-DETECT}^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)\right]$: this follows because the oracle answers to $\mathcal{E}$ when used as a subroutine by $\mathcal{U}$ are the same oracle answers $\mathcal{E}$ receives in the game weak-DETECT$^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$ (they are therefore also distributed identically). The parameters params and big brother public key $\mathsf{pk_B}$ are also the same in both situations. Because B is a strong $(t, K)$-subversion of G there exists a negligible function negl such that

$$\begin{aligned}\mathrm{Adv}^{\text{weak-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{E}) &= \alpha_0(n)\left|\Pr\left[\mathsf{true} \leftarrow \text{weak-DETECT}^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)\right] - p_0(n)\right| \\ &= \alpha_0(n)\left|\Pr\left[\mathsf{true} \leftarrow \text{strong-DETECT}^{\mathcal{U}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)\right] - p_0(n)\right| \\ &= \mathrm{Adv}^{\text{strong-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) \leq \mathsf{negl}(n),\end{aligned}$$

which proves what we want. □

We can actually reduce the number of properties required to prove that a big brother is a successful subversion. Namely, given a secure non-interactive cryptographic game G and a big brother B that satisfies strong undetectability, it is possible to prove that B also satisfy preservability i.e. it conforms to the same notion of security as G does.

**Proposition 4.2.3.** *Let* $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be a non-interactive cryptographic game and* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *a big brother for* G*. If* G *is secure and* B *is strongly undetectable, then* B *also preserves security i.e. for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\mathrm{Adv}^{\mathsf{SEC}}_{\mathsf{B}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

*Proof.* We start by constructing a reduction from preservability of B to undetectability of B. Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)$. We construct a PPT user $\mathcal{U}$, using $\mathcal{A}$ as a subroutine, that tries to distinguish between Dev and B.Dev. In constructing $\mathcal{U}$ we use the predicate function Pre, which is possible because in the strong undetectability game $\mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)$, $\mathcal{U}$ is given the secret transcript $\tau_s$. A detailed description of $\mathcal{U}$ is given below: in the description we use $\mathcal{O}$ to refer to the device oracle in the game $\mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)$.

```
User $\mathcal{U}$
Input: $(\mathsf{params}, \mathsf{pk_B})$
1: Query: $(\tau_p, \tau_s) \leftarrow \mathcal{O}$
2: $\delta \leftarrow \mathcal{A}(\mathsf{pk_B}, \mathsf{params}, \tau_p)$
3: $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
4: if $\mathsf{bool} = \mathsf{true}$ then
5:     Return $1$
6: else
7:     Return $0$
```

Because $\mathcal{A}$ and Pre are polynomial-time algorithms, $\mathcal{U}$ is also a polynomial-time algorithm. Note that we only need one query from $\mathcal{O}$. Hence it does not matter how many queries $\mathcal{U}$ is allowed to make (a distinguisher is always allowed to make at least one). Conditioning on the value of $b$ we obtain

$$\begin{aligned}
\mathrm{Adv}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathsf{strong\text{-}DETECT}}(\mathcal{U}) &= 2 \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)\right] - \frac{1}{2} \right| \\
&= 2 \cdot \left| \frac{1}{2} \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n) \mid b = 1\right] + \right. \\
&\qquad \left. \frac{1}{2} \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n) \mid b = 0\right] - \frac{1}{2} \right| \\
&= 2 \cdot \left| \frac{1}{2} \left(\Pr\left[\mathsf{bool} = \mathsf{true} \mid b = 1\right] + \Pr\left[\mathsf{bool} = \mathsf{false} \mid b = 0\right]\right) - \frac{1}{2} \right| \\
&= \left| \Pr\left[\mathsf{bool} = \mathsf{true} \mid b = 1\right] + \Pr\left[\mathsf{bool} = \mathsf{false} \mid b = 0\right] - 1 \right|. \qquad (4.1)
\end{aligned}$$

Let us now consider the two conditional probabilities separately. The first conditions on $b = 1$ which implies that the transcript returned by $\mathcal{O}$ is generated by B.Dev with input $\mathsf{params}$ and $\mathsf{pk_B}$. That is, the input to $\mathcal{A}$ has the same distribution as in the game $\mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)$. Therefore, the event that $\mathsf{bool} = \mathsf{true}$, has the same probability as $\mathcal{A}$ outputting a valid $\delta$ i.e.

$$\Pr\left[\mathsf{bool} = \mathsf{true} \mid b = 1\right] = \Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right]. \qquad (4.2)$$

The second probability conditions on $b = 0$, which means that the answer from $\mathcal{O}$ is computed by running Dev on input $\mathsf{params}$. The distribution of the input to $\mathcal{A}$ is therefore identical to the distribution of the input given to $\mathcal{A}$ in the modified game $\overline{\mathsf{SEC}}_{\mathsf{G}}^{\mathcal{A}}(n)$ defined in Figure 4.3. Note that this is a modification of the security game for G, where $\mathsf{pk_B}$ is given as additional input to the adversary $\mathcal{A}$.

Thus we have the following identity

$$\Pr\left[\mathsf{bool} = \mathsf{false} \mid b = 0\right] = \Pr\left[\mathsf{false} \leftarrow \overline{\mathsf{SEC}}_{\mathsf{G}}^{\mathcal{A}}(n)\right].$$

```
Game $\overline{\mathsf{SEC}}_{\mathsf{G}}^{\mathcal{A}}(n)$
    $\mathsf{params} \leftarrow \mathsf{Par}(1^n)$
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Dev}(1^n, \mathsf{params})$
    $\delta \leftarrow \mathcal{A}(\mathsf{pk_B}, \mathsf{params}, \tau_p)$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 4.3:** Modified security game for a non-interactive cryptographic game G.

But obviously since params and $\tau_p$ is completely independent of $\mathsf{pk_B}$ (and $\mathsf{sk_B}$) $\mathcal{A}$ learns no extra information compared to the original game. We can easily define a new PPT adversary $\mathcal{A}'$ for the game $\mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)$ that uses $\mathcal{A}$ as a subroutine to attack the security of G (we can generate identically distributed input since $\tau_p$ is independent of $\mathsf{pk_B}$), such that the following will be the case

$$\Pr\left[\mathsf{false} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right] = \Pr\left[\mathsf{false} \leftarrow \overline{\mathsf{SEC}}_{\mathsf{G}}^{\mathcal{A}}(n)\right]. \tag{4.3}$$

Using identities 4.1, 4.2 and 4.3 we obtain

$$\begin{aligned}
\mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}DETECT}}(\mathcal{U}) &= \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] + \Pr\left[\mathsf{false} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right] - 1\right| \\
&= \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] + \left(1 - \Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right]\right) - 1\right| \\
&= \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] - \Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right]\right| \\
&= \left|\left(\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] - p_0(n)\right) - \left(\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right] - p_0(n)\right)\right| \\
&\geq \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] - p_0(n)\right| - \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right] - p_0(n)\right|.
\end{aligned}$$

By multiplying $\alpha_0(n)$ $(> 0)$ on both sides and rearrange things a bit we get

$$\begin{aligned}
\alpha_0(n) \cdot \mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}DETECT}}(\mathcal{U}) + \mathrm{Adv}_{\mathsf{G}}^{\mathsf{SEC}}(\mathcal{A}') &= \alpha_0(n) \cdot \mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}DETECT}}(\mathcal{U}) + \\
&\qquad \alpha_0(n) \cdot \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}'}(n)\right] - p_0(n)\right| \\
&\geq \alpha_0(n) \cdot \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SEC}_{\mathsf{B}}^{\mathcal{A}}(n)\right] - p_0(n)\right| \\
&\geq \mathrm{Adv}_{\mathsf{B}}^{\mathsf{SEC}}(\mathcal{A}).
\end{aligned}$$

Since $\alpha_0(n)$ is a polynomial, the function $\alpha_0(n) \cdot \mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}DETECT}}(\mathcal{U})$ is negligible in $n$ because B is assumed strongly undetectable. Since we also assumed that G is secure, $\mathrm{Adv}_{\mathsf{G}}^{\mathsf{SEC}}(\mathcal{A}')$ is also negligible in $n$. Negligibility is invariant under addition implying that the left-hand side above is negligible in $n$. We conclude that $\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SEC}}(\mathcal{A})$ is negligible in $n$ as well and that B preserves security. □

By Propositon 4.2.3, we do not need to prove preservability when proving that a big brother is a strong subversion. It would be handy if we could get any of the other two properties in Def. 4.2.1 for free too. The examples below show that this is impossible without weakening the definition. The first example shows that there exists a big brother which is strongly undetectable but not recoverable while the second example provides an example of a big brother that is recoverable but not undetectable.

*Example* 4.2.4. Assume $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is a secure non-interactive cryptographic game. Then we define $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ as $\mathsf{B.Dev} \coloneqq \mathsf{Dev}$ (on all inputs) i.e. B.Dev does not utilize $\mathsf{pk_B}$. Clearly it is impossible to distinguish between Dev and B.Dev because they return the exact same output generated in the exact same way. But it is also impossible to construct a recovery function whose advantage is non-negligible because the existence of such an algorithm would imply that G is not secure: B.Rec can be used as a regular adversary (because B.Dev does not use $\mathsf{pk_B}$) for G and win the security game $\mathsf{SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$ with non-negligible probability.

*Example* 4.2.5. Assume $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is a secure non-interactive cryptographic game. Then we define B.Dev as Dev except that we define the output $(\tau_p^{\mathsf{B}}, \tau_s^{\mathsf{B}})$ of B.Dev as $\tau_p^{\mathsf{B}} = \tau_p || \tau_s$ ($||$ denotes concatenation of two bit strings) and $\tau_s^{\mathsf{B}} = \perp$, where $(\tau_p, \tau_s)$ was the original output defined by Dev. Obviously it is easy for a recovery algorithm to win the recovery game given $\tau_p^{\mathsf{B}}$ because B.Rec has access to all knowledge. On the other hand, it is trivial to distinguish between Dev and B.Dev.

Example 4.2.5 does not exactly follow the functional requirements meaning that by simply parsing the output from Dev it is possible for a distinguisher to distinguish between Dev and B.Dev. The construction in Figure 4.4 provides another example that satisfy functional requirements, and is recoverable but not undetectable (although it is a bit artificial). In the description, we have omitted the predicate function, base probability and base factor since they are not important.

```
Par(1^n)
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    params := $(\mathbb{G}, q, g)$
    Return params
Dev(1^n, params)
    Parse: params = $(\mathbb{G}, q, g)$
    $x \leftarrow_R \mathbb{Z}_q$
    $x' \leftarrow_R \mathbb{Z}_q$
    $y := g^x$
    $\tau_p := (\mathbb{G}, q, g, y, x')$
    $\tau_s := (\mathbb{G}, q, g, x)$
    Return $(\tau_p, \tau_s)$
```

A non-interactive cryptographic game G

```
Par(1^n)
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    params := $(\mathbb{G}, q, g)$
    Return params
B.Dev(1^n, params, pk_B)
    Parse: params = $(\mathbb{G}, q, g)$
    $x \leftarrow_R \mathbb{Z}_q$
    $x' \leftarrow_R \mathbb{Z}_q$
    $y := g^x$
    $\tau_p := (\mathbb{G}, q, g, y, x)$
    $\tau_s := (\mathbb{G}, q, g, x)$
    Return $(\tau_p, \tau_s)$
```

A big brother for the game G

**Figure 4.4:** Example of a big brother that is recoverable but not undetectable.

### 4.2.3 Interactive Subversion

Subversion of an interactive cryptographic game roughly follows the definition of a subversion of a non-interactive cryptographic game, but makes the games interactive.

**Definition 4.2.6.** Given a secure interactive cryptographic game $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$. A big brother $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ for G is a weak/strong $(t, K)$-subversion of G if

1. Big brother can recover information: define recovery advantage as

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{IN\text{-}REC}}(\mathsf{B.Rec}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}REC}_{\mathsf{B}}^{\mathsf{B.Rec}}(n)\right] - p_0(n) \right|,$$

where the probability is over randomness used in game $\mathsf{IN\text{-}REC}_{\mathsf{B}}^{\mathcal{A}}(n)$, which is defined in Figure 4.6. B recovers information if there exists $N \in \mathbb{N}$ such that

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{IN\text{-}REC}}(\mathsf{B.Rec}) > K$$

for all $n > N$ and B.Rec is a PPT party.

2. Detection is impossible: for a PPT distinguisher $\mathcal{D}$, we define

$$\mathrm{Adv}^{\textsf{type-IN-DETECT}}_{\textsf{Dev},\textsf{B.Dev}}(\mathcal{D}) := 2\left|\Pr\left[\textsf{true} \leftarrow \textsf{type-IN-DETECT}^{\mathcal{D}}_{\textsf{Dev},\textsf{B.Dev}}(n)\right] - \frac{1}{2}\right|$$

for type $\in$ {weak, strong} and probability is taken over the randomness used in the game type-IN-DETECT$^{\mathcal{D}}_{\textsf{Dev},\textsf{B.Dev}}(n)$ defined in Figure 4.5. A PPT distinguisher $\mathcal{D}$ can not detect the subversion B if there exists a negligible function negl such that

$$\mathrm{Adv}^{\textsf{type-IN-DETECT}}_{\textsf{Dev},\textsf{B.Dev}}(\mathcal{D}) \leq \textsf{negl}(n),$$

where $\mathcal{D}$ is allowed to make $t$ queries to oracle $\mathcal{O}$.

```
Game weak-IN-DETECT$^{\mathcal{E}}_{\textsf{Dev},\textsf{B.Dev}}(n)$
    Run Par$(1^n)$
    Let params be the initial state
    $(\textsf{pk}_\textsf{B}, \textsf{sk}_\textsf{B}) \leftarrow \textsf{B.Gen}(1^n, \textsf{params})$
    Send $\textsf{pk}_\textsf{B}$ to $\mathcal{E}$
    $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}}$
    Return $(\hat{b} \overset{?}{=} b)$
$\mathcal{O}$
    If $b = 0$: { Run Dev with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \textsf{Return}_\textsf{Dev}(\sigma_\textsf{Dev})$}
    If $b = 1$:{ Run B.Dev$(\textsf{pk}_\textsf{B})$ with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \textsf{Return}_\textsf{B.Dev}(\sigma_\textsf{B.Dev})$}
    Return $\tau_p$
```

$\mathcal{E}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$.

Device and $\mathcal{E}$ start with initial state params for each query.

```
Game strong-IN-DETECT$^{\mathcal{U}}_{\textsf{Dev},\textsf{B.Dev}}(n)$
    Run Par$(1^n)$
    Let params be the initial state
    $(\textsf{pk}_\textsf{B}, \textsf{sk}_\textsf{B}) \leftarrow \textsf{B.Gen}(1^n, \textsf{params})$
    Send $\textsf{pk}_\textsf{B}$ to $\mathcal{U}$
    $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}}$
    Return $(\hat{b} \overset{?}{=} b)$
$\mathcal{O}$
    If $b = 0$: { Run Dev with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \textsf{Return}_\textsf{Dev}(\sigma_\textsf{Dev})$}
    If $b = 1$:{ Run B.Dev$(\textsf{pk}_\textsf{B})$ with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \textsf{Return}_\textsf{B.Dev}(\sigma_\textsf{B.Dev})$}
    Return $(\tau_p, \tau_s)$
```

$\mathcal{U}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$

Device and $\mathcal{U}$ start with initial state params for each query.

**Figure 4.5:** Distinguish game for weak/strong subversion.

3. Preserve security: for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\mathrm{Adv}^{\textsf{IN-SEC}}_{\textsf{B}}(\mathcal{A}) \leq \textsf{negl}(n).$$

The left-hand side is defined as

$$\mathrm{Adv}^{\textsf{IN-SEC}}_{\textsf{B}}(\mathcal{A}) := \alpha_0(n) \cdot \left|\Pr\left[\textsf{true} \leftarrow \textsf{IN-SEC}^{\mathcal{A}}_{\textsf{B}}(n)\right] - p_0(n)\right|$$

where the probability is over the randomness used in the game IN-SEC$^{\mathcal{A}}_{\textsf{B}}(n)$ defined in Figure 4.6.

$t$ is a polynomial in $n$ and $K \in (0, 1]$. If (1) is true for arbitrary $t > 0$ we call B a weak/strong $K$-subversion of G and we call B a weak/strong subversion of G if there exist some constant $K$ such that B is a weak/strong $K$-subversion. If properties 1, 2 and 3 are satisfied we call B *recoverable*, *undetectable* and *security preserving*. Property 1, 2 and 3 above are referred to as *recoverability*, *undetectability* and *preservability*, respectively.

We give some comments on each game below:
**Recovery game:** The recovery game first runs the setup algorithm Par and distributes the result to both $\mathcal{A}$ and B.Dev. Thereafter, the big brother keys $(\textsf{pk}_\textsf{B}, \textsf{sk}_\textsf{B})$ are generated and $\textsf{pk}_\textsf{B}$ is given

```
Game IN-REC$_{\mathsf{B}}^{\mathcal{A}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Send $\mathsf{pk_B}$ to $\mathcal{A}$
    Run $\mathsf{B.Dev}(\mathsf{pk_B})$ with $\mathcal{A}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return}_{\mathsf{B.Dev}}(\sigma_{\mathsf{B.Dev}})$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}}, \mathsf{sk_B})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

```
Game IN-SEC$_{\mathsf{B}}^{\mathcal{A}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Send $\mathsf{pk_B}$ to $\mathcal{A}$
    Run $\mathsf{B.Dev}(\mathsf{pk_B})$ with $\mathcal{A}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return}_{\mathsf{B.Dev}}(\sigma_{\mathsf{B.Dev}})$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 4.6:** Recovery (left) and preservability (right) game for big brother B.

to $\mathcal{A}$. Afterwards $\mathcal{A}$ and B.Dev starts their interaction. In the end $\mathcal{A}$ must output information $\delta$ that will make the predicate function return true. As mentioned in Section 4.1.2 $\mathcal{A}$ can adaptively choose its messages to B.Dev, making it very powerful.

**Detection game:** In the strong undetectability game, the user $\mathcal{U}$ or eavesdropper $\mathcal{E}$ are first given the parameters params before starting the interaction with one of the two devices Dev or B.Dev. $\mathcal{U}$ or $\mathcal{E}$ can at most start $t$ interactions with a device and for each interaction, both the device and $\mathcal{U}$ (or $\mathcal{E}$), must start from the initial state params (i.e. they reset). In addition, for each interaction user $\mathcal{U}$ both learns the public output from the device and the private input while eavesdropper $\mathcal{E}$ learns only $\tau_p$.

**Security game:** As in the non-interactive case the only difference between $\mathsf{IN\text{-}SEC}_{\mathsf{B}}^{\mathcal{A}}(n)$ and $\mathsf{IN\text{-}REC}_{\mathsf{B}}^{\mathcal{A}}(n)$ is that in the latter $\mathcal{A}$ is given the big brother secret key while in the former this is not the case.

We now aim to prove the same propositions as in the non-interactive case. We begin by proving that a strong subversion implies a weak subversion.

**Proposition 4.2.7.** *Let* $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be an interactive cryptographic game and assume* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *is a strong* $(t, K)$*-subversion of* G*. Then* B *is also a weak* $(t, K)$*-subversion of* G.

The proof is a little different from Propositon 4.2.2 because the distinguisher can now interact with each device and choose its own messages. This is fixed by constructing a user that acts like a proxy between a weak distinguisher and the device.

*Proof.* Property (1) and (3) of Def. 4.2.6 are clearly satisfied. Let $\mathcal{E}$ be a PPT eavesdropper and consider Game $\mathsf{weak\text{-}IN\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{E}}(n)$. We will construct a user $\mathcal{U}$ that attacks the strong detection game using $\mathcal{E}$ as a subroutine. We prove

$$\mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{weak\text{-}IN\text{-}DETECT}}(\mathcal{E}) = \mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}IN\text{-}DETECT}}(\mathcal{U}). \tag{4.4}$$

We give the details of $\mathcal{U}$ below: $\mathcal{O}$ is the device oracle in the game $\mathsf{strong\text{-}IN\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{U}}(n)$.

It is obvious that $\mathcal{U}$ is a PPT party because $\mathcal{E}$ is. Now consider the answers $\mathcal{E}$ receives in its interaction with $\mathcal{U}$. These answers are equal (and therefore distributed identically) to the answers $\mathcal{E}$ would receive in the game $\mathsf{weak\text{-}IN\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{E}}(n)$. Since $\mathcal{U}$ outputs $\hat{b} = b$ if and only if $\mathcal{E}$ outputs $\hat{b} = b$ we have

$$\begin{aligned}
\mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{weak\text{-}IN\text{-}DETECT}}(\mathcal{E}) &= \alpha_0(n) \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{weak\text{-}IN\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{E}}(n)\right] - p_0(n) \right| \\
&= \alpha_0(n) \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}IN\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{U}}(n)\right] - p_0(n) \right| \\
&= \mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{strong\text{-}IN\text{-}DETECT}}(\mathcal{U}) \leq \mathsf{negl}(n)
\end{aligned}$$

```
User $\mathcal{U}$
Input: $(\mathsf{pk_B}, \mathsf{params})$
1: Give $(\mathsf{pk_B}, \mathsf{params})$ to $\mathcal{E}$
2: $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}_\mathcal{E}}$
3: Return $\hat{b}$

Device oracle: $\mathcal{O}_\mathcal{E}$ $i$'th query

1: Query the oracle $\mathcal{O}$ that starts an interaction with device $D$, which is either device Dev or device B.Dev, and at the same time $\mathcal{U}$ starts an interaction with $\mathcal{E}$
2: $\mathcal{U}$ acts as a proxy between the two interactions: messages received from $\mathcal{E}$ are send to $D$, and messages received from $D$ are send to $\mathcal{E}$
3: When $\mathcal{U}$ in the end receives the public and secret transcripts from the interaction with $D$, send the public transcript to $\mathcal{E}$
```

for some negligible function negl. Hence, 4.4 follows directly. □

Next we prove an interactive equivalent to Propositon 4.2.3. The proof is very similar to the proof of the proposition in the non-interactive case except that $\mathcal{U}$ must now act as a proxy.

**Proposition 4.2.8.** *Let* $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be an interactive cryptographic game and* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *a big brother for* G. *If* G *is secure and* B *is strongly undetectable, then* B *also preserves security i.e. for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{IN\text{-}SEC}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

*Proof.* Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathsf{IN\text{-}SEC}_{\mathsf{B}}^{\mathcal{A}}(n)$. We construct a PPT user $\mathcal{U}$, using $\mathcal{A}$ as a subroutine, that tries to distinguish between devices Dev and B.Dev in the game $\mathsf{strong\text{-}IN\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)$. A detailed description of $\mathcal{U}$ is given below: in the description we use $\mathcal{O}$ to refer to the device oracle in the game $\mathsf{strong\text{-}IN\text{-}DETECT}_{\mathsf{Dev},\mathsf{B.Dev}}^{\mathcal{U}}(n)$.

```
User $\mathcal{U}$
Input: $(\mathsf{pk_B}, \mathsf{params})$
1: Give $(\mathsf{pk_B}, \mathsf{params})$ to $\mathcal{A}$
2: Query $\mathcal{O}$ that starts an interaction between $\mathcal{U}$ and either Dev or B.Dev, and $\mathcal{U}$ simultaneously starts an interaction with $\mathcal{A}$
3: Let $\mathcal{U}$ act as a proxy
4: $\delta \leftarrow \mathsf{Return}_\mathcal{A}(\sigma_\mathcal{A})$
5: $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
6: if $\mathsf{bool} = \mathsf{true}$ then
7:     Return $1$
8: else
9:     Return $0$
```

Conditioning on the value of $b$ we obtain

$$\mathrm{Adv}^{\mathsf{strong\text{-}IN\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) = \left|\Pr\left[\mathsf{bool} = \mathsf{true} \mid b = 1\right] + \Pr\left[\mathsf{bool} = \mathsf{false} \mid b = 0\right] - 1\right|. \tag{4.5}$$

We proceed by considering each case separately. $b = 1$ implies that the transcript returned by $\mathcal{O}$ is generated by interacting with $\mathsf{B.Dev}$ (through $\mathcal{U}$) with parameters $\mathsf{params}$ and big brother public key $\mathsf{pk_B}$. That is, the messages to $\mathcal{A}$, sent by $\mathcal{U}$, are distributed identically as in the game $\mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$. Therefore, the event that $\mathsf{bool} = \mathsf{true}$, has the same probability as $\mathcal{A}$ outputs a valid $\delta$ i.e.

$$\Pr\left[\mathsf{bool} = \mathsf{true} \mid b = 1\right] = \Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)\right]. \tag{4.6}$$

Second case is $b = 0$, which means that the answer from $\mathcal{O}$ is computed by interacting with $\mathsf{Dev}$ (through $\mathcal{U}$) with parameters $\mathsf{params}$. The distribution of the input to $\mathcal{A}$ is therefore identical to the distribution given to $\mathcal{A}$ in the modified game $\overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$ defined in Figure 7.3. Note that this is a modification of the security game for $\mathsf{G}$.

```
Game $\overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Give $\mathsf{pk_B}$ to $\mathcal{A}$
    Run Dev with $\mathcal{A}$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 4.7:** Modified security game for a non-interactive cryptographic game $\mathsf{G}$.

Thus we have the following identity

$$\Pr\left[\mathsf{bool} = \mathsf{false} \mid b = 0\right] = \Pr\left[\mathsf{false} \leftarrow \overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)\right].$$

Since $\mathsf{params}$ and $\tau_p$ are independent of $\mathsf{pk_B}$, we can define a PPT adversary $\mathcal{A}'$ for the game $\mathsf{IN\text{-}SEC}^{\mathcal{A}'}_{\mathsf{G}}(n)$ that uses $\mathcal{A}$ as a subroutine, such that

$$\Pr\left[\mathsf{false} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}'}_{\mathsf{G}}(n)\right] = \Pr\left[\mathsf{false} \leftarrow \overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)\right]. \tag{4.7}$$

Using identities 4.5, 4.6 and 4.7 we obtain

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{strong\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) &= \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)\right] + \Pr\left[\mathsf{false} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}'}_{\mathsf{G}}(n)\right] - 1\right| \\
&= \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)\right] - \Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}'}_{\mathsf{G}}(n)\right]\right| \\
&\geq \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)\right] - p_0(n)\right| - \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}'}_{\mathsf{G}}(n)\right] - p_0(n)\right|.
\end{aligned}$$

By multiplying $\alpha_0(n)$ on both sides and rearrange things a bit we get

$$\alpha_0(n) \cdot \left(\mathrm{Adv}^{\mathsf{strong\text{-}IN\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) + \mathrm{Adv}^{\mathsf{IN\text{-}SEC}}_{\mathsf{G}}(\mathcal{A}')\right) \geq \mathrm{Adv}^{\mathsf{IN\text{-}SEC}}_{\mathsf{B}}(\mathcal{A}).$$

We conclude that $\mathrm{Adv}^{\mathsf{IN\text{-}SEC}}_{\mathsf{B}}(\mathcal{A})$ is negligible in $n$ and that $\mathsf{B}$ preserves security. □

As in the non-interactive case the two first properties in Def. 4.2.6 can not be left out without changing the power of the definition.

*Example* 4.2.9. The example from Example 4.2.4 works here as well.

*Example* 4.2.10. The examples from Example 4.2.5 works here as well because one can imagine an interaction between a device and another party $\mathsf{P}$ where $\mathsf{P}$ does not send any messages.

In general a non-interactive cryptographic game can be seen as a special case of an interactive cryptographic game.

CHAPTER 5

# Hardness Assumption Subversion

In this chapter we present strong (canonical) subversions of two different non-interactive cryptographic games that models two different, but related, hardness assumptions: CDH and DDH. For each hardness assumption we define a non-interactive cryptographic game and prove that security of this cryptographic game is equivalent to the hardness assumption relative to a suitable generation algorithm. After the definition of the two non-interactive cryptographic game, we construct a big brother for each game and carefully prove that each big brother is a strong subversion.

The main idea used in this chapter is discussed in section Section 5.1Section 5.2 defines an equivalent non-interactive cryptographic game for the CDH hardness assumption. Section 5.3 defines a non-interactive cryptographic game that is equivalent to the DDH hardness assumption. In each section, we also construct a big brother for each cryptographic game. In addition it is proven that each big brother is a strong subversion. The two subversions essentially use the same trick to enable a strong subversion and their proofs are very similar. This fact is elaborated upon in Section 5.4 where we prove that undetectability of the DDH subversion follows from undetectability of the CDH subversion. This relation is further explored in Chapter 6 where we subvert the El-Gamal public-key encryption scheme whose security depends heavily on the hardness of the DDH game.

## 5.1 Main Idea

In this section, we present the main idea which is used in the next two sections. The technique is a subversion strategy that makes use of the Random Oracle Model and was first used by Young and Yung [27].

Consider the generation of two elements $g^a$ and $g^b$ in some cyclic group $\mathbb{G}$ with order $q$ and generator $g$. $a$ and $b$ are uniform elements in $\mathbb{Z}_q$. The insight of Young and Yung is that it is possible to change the generation of $g^a$ and $g^b$ in such a way that $b$ can be learned from the knowledge of $g^a$ and an El-Gamal key pair $(g^x, x)$ (the element $g^x$ is the public key and $x$ is the secret key chosen uniformly from $\mathbb{Z}_q$). In an El-Gamal encryption the public key is used, together with an ephemeral key $y \leftarrow_R \mathbb{Z}_q$, to scramble a message $m$ producing a ciphertext $c = (c_1, c_2)$ (see Def. 2.6.5). In relation with the generation of $g^a$ and $g^b$ it is possible to arrange the 'encryption' such that $g^a = c_1 = c_2$ (note $a$ is used as ephemeral key) with $g^x$ being used as the public key i.e.

there exists $\eta \in \mathbb{G}$ such that $(g^x)^a \cdot \eta = g^a$: simply choose $\eta := (g^x)^{-a} g^a$. Using a random oracle $\mathcal{R}$ and the value $\eta$, we can generate $b$: $b := \mathcal{R}(\eta)$. In summary: given $g^x$ we do

$$a \leftarrow_R \mathbb{Z}_q; \quad \eta := (g^x)^{-a} g^a; \quad b := \mathcal{R}(\eta); \quad \textbf{Return } (g^a, g^b).$$

Since $g^a$ is the ciphertext of an El-Gamal encryption of the message $\eta$ it is possible to recover $\eta$ if $x$ is known! To see this we compute (El-Gamal decryption):

$$(g^a)^{-x} g^a = (g^x)^{-a} g^a = \eta.$$

From $\eta$ it is possible to compute $b$ as advertised.

## 5.2 Computational Diffie Hellman Subversion

The computational diffie hellman (CDH) problem says that given two elements $g^x$ and $g^y$ from a cyclic group $\mathbb{G} = \langle g \rangle$ of order $q$, it is hard to compute $g^{xy}$. $x$ and $y$ are uniform elements from $\mathbb{Z}_q$. In Def. 2.5.2 we formally defined what it means for the CDH problem to be hard relative to a group-generation algorithm $\mathcal{G}$.

In the following, we define a non-interactive cryptographic game $\mathsf{G}_{\mathsf{CDH}}$ ( Figure 5.1) that models the CDH hardness game. In Theorem 5.2.1 we prove that $\mathsf{G}_{\mathsf{CDH}}$ is secure when the CDH problem is hard relative to $\mathcal{G}$. Theorem 5.2.3 proves that the big brother $\mathsf{B}_{\mathsf{CDH}}$ defined in Figure 5.2, strongly subverts $\mathsf{G}_{\mathsf{CDH}}$.

**Theorem 5.2.1.** *Let $\mathcal{G}(1^n)$ be defined as in Def. 2.5.2. The non-interactive cryptographic game $\mathsf{G}_{\mathsf{CDH}}$ defined in Figure 5.1 is secure (relative to $\mathcal{G}$) when the* CDH *problem is hard relative to $\mathcal{G}$.*

```
Par($1^n$)
        $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
        params := $(\mathbb{G}, q, g)$
        Return params
Dev_CDH($1^n$, params)
        Parse: params = $(\mathbb{G}, q, g)$
        $x, y \leftarrow_R \mathbb{Z}_q$
        $h_1 := g^x$
        $h_2 := g^y$
        $\tau_p := (\mathbb{G}, q, g, h_1, h_2)$
        $\tau_s := (\mathbb{G}, q, g, g^{xy})$
        Return $(\tau_p, \tau_s)$
Pre($\tau_p, \tau_s, \sigma$)
        Parse: $\tau_s = (\mathbb{G}, q, g, g^{xy})$
        Return $(\sigma \overset{?}{=} g^{xy})$
$(p_0(n), \alpha_0(n)) := (0, 1)$
```

**Figure 5.1:** Non-interactive cryptographic game $\mathsf{G}_{\mathsf{CDH}}$.

*Proof.* Given a PPT adversary $\mathcal{A}$. We construct another PPT adversary $\mathcal{A}'$ such that

$$\mathrm{Adv}^{\mathsf{SEC}}_{\mathsf{G}_{\mathsf{CDH}}}(\mathcal{A}) = \mathrm{Adv}^{\mathsf{CDH}}_{\mathcal{G}}(\mathcal{A}'), \tag{5.1}$$

where $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{A}')$ is the advantage of $\mathcal{A}'$ in Game $\mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ and $\mathrm{Adv}_{\mathsf{G}_{\mathsf{CDH}}}^{\mathsf{SEC}}(\mathcal{A})$ is the advantage of $\mathcal{A}$ in Game $\mathsf{SEC}_{\mathsf{G}_{\mathsf{CDH}}}^{\mathcal{A}}(n)$. We first describe the attack by $\mathcal{A}$ more concretely.

**Game 0.** We write Def. 3.3.2 more algorithmically in the context of $\mathcal{A}$ being the adversary.

```
Game 0
1: $(\mathbb{G}, q, g) \leftarrow \mathsf{Par}(1^n)$
2: $x, y \leftarrow_R \mathbb{Z}_q$; $h_1 = g^x$; $h_2 = g^y$
3: $\hat{h} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2)$
```

Let $S_0$ be the event that $\hat{h} = g^{xy}$ in Game 0. Obviously $\mathrm{Adv}_{\mathsf{G}_{\mathsf{CDH}}}^{\mathsf{SEC}}(\mathcal{A}) = \Pr[S_0]$. We proceed by defining adversary $\mathcal{A}'$, which tries to compute $g^{xy}$ using $\mathcal{A}$ as a subroutine.

```
Adversary $\mathcal{A}'$
Input: $(\mathbb{G}, q, g, h_1, h_2)$
1: $\hat{h} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2)$
2: Return $\hat{h}$
```

$\mathcal{A}'$ is a PPT algorithm because $\mathcal{A}$ is a PPT algorithm. When the input to $\mathcal{A}'$ is of the form $(\mathbb{G}, q, g, g^x, g^y)$, computation proceed just as in Game 0. Hence

$$\Pr\left[x, y \leftarrow_R \mathbb{Z}_q;\ \hat{h} \leftarrow \mathcal{A}'(\mathbb{G}, q, g, g^x, g^y)\ :\ \hat{h} = g^{xy}\right] = \Pr[S_0]\,.$$

The left-hand side is exactly the advantage of $\mathcal{A}'$ in Game $\mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ and 5.1 follows. $\square$

The above theorem is not surprising at all: the only difference between the game defining security of $\mathsf{G}_{\mathsf{CDH}}$ and the game defining CDH hardness is purely syntactical. Therefore the following remark is not surprising as well.

*Remark* 5.2.2. Given the same generation algorithm $\mathcal{G}$ as in Theorem 5.2.1. The CDH problem is hard relative to $\mathcal{G}$ if the non-interactive cryptographic game $\mathsf{G}_{\mathsf{CDH}}$ is secure (relative to $\mathcal{G}$). It is trivial to see that an adversary that can win Game $\mathsf{CDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ with non-negligible probability, can also be used as an adversary in Game $\mathsf{SEC}_{\mathsf{G}_{\mathsf{CDH}}}^{\mathcal{A}}(n)$ and win with the same non-negligible probability.

Remark 5.2.2 shows that $\mathsf{G}_{\mathsf{CDH}}$ is just a reformulation of the CDH hardness assumption relative to a suitable group generation algorithm $\mathcal{G}$. What we have gained is a convenient way of describing the hardness assumption with respect to subversion. In this context, Theorem 5.2.3 below shows that the CDH hardness assumption is completely susceptible to subversion attacks.

**Theorem 5.2.3.** *The big brother* $\mathsf{B}_{\mathsf{CDH}}$ *defined in Figure 5.2 is a strong subversion of* $\mathsf{G}_{\mathsf{CDH}}$ *if the* CDH *problem is hard relative to* $\mathcal{G}$.

*Proof.* We first prove that $\mathsf{B}_{\mathsf{CDH}}$ is recoverable and thereafter we prove that $\mathsf{B}_{\mathsf{CDH}}$ is strongly undetectable. Since $\mathsf{G}_{\mathsf{CDH}}$ is secure preservability of $\mathsf{B}_{\mathsf{CDH}}$ follows from Propositon 4.2.3. The proof of recoverability is a straightforward symbol game, where one just has to check that the computation by B.Rec is actually computing the correct values. The proof of strong undetectability

```
B.Gen($1^n$, params)
    Parse: params $= (\mathbb{G}, q, g)$
    $s \leftarrow_R \mathbb{Z}_q$
    $h := g^s$
    $\mathsf{pk_B} := (\mathbb{G}, q, g, h)$
    $\mathsf{sk_B} := (\mathbb{G}, q, g, s)$
    Return $(\mathsf{pk_B}, \mathsf{sk_B})$
B.Dev$_{\mathsf{CDH}}$($1^n$, $\mathsf{pk_B}$, params)
    Parse: params $= (\mathbb{G}, q, g)$
    Parse: $\mathsf{pk_B} = (\mathbb{G}, q, g, h)$
    $x \leftarrow_R \mathbb{Z}_q$
    $h_1 := g^x$
    $\eta := h^{-x} h_1$
    $y := \mathcal{R}(\eta)$
    $h_2 := g^y$
    $\tau_p := (\mathbb{G}, q, g, h_1, h_2)$
    $\tau_s := (\mathbb{G}, q, g, g^{xy})$
    Return $(\tau_p, \tau_s)$
B.Rec($\mathsf{sk_B}$, params, $\tau_p$)
    Parse: $\tau_p = (\mathbb{G}, q, g, h_1, h_2)$
    Parse: $\mathsf{sk_B} = (\mathbb{G}, q, g, s)$
    $\hat{\eta} := h_1^{-s} h_1$
    $\hat{y} := \mathcal{R}(\hat{\eta})$
    Return $h_1^{\hat{y}}$
$\mathcal{R}$ is a random oracle with domain and range $\mathbb{Z}_q$.
```

**Figure 5.2:** Big brother subversion $\mathsf{B_{CDH}}$ of $\mathsf{G_{CDH}}$.

is a little bit more intriguing and rely heavily on the assumption that the CDH problem is hard relative to $\mathcal{G}$ and random oracle properties.

$\mathsf{B_{CDH}}$ **is recoverable**

Because B.Rec is only doing basic operations, it is a PPT algorithm. From the execution of B.Gen we have $h = g^s$ in the cyclic group $\langle g \rangle = \mathbb{G}$ of order $q$. After executing B.Dev$_{\mathsf{CDH}}$ we have $h_1 = g^x$ and $h_2 = g^y$ where $y$ is computed by $\eta = h^{-x} h_1 = h^{-x} g^x$ and then setting $y$ equal to $\mathcal{R}(\eta)$.

When $\mathsf{B_{CDH}}$ executes B.Rec with input $(\mathbb{G}, q, g, s, g^x, g^y)$, B.Rec will first compute

$$\hat{\eta} = h_1^{-s} h_1 = g^{-sx} g^x = (g^s)^{-x} g^x = h^{-x} g^x = \eta.$$

By properties of the random oracle $\mathcal{R}$, B.Rec can then compute $\hat{y} = \mathcal{R}(\hat{\eta}) = y$. Finally, B.Rec outputs

$$h_1^y = g^{xy}.$$

B.Rec will therefore always succeed. Hence

$$\mathrm{Adv}^{\mathsf{REC}}_{\mathsf{B_{CDH}}}(\mathsf{B.Rec}) = 1.$$

$\mathsf{B_{CDH}}$ **is undetectable**

Let $\mathcal{U}$ be a PPT user trying to distinguish between $\mathsf{Dev_{CDH}}$ and B.Dev$_{\mathsf{CDH}}$. Consider the game strong-DETECT$^{\mathcal{U}}_{\mathsf{Dev_{CDH}},\mathsf{B.Dev_{CDH}}}(n)$ and let $\mathcal{O}_{\mathsf{CDH}}$ be the oracle used in the game. We may WLOG assume that $\mathcal{U}$ makes exactly $t = t(n)$ queries to $\mathcal{O}_{\mathsf{CDH}}$ (if not, we can define a new adversary that makes exactly $t$ queries but ignores some of them). We first make some syntactical changes through Games 0-1. Both $\mathcal{U}$ and $\mathcal{O}_{\mathsf{CDH}}$ queries the same random oracle $\mathcal{R}$, and so, the properties of a random oracle is consistent over the queries made from both $\mathcal{U}$ and $\mathcal{O}_{\mathsf{CDH}}$. Especially, if $\lambda$ has been queried to $\mathcal{R}$, the same value will be returned if $\lambda$ is ever queried again, no matter whether $\mathcal{U}$ or $\mathcal{O}_{\mathsf{CDH}}$ executes the query.

**Game 0:** We write Game strong-DETECT$^{\mathcal{U}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)$ more algorithmically in the context of user $\mathcal{U}$ being the distinguisher. We leave out the description of the random oracle $\mathcal{R}$, because we do not modify it going from Game 0 to Game 1.

```
Game 0
1: $(\mathbb{G}, q, g) \leftarrow \mathsf{Par}(1^n)$
2: $b \leftarrow_R \{0,1\}$
3: $s \leftarrow_R \mathbb{Z}_q$
4: $h := g^s$
5: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\mathsf{CDH}},\mathcal{R}}(\mathbb{G}, q, g, h)$

    Device oracle: $\mathcal{O}_{\mathsf{CDH}}$ $i$'th query
1: $x_i \leftarrow_R \mathbb{Z}_q$
2: $h_{1,i} := g^{x_i}$
3: if $b = 0$ then
4:     $y_i \leftarrow_R \mathbb{Z}_q$
5: else
6:     $\eta_i := h^{-x_i} h_{1,i}$
7:     $y_i := \mathcal{R}(\eta_i)$
8: $h_{2,i} := g^{y_i}$
9: Return $(\mathbb{G}, q, g, h_{1,i}, h_{2,i}, g^{x_i y_i})$
```

Let $S_0$ denote the event $\hat{b} = b$ in Game 0. Then

$$\Pr[S_0] = \Pr\left[\mathsf{true} \leftarrow \text{strong-DETECT}^{\mathcal{U}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)\right].$$

We do a syntactic transformation into Game 1.

**Game 1:** We reprogram the oracle $\mathcal{O}_{\mathsf{CDH}}$ such that $x_i$'s are computed in the beginning of the game. Knowing $x_i$ we also compute $h_{1,i}$ and $\eta_i$ in advance. Since we know that $\mathcal{U}$ will make exactly $t$ queries to $\mathcal{O}_{\mathsf{CDH}}$, we can sample a precise amount of elements. We remark that the sampling is performed using fresh randomness.

Let $S_1$ be the event that $\hat{b} = b$ in Game 1. Since the change is merely syntactic we have $\Pr[S_1] = \Pr[S_0]$. We find it easier to argue from the perspective of Game 1, than directly from Game 0. Additionally, it is now possible for the challenger to observe when one of the $\eta_i$'s is queried to the random oracle. In the execution of Game 1 we define two events:

| | | |
|---|---|---|
| Query | : | The event that, at any point during its execution, $\mathcal{U}$ queries any of $\eta_1, \dots, \eta_t$ to $\mathcal{R}$. |
| Success | : | The event that $\hat{b} = b$. |

**Game 1**

```
1: $(\mathbb{G}, q, g) \leftarrow \mathsf{Par}(1^n)$
2: $b \leftarrow_R \{0,1\}$
3: $s \leftarrow_R \mathbb{Z}_q$
4: $h := g^s$
5: $x_1, x_2, \ldots, x_t \leftarrow_R \mathbb{Z}_q$
6: $h_{1,i} := g^{x_i}$, for $i = 1, 2, \ldots, t$
7: $\eta_i := h^{x_i} h_{1,i}$, for $i = 1, 2, \ldots, t$
8: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\mathsf{CDH}}, \mathcal{R}}(\mathbb{G}, q, g, h)$
```

Device oracle: $\mathcal{O}_{\mathsf{CDH}}$ $i$'th query

```
1: if $b = 0$ then
2:     $y_i \leftarrow_R \mathbb{Z}_q$
3: else
4:     $y_i := \mathcal{R}(\eta_i)$
5: $h_{2,i} := g^{y_i}$
6: Return $(\mathbb{G}, q, g, h_{1,i}, h_{2,i}, g^{x_i y_i})$
```

Since $\mathsf{Success} = \left(\mathsf{Success} \wedge \overline{\mathsf{Query}}\right) \vee (\mathsf{Success} \wedge \mathsf{Query})$ and the two sets have empty intersection, we have

$$\begin{aligned} \Pr[S_1] &= \Pr[\mathsf{Success}] \\ &= \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] + \Pr[\mathsf{Success} \wedge \mathsf{Query}] \\ &\leq \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] + \Pr[\mathsf{Query}], \end{aligned} \tag{5.2}$$

where all probabilities are taken over the randomness used in Game 1 (internal and explicit sampling). We now want to argue that

$$\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \frac{1}{2} \quad \text{and} \tag{5.3}$$

$$\Pr[\mathsf{Query}] \leq \mathsf{negl}(n). \tag{5.4}$$

Proving 5.3 and 5.4 together with 5.2 clearly suffices to prove undetectability. We first deal with 5.3.

If $\Pr\left[\overline{\mathsf{Query}}\right] = 0$ then $\mathsf{Success} \wedge \overline{\mathsf{Query}} = \emptyset$ implying $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] = 0$. Assume $\Pr\left[\overline{\mathsf{Query}}\right] \neq 0$. Then observe (valid identity when probability of event $\mathsf{Query}$ is not 0)

$$\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] = \Pr\left[\overline{\mathsf{Query}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right] \leq \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right]$$

We argue $\Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right] = \frac{1}{2}$. When $\mathsf{Query}$ does not occur, all $y_i$'s are uniformly random from the view of $\mathcal{U}$. Hence from the view of $\mathcal{U}$ the answers returned from the oracle $\mathcal{O}_{\mathsf{CDH}}$ has the exact same distribution no matter whether $b$ is 0 or 1. Phrased differently: $\mathcal{U}$ learns no information about $b$ even given $\mathcal{U}$'s view of the big brother public key $\mathsf{pk_B}$ and parameters $\mathsf{params}$: the big brother public key $\mathsf{pk_B}$ and $g^{x_i}$'s uniquely determine the $\eta_i$'s but since $\mathcal{R}$ is a random function chosen independently of anything else, this gives no information on $\mathcal{R}(\eta_i)$. The identity $\eta_i = h^{-x_i} h_{1,i}$ impose no extra restrictions since it is true independently of $b$ and there are no relations between $\eta_i$ and $y_i$ from the perspective of $\mathcal{U}$ when $\mathsf{Query}$ does not occur. Similarly, $\mathcal{U}$'s queries to the random oracle $\mathcal{R}$ reveals no information on $b$ since Query does not occur. Therefore $\mathcal{U}$ can not do any better than random guessing, implying $\Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right] = \frac{1}{2}$.

The proof of the claim in (4.3) is more involved. Let $l = l(n)$ be the number of random oracle queries made by $\mathcal{U}$. Below we define a PPT adversary $\mathcal{A}$ for the CDH problem relative to $\mathcal{G}$. We will show that the advantage of $\mathcal{A}$ in the CDH game is related to $\Pr[\mathsf{Query}]$.

We first provide some intuition. Given an instance $(\mathbb{G}, q, g, g^x, g^s)$ of the CDH problem it is the goal of $\mathcal{A}$ to compute $g^{sx}$. Since $\mathcal{A}$ wants to use $\mathcal{U}$ as a subroutine, $\mathcal{A}$ must simulate the input that $\mathcal{U}$ receives in the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}}_{\mathsf{Dev_{CDH}},\mathsf{B.Dev_{CDH}}}(n)$. In our case, the value $g^s$ will play the role of the big brother public key, which means we don't know the associated big brother secret key. $\mathcal{A}$ will run $\mathcal{U}$ as a subroutine answering $\mathcal{U}$'s queries to $\mathcal{O}_{\mathsf{CDH}}$ and $\mathcal{R}$. Queries to $\mathcal{R}$ are easy to answer since we can just return a (uniformly) random value. At a first note, queries to $\mathcal{O}_{\mathsf{CDH}}$ are also easy to answer: sample uniform $x_i$ and $y_i$ and output $(g^{x_i}, g^{y_i}, g^{x_i y_i})$ but letting the first query be answered with $(g^x, g^{y_1}, (g^x)^{y_1})$ with $y_1$ uniform and $x_1 = x$. Because Query occurs it is true for at least one $i$ that $\eta_i = h^{-x_i} g^{x_i}$. From this equation it is possible to compute $g^{sx_i}$ by

$$g^{sx_i} = \eta_i^{-1} g^{x_i}.$$

We face a problem though: if $i \neq 1$ we do not learn $g^{sx}$. To counter this we replace $g^{x_i}$ with $g^x g^{x_i}$ in each response, for all $i$. From $\eta_i$ we can recover $g^{sx}$ by computing

$$g^{sx} = \eta_i^{-1} h^{-x_i} g^x g^{x_i},$$

which are all known values to $\mathcal{A}$. There is still one problem: when a $\eta_i$ is queried to $\mathcal{R}$ we lose consistency between the queries to $\mathcal{O}_{\mathsf{CDH}}$ and $\mathcal{R}$. If $\hat{y}$ is the answer from the query $\mathcal{R}(\eta_i)$ then $\hat{y}$ is (with high probability) not the same $y$ that is used in the output from the $i$'th query to $\mathcal{O}_{\mathsf{CDH}}$. Concretely, in Game 1 there is a probability of 1 that consistency is preserved between queries while the probability of consistency, when $\mathcal{U}$ is used as a subroutine by $\mathcal{A}$, is at most $\frac{1}{|\mathbb{G}|}$ (when $\mathcal{A}$ finish its execution). Since we do not know $s$ we can not even detect when such a query occurs[1]. As an effect, the distribution of the answers, where we can not preserve consistency, will be different from the answers in game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}}_{\mathsf{Dev_{CDH}},\mathsf{B.Dev_{CDH}}}(n)$. On further thought, however, there is no problem. Consistency will be preserved up to the point where Query occurs. But when Query occurs we have a $\eta_i$ from which we can compute $g^{sx}$. What ever $\mathcal{U}$ answers afterwards, we can neglect since we are not interested in the probability of $\mathcal{U}$ succeeding in some game, but in the event Query. We give the details of $\mathcal{A}$ below.

It is clear that $\mathcal{A}$ is a PPT algorithm because $\mathcal{U}$ is. We prove that there exists a polynomial $f$ such that

$$\mathrm{Adv}^{\mathsf{CDH}}_{\mathcal{G}}(\mathcal{A}) \cdot f(n) \geq \Pr[\mathsf{Query}]. \tag{5.5}$$

First observe that event Query is still well-defined in the execution of $\mathcal{A}$, even though $\mathcal{A}$ can not detect it. We now compare the view of $\mathcal{U}$ until Query occurs in the execution of Game 1 and the view of $\mathcal{U}$ when used as a subroutine by $\mathcal{A}$. In each case, $(\mathbb{G}, q, g)$ is output by $\mathcal{G}(1^n)$. In each case, $h$ is a uniform element because $s$ is chosen uniformly at random in the CDH game. In each case, queries to $\mathcal{O}_{\mathsf{CDH}}$ are answered with a tuple with the same distribution as long as Query has not

[1]This can be fixed by assuming the stronger GAP-CDH assumption. But we prefer the weaker CDH assumption.

```
Adversary $\mathcal{A}$
Input: $(\mathbb{G}, q, g, g^s, g^x)$
1: Initialise list $L_{\mathcal{R}}$
2: $j \leftarrow_R \{1, 2, \ldots, l\}$
3: $z \leftarrow_R \{1, 2, \ldots, t\}$
4: $h := g^s$
5: $x_1, x_2, \ldots, x_t \leftarrow_R \mathbb{Z}_q$
6: $h_{1,i} := g^x g^{x_i}$, for $i = 1, 2, \ldots, t$
7: Run $\mathcal{U}^{\mathcal{O}_{\mathsf{CDH}}, \mathcal{R}}(\mathbb{G}, q, g, h)$
8: Parse: $L_{\mathcal{R}}[j] = (\eta, y)$
9: $\hat{h} := \eta^{-1} h^{-x_z} h_{1,z}$
10: Return $\hat{h}$

   Device oracle: $\mathcal{O}_{\mathsf{CDH}}$ $i$'th query
1: $y_i \leftarrow_R \mathbb{Z}_q$
2: $h_{2,i} := g^{y_i}$
3: Return $(h_{1,i}, h_{2,i}, h_{1,i}^{y_i})$

   Random oracle: $\mathcal{R}(\lambda)$
1: if $(\lambda, y)$ is in $L_{\mathcal{R}}$ for some $y$ then
2:     Return $y$
3: else
4:     $y \leftarrow_R \mathbb{Z}_q$
5:     Store $(\lambda, y)$ in $L_{\mathcal{R}}$
6:     Return $y$
```

occurred (using same arguments as above). Finally, in each case, the queries to $\mathcal{R}$ are answered with a uniform value from $\mathbb{Z}_q$ (when Query has not occurred). When $\mathcal{R}$ is queried with a $\eta_i$, $\mathcal{R}$ will answer with a uniform value independent of $g^{y_i}$ from the point of view of $\mathcal{U}$ when used as a subroutine in $\mathcal{A}$ while the answer by $\mathcal{R}$ in Game 1 uniquely determines $g^{y_i}$. But when this query is made, event Query occurs and we can compute $g^{sx}$. Notice that when Query occurs we can recover $g^{sx}$ with probability 1: say we guess the correct query among the $l$ queries and which $x_z$ is used among the $t$ possible values $x_1, x_2, \ldots, x_t$. Then $\eta = h^{-(x+x_z)} g^{x+x_z}$ and $h_{1,z} = g^x g^{x_z}$. $\mathcal{A}$ computes

$$\hat{h} = \eta^{-1} h^{-x_z} h_{1,z} = \left(g^{-x} g^{-x_z} h^x h^{x_z}\right) h^{-x_z} \left(g^x g^{x_z}\right) = h^x = g^{sx},$$

which is the value $\mathcal{A}$ answer with.

To win, $\mathcal{A}$ must guess both in which query the event Query occurs and which $x_i$ is used. Let $S$ be the event that the sampled pair $(z, j)$ in the execution of $\mathcal{A}$ satisfy $\eta = h^{-(x+x_z)} g^{x+x_z}$ and occurs in the $j$'th query. Then

$$\begin{aligned} \mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{A}) &= \Pr\left[S \wedge \mathsf{Query}\right] \\ &= \Pr\left[S\right] \cdot \Pr\left[\mathsf{Query}\right] \\ &\geq \frac{1}{l \cdot t} \Pr\left[\mathsf{Query}\right], \end{aligned}$$

since events $S$ and Query are independent (and choices of $j$ and $z$ are also independent). Because the CDH problem is hard relative to $\mathcal{G}$, $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{U})$ is negligible in $n$. Now since $l$ and $t$ are polynomials in $n$ the product $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{U}) \cdot l \cdot t$ is thus also negligible in $n$. We conclude that $\Pr[\mathsf{Query}]$ is negligible in $n$ as we wanted to show. $\square$

When proving undetectability in the proof of Theorem 5.2.3 we construct an adversary that can solve the CDH problem. We can increase the probability of the adversary winning, and thereby making the reduction more tight, if we assume the GAP-CDH problem is hard relative to $\mathcal{G}$. The reduction would work just as above, except we will have the possibility of knowing when event Query occurs and thereby making the guess of when it occurs, redundant. The proof goes exactly like the above proof with the addition of using the DDH-oracle access provided by the GAP-CDH assumption in $\mathcal{A}$ to check when Query occurs. The upshot is the use of a stronger (and more artificial) assumption.

## 5.3 Decisional Diffie-Hellman subversion

The decisional diffie hellman (DDH) problem says that given two elements $g^x$ and $g^y$ from a cyclic $\mathbb{G} = \langle g \rangle$ of order $q$, it is hard to distinguish between $g^{xy}$ and $g^z$, where $x, y, z \leftarrow_R \mathbb{Z}_q$. A formal definition of DDH hardness is given in Def. 2.5.3.

There is a clear connection between the CDH problem and the DDH problem: if the CDH problem is easy then the DDH problem is also easy. Whether the converse is true is unknown. The former statement is proved in Propositon 2.5.5 where we construct an algorithm that can solve the DDH problem using an algorithm that can solve the CDH problem. It turns out that there is a similar connection when we consider subversions of CDH and DDH. Essentially the subversion of $\mathsf{G}_{\mathsf{DDH}}$ follows from the subversion of $\mathsf{G}_{\mathsf{CDH}}$ in the sense that the exact same subversion technique is used and indistinguishably of the subverted and non-subverted device only requires CDH hardness. This connection is elaborated upon in Section 5.4.

In Figure 5.3, we define a non-interactive cryptographic game $\mathsf{G}_{\mathsf{DDH}}$ that models a DDH game. In Theorem 5.3.1 we prove that $\mathsf{G}_{\mathsf{DDH}}$ is secure when the DDH problem is hard relative to $\mathcal{G}$. Theorem 5.3.2 proves that the big brother $\mathsf{B}_{\mathsf{DDH}}$ defined in Figure 5.4, strongly subverts $\mathsf{G}_{\mathsf{DDH}}$.

We begin by constructing big brother and proving it is secure.

**Theorem 5.3.1.** *Let $\mathcal{G}$ be defined as in Def. 2.5.3. The non-interactive cryptographic game* $\mathsf{G}_{\mathsf{DDH}}$ *defined in Figure 5.3 is secure (relative to $\mathcal{G}$) when the* DDH *problem is hard relative to $\mathcal{G}$.*

*Proof.* Given a PPT adversary $\mathcal{A}$, we construct a PPT adversary $\mathcal{A}'$ such that

$$\mathrm{Adv}_{\mathsf{G}_{\mathsf{DDH}}}^{\mathsf{SEC}}(\mathcal{A}) = \mathrm{Adv}_{\mathcal{G}}^{\mathsf{DDH}}(\mathcal{A}'), \tag{5.6}$$

where $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{DDH}}(\mathcal{A}')$ is the advantage of $\mathcal{A}'$ in Game $\mathsf{DDH}_{\mathcal{G}}^{\mathcal{A}}(n)$ and $\mathrm{Adv}_{\mathsf{G}_{\mathsf{DDH}}}^{\mathsf{SEC}}(\mathcal{A})$ is the advantage of $\mathcal{A}$ in Game $\mathsf{SEC}_{\mathsf{G}_{\mathsf{DDH}}}^{\mathcal{A}}(n)$. We first describe the attack by $\mathcal{A}$ more concretely.

```
Par(1^n)
        $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
        params $= (\mathbb{G}, q, g)$
        **Return** params
Dev_DDH(1^n, params)
        Parse: params $= (\mathbb{G}, q, g)$
        $b \leftarrow_R \{0,1\}$
        $x, y, z \leftarrow_R \mathbb{G}$
        $h_1 := g^x$
        $h_2 := g^y$
        $h_3 := g^{xy+bz}$
        $\tau_p := (\mathbb{G}, q, g, h_1, h_2, h_3)$
        $\tau_s := (\mathbb{G}, q, g, b)$
        **Return** $(\tau_p, \tau_s)$
Pre($\tau_p, \tau_s, \sigma$)
        Parse: $\tau_s = (\mathbb{G}, q, g, b)$
        **Return** $(\sigma \overset{?}{=} b)$
$(p_0(n), \alpha_0(n)) := \left(\frac{1}{2}, 2\right)$
```

**Figure 5.3:** Non-interactive cryptographic game $\mathsf{G}_{\mathsf{DDH}}$.

**Game 0.** We write Def. 3.3.2 more algorithmically in the context of $\mathcal{A}$ being the adversary.

```
Game 0
1: $(\mathbb{G}, q, g) \leftarrow$ Par$(1^n)$
2: $b \leftarrow_R \{0,1\}$
3: $x, y, z \leftarrow_R \mathbb{Z}_q$; $h_1 = g^x$; $h_2 = g^y$; $h_3 = g^{xy+bz}$
4: $\hat{b} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2, h_3)$
```

Let $S_0$ be the event that $\hat{b} = b$ in Game 0. Obviously $\text{Adv}^{\mathsf{SEC}}_{\mathsf{G}_{\mathsf{DDH}}}(\mathcal{A}) = \Pr[S_0]$. We proceed by defining a new adversary $\mathcal{A}'$, which tries to guess $b$ using $\mathcal{A}$ as a subroutine.

```
Adversary A'
Input: $(\mathbb{G}, q, g, h_1, h_2, h_3)$
1: $\hat{h} \leftarrow \mathcal{A}(\mathbb{G}, q, g, h_1, h_2, h_3)$
2: **Return** $\hat{h}$
```

Since $\mathcal{A}$ is a PPT algorithm $\mathcal{A}'$ is also a PPT algorithm. When the input to $\mathcal{A}'$ is of the form $(\mathbb{G}, q, g, g^x, g^y, g^{xy+bz})$, computation proceed just as in Game 0. Hence

$$\Pr\left[b \leftarrow_R \{0,1\}\,;\, x, y, z \leftarrow_R \mathbb{Z}_q;\, \hat{b} \leftarrow \mathcal{A}'(\mathbb{G}, q, g, g^x, g^y, g^{xy+bz}) \,:\, \hat{b} = b\right] = \Pr[S_0]\,.$$

The left-hand side is exactly the advantage of $\mathcal{A}'$ in Game $\mathsf{DDH}^{\mathcal{A}'}_{\mathcal{G}}(n)$ and equation 5.6 follows. □

Just as in Section 5.2 security of the non-interactive cryptographic game $\mathsf{G}_{\mathsf{DDH}}$ is equivalent to the game $\mathsf{DDH}^{\mathcal{A}}_{\mathcal{G}}(n)$ defining DDH hardness. That is, given an adversary for the former game, this adversary succeed with the same probability in the security game for $\mathsf{G}_{\mathsf{DDH}}$.

We now state and prove that $\mathsf{G}_{\mathsf{DDH}}$ can be strongly subverted. As in the previous section this amounts to saying that the DDH hardness assumption can be subverted.

**Theorem 5.3.2.** *The big brother* $\mathsf{B}_{\mathsf{DDH}}$ *defined in Figure 5.4 is a strong subversion of* $\mathsf{G}_{\mathsf{DDH}}$ *if the* DDH *problem is hard relative to* $\mathcal{G}$.

```
B.Gen($1^n$, params)
    Parse: params $= (\mathbb{G}, q, g)$
    $s \leftarrow_R \mathbb{Z}_q$
    $h := g^s$
    $\mathsf{pk_B} := (\mathbb{G}, q, g, h)$
    $\mathsf{sk_B} := (\mathbb{G}, q, g, s)$
    **Return** $(\mathsf{pk_B}, \mathsf{sk_B})$
B.Dev$_{\mathsf{DDH}}$($1^n$, $\mathsf{pk_B}$, params)
    Parse: params $= (\mathbb{G}, q, g)$
    Parse: $\mathsf{pk_B} = (\mathbb{G}, q, g, h)$
    $b \leftarrow_R \{0, 1\}$
    $x, z \leftarrow_R \mathbb{Z}_q$
    $h_1 := g^x$
    $\eta := h^{-x} h_1$
    $y := \mathcal{R}(\eta)$
    $h_2 := g^y$
    $h_3 := g^{xy+bz}$
    $\tau_p := (\mathbb{G}, q, g, h_1, h_2, h_3)$
    $\tau_s := (\mathbb{G}, q, g, b)$
    **Return** $(\tau_p, \tau_s)$
B.Rec($\mathsf{sk_B}$, params, $\tau_p$)
    Parse: $\tau_p = (\mathbb{G}, q, g, h_1, h_2, h_3)$
    Parse: $\mathsf{sk_B} = (\mathbb{G}, q, g, s)$
    $\hat{\eta} := h_1^{-s} h_1$
    $\hat{y} := \mathcal{R}(\eta)$
    If $(h_1^{\hat{y}} = h_3)$: **Return** 0
    Else: **Return** 1

$\mathcal{R}$ is a random oracle with domain and range $\mathbb{Z}_q$.
```

**Figure 5.4:** Big brother subversion $\mathsf{B_{DDH}}$ of $\mathsf{G_{DDH}}$.

*Proof.* We first prove that $\mathsf{B_{DDH}}$ is recoverable and afterwards that $\mathsf{B_{DDH}}$ is strongly undetectable. The first part is a straightforward symbol game, where one just has to check that the computation by B.Rec is actually computing the correct values. The second part is following the same strategy as in Theorem 5.2.3 and actually only requires that the CDH problem is hard relative to $\mathcal{G}$.

## $\mathsf{B_{DDH}}$ is recoverable

The operations done by B.Rec clearly finish in polynomial-time implying that B.Rec is a PPT algorithm. From the execution of B.Gen we have $h = g^s$ in $\mathbb{G}$. After executing B.Dev$_{\mathsf{DDH}}$ we have $h_1 = g^x$, $h_2 = g^y$ and $h_3 = g^{xy+bz}$ where $y$ is computed by $\eta = h^{-x}h_1 = h^{-x}g^x$ and then setting $y$ equal to $\mathcal{R}(\eta)$.

When $\mathsf{B_{DDH}}$ executes B.Rec with input $(\mathbb{G}, q, g, s, g^x, g^y, g^{xy+bz})$, B.Rec first computes

$$\hat{\eta} = h_1^{-s} h_1 = g^{-sx} g^x = (g^s)^{-x} g^x = h^{-x} g^x = \eta.$$

By properties of the random oracle $\mathcal{R}$ we can compute $\hat{y} = \mathcal{R}(\hat{\eta}) = y$. B.Rec can then compute

$$h_1^y = g^{xy}.$$

Comparing this value against $h_3 = g^{xy+bz}$ determines the correct value of $b$ unless $z = 0$ in which case B.Rec always guess $b = 0$. But since $z$ is uniform this happens with probability $\frac{1}{q}$ and hence is negligible (the length of $q$ is $n$). Therefore, we at least have

$$\mathrm{Adv}_{\mathsf{B_{DDH}}}(\mathsf{B.Rec}) \geq \frac{1}{2},$$

which is sufficient to conclude that $\mathsf{B}_{\mathsf{DDH}}$ recovers information. Asymptotically B.Rec will be successful with an unnoticeable probability away from 1.

**$\mathsf{B}_{\mathsf{CDH}}$ is undetectable**

Let $\mathcal{U}$ be a PPT user trying to distinguish between $\mathsf{Dev}_{\mathsf{DDH}}$ and $\mathsf{B.Dev}_{\mathsf{DDH}}$. Consider the game $\text{strong-DETECT}^{\mathcal{U}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$ and let $\mathcal{O}_{\mathsf{DDH}}$ be the oracle used in that game. Assume WLOG $\mathcal{U}$ makes exactly $t = t(n)$ queries to $\mathcal{O}_{\mathsf{DDH}}$. From Game 0 to Game 1 we make some syntactical changes. As before both $\mathcal{U}$ and $\mathcal{O}_{\mathsf{DDH}}$ have access to the same random oracle $\mathcal{R}$.

**Game 0:** We write Game $\text{strong-DETECT}^{\mathcal{U}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$ more algorithmically in the context of user $\mathcal{U}$ being the distinguisher. We leave out the description of the random oracle $\mathcal{R}$, because we do not modify it going from Game 0 to Game 1.

```
Game 0
1: $(\mathbb{G}, q, g) \leftarrow \mathsf{Par}(1^n)$
2: $b \leftarrow_R \{0,1\}$
3: $s \leftarrow_R \mathbb{Z}_q$
4: $h := g^s$
5: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\mathsf{DDH}},\mathcal{R}}(\mathbb{G}, q, g, h)$

   Device oracle: $\mathcal{O}_{\mathsf{DDH}}$ $i$'th query

1: $b_i \leftarrow_R \{0,1\}$
2: $x_i, z_i \leftarrow_R \mathbb{Z}_q$
3: $h_{1,i} := g^{x_i}$
4: if $b_i = 0$ then
5:     $y_i \leftarrow_R \mathbb{Z}_q$
6: else
7:     $\eta_i := h^{-x} h_{1,i}$
8:     $y_i := \mathcal{R}(\eta_i)$
9: $h_{2,i} := g^{y_i}$
10: $h_{3,i} := g^{x_i y_i + b_i z_i}$
11: Return $(\mathbb{G}, q, g, h_{1,i}, h_{2,i}, h_{3,i}, b_i)$
```

Let $S_0$ denote the event $\hat{b} = b$ in Game 0. Then

$$\Pr[S_0] = \Pr\left[\mathsf{true} \leftarrow \text{strong-DETECT}^{\mathcal{U}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)\right].$$

We do a syntactic transform into Game 1.

**Game 1:** We reprogram the oracle $\mathcal{O}_{\mathsf{DDH}}$ such that $b_i$'s, $x_i$'s and $z_i$'s are computed in the beginning of the game. Knowing $x_i$ we can also compute $h_{1,i}$ and $\eta_i$ in advance. Recall that $\mathcal{U}$ will make exactly $t$ queries and that we sample elements using fresh randomness.

Let $S_1$ be the event that $\hat{b} = b$ in Game 1. Since the change is merely syntactic we have $\Pr[S_1] = \Pr[S_0]$. In the execution of Game 1 we define two events:

| | | |
|---|---|---|
| Query | : | The event that, at any point during its execution, $\mathcal{U}$ queries any of $\eta_1, \ldots, \eta_t$ to $\mathcal{R}$. |
| Success | : | The event that $\hat{b} = b$. |

**Game 1**

```
1: $(\mathbb{G}, q, g) \leftarrow \mathsf{Par}(1^n)$
2: $b \leftarrow_R \{0, 1\}$
3: $s \leftarrow_R \mathbb{Z}_q$
4: $h := g^s$
5: $x_1, x_2, \ldots, x_t \leftarrow_R \mathbb{Z}_q$
6: $z_1, z_2, \ldots, z_t \leftarrow_R \mathbb{Z}_q$
7: $b_1, b_2, \ldots, b_t \leftarrow_R \mathbb{Z}_q$
8: $h_{1,i} := g^{x_i}$, for $i = 1, 2, \ldots, t$
9: $\eta_i := h^{x_i} h_{1,i}$, for $i = 1, 2, \ldots, t$
10: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\mathsf{CDH}}, \mathcal{R}}(\mathbb{G}, q, g, h)$
```

Device oracle: $\mathcal{O}_{\mathsf{DDH}}$ $i$'th query

```
1: if $b = 0$ then
2:     $y_i \leftarrow_R \mathbb{Z}_q$
3: else
4:     $y_i := \mathcal{R}(\eta_i)$
5: $h_{2,i} := g^{y_i}$
6: $h_{3,i} := g^{x_i y_i + b_i z_i}$
7: Return $(\mathbb{G}, q, g, h_{1,i}, h_{2,i}, h_{3,i}, b_i)$
```

As in the proof of Theorem 5.2.3 we have

$$\Pr[S_1] \leq \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] + \Pr[\mathsf{Query}], \tag{5.7}$$

where all probabilities are taken over the randomness used in Game 1 (internal and explicit sampling). Again we want to argue

$$\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \frac{1}{2} \quad \text{and} \tag{5.8}$$

$$\Pr[\mathsf{Query}] \leq \mathsf{negl}(n). \tag{5.9}$$

Proving 5.8 and 5.9 together with 5.7 proves undetectability. The arguments for the two inequalities above are almost identical to the arguments given in Theorem 5.2.3. We go through a shortened version in this proof because the two arguments are so similar. After the proof of Theorem 5.3.2 we show an alternative way of proving undetectability that highlights the close connection between undetectability of $\mathsf{B}_{\mathsf{CDH}}$ and undetectability of $\mathsf{B}_{\mathsf{DDH}}$.

We start with 5.8. If $\Pr\left[\overline{\mathsf{Query}}\right] = 0$ then $\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] = 0$. Assume $\Pr\left[\overline{\mathsf{Query}}\right] \neq 0$. Then observe

$$\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Query}}\right] \leq \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right].$$

We argue $\Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right] = \frac{1}{2}$. $\mathcal{U}$ learns no information about $b$ even given $\mathcal{U}$'s view of the big brother public key $\mathsf{pk_B}$ and parameters $\mathsf{params}$: the big brother public key and $g^{x_i}$'s do uniquely determine the $\eta_i$'s but since $\mathcal{R}$ is a random function chosen independently of anything else, this gives no information on $\mathcal{R}(\eta_i)$ and $\mathsf{params}$ contain no more information than the big brother public key. The identity $\eta_i = h^{-x_i} h_{1,i}$ imposes no extra restrictions since it is true in both cases and there are no relations between $\eta_i$ and $y_i$ from the perspective of $\mathcal{U}$ when $\mathsf{Query}$ does not occur. Similarly,

$\mathcal{U}$'s queries to $\mathcal{R}$ reveals no information on $b$ since Query does not occur. Therefore $\mathcal{U}$ can not do any better than random guessing, implying $\Pr\left[\mathsf{Success} \mid \overline{\mathsf{Query}}\right] = \frac{1}{2}$.

Let $l = l(n)$ be number of random oracle queries made by $\mathcal{U}$. Below we define a PPT adversary $\mathcal{A}$ for the CDH problem relative to $\mathcal{G}$ (yes, CDH problem and not DDH problem) using $\mathcal{U}$ as a subroutine. We are interested in showing that same as in 5.5. We give the details of $\mathcal{A}$ below.

```
Adversary $\mathcal{A}'$
Input: $(\mathbb{G}, q, g, g^s, g^x)$
 1: Initialise list $L_{\mathcal{R}}$
 2: $j \leftarrow_R \{1, 2, \dots, l\}$
 3: $z \leftarrow_R \{1, 2, \dots, t\}$
 4: $h := g^s$
 5: $x_1, x_2, \dots, x_t \leftarrow_R \mathbb{Z}_q$
 6: $z_1, z_2, \dots, z_t \leftarrow_R \mathbb{Z}_q$
 7: $b_1, b_2, \dots, b_t \leftarrow_R \mathbb{Z}_q$
 8: $h_{1,i} := g^x g^{x_i}$
 9: Run $\mathcal{A}^{\mathcal{O}_{\mathsf{CDH}}, \mathcal{R}}(\mathbb{G}, q, g, h)$
10: Parse: $L_{\mathcal{R}}[j] = (\eta, y)$
11: $\hat{h} := \eta^{-1} h^{-x_z} h_{1,z}$
12: Return $\hat{h}$

    Device oracle: $\mathcal{O}_{\mathsf{DDH}}$
 1: $y_i \leftarrow_R \mathbb{Z}_q$
 2: $h_{2,i} := g^{y_i}$
 3: $h_{3,i} := h_{1,i} \cdot g^{b_i z_i}$
 4: Return $(h_{1,i}, h_{2,i}, h_{3,i}, b_i)$
    Random oracle: $\mathcal{R}(\lambda)$
 1: if $(\lambda, y)$ is in $L_{\mathcal{R}}$ for some $y$ then
 2:     Return $y$
 3: else
 4:     $y \leftarrow_R \mathbb{Z}_q$
 5:     Store $(\lambda, y)$ in $L_{\mathcal{R}}$
 6:     Return $y$
```

Since $\mathcal{U}$ is a PPT algorithm, $\mathcal{A}$ will also be a PPT algorithm. First observe that event Query is still well-defined in the execution of $\mathcal{A}$, even though $\mathcal{A}$ can not detect it. We now compare the views of $\mathcal{U}$ until Query occurs in the execution of Game 1 and the view of $\mathcal{U}$ when used as a subroutine by $\mathcal{A}$. In each case, $(\mathbb{G}, q, g)$ is output by $\mathcal{G}(1^n)$. In each case, $h$ is a uniform element because $s$ is chosen uniformly at random in the CDH game. In each case, queries to $\mathcal{O}_{\mathsf{DDH}}$ are answered with a tuple with the same distribution as long as Query has not occurred (using same arguments as above). Finally, in each case, the queries to $\mathcal{R}$ are answered with a uniform value from $\mathbb{Z}_q$ (when Query has not occurred). When $\mathcal{R}$ is queried with a $\eta_i$, $\mathcal{R}$ will output a uniform value independent of $g^{y_i}$ from the view of $\mathcal{U}$ when used as a subroutine in $\mathcal{A}$ while the output in Game 1, when $\mathcal{R}$ is queried by $\eta_i$, uniquely determines $g^{y_i}$. But when this query occurs, event Query occurs.

Let $S$ be the event that the sampled pair $(z, j)$ in $\mathcal{A}$ satisfy: $\eta = h^{-(x+x_z)} g^{x+x_z}$ is queried to $\mathcal{R}$

in the $j'$th query. Then

$$\begin{aligned}\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{A}) &= \Pr[S \wedge \mathsf{Query}] \\ &= \Pr[S] \cdot \Pr[\mathsf{Query}] \\ &\geq \frac{1}{l \cdot t} \Pr[\mathsf{Query}] .\end{aligned}$$

Propositon 2.5.5 implies that the CDH problem is hard relative to $\mathcal{G}$ because the DDH problem is assumed hard relative to $\mathcal{G}$. Thus $\mathrm{Adv}_{\mathcal{G}}^{\mathsf{CDH}}(\mathcal{A})$ is negligible in $n$. Since $l$ and $t$ are polynomials in $n$, we conclude that $\Pr[\mathsf{Query}]$ is negligible in $n$. □

In the above proof we did not use any properties of $\mathsf{B}_{\mathsf{CDH}}$. In the next section we elaborate on the connection between $\mathsf{B}_{\mathsf{CDH}}$ and $\mathsf{B}_{\mathsf{DDH}}$.

## 5.4 Alternative proof of undetectability of $\mathsf{B}_{\mathsf{DDH}}$

The observant reader may have noticed the similarity between the proof of Theorem 5.2.3 and Theorem 5.3.2. It turns out that strong undetectability can be proven solely by using strong undetectability of $\mathsf{B}_{\mathsf{CDH}}$.

*Proof.* Our strategy is to use undetectability of $\mathsf{B}_{\mathsf{CDH}}$ to prove that detection of the subversion $\mathsf{B}_{\mathsf{DDH}}$ is impossible. In the following we notate $\mathsf{B}_{\mathsf{CDH}} = (\mathsf{B.Gen}_{\mathsf{CDH}}, \mathsf{B.Dev}_{\mathsf{CDH}}, \mathsf{B.Rec}_{\mathsf{CDH}})$ and $\mathsf{B}_{\mathsf{DDH}} = (\mathsf{B.Gen}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}, \mathsf{B.Rec}_{\mathsf{DDH}})$

Let $\mathcal{U}_{\mathsf{DDH}}$ be a PPT user and consider the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$. We construct a user $\mathcal{U}_{\mathsf{CDH}}$ using $\mathcal{U}_{\mathsf{DDH}}$ as a subroutine that tries to distinguish between $\mathsf{Dev}_{\mathsf{CDH}}$ and $\mathsf{B.Dev}_{\mathsf{CDH}}$ in game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{CDH}}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)$. In the sequel we will argue that there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}^{\mathsf{strong\text{-}DETECT}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(\mathcal{U}_{\mathsf{DDH}}) = \mathrm{Adv}^{\mathsf{strong\text{-}DETECT}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(\mathcal{U}_{\mathsf{CDH}}) \leq \mathsf{negl}(n), \tag{5.10}$$

which proves what we want. That inequality in 5.10 follows because $\mathsf{B}_{\mathsf{CDH}}$ is a strong subversion of $\mathsf{G}_{\mathsf{CDH}}$ proved in Theorem 5.2.3. We are left with the task of proving the equality. The user $\mathcal{U}_{\mathsf{CDH}}$ is constructed below: in the description $\mathcal{U}_{\mathsf{CDH}}$ has access to a device oracle $\mathcal{O}_{\mathsf{CDH}}$ and random oracle $\mathcal{R}_{\mathsf{CDH}}$.

The idea is pretty clear: in the role of $\mathcal{U}_{\mathsf{CDH}}$ we can easily mimic as the challenger in game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$ and simulate the input to $\mathcal{U}_{\mathsf{DDH}}$. Note that it is imperative that $\mathsf{B}_{\mathsf{CDH}}$ is a strong subversion such that we have access to the value $g^{xy}$. Implicitly we assume that $\mathsf{B.Gen}_{\mathsf{CDH}}$ and $\mathsf{B.Gen}_{\mathsf{DDH}}$ use the same algorithm $\mathcal{G}$ to generate parameters.

Let $\mathsf{Success}_{\mathsf{CDH}}$ be the event that $\mathcal{U}_{\mathsf{CDH}}$ guess the correct bit in $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{CDH}}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)$. Then

$$\Pr[\mathsf{Success}_{\mathsf{CDH}}] = \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{CDH}}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)\right].$$

Obviously the event $\mathsf{Success}_{\mathsf{CDH}}$ occurs if and only if the subroutine $\mathcal{U}_{\mathsf{DDH}}$ used by $\mathcal{U}_{\mathsf{CDH}}$ guess the correct bit, because $\mathcal{U}_{\mathsf{CDH}}$ directly use this bit as its guess. Hence, if $\mathsf{Success}_{\mathsf{DDH}}$ is the event that $\mathcal{U}_{\mathsf{DDH}}$ guess the correct bit given its view when used as a subroutine, then

$$\Pr[\mathsf{Success}_{\mathsf{DDH}}] = \Pr[\mathsf{Success}_{\mathsf{CDH}}] .$$

```
User $\mathcal{U}_{\mathsf{CDH}}$
Input: $(\mathsf{pk_B}, \mathsf{params})$
1: $\hat{b} \leftarrow \mathcal{U}_{\mathsf{DDH}}^{\mathcal{O}_{\mathsf{DDH}}, \mathcal{R}_{\mathsf{DDH}}}(\mathsf{pk_B}, \mathbb{G}, q, g)$
2: Return $\hat{b}$

    Device oracle: $\mathcal{O}_{\mathsf{DDH}}$

1: Query: $(\tau_p, \tau_s) \leftarrow \mathcal{O}_{\mathsf{CDH}}$
2: Parse: $\tau_p = (\mathbb{G}, q, g, g^x, g^y)$
3: Parse: $\tau_s = (\mathbb{G}, q, g, g^{xy})$
4: $b' \leftarrow_R \{0,1\}$
5: $z \leftarrow_R \mathbb{Z}_q$
6: $\tau_p' = (\mathbb{G}, q, g, g^x, g^y, g^{xy} \cdot g^{b'z})$
7: $\tau_s' = (\mathbb{G}, q, g, b')$
8: Return $(\tau_p', \tau_s')$
    Random oracle: $\mathcal{R}_{\mathsf{DDH}}(\lambda)$

1: Query: $r \leftarrow \mathcal{R}_{\mathsf{CDH}}(\lambda)$
2: Return $r$
```

We want to relate the probability of $\mathsf{Success}_{\mathsf{DDH}}$ with the advantage of $\mathcal{U}_{\mathsf{DDH}}$ when its view is the same as in the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}}(n)$. More precisely we prove

$$\Pr\left[\mathsf{Success}_{\mathsf{DDH}}\right] = \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}}(n)\right] \tag{5.11}$$

To do this, we show that the view of $\mathcal{U}_{\mathsf{DDH}}$ when used as a subroutine by $\mathcal{U}_{\mathsf{CDH}}$ is the same view of $\mathcal{U}_{\mathsf{DDH}}$ when playing the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}}(n)$. That is, the distribution of the input to $\mathcal{U}_{\mathsf{DDH}}$ and the answers to $\mathcal{U}_{\mathsf{DDH}}$'s queries is the same regardless of where $\mathcal{U}_{\mathsf{DDH}}$ is used. Since the event $\mathsf{Success}_{\mathsf{DDH}}$ is also well-defined given $\mathcal{U}_{\mathsf{DDH}}$'s view in the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}}(n)$, 5.11 will follow. To decrease the amount of notation, we let *case 1* be the situation where we consider the view of $\mathcal{U}_{\mathsf{DDH}}$ in the game $\mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}}, \mathsf{B.Dev}_{\mathsf{DDH}}}(n)$ and *case 2* be the situation where we consider the view of $\mathcal{U}_{\mathsf{DDH}}$ when used as a subroutine by $\mathcal{U}_{\mathsf{CDH}}$.

First we consider the input to $\mathcal{U}_{\mathsf{DDH}}$ in both cases. Let $(\mathbb{G}, q, g, h)_1$ be the input in case 1 and $(\mathbb{G}, q, g, h)_2$ the input in case 2. $(\mathbb{G}, q, g, h)_1$ is generated by

$$(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n), \quad s \leftarrow_R \mathbb{Z}_q, \quad h := g^s.$$

$(\mathbb{G}, q, g, h)_2$ is generated by

$$(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n), \quad s \leftarrow_R \mathbb{Z}_q, \quad h := g^s.$$

Obviously the two generations above impose the same distributions.

We now look at the answers from the device oracle queries, which all have the same form. Let $(g^x, g^y, g^{xy+b'z})_1$ be an answer in case 1 and $(g^x, g^y, g^{xy+b'z})_2$ be an answer in case 2 (we neglect the part of the answer containing the group instance). We consider separately the two cases $b = 0$ and $b = 1$ i.e the non-subverted, respectively, subverted case.

When $b = 0$, $(g^x, g^y, g^{xy+b'z})_1$ is generated by

$$b' \leftarrow_R \{0,1\}, \quad x, y, z \leftarrow_R \mathbb{Z}_q.$$

$(g^x, g^y, g^{xy+b'z})_2$ is generated by

$$b' \leftarrow_R \{0,1\}, \quad x, y, z \leftarrow_R \mathbb{Z}_q$$

with $x$ and $z$ sampled in the query to $\mathcal{O}_{\mathsf{CDH}}$ and $b'$ and $z$ are sampled by $\mathcal{U}_{\mathsf{CDH}}$. Notice that $g^{xy+b'z}$ is computed in the same way with exponents sampled uniformly at random in both cases. Hence, when $b = 0$, the answers from the device oracle queries are distributed equally in both cases.

When $b = 1$, $(g^x, g^y, g^{xy+b'z})_1$ is generated by

$$b' \leftarrow_R \{0,1\}, \quad x, z \leftarrow_R \mathbb{Z}_q, \quad y := \mathcal{R}(h^{-x}g^x),$$

where $\mathcal{R}$ is the random oracle used in the game strong-$\mathsf{DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$. $(g^x, g^y, g^{xy+b'z})_2$ is generated by

$$b' \leftarrow_R \{0,1\}, \quad x, z \leftarrow_R \mathbb{Z}_q, \quad y := \mathcal{R}_{\mathsf{CDH}}(h^{-x}g^x)$$

with $x$ and $y$ sampled in the query to $\mathcal{O}_{\mathsf{CDH}}$ and $b'$ and $z$ are sampled by $\mathcal{U}_{\mathsf{CDH}}$. Notice that $g^{xy+b'z}$ is computed in the same way with identically distributed exponents, because $\mathcal{R}_{\mathsf{CDH}}$ and $\mathcal{R}$ are random functions. Hence, when $b = 1$, the answers from the device oracle queries are distributed equally in both cases.

The final thing we need to consider is the queries to the random oracle. The queries needs to be consistent with the answers from the device oracles. This is clear in case 1. In case 2 we achieve this by routing the random oracle query to $\mathcal{R}_{\mathsf{DDH}}$, to the random oracle $\mathcal{R}_{\mathsf{CDH}}$. That is, $y$ is computed from the answer by $\mathcal{R}_{\mathsf{CDH}}$. This makes the queries to $\mathcal{R}_{\mathsf{DDH}}$ consistent with the answers from the device oracle $\mathcal{O}_{\mathsf{DDH}}$ used by $\mathcal{A}$.

In total we have proved that it is possible for $\mathcal{U}_{\mathsf{CDH}}$ to simulate the challenger in the game strong-$\mathsf{DETECT}^{\mathcal{U}_{\mathsf{DDH}}}_{\mathsf{Dev}_{\mathsf{DDH}},\mathsf{B.Dev}_{\mathsf{DDH}}}(n)$ such that the view of $\mathcal{U}_{\mathsf{DDH}}$ in the game is distributed identically to $\mathcal{U}_{\mathsf{DDH}}$'s view when used a subroutine by $\mathcal{U}_{\mathsf{CDH}}$. $\square$

What we have just proved is that, if we can detect a DDH subversion, then we can also detect a CDH subversion. That is, contrary to the situation of CDH versus DDH where DDH hardness is a stronger assumption than CDH hardness, we have that strong undetectability in CDH subversion implies strong undetectability in the DDH subversion. From this perspective, a strong subversion of CDH is stronger than a strong subversion of DDH. From another perspective we have also proved that strong undetectability of the CDH subversion is not necessary to achieve strong undetectability of DDH subversion, because we can prove strong undetectability of the DDH subversion by just assuming DDH is hard relative to $\mathcal{G}$.

*Remark* 5.4.1. The idea in the proof above is that given a CDH instances, we can generate a matching DDH instance by just sampling a uniform random bit. Because we had access to the whole instance (the secret information) this allowed us to construct a reduction. Consider the reverse scenario: Generate a CDH instance given a DDH instance (with knowledge of the secret information as well). In this case we know the elements $(g^x, g^y, g^{xy+bz})$ and bit $b$. If $b = 0$ it is possible to generate the CDH instance. But if $b = 1$ then $g^{xy}g^z$ gives no information on $g^{xy}$ and another trick is needed to generate a correctly distributed CDH instance.

CHAPTER 6

# Public-Key Encryption Subversion

In this chapter we subvert cryptographic games that relates to public-key encryption. In particular, we will consider the case of chosen-ciphertext security games in relation to the El-Gamal public-key encryption scheme.

The section in this chapter contains a subversion of the El-Gamal public-key encryption scheme in the CPA security game. To construct the subversion, we abuse the fact that the El-Gamal public-key encryption scheme surfaces a CDH instance.

## 6.1 El-Gamal subversion

We present a subversion of El-Gamal in the CPA security game. On a conceptual plane the subversions shows that if the El-Gamal encryption algorithm has access to the private key during encryption, then the El-Gamal public-key encryption scheme can be subverted. Having access to the private key is not the usual scenario of public-key encryption but if this situation occurs, you are doomed.

The subversion of El-Gamal works by recovering a CDH instance after an encryption. First we present the interactive cryptographic game.

**Theorem 6.1.1.** *Let $\mathcal{G}(1^n)$ be a polynomial-time algorithm that on input $1^n$ outputs a cyclic group $\mathbb{G}$ of prime order $q$, $|q| = n$ and generator $g$. The interactive cryptographic game $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ defined in Figure 6.1 is secure (relative to $\mathcal{G}$) when the* DDH *problem is hard relative to $\mathcal{G}$.*

Note that, we do not consider the secret key $\mathsf{sk}$ as part of the secret transcript $\tau_s$. Remark 6.1.3 elaborate on this assumption.

*Proof.* Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathsf{IN\text{-}SEC}_{\mathsf{G}}^{\mathcal{A}}(n)$. It is clear that the adversary $\mathcal{A}$ is successful if and only if $\mathcal{A}$ can guess the bit $b$ knowing only the public key $\mathsf{pk}$ and one ciphertext $c$, which is the encryption of $m_b$. If follows that security of $\mathsf{G}$ is equivalent to proving that the El-Gamal public-key encryption scheme is CPA-secure. Since the DDH problem is assumed to be hard relative to $\mathcal{G}$, security of $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ follows from Propositon 2.6.6. $\square$

We proceed by presenting a big brother that strongly subverts $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$. Having already established the CDH subversion in Section 5.2 it should be obvious how we would construct such a big brother. The transcripts produced by $\mathsf{Dev}_{\mathsf{CDH}}$ in $\mathsf{G}_{\mathsf{CDH}}$ are almost identical to the transcripts

```
$\mathsf{Par}(1^n)$
    $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
    $\mathsf{params} := (\mathbb{G}, q, g)$
    **Return** params

$\mathsf{Pre}(\tau_p, \tau_s, \sigma)$
    Parse: $\tau_s = (\mathbb{G}, q, g, b)$
    Parse: $\sigma = \hat{b}$
    If $(\hat{b} = b)$ **Return** true
    Else **Return** false

$p_0(n) := \frac{1}{2}$
$\alpha_0(n) := 2$
```

```
P                                                  $\mathsf{Dev}_{\mathsf{El\text{-}Gamal}}$
input : params                                     input : params
                                                   $s \leftarrow \mathbb{Z}_q,\ h := g^s$
                                                   $\mathsf{pk} := (\mathbb{G}, q, g, h),\ \mathsf{sk} := (\mathbb{G}, q, g, s)$
                    $\xleftarrow{\mathsf{pk}}$
Choose $m_0, m_1 \in \mathbb{G}$
$|m_0| = |m_1|$
                    $\xrightarrow{(m_0, m_1)}$
                                                   $b \leftarrow_R \{0,1\}$
                                                   $y \leftarrow_R \mathbb{Z}_q$
                                                   $c := (g^y, h^y \cdot m_b)$
                    $\xleftarrow{c}$
                                                   $\tau_p := (\mathbb{G}, q, g, \mathsf{pk}, c)$
                                                   $\tau_s := (\mathbb{G}, q, g, b)$
**Return** $\hat{b}$                               **Return** $(\tau_p, \tau_s)$
```

**Figure 6.1:** Interactive cryptographic game $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$.

produced by Dev in G but with the twist that $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ modifies the CDH instance by multiplying a message on one element of the instance. It turns out that this does not give much trouble.

**Theorem 6.1.2.** *The big brother adversary* $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ *defined in Figure 6.2, strongly subverts* $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ *if the* DDH *problem is hard relative to* $\mathcal{G}$.

*Proof.* We first prove that $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ is recoverable and afterwards that $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ is strongly undetectable. The first part is a straightforward symbol game, where one just have to check that the computation by B.Rec is actually computing the correct bit $b$. The second part is a reduction to strong undetectability of $\mathsf{B}_{\mathsf{CDH}}$.

$\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ **is recoverable**

It is obvious that B.Rec is a PPT party. We will check that $\hat{b} = b$ happens with probability 1, which implies

$$\mathrm{Adv}^{\mathsf{IN\text{-}SEC}}_{\mathsf{B}_{\mathsf{El\text{-}Gamal}}}(\mathsf{B.Rec}) = 1.$$

```
$\underline{\mathsf{B.Gen}(1^n, \mathsf{params})}$
    Parse: $\mathsf{params} = (\mathbb{G}, q, g)$
    $s_\mathsf{B} \leftarrow_R \mathbb{Z}_q$
    $h_\mathsf{B} := g^{s_\mathsf{B}}$
    $\mathsf{pk}_\mathsf{B} := (\mathbb{G}, q, g, h_\mathsf{B})$
    $\mathsf{sk}_\mathsf{B} := (\mathbb{G}, q, g, s_\mathsf{B})$
    **Return** $(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{B})$
```

```
$\underline{\mathsf{Return}_{\mathsf{B.Rec}}(\mathsf{sk}_\mathsf{B}, \tau_p)}$
    Parse: $\tau_p = (\mathbb{G}, q, g, h, c)$
    Parse: $\mathsf{sk}_\mathsf{B} = (\mathbb{G}, q, g, s_\mathsf{B})$
    Parse: $c = (g^y, h^y \cdot m_b)$
    $\hat{\eta} := h^{-s_\mathsf{B}} h$
    $\hat{y} := \mathcal{R}(\eta)$
    $\hat{m} := \left(h^{\hat{y}}\right)^{-1} \cdot h^y \cdot m_b$
    If $\hat{m} = m_0$ **Return** $0$
    Else **Return** $1$
```

B.Rec can choose messages $m_0$ and $m_1$ with no special structure

| P | | $\mathsf{B.Dev}_{\mathsf{El\text{-}Gamal}}$ |
|---|---|---|
| input : params | | input : (params, $\mathsf{pk}_\mathsf{B}$) |
| | | $s \leftarrow \mathbb{Z}_q,\ h := g^s$ |
| | | $\mathsf{pk} := (\mathbb{G}, q, g, h),\ \mathsf{sk} := (\mathbb{G}, q, g, s)$ |
| | $\xleftarrow{\mathsf{pk}}$ | |
| Choose $m_0, m_1 \in \mathbb{G}$ | | |
| $\lvert m_0 \rvert = \lvert m_1 \rvert$ | | |
| | $\xrightarrow{(m_0, m_1)}$ | |
| | | $b \leftarrow_R \{0,1\}$ |
| | | $\eta := h_\mathsf{B}^{-s} \cdot h,\ y := \mathcal{R}(\eta)$ |
| | | $c := (g^y, h^y \cdot m_b)$ |
| | $\xleftarrow{c}$ | |
| | | $\tau_p := (\mathbb{G}, q, g, \mathsf{pk}, c)$ |
| | | $\tau_s := (\mathbb{G}, q, g, b)$ |
| **Return** $\hat{b}$ | | **Return** $(\tau_p, \tau_s)$ |

**Figure 6.2:** Big brother $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ of $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$.

After running the interaction between B.Rec and $\mathsf{B.Dev}_{\mathsf{El\text{-}Gamal}}$ and the start of the game $\mathsf{IN\text{-}REC}_\mathsf{B}^\mathcal{A}(n)$, we have $\mathsf{pk} = g^s$, $\mathsf{pk}_\mathsf{B} = h_\mathsf{B}^{s_\mathsf{B}}$, $c = (g^y, h \cdot m_b)$, $\eta = h_\mathsf{B}^{-s} h$ and $y = \mathcal{R}(\eta)$. The return function $\mathsf{Return}_{\mathsf{B.Rec}}$ then computes

$$\begin{aligned}\hat{\eta} &= h^{-s_\mathsf{B}} h = g^{-s s_\mathsf{B}} h = h_\mathsf{B}^{-s} h = \eta,\\ \hat{y} &= \mathcal{R}(\hat{\eta}) = \mathcal{R}(\eta) = y,\\ \hat{m} &= (h^{\hat{y}})^{-1} h^y m_b = m_b.\end{aligned}$$

That is, $\mathsf{Return}_{\mathsf{B.Rec}}$ will output $\hat{b} = b$ with non-negligible probability.

$\mathsf{B}_{\text{El-Gamal}}$ **is undetectable**

Let $\mathcal{U}_{\text{El-Gamal}}$ be a PPT user trying to distinguish $\mathsf{Dev}_{\text{El-Gamal}}$ and $\mathsf{B.Dev}_{\text{El-Gamal}}$ in Game $\text{strong-IN-DETECT}^{\mathcal{U}}_{\mathsf{Dev}_{\text{El-Gamal}},\mathsf{B.Dev}_{\text{El-Gamal}}}(n)$. We construct a reduction $\mathcal{U}_{\text{CDH}}$ to strong undetectability of $\mathsf{B}_{\text{CDH}}$, using $\mathcal{U}_{\text{El-Gamal}}$ as a subroutine, such that

$$\text{Adv}^{\text{strong-IN-DETECT}}_{\mathsf{Dev}_{\text{El-Gamal}},\mathsf{B.Dev}_{\text{El-Gamal}}}(\mathcal{U}_{\text{El-Gamal}}) = \text{Adv}^{\text{strong-DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}_{\text{CDH}}). \tag{6.1}$$

where $\mathsf{Dev}$ and $\mathsf{B.Dev}$ are from $\mathsf{G}_{\text{CDH}}$ and $\mathsf{B}_{\text{CDH}}$ respectively (saving the subscripts). The messages $\mathcal{U}_{\text{El-Gamal}}$ expects to see are the public key $\mathsf{pk}$ and encryptions of either $m_0$ or $m_1$, where $m_0$ and $m_1$ are chosen by $\mathcal{U}_{\text{El-Gamal}}$ before each encryption. We will emulate these encryptions using the messages $\mathcal{U}_{\text{CDH}}$ receives from its oracle in Game $\text{strong-DETECT}^{\mathcal{U}_{\text{CDH}}}_{\mathsf{Dev}_{\text{CDH}},\mathsf{B.Dev}_{\text{CDH}}}(n)$.

Consider an answer to a query by $\mathcal{U}_{\text{CDH}}$ in the game $\text{strong-DETECT}^{\mathcal{U}_{\text{CDH}}}_{\mathsf{Dev}_{\text{CDH}},\mathsf{B.Dev}_{\text{CDH}}}(n)$: $\tau_p = (g^x, g^y)$ and $\tau_s = g^{xy}$. We can start by given $\mathsf{pk} = g^x$ to $\mathcal{U}_{\text{El-Gamal}}$ who then responds with messages $m_0, m_1$ ($\mathcal{U}_{\text{CDH}}$ is playing the role of the challenger (device) in $\text{strong-IN-DETECT}^{\mathcal{U}_{\text{El-Gamal}}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$). We then have to construct a response to $\mathcal{U}_{\text{El-Gamal}}$, which should be the El-Gamal encryption of $m_b$ under the public key $g^x$ and ephemeral key $g^y$. We can do this because the secret transcript $\tau_s$ (which $\mathcal{U}_{\text{CDH}}$ receives because we consider strong undetectability) contains the element $g^{xy}$. The simple computation $g^{xy} \cdot m_b$ will compute the encryption that is needed. A detailed description of $\mathcal{U}_{\text{CDH}}$ is given below: in the description $\mathcal{O}_{\text{CDH}}$ and $\mathcal{R}_{\text{CDH}}$ are the, respectively, device oracle and random oracle in Game $\text{strong-DETECT}^{\mathcal{U}_{\text{CDH}}}_{\mathsf{Dev}_{\text{CDH}},\mathsf{B.Dev}_{\text{CDH}}}(n)$; $\mathcal{O}_{\text{El-Gamal}}$ and $\mathcal{R}_{\text{El-Gamal}}$ emulates, respectively, device oracle and random oracle in the game $\text{strong-IN-DETECT}^{\mathcal{U}_{\text{El-Gamal}}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$.

```
User $\mathcal{U}_{\text{CDH}}$
Input: $(\mathsf{params}, \mathsf{pk}_\mathsf{B})$
 1: Give $(\mathsf{params}, \mathsf{pk}_\mathsf{B})$ to $\mathcal{U}_{\text{El-Gamal}}$
 2: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\text{El-Gamal}}}_{\text{El-Gamal}}$
 3: Return $\hat{b}$

    Device oracle: $\mathcal{O}_{\text{El-Gamal}}$

 1: Query: $(\tau_p^{\text{CDH}}, \tau_s^{\text{CDH}}) \leftarrow \mathcal{O}_{\text{CDH}}$
 2: Parse: $\tau_p^{\text{CDH}} = (g^x, g^y)$, $\tau_s^{\text{CDH}} = g^{xy}$
 3: $\mathsf{pk} := g^x$
 4: Send $\mathsf{pk}$ to $\mathcal{U}_{\text{El-Gamal}}$
 5: Receive $(m_0, m_1) \leftarrow \mathcal{U}_{\text{El-Gamal}}$
 6: $b' \leftarrow_R \{0,1\}$
 7: $c := (g^y, g^{xy} \cdot m_{b'})$
 8: Send $c$ to $\mathcal{U}_{\text{El-Gamal}}$
 9: $\tau_p := (\mathsf{pk}, c)$, $\tau_s := b'$
10: Return $(\tau_p, \tau_s)$
    Random oracle: $\mathcal{R}_{\text{El-Gamal}}(\lambda)$

 1: Query: $z \leftarrow \mathcal{R}_{\text{CDH}}(\lambda)$
 2: Return $z$
```

We now argue that the view of $\mathcal{U}_{\text{El-Gamal}}$ when used as a subroutine by $\mathcal{U}_{\text{CDH}}$, is identical to the view in the game $\text{strong-IN-DETECT}^{\mathcal{U}_{\text{El-Gamal}}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. We consider two cases: $b = 0$ and $b = 1$. In both cases the parameters $\mathsf{params} = (\mathbb{G}, q, g)$ given to $\mathcal{U}_{\text{El-Gamal}}$ is distributed identically in both views. If $b = 0$ the tuple $(g^x, g^y, g^{xy})$ returned by $\mathcal{O}_{\text{CDH}}$ is computed by sampling $x$ and $y$ uniformly at random from $\mathbb{Z}_q$. Since $b'$ is also a uniform element, $(\tau_p, \tau_s)$ is distributed identically from the view of $\mathcal{U}_{\mathsf{El\text{-}Gamal}}$. If $b = 1$ the tuple $(g^x, g^y, g^{xy})$ returned by $\mathcal{O}_{\mathsf{CDH}}$ is computed by sampling $x$ uniformly at random from $\mathbb{Z}_q$ and setting $\eta := h_{\mathsf{B}}^{-s}h$ and $y = \mathcal{R}(\eta)$. This is identical to the computation by $\mathsf{B.Dev}$. Hence, from the view of $\mathcal{U}_{\mathsf{El\text{-}Gamal}}$ the transcript $(\tau_p, \tau_s)$ is also distributed identically when $b = 1$.

To finish the proof, notice that $\mathcal{U}_{\mathsf{CDH}}$ wins the game strong-DETECT$^{\mathcal{U}_{\mathsf{CDH}}}_{\mathsf{Dev}_{\mathsf{CDH}},\mathsf{B.Dev}_{\mathsf{CDH}}}(n)$ if and only if $\mathcal{U}_{\mathsf{El\text{-}Gamal}}$ returns a bit $\hat{b}$ that is equal to $b$. The probability of $\mathcal{U}_{\mathsf{CDH}}$ outputting $\hat{b}$ such that $\hat{b} = b$ is $\Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}IN\text{-}DETECT}^{\mathcal{U}_{\mathsf{El\text{-}Gamal}}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)\right]$. Therefore

$$\Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}IN\text{-}DETECT}^{\mathcal{U}_{\mathsf{El\text{-}Gamal}}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)\right] = \Pr\left[\mathsf{true} \leftarrow \mathsf{strong\text{-}DETECT}^{\mathcal{U}_{\mathsf{El\text{-}Gamal}}}_{\mathsf{Dev}_{\mathsf{El\text{-}Gamal}},\mathsf{B.Dev}_{\mathsf{El\text{-}Gamal}}}(n)\right].$$

Equation 6.1 follows accordingly. □

Notice that $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$ models a multi-user scenario, where each distributed device has been subverted with the same public key. Even if these users group together to form a stronger distinguisher it would still not be impossible to detect $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$. Additionally, if the encryption algorithm has access to persistent memory, one can perform an equivalent subversion by modifying two consecutive encryptions (at an arbitrary point during encryption of multiple messages). This however, can only recover the seconded encrypted messages and not the first. The use-case for this subversion is more realistic, since the encryption does not need to be performed at the same location as the key generation.

*Remark* 6.1.3. The observant reader might have noticed that the secret key $\mathsf{sk}$ is not included in the secret transcript $\tau_s$ in $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$. One could argue that $\mathsf{sk}$ should be included but if we constructed $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ in this way, it would not be possible to do the reduction to strong undetectability of $\mathsf{B}_{\mathsf{CDH}}$. This is because we use $g^x$ as the public key in our El-Gamal instance and we have no way of computing $x$ (the DDH problem is assumed hard relative to $\mathcal{G}$, which implies that the DL problem is hard relative to $\mathcal{G}$ by Propositon 2.5.4 and Propositon 2.5.5). This does not exclude the possibility that the modified $\mathsf{G}_{\mathsf{El\text{-}Gamal}}$ and correspondingly modified big brother $\mathsf{B}_{\mathsf{El\text{-}Gamal}}$, can be proven strongly undetectable by some other means. Instead of strong undetectability one can obtain weak undetectability, which is proven in the same way as above.

Remark 6.1.3 contains a description of a candidate interactive cryptographic game that is weakly undetectable but not strongly undetectable.

CHAPTER 7

# Symmetric Subversion

Here we define a cryptographic subversion theory for when we are allowed to use symmetric big brother keys instead of asymmetric keys. The technical modifications needed are few: we simply do not give the big brother key to the distinguisher in the detect game and the adversary in the security game (essentially). In Section 4.1 we discussed that a successful reverse-engineering of a subverted device does not imply that it is possible to detect subversions of other devices. This is not the case in the symmetric model where disclosure of the big brother key would allow trivial detection of all devices subverted with the same big brother key.

The model presented in this chapter is directly inspired by [8]. To show that it translates successfully, we recover a result from [8] in Section 7.3. The result states that all IV-surfacing, probabilistic, stateless symmetric encryption schemes are prone to cryptographic subversion attacks.

It should be noted that the work of Bellare, Paterson and Rogaway [8] has undergone some critique lately [14]. To our defence, the critique in the latter paper concerns the definition of subversion resilience and is tightly connected to the problems discussed in the last paragraph of Section 4.2.1.

The chapter starts by defining a symmetric big brother in Section 7.1 and proceeds to define the properties of a cryptographic subversion in the symmetric case in Section 7.2. As a final act, Section 7.3 will describe the result mentioned above.

## 7.1 Symmetric big brother

We only present an interactive version of a symmetric big brother because this is what we will need later. It is equally easy to define a non-interactive symmetric big brother.

We must slightly modify Def. 4.1.2. Namely, the big brother key generation algorithm $\mathsf{B.Gen}$ must now output a symmetric key $\mathsf{k_B}$, the subverted device $\mathsf{B.Dev}$ must take as input a symmetric key instead of a public key and the big brother recovery algorithm $\mathsf{B.Rec}$ must as well take as input a symmetric key. Details are as follows.

**Definition 7.1.1.** A **symmetric big brother** for an interactive cryptographic game $\mathsf{G} = (\mathsf{Par}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ is a 3-tuple $(\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ denoted by $\mathsf{B}$, where

- $\mathsf{B.Gen}$ is a PPT algorithm that we call the *symmetric big brother key generation* algorithm, which on input the security parameter $n$ and parameter $\mathsf{params}$ outputs a symmetric big brother key $\mathsf{k_B} \in \{0,1\}^*$ with length at least $n$. We write $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$.
- $\mathsf{B.Dev}$ is a PPT party that we call the symmetric subversion of $\mathsf{Dev}$. $\mathsf{B.Dev}$ functions the same way as $\mathsf{Dev}$ in its interaction with another party (i.e it must adhere to the same message schedules as $\mathsf{Dev}$ and return a public and secret transcript) but in addition to the input given to $\mathsf{Dev}$ it is also given the symmetric big brother key $\mathsf{k_B} \in \{0,1\}^*$.
- $\mathsf{B.Rec}$ is a PPT party that we call the *adversarial recovery algorithm*. $\mathsf{B.Rec}$ functions like a PPT party $\mathsf{P}$ but apart from given $\mathsf{params}$ as input, it also receives the symmetric big brother key $\mathsf{k_B}$ as input to its return algorithm $\mathsf{Return_{B.Rec}}$. The return function of $\mathsf{B.Rec}$ outputs information $\delta \in \{0,1\}^*$.

## 7.2 Subversion in the symmetric case

We now aim to define what it means to subvert an interactive cryptographic game when using a symmetric key. Comments on the changes compared to the asymmetric case is given after the definition.

**Definition 7.2.1.** Given a secure interactive cryptographic game $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$. A symmetric big brother $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ for $\mathsf{G}$ is a weak/strong symmetric $(t, K)$-subversion of $\mathsf{G}$ if

1. Big brother can recover information: define recovery advantage as

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SYM\text{-}REC}}(\mathsf{B.Rec}) := \alpha_0(n) \cdot \left| \Pr\left[\mathsf{true} \leftarrow \mathsf{SYM\text{-}REC}_{\mathsf{B}}^{\mathsf{B.Rec}}(n)\right] - p_0(n) \right|,$$

   where the probability is over randomness used in game $\mathsf{SYM\text{-}REC}_{\mathsf{B}}^{\mathcal{A}}(n)$, which is defined in Figure 7.2. $\mathsf{B}$ recovers information if there exists $N \in \mathbb{N}$ such that

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SYM\text{-}REC}}(\mathsf{B.Rec}) > K$$

   for all $n > N$ and $\mathsf{B.Rec}$ is a PPT party.

2. Detection is impossible: for a PPT distinguisher $\mathcal{D}$, we define

$$\mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{type\text{-}SYM\text{-}DETECT}}(\mathcal{D}) := 2\left| \Pr\left[\mathsf{true} \leftarrow \mathsf{type\text{-}SYM\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{D}}(n)\right] - \frac{1}{2} \right|$$

   for $\mathsf{type} \in \{\mathsf{weak}, \mathsf{strong}\}$ and probability is taken over the randomness used in the game $\mathsf{type\text{-}SYM\text{-}DETECT}_{\mathsf{Dev,B.Dev}}^{\mathcal{D}}(n)$ defined in Figure 7.1. A PPT distinguisher $\mathcal{D}$ can not detect the subversion $\mathsf{B}$ if there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathsf{Dev,B.Dev}}^{\mathsf{type\text{-}SYM\text{-}DETECT}}(\mathcal{D}) \leq \mathsf{negl}(n),$$

   where $\mathcal{D}$ is allowed to make $t$ oracle queries.

3. Preserve security: for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}_{\mathsf{B}}^{\mathsf{SYM\text{-}SEC}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

```
Game weak-SYM-DETECT$^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    $b \leftarrow_R \{0,1\}$
    $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}}$
    Return $(\hat{b} \overset{?}{=} b)$
$\mathcal{O}$
    If $b = 0$: { Run Dev with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{Dev}}(\sigma_{\mathsf{Dev}})$}
    If $b = 1$:{ Run $\mathsf{B.Dev}(\mathsf{k_B})$ with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{B.Dev}}(\sigma_{\mathsf{B.Dev}})$}
    Return $\tau_p$
```

```
Game strong-SYM-DETECT$^{\mathcal{U}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    $b \leftarrow_R \{0,1\}$
    $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}}$
    Return $(\hat{b} \overset{?}{=} b)$
$\mathcal{O}$
    If $b = 0$: { Run Dev with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{Dev}}(\sigma_{\mathsf{Dev}})$}
    If $b = 1$:{ Run $\mathsf{B.Dev}(\mathsf{k_B})$ with $\mathcal{E}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{B.Dev}}(\sigma_{\mathsf{B.Dev}})$}
    Return $(\tau_p, \tau_s)$
```

| | |
|---|---|
| $\mathcal{E}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$. | $\mathcal{U}$ is allowed to make $t$ queries to the oracle $\mathcal{O}$ |
| Device and $\mathcal{E}$ start with initial state params for each query. | Device and $\mathcal{U}$ start with initial state params for each query. |

**Figure 7.1:** Distinguish game for weak/strong symmetric subversion.

The left-hand side is defined as

$$\mathrm{Adv}^{\mathsf{SYM\text{-}SEC}}_{\mathsf{B}}(\mathcal{A}) := \alpha_0(n) \cdot \left|\Pr\left[\mathsf{true} \leftarrow \mathsf{SYM\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)\right] - p_0(n)\right|,$$

where the probability is over the game $\mathsf{SYM\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$ defined in Figure 7.2.

```
Game SYM-REC$^{\mathcal{A}}_{\mathsf{B}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Run $\mathsf{B.Dev}(\mathsf{k_B})$ with $\mathcal{A}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{B.Dev}}(\sigma_{\mathsf{B.Dev}})$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}}, \mathsf{k_B})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

```
Game SYM-SEC$^{\mathcal{A}}_{\mathsf{B}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Run $\mathsf{B.Dev}(\mathsf{k_B})$ with $\mathcal{A}$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return_{B.Dev}}(\sigma_{\mathsf{B.Dev}})$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 7.2:** Recovery (left) and preservability (right) game for big brother B.

$t$ is a polynomial in $n$ and $K \in (0,1]$.

If (1) is true for arbitrary $t$ we call B a weak/strong $K$-subversion of G and we call B a weak/strong subversion of G if there exist some constant $K$ such that B is a weak/strong $K$-subversion. If properties 1, 2 and 3 are satisfied we call B *recoverable*, *undetectable* and *security preserving*. Property 1, 2 and 3 above are referred to as *recoverability*, *undetectability* and *preservability*, respectively.

Comparing the above definition to Def. 4.2.6, we report on the following changes:

**Recovery game:** Instead of given a public key to $\mathcal{A}$ and B.Dev, we give them a symmetric key (basically the only thing that changes is the notation used).

**Detect game:** We do not give the big brother key to the distinguisher.

**Security game:** As in the detection game, we do not give the big brother key to the adversary $\mathcal{A}$. This is the only difference between the $\mathsf{SYM\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$ and $\mathsf{SYM\text{-}REC}^{\mathcal{A}}_{\mathsf{B}}(n)$.

We now want to prove the same propositions as in Section 4.2. Specifically, prove that a strong subversion implies a weak subversion and that a secure interactive cryptographic game together with a strong undetectable subversion implies that the subversion is also preservable.

**Proposition 7.2.2.** *Let* $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be an interactive cryptographic game and assume* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *is a strong symmetric* $(t, K)$*-subversion of* $\mathsf{G}$*. Then* $\mathsf{B}$ *is also a weak symmetric* $(t, K)$*-subversion of* $\mathsf{G}$*.*

*Proof.* It is enough to prove property (2) in Def. 7.2.1. Given a PPT eavesdropper $\mathcal{E}$, we consider the game $\mathsf{weak\text{-}SYM\text{-}DETECT}^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. We construct a user $\mathcal{U}$ such that

$$\mathrm{Adv}^{\mathsf{strong\text{-}SYM\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) = \mathrm{Adv}^{\mathsf{weak\text{-}SYM\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{E}). \tag{7.1}$$

Below we construct $\mathcal{U}$: in the description the oracle $\mathcal{O}$ is the oracle from $\mathsf{strong\text{-}SYM\text{-}DETECT}^{\mathcal{U}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$.

```
User $\mathcal{U}$
Input: params
1: Give params to $\mathcal{E}$
2: $\hat{b} \leftarrow \mathcal{E}^{\mathcal{O}_\mathcal{E}}$
3: Return $\hat{b}$

Device oracle: $\mathcal{O}_\mathcal{E}$ $i$'th query
1: Query the oracle $\mathcal{O}$ that starts an interaction with device $D$, which is either device Dev or device B.Dev, and at the same time starts an interaction with $\mathcal{E}$
2: $\mathcal{U}$ acts as a proxy between the two interactions: messages received from $\mathcal{E}$ are send to $D$, and messages received from $D$ are send to $\mathcal{E}$
3: When $\mathcal{U}$ in the end receives the public and secret transcripts from the interaction with $D$, send the public transcript to $\mathcal{E}$
```

Since $\mathcal{E}$ must adhere to the same messages schedule in its interaction with $\mathcal{U}$ as $\mathcal{U}$ must in its interaction with $D$ there is consistency between the messages that are received and messages that are send from all three parties. It clear that $\mathcal{U}$ is a PPT party because $\mathcal{E}$, Dev and B.Dev are.

As in the proof of Propositon 4.2.7 the equality 7.1 follows immediately because the messages that $\mathcal{E}$ receives when used as a subroutine by $\mathcal{U}$ are the same messages as $\mathcal{E}$ receives in the game $\mathsf{weak\text{-}SYM\text{-}DETECT}^{\mathcal{E}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. □

The proposition above follows in the same way as Propositon 4.2.7. The same is true for the proposition below, that only requires small changes compared to Propositon 4.2.8.

**Proposition 7.2.3.** *Let* $\mathsf{G} = (\mathsf{Setup}, \mathsf{Dev}, \mathsf{Pre}, p_0, \alpha_0)$ *be an interactive cryptographic game and* $\mathsf{B} = (\mathsf{B.Gen}, \mathsf{B.Dev}, \mathsf{B.Rec})$ *a symmetric big brother for* $\mathsf{G}$*. If* $\mathsf{G}$ *is secure and* $\mathsf{B}$ *is strongly undetectable, then* $\mathsf{B}$ *also preserves security.*

*Proof.* As in Section 4.2, we must prove that there exists a negligible function $\mathsf{negl}$ such that

$$\mathrm{Adv}^{\mathsf{SYM\text{-}SEC}}_{\mathsf{B}}(\mathcal{A}) \leq \mathsf{negl}(n).$$

Our strategy follows the exact same strategy as in the proof of Propositon 4.2.8. Instead of writing the same computations again, we simply just construct the user $\mathcal{U}$ and the modified game

$\overline{\mathsf{SYM\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$, which are both used in the proof. The computations and arguments are exactly the same as in the case of an interactive big brother using asymmetric keys.

Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathsf{SYM\text{-}SEC}^{\mathcal{A}}_{\mathsf{B}}(n)$. We construct user $\mathcal{U}$ below: in the description $\mathcal{O}$ is the oracle from $\mathsf{strong\text{-}SYM\text{-}DETECT}^{\mathcal{U}}_{\mathsf{Dev,B.Dev}}(n)$.

```
User $\mathcal{U}$
Input: params
1: Give params to $\mathcal{A}$
2: Query $\mathcal{O}$ that starts an interaction
   between $\mathcal{U}$ and either Dev or B.Dev,
   and $\mathcal{U}$ simultaneously start an in-
   teraction with $\mathcal{A}$
3: Let $\mathcal{U}$ act as a proxy
4: $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
5: $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
6: if bool = true then
7:     Return 1
8: else
9:     Return 0
```

Below we construct the modified security game $\overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$.

```
Game $\overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$
    Run $\mathsf{Par}(1^n)$
    Let params be the initial state
    $\mathsf{k_B} \leftarrow \mathsf{B.Gen}(1^n, \mathsf{params})$
    Run Dev with $\mathcal{A}$
    $\delta \leftarrow \mathsf{Return}_{\mathcal{A}}(\sigma_{\mathcal{A}})$
    $(\tau_p, \tau_s) \leftarrow \mathsf{Return}_{\mathsf{Dev}}(\sigma_{\mathsf{Dev}})$
    $\mathsf{bool} \leftarrow \mathsf{Pre}(\tau_p, \tau_s, \delta)$
    Return bool
```

**Figure 7.3:** Modified security game for a non-interactive cryptographic game G

In this case, it is even more obvious that the modified game and the original game have equal output distributions i.e.

$$\Pr\left[\mathsf{false} \leftarrow \mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{G}}(n)\right] = \Pr\left[\mathsf{false} \leftarrow \overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)\right].$$

In the proof Propositon 4.2.8, we had to define a new adversary $\mathcal{A}'$ from $\mathcal{A}$. That step is redundant in this case, since $\mathcal{A}$ receives the same input in both $\overline{\mathsf{IN\text{-}SEC}}^{\mathcal{A}}_{\mathsf{G}}(n)$ and $\mathsf{IN\text{-}SEC}^{\mathcal{A}}_{\mathsf{G}}(n)$. $\square$

We could also give examples that shows that the two assumptions recoverability and undetectability can not be left out. But the examples from Section 4.2.3 does also apply in the case of symmetric cryptographic subversion.

## 7.3 Recover previous result

In this section, we describe a result from [8], namely Theorem 1. We define an interactive cryptographic game that is secure when we consider a symmetric encryption scheme that has indistinguishable encryption in the presence of an eavesdropper. The cryptographic game will be standardless.

The symmetric encryption schemes being considered must satisfy some extra requirements compared to general symmetric encryption schemes: first, they must be both randomised and stateless; second, they must be IV-surfacing. The last requirement is described more precisely below

**Definition 7.3.1.** Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme that employ the use of an initialisation vector $\mathsf{IV} \in \{0,1\}^*$ (of some fixed length in n). We write $c \leftarrow \mathsf{Enc}_\mathsf{k}(m; \mathsf{IV})$. $\Pi$ is IV-surfacing if there exists an efficient algorithm $\mathsf{f}$ such that $\mathsf{f}(\mathsf{Enc}_\mathsf{k}(m; \mathsf{IV})) = \mathsf{IV}$ for all $\mathsf{k}, m$ and IV.

The condition above says that f can recover the IV from the ciphertext. There exists many examples of IV-surfacing symmetric encryption schemes: CBC flavoured schemes with random IV are probably the best known schemes.

Below we construct the interactive game $\mathsf{G}_\mathsf{sym}$ and prove that it is secure if the scheme considered is EAV-secure.

**Theorem 7.3.2.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dev})$ *be a randomised, stateless and* IV*-surfacing symmetric encryption scheme. Assume* $\mathsf{Gen}(1^n)$ *on input* $1^n$ *outputs a uniform bit-string from* $\{0,1\}^n$*. If* $\Pi$ *has indistinguishable encryptions under the presence of an eavesdropper then* $\mathsf{G}_\mathsf{sym}$ *defined in Figure 7.4, is a secure interactive cryptographic game.*

*Proof.* It is obvious that the only difference between the game defining security of $\mathsf{G}_\mathsf{sym}$ and the game defining EAV-security of $\Pi$ is syntactic. Hence, since $\Pi$ is assumed EAV-secure the interactive cryptographic game $\mathsf{G}_\mathsf{sym}$ is also secure. $\square$

Next we present a strong subversion of $\mathsf{G}_\mathsf{sym}$. The idea is to produce the IV from the key k by encrypting k using a block cipher under the big brother symmetric key $\mathsf{k}_\mathsf{B}$. In the subversion we utilise a block cipher $\mathsf{Enc}^\mathsf{block}$ that is assumed to be a PRF and we assume that the initialisation vector IV and key k has the same length as the block size.

**Theorem 7.3.3.** *Let* $\mathsf{Enc}^\mathsf{block} : \{0,1\}^n \times \{0,1\}^n \leftarrow \{0,1\}^n$ *be a block cipher with block size* $n$ *and key-space* $\{0,1\}^n$*. Assume* $\mathsf{Enc}^\mathsf{block}$ *is a* PRF*. Then the big brother* $\mathsf{B}_\mathsf{sym}$ *defined in Figure 7.5, strongly subverts* $\mathsf{G}_\mathsf{sym}$*.*

*Proof.* We follow the usual process of first proving that $\mathsf{B}_\mathsf{sym}$ is recoverable and, afterwards, that $\mathsf{B}_\mathsf{sym}$ is undetectable.

**$\mathsf{B}_\mathsf{sym}$ is recoverable**

Since f and $\mathsf{Enc}^\mathsf{block}_\mathsf{k}$ are PPT algorithms, B.Rec is a PPT party. We will prove

$$\mathrm{Adv}^{\mathsf{SYM\text{-}REC}}_{\mathsf{B}_\mathsf{sym}}(\mathsf{B.Rec}) = 1, \tag{7.2}$$

which proves recoverability. The recovery game $\mathsf{SYM\text{-}REC}^{\mathsf{B.Rec}}_{\mathsf{B}_\mathsf{sym}}(n)$ starts by B.Gen computing a symmetric key $\mathsf{k}_\mathsf{B} \in \{0,1\}^n$. The interaction between B.Rec and B.Dev then happens where B.Rec choose messages $m_0$ and $m_1$ (with no special structure or dependence). After the interaction between B.Rec and B.Dev, we have: the symmetric key k, initialisation vector $\mathsf{IV} = \mathsf{Enc}^\mathsf{block}_{\mathsf{k}_\mathsf{B}}(\mathsf{k})$ and

```
Pre($\tau_p, \tau_s, \sigma$)                    $p_0(n) := \frac{1}{2}$
    Parse: $\tau_s = (\mathbb{G}, q, g, b)$       $\alpha_0(n) := 2$
    Parse: $\sigma = \hat{b}$
    If ($\hat{b} = b$) Return true
    Else Return false
```

| P | | Dev |
|---|---|---|
| input : $\perp$ | | input : $\perp$ |
| Choose $m_0, m_1 \in \{0,1\}^n$ | | |
| $m_0 \neq m_1$, $\lvert m_0 \rvert = \lvert m_1 \rvert$ | | |
| | $\xrightarrow{(m_0, m_1)}$ | |
| | | $\mathsf{k} \leftarrow_R \{0,1\}^n$ |
| | | $\mathsf{IV} \leftarrow_R \{0,1\}^n$ |
| | | $b \leftarrow_R \{0,1\}$ |
| | | $c \leftarrow \mathsf{Enc}_{\mathsf{k}}(m_b; \mathsf{IV})$ |
| | $\xleftarrow{c}$ | |
| | | $\tau_p := c$ |
| | | $\tau_s := b$ |
| **Return** $\hat{b}$ | | **Return** $(\tau_p, \tau_s)$ |

**Figure 7.4:** Interactive cryptographic game $\mathsf{G}_{\mathsf{sym}}$.

ciphertext $c = \mathsf{Enc}_{\mathsf{k}}(m_b; \mathsf{IV})$. The return algorithm of B.Rec then first computes $\hat{\mathsf{IV}} = \mathsf{f}(c)$. By the property of f, we have $\hat{\mathsf{IV}} = \mathsf{IV}$. Then $\hat{\mathsf{k}} \leftarrow \mathsf{Enc}^{\mathsf{block}}_{\mathsf{k_B}}(\mathsf{IV})$, where again $\hat{\mathsf{k}} = \mathsf{k}$. $\mathsf{Return}_{\mathsf{B.Rec}}$ proceeds by computing $\hat{m} = \mathsf{Dev}_{\mathsf{k}}(c; \mathsf{IV})$. We have $\hat{m} = m_b$ with probability 1 because of perfect correctness. The last conditional branch performed by B.Rec implies 7.2.

$\mathsf{B}_{\mathsf{sym}}$ **is undetectable**

To prove strong undetectability, we construct a distinguisher $\mathcal{D}$ that tries to distinguish between $\mathsf{Enc}^{\mathsf{block}}$ and a random function $f$, thereby attacking the PRF assumption. Given a PPT user $\mathcal{U}$. Consider the game strong-SYM-DETECT$^{\mathcal{U}}_{\mathsf{Dev,B.Dev}}(n)$. Below we construct $\mathcal{D}$ that uses $\mathcal{U}$ as a subroutine: in the description $\mathcal{D}$ has access to an oracle $\mathcal{O}_{\mathcal{D}}(\mathsf{k}_i)$, which takes a bit-string $\mathsf{k}_i \in \{0,1\}^n$ as input and outputs either $\mathsf{IV}_i := \mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\mathsf{k}_i)$ for $\mathsf{k} \leftarrow_R \{0,1\}^n$ or $\mathsf{IV}_i := f(\mathsf{k}_i)$ for $f \leftarrow_R \mathsf{Func}_n$.

Because $\mathcal{U}$ is a PPT user $\mathcal{D}$ is a PPT distinguisher. We may assume that $\mathcal{U}$ makes exactly $t = t(n)$ queries to the oracle $\mathcal{O}_{\mathcal{U}}$. In the execution of Game strong-SYM-DETECT$^{\mathcal{U}}_{\mathsf{Dev,B.Dev}}(n)$, we define two events:

| | | |
|---|---|---|
| Repeat | : | The event that a repetition occurs when sampling the keys k in the queries to $\mathcal{O}$. |
| Success | : | The event that $b = \hat{b}$. |

```
B.Gen(1^n)
    k_B ←_R {0,1}^n
    Return k_B
```

```
Return_B.Rec(τ_p, k_B)
    Parse: τ_p = c
    IV^ := f(c)
    k^ ← Enc^block k_B(IV)
    m^ := Dec_k(c; IV)
    If m^ = m_0 Return 0
    Else Return 1
```

B.Rec can choose messages $m_0$ and $m_1$ with no special structure

| P | | B.Dev |
|---|---|---|
| input : $\perp$ | | input : $\mathsf{k_B}$ |
| Choose $m_0, m_1 \in \{0,1\}^n$ | | |
| $\lvert m_0\rvert = \lvert m_1\rvert$ | | |
| | $\xrightarrow{(m_0,m_1)}$ | |
| | | $\mathsf{k} \leftarrow_R \{0,1\}^n$ |
| | | $\mathsf{IV} \leftarrow \mathsf{Enc}^{\mathsf{block}}_{\mathsf{k_B}}(\mathsf{k})$ |
| | | $b \leftarrow_R \{0,1\}$ |
| | | $c \leftarrow \mathsf{Enc_k}(m_b; \mathsf{IV})$ |
| | $\xleftarrow{c}$ | |
| | | $\tau_p := c$ |
| | | $\tau_s := b$ |
| **Return** $\hat{b}$ | | **Return** $(\tau_p, \tau_s)$ |

**Figure 7.5:** Big brother $\mathsf{B.Dev_{sym}}$ for $\mathsf{G_{sym}}$

The event Success is equivalent to $\mathcal{U}$ winning the game strong-SYM-DETECT$^{\mathcal{U}}_{\mathsf{Dev,B.Dev}}(n)$. Using the two events defined above, we can do the following computations

$$\begin{aligned}
\mathrm{Adv}^{\text{strong-SYM-DETECT}}_{\mathsf{Dev,B.Dev}}(\mathcal{U}) &= 2 \cdot \left| \Pr\left[\mathsf{true} \leftarrow \text{strong-SYM-DETECT}^{\mathcal{U}}_{\mathsf{Dev,B.Dev}}(n)\right] - \frac{1}{2} \right| \\
&= 2 \cdot \left| \Pr\left[\mathsf{Success}\right] - \frac{1}{2} \right| \\
&= 2 \cdot \left| \Pr\left[\mathsf{Success} \wedge \mathsf{Repeat}\right] + \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}}\right] - \frac{1}{2} \right| \\
&\le 2 \cdot \left| \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}}\right] - \frac{1}{2} \right| + 2 \cdot \Pr\left[\mathsf{Success} \wedge \mathsf{Repeat}\right] \\
&\le 2 \cdot \left| \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}}\right] - \frac{1}{2} \right| + 2 \cdot \Pr\left[\mathsf{Repeat}\right]. \qquad (7.3)
\end{aligned}$$

The first inequality is simply the triangle inequality. We now only consider the first part of the summation in equation 7.3. In the following we use that the events $\overline{\mathsf{Repeat}}$ and $b = 0$ (or $b = 1$)

**Distinguisher $\mathcal{D}$**

```
1: $\mathsf{k}_1, \mathsf{k}_2, \ldots, \mathsf{k}_t \leftarrow_R \{0,1\}^n$
2: Query: $\mathsf{IV}_i \leftarrow \mathcal{O}_{\mathcal{D}}(k_i)$ for $i = 1, 2, \ldots, t$
3: $\hat{b} \leftarrow \mathcal{U}^{\mathcal{O}_{\mathcal{U}}(m_0, m_1)}$
4: **if** $\hat{b} = 0$ **then**
5:     **Return** true
6: **else if** $\hat{b} = 1$ **then**
7:     **Return** false
```

Device oracle: $\mathcal{O}_{\mathcal{U}}(m_0, m_1)$ $i$'th query

```
1: $b' \leftarrow_R \{0,1\}$
2: $c \leftarrow \mathsf{Enc}_{k_i}(m_{b'}; \mathsf{IV}_i)$
3: $\tau_p := c$
4: $\tau_s := b'$
5: **Return** $(\tau_p, \tau_s)$
```

are independent.

$$\begin{aligned}
2 \cdot \left| \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}}\right] - \frac{1}{2} \right| &= 2 \cdot \left| \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}} \wedge (b = 0)\right] + \Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}} \wedge (b = 1)\right] - \frac{1}{2} \right| \\
&= 2 \cdot \Bigg| \Pr\left[b = 0 \wedge \overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 0)\right] \\
&\qquad + \Pr\left[b = 1 \wedge \overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 1)\right] - \frac{1}{2} \Bigg| \\
&= 2 \cdot \Bigg| \Pr[b = 0] \cdot \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 0)\right] \\
&\qquad + \Pr[b = 1] \cdot \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 1)\right] - \frac{1}{2} \Bigg| \\
&= 2 \cdot \Bigg| \frac{1}{2} \cdot \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 0)\right] \\
&\qquad + \frac{1}{2} \cdot \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 1)\right] - \frac{1}{2} \Bigg| \\
&= \Bigg| \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 0)\right] + \\
&\qquad \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 1)\right] - 1 \Bigg|. \qquad (7.4)
\end{aligned}$$

Consider the first conditional probability that conditions on $\overline{\mathsf{Repeat}}$ and $b = 0$. That is, we consider the event where $\mathcal{U}$ outputs 0 with no repetitions of keys. Then the view of $\mathcal{U}$ when used as a subroutine by $\mathcal{D}$ is distributed identically as the view from $\mathcal{U}$ in the game $\mathsf{strong\text{-}SYM\text{-}DETECT}^{\mathcal{D}}_{\mathsf{Dev},\mathsf{B.Dev}}(n)$. Since $\mathcal{U}$ outputs $\hat{b} = 0$ if and only if $\mathcal{D}$ outputs true (both conditioned on $\overline{\mathsf{Repeat}}$) when queries to $\mathcal{O}_{\mathcal{D}}$ return values computed by $\mathsf{Enc}^{\mathsf{block}}$, we have

$$\Pr\left[\mathsf{Success} \mid \overline{\mathsf{Repeat}} \wedge (b = 0)\right] = \Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \mid \overline{\mathsf{Repeat}}\right].$$

Note that the event $\overline{\mathsf{Repeat}}$ is also well-defined on the right-hand side (i.e the values $\mathsf{k}_1, \mathsf{k}_2, \ldots, \mathsf{k}_t$ queried by $\mathcal{D}$ are distinct) and that the key used by $\mathsf{Enc}^{\mathsf{block}}$ is a uniform bit-string from $\{0,1\}^n$, which matches the generation of the big brother secret key $\mathsf{k}_\mathsf{B}$. Consider now the second conditional probability in equation 7.4 that conditions on $\overline{\mathsf{Repeat}}$ and $b = 1$. Using the same arguments as before, we get

$$\Pr\left[\mathsf{Success} \,\middle|\, \overline{\mathsf{Repeat}} \wedge (b=1)\right] = \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{false} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right],$$

where we noticed that $\mathcal{U}$ outputs $\hat{b} = 1$ if and only if $\mathcal{D}$ outputs $\mathsf{false}$. Continuing from equation 7.4, we obtain

$$\begin{aligned}
2 \cdot \left|\Pr\left[\mathsf{Success} \wedge \overline{\mathsf{Repeat}}\right] - \frac{1}{2}\right| &= \Bigg|\Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] \\
&\quad + \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{false} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] - 1\Bigg| \\
&= \Bigg|\Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] \\
&\quad + \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \left(1 - \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right]\right) - 1\Bigg| \\
&= \Bigg|\Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] \\
&\quad - \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] + \left(\Pr\left[\overline{\mathsf{Repeat}}\right] - 1\right)\Bigg| \\
&\leq \Bigg|\Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] \\
&\quad - \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right]\Bigg| + \left|\Pr\left[\overline{\mathsf{Repeat}}\right] - 1\right| \\
&= \Pr\left[\overline{\mathsf{Repeat}}\right] \cdot \Bigg|\Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] \\
&\quad - \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right]\Bigg| + 1 - \Pr\left[\overline{\mathsf{Repeat}}\right] \\
&\leq \Bigg|\Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] - \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right]\Bigg| \\
&\quad + \Pr\left[\mathsf{Repeat}\right] \qquad (7.5)
\end{aligned}$$

Because $\mathsf{Enc}^{\mathsf{block}}$ is assumed to be a PRF and $\mathcal{D}$ is a PPT algorithm there exists a negligible function $\mathsf{negl}$ such that

$$\left|\Pr_{\mathsf{k} \leftarrow_R \{0,1\}^n}\left[\mathsf{true} \leftarrow \mathcal{D}^{\mathsf{Enc}^{\mathsf{block}}_{\mathsf{k}}(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right] - \Pr_{f \leftarrow_R \mathsf{Func}_n}\left[\mathsf{true} \leftarrow \mathcal{D}^{f(\cdot)}(n) \,\middle|\, \overline{\mathsf{Repeat}}\right]\right| \leq \mathsf{negl}(n). \quad (7.6)$$

Combining equations 7.3, 7.5 and 7.6, we get

$$\mathrm{Adv}^{\mathsf{strong\text{-}SYM\text{-}DETECT}}_{\mathsf{Dev},\mathsf{B.Dev}}(\mathcal{U}) \leq \mathsf{negl}(n) + 3 \cdot \Pr\left[\mathsf{Repeat}\right] \leq \mathsf{negl}(n) + 3\frac{t^2}{2^{n+1}} \leq \mathsf{negl}'(n),$$

where $\mathsf{negl}'(n) := \mathsf{negl}(n) + 3\frac{t^2}{2^{n+1}}$ is negligible because $t$ is a polynomial in $n$.

□

In the very last part of the proof of Theorem 7.3.2, we used the inequality $\Pr\left[\mathsf{Repeat}\right] \leq \frac{t^2}{2 \cdot 2^n}$, which is just an upper bound of the probability of a collision between $t$ independent uniformly chosen values from a set of size $2^n$.

CHAPTER 8

# Conclusion and Future Work

The ultimate conclusion of this thesis is that unverifiable algorithmic choice can be a significant liability. A subversion attack highlight an attack strategy that is different from conventional attacks: the takeover of a user's software for malicious purposes. The usual protection mechanisms such as testing, auditing or monitoring can not in general ensure detection and security since the attacker can use cryptographic methods to hide its modifications.

To investigate the state of possible cryptographic subversions, we define a framework for defining cryptographic subversions in a coherent manner. In this framework a technique of canonical cryptographic subversion is defined that is used to subvert the El-Gamal public-key encryption scheme. In addition, we frame a previous described cryptographic subversion in out new framework.

There are mainly two interesting directions for future work. First is to describe more previous work in the new framework. Second is to come up with more canonical subversions and show they can be used to subvert lower-level schemes. Interesting opportunities for new canonical subversions are the RSA and LWE hardness assumptions. The latter would give a novel method of subverting state-of-the-art lattice based public-key encryption schemes.

# Bibliography

[1] Nemid. https://www.nemid.nu/dk-en/.

[2] Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 626–642. Springer Berlin Heidelberg, 2012.

[3] Underhanded crypto contest, 2014. https://underhandedcrypto.com/.

[4] Vulnerability summary for cve-2014-1260 (heardbleed). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1260, April 2014.

[5] Vulnerability summary for cve-2014-1266 (goto fail). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266, February 2014.

[6] Vulnerability summary for cve-2014-6271 (shellshock). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271, September 2014.

[7] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. Cryptology ePrint Archive, Report 2015/517, 2015. http://eprint.iacr.org/.

[8] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 1–19, 2014.

[9] Daniel J. Bernstein and Tanja Lange. Choosing safe curves for elliptic-curve cryptography. http://safecurves.cr.yp.to/. Accessed: 2015-30-05.

[10] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 127–144, 1998.

[11] Mike Burmester, Yvo Desmedt, Toshiya Itoh, Kouichi Sakurai, Hiroki Shizuya, and Moti Yung. A progress report on subliminal-free channels. In *Information Hiding, First International Workshop, Cambridge, U.K., May 30 - June 1, 1996, Proceedings*, pages 157–168, 1996.

[12] MikeV.D. Burmester and Yvo Desmedt. All languages in np have divertible zero-knowledge proofs and arguments under cryptographic assumptions. In IvanBjerre Damgård, editor,

*Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 1991.

[13] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual ec in tls implementations. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 319–335, San Diego, CA, August 2014. USENIX Association.

[14] Jean Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass survillance. *Fast Software Encryption - (FSE), 22nd International Workshop on Fast Software Encryption*, 2015.

[15] Yvo Desmedt. Abuses in cryptography and how to fight them. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 375–389, 1988.

[16] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 101–126, 2015.

[17] Irene Giacomelli, Ruxandra F. Olimid, and Samuel Ranellucci. Security of linear secret-sharing schemes against mass surveillance. Cryptology ePrint Archive, Report 2015/683, 2015. `http://eprint.iacr.org/`.

[18] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA, 2006.

[19] Glenn Greenwald. No place to hide: Edward snowden, the nsa, and the u.s. surveillance state. *Metropolitan Books*, May 2013.

[20] Julian Borger James Ball and Glenn Greenwald. Revealed: how us and uk spy agencies defeat internet privacy and security. *Guardian Weekly*, September 2013.

[21] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[22] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 657–686, 2015.

[23] Jakob Møllerhøj. Smugkig på lukket test: Se en demo af nemid javascript her (danish). *Version 2*, June 2014.

[24] Jeff Larson Nicole Perlroth and Scott Shane. N.s.a able to foil basic safeguards of privacy on web. *The New York Times*, September 2013.

[25] Gustavus J. Simmons. Subliminal channels; past and present. *European Transactions on Telecommunications*, 5(4):459–474, 1994.

[26] GustavusJ. Simmons. The subliminal channel and digital signatures. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 364–378. Springer Berlin Heidelberg, 1985.

[27] Adam L. Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 89–103, 1996.

[28] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 62–74, 1997.

[29] Adam L. Young and Moti Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 264–276, 1997.

[30] Adam L. Young and Moti Yung. Bandwidth-optimal kleptographic attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, number Generators, pages 235–250, 2001.