Reduction Techniques in Computer Science

By

NGOC KHANH NGUYEN

School of Mathematics
UNIVERSITY OF BRISTOL

Supervisors: Dr. Andrew Booker & Prof. Bogdan Warinschi

A dissertation submitted to the University of Bristol in accordance with the requirements of the 40cp Project (MATHM2204).

DATE OF HANDING IN: 30TH APRIL 2018

Acknowledgement of Sources

For all ideas taken from other sources (books, articles, internet), the source of the ideas is mentioned in the main text and fully referenced at the end of the report.

All material which is quoted essentially word-for-word from other sources is given in quotation marks and referenced.

Pictures and diagrams copied from the internet or other sources are labelled with a reference to the web page,book, article etc.

Signed: Ngoc Khanh Nguyen

Dated: 30/04/2018

Acknowledgements

First of all, I would like to thank Dr. Andrew Booker for being my official supervisor and letting me do a project with collaboration of Department of Computer Science. I found the topic of this project, which combines knowledge of mathematics with real-world applications in computer science, very fascinating.

Secondly, I want to thank Prof. Bogdan Warinschi for being a great supervisor, providing help as well as giving useful comments about my progress on this project. He showed me how research in cryptography looks like and consequently I think I will continue doing research in this field.

Last but not least, I would like to thank Prof. Dennis Hofheinz for introducing me the notion of tight reductions and tightly secure primitives. We plan to submit the results from Sections 3.1 and 3.4 to the *Theory of Cryptography Conference* (TCC) 2018.

Abstract

The concept of *reduction* plays a big part in theoretical computer science. It is commonly used to show that the hardness of one problem implies the hardness of another one. For instance, in order to show that a problem A is NP-hard we need to prove that every problem in NP can be reduced to A. Furthermore, if we want to show that some language L is undecidable, we only need to reduce it to the Halting Problem. Another example would be that most of cryptographic primitives and protocols rely on some standard assumptions i.e. their security can be reduced to some mathematical problem. Different notions of reductions are defined in computability theory, complexity theory and cryptography. In this thesis we give an overview of reductions in these areas and also study the standard techniques to derive a reduction used in the literature.

The first part of the thesis provides a review of reductions in computability and complexity theory. By taking the notion of a Turing Machine as a model of computation, we look at the concept of Turing reductions and its applications to showing undecidability of a language. Next, we focus on polynomial-time (or Karp) reductions and see how they are used to show if a specific problem is harder than the other or if a problem is in a particular complexity class. Eventually, we study approximation algorithms and look at approximation-preserving reductions.

In the second part of the thesis we focus on reductions applied in cryptography. First, we study formal definitions of cryptographic reductions proposed in the literature. Then, we give an overview of standard techniques to derive reductions in this area e.g. hybrid argument, self-reducibility, information-theoretic argument and more.

Last but not least, we propose our new results in the third part of the report. Firstly, we extend the framework by Reingold et al. (TCC 2004) and define formally the notion of a tight reduction (Bellare et al., EUROCRYPT 2000). We introduce the notion of *tight extensibility*, i.e. a primitive P is tightly extensible w.r.t. Q if a reduction from P in the multi-instance setting to Q admits the same security loss as the single-instance reduction from P to Q. Then, we provide examples of tightly extensible primitives w.r.t. more general primitives. The second main result concentrates on the relationship between reductions in standard model and reduction in an external model (e.g. in the random oracle model). We look into sufficient conditions where we can conclude that standard model reduction can imply a reduction in an external model.

Contents

Chapter 1

Introduction

1.1 Motivation

A common way to derive a relation between two problems, especially to check whether one is harder than the other, is to use a technique called *reduction*. The key idea is that in order to show that problem A is harder than a problem B, one assumes existence of a solution for A and uses it to get a solution for B. We can illustrate this notion on a real-life example as follows. Suppose we want to book flight tickets to Vietnam for £500. We argue that this problem is at least as hard as getting a scholarship worth £500. Indeed, if we get the scholarship then we can use the funding we received and buy flight tickets to Vietnam. Hence, we *reduce* the problem of booking the tickets to getting a scholarship.

Mathematically, the concept of reductions is similar to *reductio ad absurdum*, i.e. the proof by contradiction. In order to show it, let us define a sentence P_X, for $X \in \{A, B\}$, which says that X is hard. We want to prove that B is as hard as A which can be written as $P_B \implies P_A$. In order to do so, we first assume that B is easy ($\neg P_B$) and use its solution to show that A is also easy ($\neg P_A$). The whole process corresponds to $\neg P_B \implies \neg P_A$ and it indeed represents a proof by contraposition. In particular, we can compose two reductions since $(P_C \implies P_B) \wedge (P_B \implies P_A) \implies (P_C \implies P_A)$. To illustrate it, let us continue with our real-life example above (which might not be true in general). Assume there is a scholarship for the best dissertation by a University of Bristol final year student worth £500. We claim that getting it is as hard as writing an excellent thesis. This is because if we can write an outstanding dissertation then with high probability we receive the scholarship. Now, using the example in the first paragraph combined with the composition property we have that booking flight tickets to Vietnam is at least as hard as writing an excellent dissertation (which in practice is much harder than the former).

In this thesis, we study various techniques to derive a reduction. So far, we have not specified how we modify and use the solution for one problem to obtain another valid solution. Here, we focus on theoretical areas of computer science and give an overview of how reductions are presented in computability theory, complexity theory as well as cryptography.

1.2 Reductions in Complexity Theory

In theoretical computer science, one thinks of a solution to some particular problem as an *algorithm.* The algorithm [1], which might be presented in pseudo-code or in language C, takes some input and parameters related to the problem and outputs a valid solution. Then, one can define a reduction as follows. We say that problem A reduces to B if given an algorithm which solves A, we can construct an algorithm which solves B. Let us provide the following simple example. Denote A to be the problem of finding all divisors of $n \in \mathbb{N}$ given n and B be the problem of determining whether n is prime. We claim that A can be reduced to B. Indeed, suppose that there exists an algorithm M_1 which solves A. So, given n, M returns a list $(p_1, p_2, ..., p_m)$ of divisors of n. Then, we can construct an algorithm M_2 which solves B as follows: run M_1 on input n and check if the list output by M_1 contains only 1 and n. If so, return `"`n` is prime"` and `"`n` is not prime"` otherwise. Note that we do not modify the code or behaviour of M_1 when building M_2 but we only use the output from M_1. This observation motivates the more specific definition of a reduction. Namely, let L_1, L_2 be the sets of all possible (valid and invalid) solutions for A and B respectively. Then, A can be reduced to B if there exists a function $f : L_1 \to L_2$ such that:

$$x \text{ is a valid solution for } A \iff f(x) \text{ is a valid solution for } B.$$

This definition is called in literature a *many-one reduction* or *Karp reduction* [57]. There are a few formalities related to sets L_1, L_2 and a function f but we omit them in this introduction. In the example above, f would take a list of divisors of n, then check if the only elements in the list are 1 and n and eventually return an appropriate answer.

Reductions are in particular interesting in case where we can reduce one problem A to another problem B and we know that A is already proven to be *hard.* Then, we can simply conclude that B is *hard* too. Hence, if one wants to show that some problem B is generally *hard*, one could find another problem A which is known to be *hard* and try to reduce to B. Now, it is just a matter of defining what *hard* is. In computability theory, *hard* means usually *undecidable.* That is, problem is undecidable if there is no algorithm which can solve it. The Halting Problem [24] is an example of an undecidable problem. In Section 2.2 we present different problems which are also undecidable by reducing them to the Halting Problem. In complexity theory, however, a problem A is $C-$*hard*, where C is a set of problems, if each problem in C *efficiently* reduces to A. The Cook-Levin Theorem states that the 3SAT problem is **NP**-hard [24, 80]. Thus,

[1] Formally, we define an algorithm to be a Turing Machine. More on this on Chapter 2.

in order to show that problem B is **NP**-hard we just need to reduce from the 3SAT to B efficiently. Examples of such reductions are shown in Section 2.3.

So far, we only considered reductions where the hardness property is preserved, meaning that A is hard $\implies$ B is hard. One could possibly think about a specific type of reductions which preserves the *distance* to an optimal solution. For instance, let x be the optimal solution for problem A (e.g. the length of the shortest path from one point to another) and y be the optimal solution for B. Then, a possible gap-preserving reduction would take any solution u for A satisfying $x \leq u \leq 2x$ and transform it into $f(u)$ such that $y \leq f(u) \leq 2$. In this case, the solution output by the reduction is still smaller than twice the optimal solution. This variant of reduction is called *approximation-preserving* [27] and we discuss them formally in Section 2.4.

1.3 Cryptographic Reductions

Most of cryptographic primitives (e.g. encryption schemes or signature schemes) cannot be proven to be secure using current techniques since that would imply that $\mathbf{P} \neq \mathbf{NP}$. Therefore, their security is often shown by *reduction* to a suitable computational assumption, such as the discrete logarithm or factoring. More precisely, we prove that if there exists an adversary $\mathcal{A}$, which breaks a primitive with *non-negligible* probability p_a then we can construct an algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to break the underlying assumption with *non-negligible* probability p_b. Negligible here means that re-running the algorithm (polynomially) many times does not improve the overall probability. Ideally, we would want that $p_a \approx p_b$. Unfortunately, many of security reductions suffer from a non-trivial *security loss* L meaning that it is only possible to show $p_a \leq L \cdot p_b$.

The reason why probability theory is involved here (unlike in the previous sections) is the Kerkchoff's principle (1883):

> A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

In particular, security of a cryptographic scheme should strongly rely on randomness it generates and hence, we say that an algorithm solves a problem with (negligible/non-negligible) probability.

We give an overview of different notions of reductions. One can distinguish different types of reductions by the way they interact with the subroutine. If we are only interested in the input/output behaviour of the underlying algorithm then this type of reduction is called *fully black-box* and is very similar to many-one reductions. The other popular variant is when we are allowed to see the code of our subroutine and is called *semi black-box.* We study these notions and provide practical examples.

We also consider cryptographic reductions which admit a small security loss L. Note that the issue of not having small loss becomes problematic in case of analysing security of primitives in the multi-instance setting, such as public key encryption schemes (PKE) [8] in the multi-user setting. Even though it can be shown [8, 10] that a secure PKE scheme implies a secure multi-user PKE scheme, the generic reduction suffers from a large security loss $L = n \cdot q$, where n and q are the number of users and ciphertexts per user respectively. As a consequence, if a PKE scheme is secure in the commonly considered one-user one-ciphertext setting, it might happen that its security level is significantly lower in the case for many users (e.g. [9]) which is more realistic in terms of practical applications. Moreover, for a large loss L we would have to make our primitives more complex in order to keep it secure and thus we end up with more inefficient scheme. Hence, we say that a reduction is *tight* [8, 25, 51] if the corresponding reduction loss L is "small" (preferably constant); in particular, it is independent of the number of users or queries made by adversary. When $L = \mathsf{poly}(\lambda)$, we say that the reduction is *almost tight.*

Some cryptographic schemes are proven to be secure (under some certain mathematical assumption) if they have access to an ideal object. A simple example would be for the scheme to use the *random oracle* (RO) [13, 54] instead of a hash function. Random oracle is a black-box oracle which responds to every unique query with a uniformly random response chosen from its output domain. Obviously, an attacker also has access to RO just like it would have access to a public hash functions. In this particular case, we would say that the scheme is secure in the *random oracle model.* There are also other external models used in cryptography for various purposes, e.g. common reference string model [31, 38], ideal ciphers [16, 26], generic group model [15, 78] or even quantum random oracle model [19]. In this thesis, we look at deriving reductions in an external model and how we could formalise them in general.

1.4 Our contribution

We first give a summary of reduction techniques used in theoretical computer science. We start off by explaining the concepts of Turing reductions as well as many-one reductions. Then, we move on to the standard notions of complexity theory and look at polynomial-time and approximation-preserving reductions. Later on, we discuss well-known techniques to derive cryptographic reductions. This includes the hybrid argument, self-reducibility and information-theoretic argument. Eventually, we talk about metareductions and how they are useful in showing a non-existence of a reduction.

In this thesis we present two new results about specific types of cryptographic reductions. They are described in more details below.

1.4.1 Tight Reductions

We provide the first general definition of a tight reduction, and revisit several classical reductions (with an emphasis on their tightness). We obtain the following results:

- We obtain a new (and tighter) security reduction for the classical construction of signatures from one-way functions [75, 58, 47].

- We also obtain tightly secure pseudorandom generators by instantiating the classical construction of Blum and Micali [18] with a suitable (i.e., rerandomisable) hard-core predicate.
- Finally, we show that the DDH-based lossy trapdoor functions of Peikert and Waters [71] are tightly secure in a multi-instance scenario.

Our new definition. Our definition of tight security adapts the general definition of reductions due to Reingold, Trevisan, and Vadhan [73]. First, we will consider the tightness of a reduction as an additional property of that reduction. Additionally, we will formalise the "multi-instance version" of a given primitive (taking into account a suitably modified multi-instance security game and potential global parameters).

Perhaps most interestingly, this modification allows to define the notion of a "tightly extensible primitive". Intuitively, a primitive X is tightly extensible relative to another primitive Y if the multi-instance version of X can be tightly reduced to Y. For instance, it is easy to see that one-way functions are tightly extensible relative to collision-resistant hash functions (CRHFs). In fact, a simple extension of an argument of Damgård [29] shows that any compressing CRHF h already is a one-way function in the multi-instance setting: suppose an algorithm A successfully finds a preimage x'_i for one of potentially many given images $h(x_i)$. Since h is compressing, we have $x'_i \neq x_i$ with probability at least $1/2$, so that (x_i, x'_i) forms a collision.

Our results. We now outline the results mentioned above.

First, we revisit the classical construction of signatures from one-way functions from [75, 58, 47]. This construction uses the one-time signature scheme of Lamport [61], which in turn uses many (i.e., $L = O(\lambda)$, where λ is the security parameter) instances of a given one-way function f. Each forged (one-time) signature implies an inversion of one instance of f. The problem here is that it is not clear a priori *which* instance is inverted. Hence, the corresponding security reduction for the one-time signature scheme (as formalised, e.g., in [43]) loses a factor of L, which of course is inherited by the security reduction of the overall signature scheme.

This loss of L can essentially be avoided if we assume that f is collision-resistant. Concretely, recall our observation above that f (when viewed as a one-way function) is tightly extensible relative to itself (when viewed as a CRHF). In particular, an adversary that inverts one out of many f-instances can be turned into a collision finder for f with a reduction loss of only 2. Hence, any forged one-time signature can be converted into an f-collision with probability of at least $1/2$, and we can save a factor of $L/2$ in the overall reduction.

Next, we consider pseudorandom generators (PRGs) G that are tightly extensible (relative to themselves). In other words, we are looking for a G such that the pseudorandomness of many $G(x_i)$ instances (for independently chosen seeds x_i) can be tightly reduced to the pseudorandomness of a single $G(x)$. This property leads to tighter reductions whenever G is used multiple times (e.g., in one or many instances of a larger scheme).

Note that an almost trivial solution to this problem can be found under the DDH assumption (assuming groups with dense representations, such that random group elements are random bitstrings). Namely, recall that the DDH assumption states that for a generator g and random exponents a, b, c, the tuple (g^a, g^b, g^{ab}) is computationally indistinguishable from (g^a, g^b, g^c). Now the DDH assumption is known to be rerandomisable (e.g., [8, Lemma 1]), in the sense a distinguisher between many $(g^{a_i}, g^{b_i}, g^{a_i b_i})$ and many $(g^{a_i}, g^{b_i}, g^{c_i})$ can be converted into a DDH distinguisher, with (almost) no reduction loss. Hence, defining $G(a, b) = g^a || g^b || g^{ab}$ yields a tightly extensible PRG.

Here, however, we are interested in constructions from (potentially) weaker assumptions. To this end, we revisit the PRG of Blum and Micali [18]. This PRG assumes a one-way permutation f with a hard-core predicate b, and defines $G(x) = f(x)||b(x)$.[2] We set $f(x) = g^x$ (which also means we require a group with dense representations), and $b(x)$ to be the Legendre symbol $(\frac{x}{p})$ of x modulo the group order p.[3] Under a suitable computational assumption (that appears to lie in between the CDH and DDH assumptions), b is indeed a hard-core predicate of f. Most importantly, and unlike with other hard-core predicates, f and b are rerandomisable: given $f(x)$ and $b(x)$, it is easy to compute $f(ax)$ and $b(ax)$ (for a known random a). Hence, by rerandomising PRG images, we can show the tight extensibility of this G.[4]

Finally, we consider the tight extensibility of lossy trapdoor functions (LTDF) relative to themselves. Our motivation to consider LTDFs is that they form an abstract tool which is already known to imply tightly (IND-CPA-)secure encryption in the single-user (but multi-ciphertext) setting [71]. A tightly extensible LTDF can be additionally useful in settings with many instances (e.g., users). Here, our main result is that the DDH-based LTDF construction of Peikert and Waters [71] is already tightly extensible. The corresponding argument is somewhat technical, but relies on the rerandomisability of the DDH assumption (as outlined already above).

We note that this last result does not yield interesting new tightly secure encryption schemes. In fact, already the ElGamal scheme is tightly IND-CPA-secure under the DDH assumption [8]. Rather, we view our last result as conceptual: it shows that an abstract building block (that was already known to enable "partially tight" reductions) is tightly secure even in a multi-instance setting.

[2]For simplicity, we only consider a PRG that stretches its seed x by one bit.

[3]Slightly simplifying, we ignore the unlikely case $x = 0$ and treat $(\frac{x}{p})$ as a bit.

[4]We note that the Legendre symbol has already been considered as a hard-core predicate by Damgård [30], although, to the best of our knowledge, its rerandomisability has not been investigated before.

1.4.2 Reductions in External Models

We use our new framework of reductions to define reductions in external model, especially in random oracle model. For simplicity, we consider non-programmable oracles. We obtain the following results:

- We introduce a new notion of an *extension of a primitive* with respect to an external model. We also define a fully black-box non-programmable reduction using the framework from previous sections.
- We look into the following problem: if a reduction holds in the standard model, then does it hold in an external (e.g. random oracle) model? We provide a few sufficient conditions for which the answer for this question is positive and also prove that it holds for a certain class of adversaries.

Extension of Primitives. Suppose we have a cryptographic primitive P in standard model. How can we define this primitive in an external model? For instance, let P be a public-key encryption scheme. We want to define an encryption scheme P' in random oracle model. Roughly speaking, it is still a triple of algorithms: key generation, encryption and decryption - these algorithms have access to the random oracle. But P and P' still have one thing in common; the algorithms have to satisfy the correctness condition. Namely, decryption of an encrypted message gives us that message. This simple example gives us motivation to characterise implementations of a primitive by the correctness conditions. We implement this idea using our framework introduced earlier.

Extending Reduction to External Models. We consider the following problem: if there exists a reduction from P to Q in the standard model, does it also hold in an external model (e.g. random oracle model)? Intuitively, the answer seems to be positive. Indeed, a reduction can just stay the same as in the standard model, and if adversary wants to query an external oracle, the reduction just passes the queries. It is not obvious, however, if such reduction still works since in an external model, both implementation and adversary have access to a certain "shared state" (which is an external oracle). This is not the case in the standard model.

We provide sufficient conditions for which this approach works. The first one is as follows. If for each implementation f of Q in an external model, there exists an *equivalent* implementation f' in the standard model then the answer is positive. Informally, this is because we can substitute f by f' and then use the assumption that the reduction works in the standard model. Another condition relies on the equivalence of two certain algorithms which combine the security games and the reduction. Due to its complexity in details, we skip this condition in the introduction.

We also show that this conjecture holds for a certain class of adversaries as long as pseudorandom functions (PRFs) exist. The proof essentially involves swapping a random oracle with a PRF (this is allowed since adversaries have

x	y	$\wedge$	$\vee$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

x	$\neg$
0	1
1	0

Figure 1.1: Boolean algebra.

the additional advice which can be treated as a PRF key) and then using the assumption that the reduction works in the standard model.

1.5 Related Works

The first formal definition of reducibility was given by Alan Turing in 1937 in terms of oracle machines [82, 80]. Later on, Stephen Kleene [59, 60] defined an equivalent concept using the notion of recursive functions. In the meantime, other definitions of Turing reductions were proposed [32, 74, 81] including the most popular nowadays, many-one reductions.

Polynomial-time reductions were studied heavily in the second half of the 20th century. More precisely, Cook [24] defined a polynomial-time Turing reduction and Karp [57] provided a *polynomial-time* version of many-one reductions. Also, the notion of polynomial-time truth-table reduction was introduced [74]. So far, the most commonly used in practice is the Karp's definition.

The notion of a reduction has been formalised early on in cryptography (e.g., in the context of black-box separations such as [7, 55, 79, 73]).[5] We note that these works were mostly interested in the (non-)existence of reductions for certain types of schemes, and do not take into account reduction loss. Hence, currently there is no *general* (i.e., formal but primitive-independent) definition of a tight reduction. Also, the papers mentioned above do not consider reductions in an general external model, although reductions in some particular models have been formalised, e.g. the random oracle model [40].

The tightness of reductions (in particular for schemes in a multi-instance scenario) has first been considered by Bellare, Boldyreva, and Micali [8]. Since their work, a variety of tightly secure constructions for concrete cryptographic building blocks (such as encryption [51, 1, 62, 63, 41, 50, 42], identity-based encryption [23, 17, 52, 4, 46], digital signatures [62, 63, 49, 2], or zero-knowledge proofs [51, 41]) have been proposed.

1.6 Preliminaries

In this section we review standard notation and mathematical definitions we use in later chapters.

[5]We also remark that other formalisation of cryptographic *assumptions* exist, e.g., [66, 34].

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{Z}_n$ be the set of integers modulo n. For a finite set S, we denote its cardinality by $|S|$ and write $s \leftarrow_{\$} S$ meaning that we choose an element s from S uniformly at random. For a function $f : A \to B$ and $C \subset A$, we define $f|_C : C \to B$ as $f|_C(x) = f(x)$. We write $\mathsf{poly}(\lambda)$ to denote an unspecified polynomial in λ. Throughout the thesis, $\perp$ denotes an error symbol.

We mentioned briefly what it means for probability to be negligible. Here, we define a negligible function.

Definition 1.1. *A function $v : \mathbb{N} \to \mathbb{R}^{\geq 0}$ is negligible if for any $c \in \mathbb{N}$:*

$$\lim_{\lambda \to \infty} v(\lambda)\lambda^c = 0.$$

We let $\mathsf{negl}(\lambda)$ denote an unspecified negligible function in λ.

Note that if the probability of breaking a scheme is $v(\lambda)$, which is negligible, then re-running the algorithm polynomially many times still gives negligible probability. Indeed, this is because $\mathsf{poly}(\lambda) \cdot v(\lambda)$ is still negligible.

Let us now review the big-O notation.

Definition 1.2. *Let $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. Then, $f(\lambda) = O(g(\lambda))$ if $\exists c \geq 0$ such that:*

$$\lim_{\lambda \to \infty} \frac{f(\lambda)}{g(\lambda)} = c.$$

For example, we have that $x^2 + 5x + 2018 = O(x^2)$ and $5 \cdot 2^n + n^{2018} = O(2^n)$.

We sometimes call a finite set A an *alphabet*. We say that a is a *letter* of A if $a \in A$. For $n \in \mathbb{N}$, we define $A^n = \underbrace{A \times A \times ... \times A}_{n}$. Then, we denote:

$$A^* = \bigcup_{n \in \mathbb{N}} A^n.$$

This operation is called the Kleene star [36].

A graph G is a pair of finite sets (V, E) where $E \subseteq V \times V$. We say that G is undirected if $(u, v) \in E \iff (v, u) \in E$. Let $V = \{v_1, ..., v_n\}$. An adjacency matrix of a graph G is a $|V| \times |V|$ matrix $\mathbf{A}$ defined by:

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

We briefly recap the Boolean algebra. Let us define the standard logical operations: and ($\wedge$), or ($\vee$), and not ($\neg$) in Fig. 1.1. We will also use the fact that for any logical values p, q we have:

$$(p \implies q) \iff (\neg q \implies \neg p) \iff (\neg p \vee q).$$

Chapter 2

Theory of Computation

We give an overview of common notions of reductions used in computability and complexity theory. We start by defining formally the model of computation we use in this thesis, namely Turing Machines (Section 1). Then, we discuss Turing reductions and how they are useful to show undecidability of a language (Section 2). Next, we focus on efficient algorithms and polynomial time reductions (Section 3). Eventually, we have a look at various approximation algorithms and approximation-preserving reductions which preserve the distance between a solution and the optimal one (Section 4).

2.1 Background

In computer science, a solution to a certain problem is presented in the form of an algorithm. Usually, we provide it in a pseudo-code or code in some programming language (e.g. C or Python). However, in order to study the notion of reduction formally, we need to define what *algorithm* and *problem* really mean. Let us first consider the latter. Note that we can divide problems into decision and optimisation ones. The first class consists of problems which can be formulated by true-false questions and answered only by `true` or `false`. A generic example of such problems is as follows: given a directed graph G and two vertices $u \neq v$, is there a path from u to v? On the other hand, an optimisation problem asks for the specific number which solves the problem. For instance, given a directed graph G and two vertices $u \neq v$, what is the shortest path from u to v? One can observe the relationship between the decision and the optimised version of a problem. Namely, if the optimisation version is easy then the decision one is also easy. For some cases the converse also holds.

In the first few sections we only focus on decision problems, i.e. true-false questions. We can define such problems using set theory. Indeed, for a problem P, we introduce a *language* L_P containing all the inputs for which the answer for P is `true`. The formal definition of a language is as follows.

Definition 2.1. *A language L over an alphabet Σ is a subset of Σ^*.*

Hence, any decision problem can be modified to be a membership problem, i.e. "does this input belong to the set of all inputs such that the answer is `true`?". Let us illustrate it with the example with graph G and vertices u, v above. Define language L_P as:

$$L_P = \{(G', u', v') : \text{ there is a path from vertex } u' \text{ to } v' \text{ in the graph } G'\}.$$

Then, this problem simply corresponds to determining whether $(G, u, v) \in L_P$.

Now, we formalise the notion of an algorithm using Turing Machines [82, 80]. This notion has been heavily used as a model of computation and represents a standard, modern computer. Roughly speaking, a Turing Machine is a model which contains a tape, from which it can read (e.g. input) and also write, as well as a list of instructions given a state it is currently in. Formally, we define it as follows.

Definition 2.2. *A Turing Machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q, Σ, Γ are finite sets and:*

- *Q is a set of states*
- *Σ is an input alphabet, where the empty sign $\llcorner\!\lrcorner$ does not belong to Σ,*
- *Γ is a tape alphabet where $\llcorner\!\lrcorner \in \Gamma$ and $\Sigma \subseteq \Gamma$*
- *$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a transition function,*
- *q_0 is a start state,*
- *$q_{acc} \in Q$ is the accept state,*
- *$q_{rej} \in Q$ is the reject state, where $q_{rej} \neq q_{acc}$.*

We describe the computation of a Turing Machine as follows. It is first given input $w = w_1 w_2 ... w_n \in \Sigma^*$ which is stored in the first leftmost n cells of the tape. The rest is filled with the blank space sign. Note that the condition $\llcorner\!\lrcorner \notin \Sigma$ makes sure that we can find out when the input ends. At the beginning, the head of the tape is set on the leftmost square. The machine starts from the start state q_0 and follows the transition function δ. Concretely, $\delta(q, a) = (r, b, X)$ means that the machine replaces a on the tape with b, the next state becomes r and the head of the tape moves in the direction defined by X (L - left, R - right). However, if it is already at the start of the tape and has to move left, it stays in the same place.

For strings $u, v \in \Gamma^*$ and a state $q \in Q$ we say that $C = uqv$ is a configuration of a Turing Machine M. We can think of a configuration as the current state of M. That is, the tape consists of uv followed by blank space signs, the current state is q and the head of the tape is at the $|u|$-th cell (if we count leftmost cells from 0). Now, let $a, b, c \in \Gamma$ and $q_1, q_2 \in Q$. We say that the configuration uaq_1bv yields uq_2cv if $\delta(q_1, b) = (q_2, c, L)$. Similarly, uaq_1bv yields $uacq_2v$ if $\delta(q_1, b) = (q_2, c, R)$.

The starting configuration of M is the configuration $C_0 = uq_0v$, where $u = \varepsilon$ (i.e. an empty string) and v is the input. We say that a configuration is accepting (resp. rejecting) if the state of the configuration is the accept (resp. reject) state. Now, a Turing Machine M accepts an input w if there exists a finite sequence of configurations $C_0, C_1, ..., C_k$ such that

- C_0 is a starting configuration,
- C_i yields C_{i+1} for $i = 0, ..., k-1$,
- C_k is an accepting configuration.

Similarly, we say that M rejects w if C_k is a rejecting configuration. We write $L(M)$ for the set of all words accepted by M.

Definition 2.3. *A language L is Turing-recognisable if there exists a Turing Machine M which recognises L, i.e. $L = L(M)$.*

We observe that given input w, a Turing Machine can do one of three possible things: accept, reject or *loop*. Here, loop means that it runs forever and never terminates. We look at concrete Turing Machines which terminate for every input and call them *deciders*. We say that a decider, which recognises some language L, *decides* L.

Definition 2.4. *A language L is Turing-decidable (or decidable) if there exists a Turing Machine M which decides L.*

Let us look at some specific examples of Turing Machines.

Example 2.5. Define a Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ as in Fig. 2.1, where $Q = \{q_0, q_1, q_{acc}, q_{rej}\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \sqcup\}$ and δ is defined as:

$$\delta(q_i, b) = (q_{i\oplus 1}, b, R) \text{ for } i = 0, 1 \text{ and } b \in \{0, 1\},$$

$$\delta(q_0, \sqcup) = (q_{acc}, \sqcup, R) \text{ and } \delta(q_1, \sqcup) = (q_{rej}, \sqcup, R).$$

One observes that M only reads input from the tape without any changes. We want to compute $L(M)$. By observing which words are accepted by M, we claim that a word is accepted by M if and only if it has even number of `1`'s. Indeed, if the current state is q_0 then the machine has read even number of `1`'s. Conversely, being in q_1 means that we read odd number of `1`'s. Hence, the only way to accept a word is when M finishes reading the input and is in the state q_0. Also, note that M decides $\{w | w \text{ has even number of } 1\text{'s}\}$ since after it finishes reading the input, it is either in q_0 (where it accepts) or in q_1 (where it rejects).

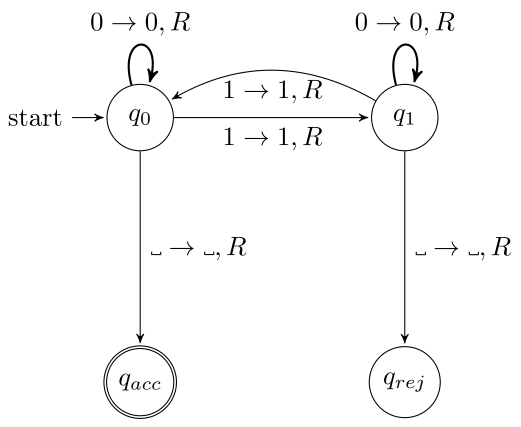

Figure 2.1: Turing Machine for Example 2.5.

In some cases we are interested in the actual content of the tape after a Turing Machine terminates. This is useful in solving optimisation problem, or simply problems for which we want more precise answer than just yes/no. The next example simply shows how to shift the input by one cell.

Example 2.6. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_3, q_{rej})$ be a Turing Machine as in Fig. 2.2 where $Q = \{q_0, q_a, q_b, q_2, q_3, q_{rej}\}, \Sigma = \{a, b\}$ and $\Gamma = \{a, b, \sqcup\}$. Here, M never rejects. We claim that M shifts the input by one cell (on the right) leaving the first leftmost cell blank. Indeed, being in the state q_x, where $x \in \Sigma$, means that we will write x on the tape in the next turn. After shifting the whole input, we end up in the state q_2 which makes the head of the tape go back to the very beginning of the tape.

When showing that a language is decidable or recognisable, one usually does not define $Q, \Sigma, \Gamma, \delta$ and so on but describes its behaviour in high-level. Specifically, we often provide a list of instructions (often in a *pseudocode*) that our Turing Machine will execute. This is because we believe that they represent the general notion of an algorithm. The famous Church-Turing thesis indeed states that every type of an algorithm can be computed by a Turing Machine.

It is a well-known fact that not all languages are decidable. It follows from the *diagonalisation* method by George Cantor (1873). This means that there exist problems which are algorithmically unsolvable. We call them *undecidable.* We present an example of such problem.

Theorem 2.7. *Let $A_{TM} = \{\langle M, w\rangle | M$ is a Turing Machine and M accepts $w\}$. Then, A_{TM} is undecidable.*

Proof. Suppose there exists a Turing Machine H which decides A_{TM}. Define a Turing Machine D which uses H as follows. On input $\langle M\rangle$, where $\langle M\rangle$ is a description of a Turing Machine M, D runs H on input $\langle M, \langle M\rangle\rangle$ and accepts if H rejects and rejects if H accepts. Now, consider the case when D is given a

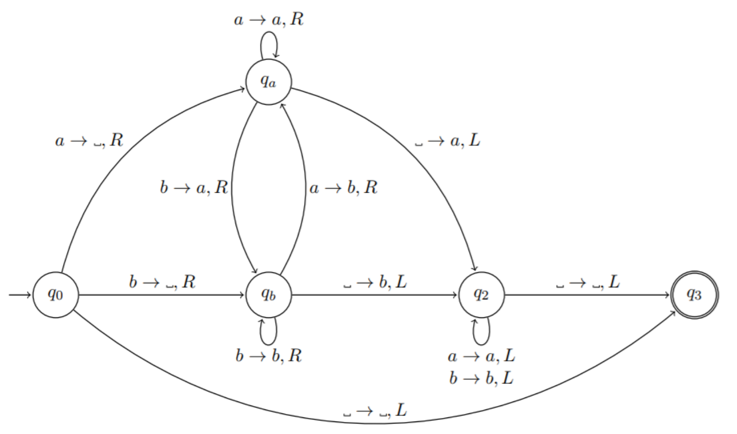

Figure 2.2: Turing Machine for Example 2.6.

description of itself, i.e. $\langle D\rangle$. If H accepts $\langle D, \langle D\rangle\rangle$ then by definition of H, D accepts $\langle D\rangle$. However, this also implies that D rejects $\langle D\rangle$ by the definition of D. Hence, we get a contradiction. In a similar fashion we can show contradiction in case when H rejects $\langle D, \langle D\rangle\rangle$. □

2.2 Turing Reductions

In this subsection we introduce the notion of Turing reductions. Informally, *reducing* a language L_1 to another language L_2 means that a decider for L_1 can be transformed into a decider for L_2. Formally, we say that there is a *Turing reduction* from L_1 to L_2 and write $L_1 \leq_T L_2$ if existence of a Turing Machine A, which decides L_2, implies the existence of a Turing Machine B (which might use A as a subroutine) which decides L_1. Note that the relation $\leq_T$ is reflexive as well as transitive. Indeed, let L_1, L_2, L_3 be languages. Clearly, $L_1 \leq_T L_1$ by definition. Now, assume that $L_1 \leq_T L_2$ and $L_2 \leq_T L_3$. Suppose that A decides L_3. Then, there exists a decided B for L_2. However, this also implies that there exists a decider C for L_1, therefore $L_1 \leq_T L_3$.

Notion of reducibility is very useful if we want to show undecidability of a language. We already know that A_{TM} is undecidable. Hence, one could prove that a language L is undecidable by just reducing it to A_{TM}. This is because if there was a decider for L then there would also be a decider for A_{TM} which leads to the contradiction. We look at the specific instance of such L, namely the Halting Problem (1936). We define it as a language HALT as follows:

$$\mathsf{HALT} = \{\langle M, w\rangle | M \text{ is a Turing Machine and } M \text{ halts on input } w\}.$$

Proposition 2.8. *The Halting Problem is undecidable.*

Proof. We just need to show $\mathsf{HALT} \leq_T A_{TM}$. Suppose that a Turing Machine D decides HALT. We construct a machine H which works as follows.

```
On input $\langle M, w\rangle$ where $M$ is a Turing Machine and $w$ is a word:
    1. Run $D$ on input $\langle M, w\rangle$.
    2. If $D$ rejects, reject.
    3. If $D$ accepts then run $M$ on $w$.
    4. If $M$ accepts, accept. Otherwise, reject.
```

We claim that H decides A_{TM}. Let us run H on $\langle M, w\rangle$. First, suppose that M accepts w. Then, we observe that D accepts (Step 3) since M halts on w. Next, H runs M on w and accepts since M also accepts (Step 4). Now, consider the case when M rejects w. Then, D accepts in Step 3 since M halts. However, H will output `reject` in Step 4 because M rejects w. The last case is when M loops on w. Then, D returns `reject` and consequently, H rejects too. By considering all the cases, we conclude that H decides A_{TM}. $\square$

We note that the transitivity property of $\leq_T$ allows us to obtain undecidability of a language just by reducing it to HALT.

Example 2.9. Consider the following language E_{TM}:

$$E_{TM} = \{\langle M\rangle | M \text{ is a Turing Machine and } L(M) = \emptyset\}.$$

We want to prove that E_{TM} is undecidable. The idea is, again, to reduce it to A_{TM}. This will imply that if there is a decider for E_{TM} then we can build a decider for A_{TM} which leads to a contradiction. Let D be a Turing Machine which decides E_{TM}. We first define a machine D_1, which takes M and w as parameters, as follows.

```
On input $x$:
    1. If $x \neq w$, reject.
    2. If $x = w$, run $M$ on $x$ and accept if $M$ accepts.
```

Note that D_1 accepts if and only x is equal to the parameter w and if M accepts w. Now, we construct the decider H for A_{TM}:

```
On input $\langle M, w\rangle$ where $M$ is a Turing Machine and $w$ is a word:
    1. Use the description of $M$ and $w$ to construct $D_1$.
    2. Run $D$ on input $\langle D_1\rangle$.
    3. If $D$ accept, reject.
    4. If $D$ rejects, accept.
```

We need to argue that $L(H) = A_{TM}$. First, suppose that a Turing Machine M accepts w. Then, D_1 accepts w and consequently $L(D_1) \neq \emptyset$. This means that D rejects $\langle D_1 \rangle$ and hence, H accepts. Conversely, if M does not accept w, then $L(D_1) = \emptyset$. Then, D accepts $\langle D_1 \rangle$ and thus, H rejects. Therefore, the result holds.

We observe from the two examples of reduction that we do not modify the actual code or behaviour of a decider D for language we reduce from. Instead, we only look at the output it returns and modify it to our needs. This motivates us to define a reduction from L_1 to L_2 as a function f such that $w \in L_1 \iff f(w) \in L_2$. This type of reduction is called *many-one* or mapping reducibility. Note that f cannot be any function. Ideally, it needs to be computable by a Turing Machine. We call such a function *computable*.

Definition 2.10. *A function $f : \Sigma^* \to \sigma^*$ is a computable function if there exists a Turing Machine M such that for every input w, M halts with $f(w)$ on its tape.*

We finally introduce the notion of a many-one reduction.

Definition 2.11. *Let L_1, L_2 be languages over the alphabet Σ respectively. Then, there is a many-one reduction from L_1 to L_2 (or $L_1 \leq_m L_2$) if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that:*

$$\forall w \in \Sigma^*, w \in L_1 \iff f(w) \in L_2.$$

We focus on some simple, but useful properties of many-one reductions. Firstly, it is a special case of a Turing reduction.

Lemma 2.12. *Every many-one reduction is a Turing reduction.*

Proof. Suppose that $L_1 \leq_m L_2$ and let f be the function stated at the definition of a many-one reduction. Also, denote M to be a Turing Machine which computes f. Assume that there exists a decider D for L_2. We define a decider H for L_2 as follows.

```
On input $w$:
  1. Compute $f(w)$ by running $A$ on input $w$.
  2. Run $D$ on input $f(w)$ and return what $D$ outputs.
```

Now, if $w \in L_1$ then $f(w) \in L_2$ so D accepts $f(w)$. Hence, H accepts w. On the other hand, if $w \notin L_1$ then $f(w) \notin L_2$ and therefore D rejects $f(w)$. Thus, H rejects. We conclude that $L(H) = L_1$. □

In a similar fashion one can show that $\leq_m$ is reflexive and transitive.

Proposition 2.13. *If $L_1 \leq_m L_2$ and L_2 is decidable then L_1 is decidable.*

Proof. Let D be a decider for L_2. We define a Turing Machine H as before. Then, it is easy to see that H decides L_1. □

The next result has been a main tool for showing undecidability and follows from the Proposition 2.13 by contraposition.

Corollary 2.14. *If $L_1 \leq_m L_2$ and L_2 is undecidable then L_1 is undecidable.*

Example 2.15. We proved that $\mathsf{HALT} \leq_T A_{TM}$. Here, we want to show that $A_{TM} \leq_m \mathsf{HALT}$. Note that by Lemma 2.12 we have that $A_{TM} \leq_T \mathsf{HALT}$, thus they are *Turing-equivalent*.

Just like in the proof of undecidability of E_{TM}, we first define a simple Turing Machine M_1 which takes the description of a Turing Machine M as a parameter:

```
On input $x$:
    1. Run $M$ on $x$.
    2. If $M$ accepts, accept.
    3. If $M$ rejects, enter the loop.
```

We construct a Turing Machine F as follows.

```
On input $\langle M, w \rangle$:
    1. Construct the machine $M_1$ with parameter $\langle M \rangle$.
    2. Output $\langle M_1, w \rangle$.
```

Let f be the function which is computable by F. We claim that f is a many-one reduction from A_{TM} to HALT. First, suppose that $\langle M, w \rangle \in A_{TM}$. This means that M accepts w. We also have that $f(\langle M, w \rangle) = \langle M_1, w \rangle$. We need to prove that $f(\langle M, w \rangle) \in \mathsf{HALT}$, i.e. M_1 halts on w. Note that if M accepts w then M_1 accepts, in particular M_1 halts. Now, assume that $\langle M, w \rangle \notin A_{TM}$. If M loops on w, then M_1 loops on w. Also, if M rejects w then M_1 also loops. Hence, in all cases M_1 does not halt. This means that $f(\langle M, w \rangle) = \langle M_1, w \rangle \notin \mathsf{HALT}$. Thus, $A_{TM} \leq_m \mathsf{HALT}$.

2.3 Polynomial-time Reductions

Even though some problems are decidable and therefore, solvable by a Turing Machine, the algorithms themselves might not be good enough to be implemented in practice, mostly due to the time and space restrictions. Hence, one should look for efficient and fast algorithms, i.e. Turing Machines which make *small* number of steps. In this subsection we define the time complexity of an algorithm and describe how useful the notion of reductions is in complexity theory.

We first introduce the notion of *running time* of a Turing Machine.

Definition 2.16. *Let M be a Turing Machine which halts on all inputs. Then, the running time of M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M then we say that M runs in time $f(n)$.*

Let us calculate the running time for Turing Machines in Examples 2.5 and 2.6. In the former one, the machine M reads the whole input of size n and then goes to the accept state if the input has even number of ones, or the reject state otherwise. This means that M runs in time $f(n) = n + 1$. In case of Example 2.6, the machine M first goes through the whole input and then goes back to the beginning of the tape. Eventually, it makes one more step to get to the accept state. Thus, it runs in time $f(n) = 2n + 1$.

Calculating the exact running time of an algorithm, which solves more complicated problems (e.g. finding the shortest path in the graph), can be extremely tedious. What we are actually interested in is its asymptotic running time and how fast it runs for large input length. We now define the complexity class $\mathbf{TIME}(f(n))$ that consists of problems which can be solved with *roughly* the same running time.

Definition 2.17. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function. Then, $\boldsymbol{TIME}(f(n))$ is the set of all languages that can be decided by a Turing Machine which runs in time $O(f(n))$.*

One observes that the machines defined in Examples 2.5 and 2.6 belong to $\mathbf{TIME}(n)$ since $n + 1 = O(n)$ and $2n + 1 = O(n)$.

The next definition introduces a class of all problems which can be solved efficiently, namely in polynomial time.

Definition 2.18. *$\boldsymbol{P}$ is the class of languages which can be decided by a Turing Machine which runs in polynomial time, i.e. $\boldsymbol{P} = \bigcup_{k=1}^{\infty} \boldsymbol{TIME}(n^k)$.*

We provide a few interesting problems which are in $\mathbf{P}$.

Example 2.19. We show that the problem of determining whether there exists a path between two vertices belongs to $\mathbf{P}$. Let us define the following PATH language:

$$\mathsf{PATH} = \{\langle G, s, t\rangle \mid \text{there exists a path from vertex } s \text{ to } t \text{ in the directed graph } G\}.$$

We construct a Turing Machine which decides PATH in polynomial time. Note that we can represent a graph G using its adjacency matrix of size $O(n^2)$ where n is the number of vertices of G.

We present a rather slow (but still polynomial time) solution for PATH. The first observation is that if there exists a path from s to t then the shortest path from s to t has length at most n. Indeed, suppose that $s_0 \to s_1 \to ... \to s_k$, where $s_0 = s$, $s_k = t$ and $k \geq n+1$, is a shortest path from s to t. Then, by the Pigeonhole Principle, there exist $0 \leq i < j \leq k-1$ such that $s_i = s_j$. Thus, $s_0 \to s_1 \to ... \to s_i \to s_{j+1} \to ... \to s_k$ is also a valid path with length strictly less than k, contradiction.

Let us define the following machine M:

```
On input $\langle G, s, t\rangle$:
1. Mark the vertex $s$.
2. For $i = 1, 2, ..., n$:
3.     For every edge $(a, b)$ of $G$:
4.         If $a$ is marked and $b$ is not marked, mark $b$.
5. If $t$ is marked, accept. Otherwise, reject.
```

Firstly, we need to argue that M indeed decides PATH. Note that at the i-th time we enter the loop in Step 2, we will find all the vertices which are reachable from s within the distance i (this follows by an easy induction proof). By the previous observation, we should only look at paths of size at most n. Therefore, when M halts, all vertices reachable from s are marked.

Let us compute the running time of M. We do Steps 1 and 5 once. Next, we enter the loop in Step 2 n times. Then, we look at every edge of G, so we iterate $O(n^2)$ times. Eventually, brute-force marking should take no longer than $O(n)$ time. Hence, we conclude that this algorithm runs in time $O(n^4)$. Note that if we denote m to be the size of the input, then $m = \Theta(n^2)$. This means that M runs in $O(m^2)$, so in polynomial time. We would like to mention that there are much faster algorithms for solving the PATH problem such as Breadth First Search (BFS) or Depth First Search (DFS).

Example 2.20. We look at the well-known satisfiability problem. Recall that the Boolean variables take values `true` (1) or `false` (0). Let us use the notation $\bar{x}$ for $\neg x$.

A *Boolean formula* is an expression which involves Boolean variables and operations. The following example is a Boolean formula:

$$\phi = (x \vee \bar{y}) \wedge (y \vee z) \wedge (\bar{x} \vee \bar{z} \vee y).$$

We say that a Boolean fomula ϕ is *satisfiable* if there exists an assignment of 0s and 1s for the Boolean variables involved in ϕ such that $\phi = 1$. In the example above, ϕ is satisfiable since for $x = 0, y = 0, z = 1$, ϕ evaluates to 1.

The satisfiability problem SAT is to determine whether a Boolean formula is satisfiable.

$$\mathsf{SAT} = \{\phi | \phi \text{ is a satisfiable Boolean formula}\}.$$

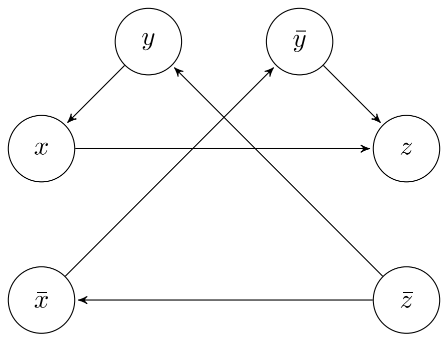

Figure 2.3: Graph G for the Boolean formula $\phi = (x \vee \bar{y}) \wedge (\bar{x} \vee z) \wedge (y \vee z)$. It is satisfiable for $x = 0, y = 0$ and $z = 1$.

In this example we look at the special case of SAT. We define a *literal* to be a Boolean variable or a negated Boolean variable, i.e. x or $\bar{x}$. A *clause* is a formula which connects all the literals with $\vee$s, e.g. $(x_1 \vee \bar{x}_2 \vee x_3)$. We say that a Boolean formula ϕ is in *conjuctive normal form* (CNF) if it consists of clauses connected with $\wedge$s, e.g.

$$\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4).$$

Moreover, a Boolean formula ϕ is a $k-$CNF formula if it is in conjuctive normal form and each clause consists of exactly k literals. We define the following language:

$$k\mathsf{SAT} = \{\phi | \phi \text{ is a satisfiable } k - \text{CNF formula}\}.$$

We first show that $1\mathsf{SAT} \in \mathbf{P}$. Let ϕ be a $1-$CNF formula. That is, ϕ consists of literals connected with $\wedge$s. Note that if there exists a Boolean variable x such that both literals x and $\bar{x}$ are involved in ϕ then ϕ is not satisfiable. This is because for any assignment of x, one of x and $\bar{x}$ will be equal to 0. Otherwise, a decider for 1SAT can just take each literal on the go and assign the corresponding variable such that the literal has the value 1. This can obviously be done in polynomial time in the input size.

Now, we give a proof sketch for $2\mathsf{SAT} \in \mathbf{P}$. Let ϕ be a $2-$CNF formula with n variables $x_1, ..., x_n$ and m clauses. Define a directed graph $G = (V, E)$ such that $V = \{x_1, ..., x_n, \bar{x}_1, \bar{x_2}, ..., \bar{x}_n\}$. Intuitively, for each variable x_i we create a true and false literal vertices x_i and $\bar{x}_i$. Now, for each clause $(a \vee b)$ of ϕ, we add an edge from $\bar{a}$ to b and $\bar{b}$ to a. One can interpret an edge as an implication. That is, if we want for $a \vee b$ to hold, then this is equivalent to $\bar{b} \implies a$ and $\bar{a} \implies b$, see Fig. 2.3.

The main observation is that ϕ is not satisfiable if and only if there exists a path from x_i to $\bar{x}_i$ and vice versa, for some i. Hence, the decider for 2SAT could check for each variable x_i if there exists a path from x_i to $\bar{x}_i$ and from $\bar{x}_i$ to x_i. Note that this can be done efficiently since this is just a PATH problem. Therefore, $2\mathsf{SAT} \in \mathbf{P}$. One could actually note that we reduced the 2SAT to PATH which we know is in $\mathbf{P}$.

It is still a big open question whether 3SAT belongs to **P**. Currently, no one has found a polynomial-time algorithm which could solve it. We observe, however, that given an assignment for variables, it is easy to check whether that assignment evaluates the 3-CNF formula to 1. Indeed, one could just plug in the values to the formula and determine if it equals to 1. Therefore, we can *verify* if a solution is correct in polynomial time. We now define the notion of a *verifier.*

Definition 2.21. *A verifier V for a language L is a Turing Machine which satisfies the following condition:*

$$L = \{w : \ V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

We say that V is a polynomial time verifier if V runs in the polynomial time in length of w. Also, we say that a language L is polynomially verifiable if there exists a polynomial time verifier for L.

We introduce the class **NP** that consists of problems which can be verified efficiently, meaning that we can quickly determine whether a solution to the problem is correct or not.

Definition 2.22. ***NP** is the class of polynomially verifiable languages.*

From the argument above, we can see that 3SAT $\in$ **NP** and SAT $\in$ **NP**. Also, all problems in **P** belong to **NP**. This is because a we can construct a verifier V for language $L \in \mathbf{P}$ as follows. Firstly, there exists a polynomial-time Turing Machine M which decides L. So, given $\langle w, c\rangle$, V would ignore c and just run M on w. Clearly, V verifies L in polynomial-time. Therefore, $\mathbf{P} \subseteq \mathbf{NP}$. However, it is still an open question whether $\mathbf{P} = \mathbf{NP}$.

Let us define the *polynomial-time reductions* (or Karp reductions). They are based on the definition of many-one reductions from Section 2.2 but this time, we want the reduction to run efficiently. First, we introduce a notion of polynomial-time computable function.

Definition 2.23. *A function $f : \Sigma^* \to \Sigma^*$ is a polynomial-time computable function if there exists a polynomial-time Turing Machine M which on input n halts with $f(n)$ on the tape.*

Now, we give a full definition of a polynomial-time reduction. Informally, we can reduce a language A to B in polynomial time if we can modify the instance of A to be an instance of B efficiently, e.g. Example 2.20 shows that we can reduce 2SAT to PATH in polynomial time.

Definition 2.24. *Language A is polynomial-time reducible to language B, written as $A \leq_P B$, if there exists a polynomial time computable function f such that*

$$\forall w \in \Sigma^*, w \in A \iff f(w) \in B.$$

The reflexive property and transitivity of $\leq_P$ follow straight from the definition. We now prove the following simple property of $\leq_P$.

Lemma 2.25. *If $A \leq_P B$ and $B \in \boldsymbol{P}$ then $A \in \boldsymbol{P}$.*

Proof. Suppose that $A \leq_P B$ and let f be the function stated at the definition of a polynomial-time reduction. Also, denote M to be a Turing Machine which computes f. Assume that there exists a polynomial-time decider D for B. We define a decider H for A as follows.

```
On input $w$:
  1. Compute $f(w)$ by running $M$ on input $w$.
  2. Run $D$ on input $f(w)$ and return what $D$ outputs.
```

It is easy to see that if D decides B then H decides A. Also, the runtime of H is equal to the sum of the runtime of M and D which is still polynomial, since M and D run in polynomial time. □

We introduce the notion of *the hardest problem* in **NP**. That is, it belongs to **NP** and every other problem in **NP** can be reduced to it. Intuitively, we can think of it as being the hardest since if there exists a solution to it then there exists a solution to any other problem in **NP**. Formally, we call such problems **NP**-complete.

Definition 2.26. *A language B is **NP**-complete if:*

- $B \in \boldsymbol{NP}$,
- $\forall A \in \boldsymbol{NP}, A \leq_P B$.

*We say that B is **NP**-hard if only the second condition is satisfied.*

Cook and Levin [24, 80] gave an example of a **NP**-complete language, namely 3SAT. The proof consists of showing that 3SAT $\in$ **NP** (which we showed earlier) and that every language in **NP** can be reduced to 3SAT in polynomial time. The second part is shown using the alternative definition of the class **NP** and converting a computation of a non-deterministic Turing Machine into a Boolean formula. We skip this proof since it is out of scope of this thesis.

Theorem 2.27 (Cook-Levin)**.** *SAT problem is **NP**-complete.*

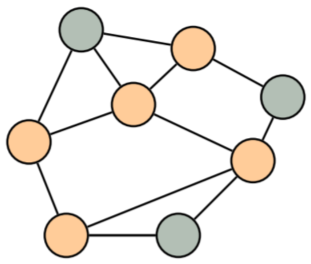

Figure 2.4: Example of a vertex cover of a graph (orange vertices).

One observes that in order to show that a language L is **NP**-complete, we just need to reduce 3SAT to L in polynomial time. The following lemma is a generalisation of this observation.

Lemma 2.28. *If A is **NP**-complete and $A \leq_P B$ for some $B \in$ **NP** then B is **NP**-complete.*

Proof. We already assumed that $B \in \mathbf{NP}$. So, we just need to show that for every $C \in \mathbf{NP}, C \leq_P B$. However, since A is **NP**-complete, we know that $C \leq_P A$. Also, $A \leq_P B$. Therefore, by the transitivity property, $C \leq_P B$, so the result holds. □

We look at some examples of **NP**-complete languages which can be reduced from 3SAT.

Example 2.29. We define a vertex cover of an undirected graph $G = (V, E)$ to be a subset S of V such that every edge in E has at least one endpoint in V (see Fig. 2.4). The optimisation version of the Vertex Cover problem (VER-COVER) is as follows: given a graph G, find a vertex cover of G of the smallest size. On the other hand, the decision version is to determine whether there exists a vertex cover of G of size k. In this example we only consider the latter version.

We claim that VER-COVER is **NP**-complete. Clearly, this problem is in **NP**. Indeed, given a subset S of vertices of a graph G, a polynomial-time verifier can check for every edge e if any endpoint of e belongs to S. Now, we just have to reduce this from 3SAT.Let ϕ be a 3CNF formula with n variables $x_1, ..., x_n$ and k clauses. We construct an undirected graph G_ϕ as follows. For every variable x_i, we introduce two vertices: x_i and $\neg x_i$. Then, we create an edge $(x_i, \neg x_i)$. We call this subgraph a *variable gadget*. Now, for the i-th clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$, we create new vertices $v_{i_1}, v_{i_2}, v_{i_3}$ and add edges between them. We call this a *clause gadget*. Eventually, we add edges $(v_{i_1}, \lambda_1), (v_{i_2}, \lambda_2), (v_{i_3}, \lambda_3)$.

We now have to prove that this reduction is correct. Specifically, ϕ is satisfiable if and only if G_ϕ has a vertex cover of size $n + 2k$. Suppose that ϕ is satisfiable. For each variable gadget, let us cover the vertex of the literal with value 1. Then, for each clause gadget, we select the vertex which is connected to the true literal in the variable gadget and cover the other two vertices in the clause. One can easily check that this set is a vertex cover of G_ϕ of size $n+2k$. Now, suppose that there exists a vertex cover of G_ϕ of size $n+2k$. Note that the minimum size of a vertex cover of G_ϕ is $n+2k$. So, if there exists a vertex cover of size exactly $n+2k$, then each variable gadget (resp. clause gadget) has to have *exactly* one vertex (resp. two vertices) covered. So, for $i = 1, ..., n$, assign $x_i = 1$ if x_i is covered or $x_i = 0$ if $\neg x_i$ is covered. We claim that such an assignment is satisfiable. Note that in each clause gadget, there exists exactly one uncovered vertex v. Also, there exists some vertex u in a variable gadget which is connected to v. This means that v must be covered. Since we assign the literal v to be true, this clause must also be true. The same reasoning can be done for all other clauses. Hence, ϕ is satisfiable.

Example 2.30. A clique of a graph $G = (V, E)$ is a subset $S \subseteq V$ of vertices such that each pair of vertices in S is connected by an edge in E. The decision problem of the CLIQUE problem is formulated as follows: given a graph G and an integer k, determine whether there exists a clique of size k.

We show that CLIQUE is **NP**-complete. Firstly, it is in **NP** since given a subset $S \subseteq V$ of vertices, we can check efficiently if each pair of vertices in S is connected by an edge. Now, we want to give a reduction from VER-COVER to CLIQUE. Let $G = (V, E)$ be graph and k be an integer. We define the complement of G as $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(u, v) | u, v \in V, u \neq v, (u, v) \notin E\}$. We claim that $\bar{G}$ has a clique of size $|V| - k$ if and only if G has a vertex cover of size k. First, assume that G has a vertex cover S of size k. This means that $\forall u, v \in V, (u, v) \in \bar{E}$ then $u \in S$ or $v \in S$. The contrapositive is that $\forall u, v \in V$, if $u \notin S$ and $v \notin S$ then $(u, v) \notin E$, so $(u, v) \in \bar{E}$. Therefore, $V \backslash S$ is a clique of size $|V| - k$ by definition. In the similar manner we can show the implication in the other direction. This proves that if we can solve the CLIQUE problem then we can also solve VER-COVER. Example 2.29, however, shows that it is **NP**-complete. Hence, by the transitivity property we get that CLIQUE is **NP**-complete.

2.4 Approximation-Preserving Reductions

In the previous sections we only considered decision problems. Here, we focus on optimisation problems i.e. the task is to find a minimum or maximum rather than yes/no. For example, we would have to return the smallest size of a vertex cover or the shortest path between two vertices.

There are dozens of **NP**-hard problems for which it seems to be difficult to solve them in polynomial time. So, the next goal would be to find an efficient algorithm which gives an answer *close* to the optimal solution. We call such solvers *approximation algorithms.* In case of VER-COVER, one could come up with an approximation algorithm which returns a number which is less than twice the minimal vertex cover size.

In this section we focus on approximation-preserving reductions. Informally, they are the reductions which preserve the distance to the optimal solutions. Note that defining this idea with a standard many-one reduction does not work since we not only want to map an instance of a problem to another instance, but we also need another function which map solutions to solutions. Due to the complexity of this idea, many types of approximation-preserving reductions have been introduced. We first present their formal definitions introduced in the literature [3, 5, 27] and then show their applications in concrete problems.

We start by formally defining a **NP**-optimisation problem.

Definition 2.31. *A **NP**-optimisation problem A is a tuple $\langle L_A, \mathsf{sol}, m, \mathsf{type}\rangle$ where:*

- *L_A is a set of instances of A which is recognisable by a polynomial-time Turing Machine,*
- *For $x \in L_A$, $\mathsf{sol}(x)$ is a set of feasible solutions of x. Elements of $\mathsf{sol}(x)$ are short, i.e. there exists a polynomial P such that $\forall y \in \mathsf{sol}(x), |y| \leq P(|x|)$. Moreover, for any x, y satisfying $|y| \leq P(|x|)$, it is decidable in polynomial time whether $y \in \mathsf{sol}(x)$,*
- *Given $x \in L$ and $y \in \mathsf{sol}(x)$, $m(x, y)$ is an objective function and denotes the positive measure integer y. We require m to polynomial-time computable,*
- *$\mathsf{type} \in \{\min, \max\}$.*

We look at some examples of **NP**-optimisation problems derived from SAT.

Example 2.32. Let ϕ be a Boolean formula in a CNF form. The MAX SAT problem is to find the largest number of clauses of ϕ satisfied by some assignment. Formally, an instance of MAX SAT is a Boolean formula ϕ in a CNF form, the solution is a truth assignment for ϕ and the measure m is the number of clauses satisfied by the truth assignment. Similarly, we can define the MAX kSAT problem where its instances are kCNF Boolean formulas. Clearly, both problems are **NP**-optimisation problems.

We introduce the class of **NP**-optimisation problems.

Definition 2.33. ***NPO** is the class of **NP**-optimisation problems.*

Let us define formally an *optimal solution*.

Definition 2.34. *Let $A = \langle L_A, \mathsf{sol}, m, \mathsf{type}\rangle$ be a **NP**-optimisation problem. For $x \in L_A$, we denote $\mathsf{opt}(x)$ to be a solution y' which satisfies*

$$m(x, y') = \mathsf{type}\{m(x, y) : y \in \mathsf{sol}(x)\}.$$

An approximation algorithm aims to return a result which is as close to the optimal solution as possible. This idea is captured by the notion of the *performance ratio.*

Definition 2.35. *Let $A = \langle L_A, \mathsf{sol}, m, \mathsf{type}\rangle$ be a **NP**-optimisation problem and $x \in L_A$. For $y \in \mathsf{sol}(x)$, we define the performance ratio of y with respect to x as:*

$$R(x, y) = \max\{\frac{m(x, y)}{\mathsf{opt}(x)}, \frac{\mathsf{opt}(x)}{m(x, y)}\}.$$

Now, we define the notion of a r-approximation algorithm.

Definition 2.36. *Let $A = \langle L_A, \mathsf{sol}, m, \mathsf{type}\rangle \in$ **NPO** and T be a polynomial-time Turing Machine which for all $x \in L_A$ it returns a feasible solution $T(x) \in \mathsf{sol}(x)$ in polynomial time. Given $r > 1$, we say that T is an r-approximation algorithm for A if for all $x \in L_A$, the performance ratio of $T(x)$ w.r.t. x is less or equal to r, i.e. $R(x, T(x)) \leq r$.*

Example 2.37. We construct an efficient 2-approximation algorithm T for the optimisation version of VER-COVER as follows.

```
On input: graph $G = (V, E)$:
  1. $C \leftarrow \emptyset$.
  2. $E' = E$.
  3. while $E'$ is not empty:
  4.       Let $(u, v) \in E'$.
  5.       $C \leftarrow C \cup \{u, v\}$.
  6.       Delete from $E'$ any edge incident to either $u$ or $v$.
  7. return  $C$.
```

Let C^* be a vertex cover of G with the minimal size. We need to show that $|C^*| \leq |C| \leq 2|C^*|$. The first inequality holds since C^* has the smallest size. Denote A to be the set of edges picked by T. Then, each edge in A contains an endpoint which belongs to C^*. Also, edges in A do not share any endpoints. Therefore, $|A| \leq |C^*|$. On the other hand, when T picks an edge, it adds exactly two vertices to C. Therefore, $|C| = 2|A|$ and consequently, $C \leq 2|C^*|$.

We now introduce two complexity classes **APX** (approximable) and **PTAS** (polynomial-time approximation scheme).

Definition 2.38. *A **NP**-optimisation problem A belongs to the class **APX** if there exists $\epsilon > 0$ for which there is an ϵ-approximation algorithm for A.*

Clearly, we have VER-COVER $\in$ **APX** (choose $\epsilon = 2$).

Definition 2.39. *A* ***NP****-optimisation problem A belongs to the class* ***PTAS*** *if for any $\epsilon > 0$, there exists a polynomial-time algorithm T which is a $(1+\epsilon)$-approximation algorithm for A.*

Arora [3] proved that the well-known Travelling Salesman Problem in the Euclidean plane belongs to **PTAS**.

In the following lemma, we present the relationship between the three classes defined in the section.

Lemma 2.40. ***PTAS*** $\subseteq$ ***APX*** $\subseteq$ ***NPO****.*

Proof. Firstly, any problem in **APX** is a **NP**-optimisation problem by definition. Therefore, **APX** $\subseteq$ **NPO**. Now, take a problem A in **PTAS**. Also, set $\epsilon = 1$. Then, there exists a polynomial-time algorithm T which is a 2-approximation algorithm for A. Hence, $A \in$ **APX**. □

The major open question in complexity theory is whether these inclusions are strict or not. We know, however, that **P** $\neq$ **NP** implies **PTAS** $\subset$ **APX**.

In the rest of this section we look at approximation-preserving reductions. Due to the large number of different types of such reductions [27, 5], we only consider the most practical ones, $L-$,$PTAS-$ and AP-reductions.

Definition 2.41. *Let $A = \langle L_A, \mathsf{sol}_A, m_A, \mathsf{type}_A \rangle$, $B = \langle L_B, \mathsf{sol}_B, m_B, \mathsf{type}_B \rangle$ be* ***NP****-optimisation problems. We say that there is a L-reduction from A to B and write $A \leq_L B$ if there exists a pair of polynomial time computable functions (f, g) and some positive constants α, β which satisfy the following conditions:*

- $\forall x \in L_A, \mathsf{opt}_B(f(x)) \leq \alpha \mathsf{opt}_A(x)$,
- $\forall x \in L_A, \forall y \in \mathsf{sol}_B(x), E_A(x, g(x,y)) \leq \beta E_B(f(x), y)$, *where:*

$$E(x,y) := |\mathsf{opt}(x) - m(x,y)|.$$

One can think of a L-reduction as a transformation of optimisation problems which linearly preserves approximability features. The following result shows that $L-$reducibility preserves the membership in **PTAS**.

Proposition 2.42. *If $A \leq_L B$ and $B \in$* ***PTAS*** *then $A \in$* ***PTAS****.*

Proof. Let (f, g, α, β) be the L-reduction from A to B. First, suppose that there exists a computable function $c : \mathbb{Q} \cap (1, \infty) \to \mathbb{Q} \cap (1, \infty)$ such that for any $x \in I_A$, for any $y \in \mathsf{sol}_B(f(x))$ and for any $r > 1$:

$$R_B(f(x), y) \leq c(r) \implies R_A(x, g(x,y)) \leq r.$$

Let $\epsilon > 0$. Take $\delta = c(1+\epsilon)$. Then, there exists an efficient algorithm T which δ-approximates B since $B \in$ **PTAS**. Define an algorithm $T'(x) = g(x, T(f(x)))$.

We know that that $R_B(f(x), T(f(x))) \leq \delta = c(1+\epsilon)$. But this implies that $R_B(x, T'(x)) \leq 1+\epsilon$ by the definition of T'. Hence, T' approximates $1+\epsilon$.

We now have to show the existence of such c. First, suppose that A is a minimisation problem. Since (f, g, α, β) is a L-reduction, we know that:

$$\frac{E_A(x, g(x,y))}{\mathsf{opt}_A(x)} \leq \alpha\beta \frac{E_F(f(x), y)}{\mathsf{opt}_B(f(x))}.$$

Note that:

$$\frac{E_F(f(x), y)}{\mathsf{opt}_B(f(x))} \leq R_B(f(x), y) - 1$$

. Consequently, we have that

$$R_A(x, g(x,y)) \leq 1 + \alpha\beta(R_B(f(x), y) - 1).$$

One can now choose $c(r) = 1 + \frac{r-1}{\alpha\beta}$ and this would satisfy the properties of c. Similarly, we can define $c(r) = 1 + \frac{r+1}{\alpha\beta r}$ for the maximisation problem. □

In the similar manner we can prove the following lemma.

Lemma 2.43. *Suppose that $A \leq_L B$ and $B \in$ **APX**. If A is a minimisation problem then $A \in$ **APX**.*

Unfortunately, if A is a maximisation problem then it is not certain that A belongs to **APX**. A concrete counterexample can be found in [28].

We now look at the next type of an approximation-preserving reduction, called PTAS-reduction. Not only does it preserve the membership in **PTAS** but it is also used to define completeness for certain classes of optimisation problems such as **APX**.

Definition 2.44. *Let $A = \langle L_A, \mathsf{sol}_A, m, \mathsf{type}_A \rangle$, $B = \langle L_B, \mathsf{sol}_B, m_B, \mathsf{type}_B \rangle$ be **NP**-optimisation problems. We say that there is a PTAS-reduction from A to B and write $A \leq_{PTAS} B$ if there exists a triple of polynomial time computable functions (f, g, c), where $c : \mathbb{Q} \cap (1, \infty) \to \mathbb{Q} \cap (1, \infty)$, which satisfies the following conditions:*

- *$\forall x \in L_A, \forall r > 1, f(x, r) \in L_B$ is computable in polynomial time,*
- *$\forall x \in L_A, \forall r > 1, \forall y \in \mathsf{sol}_B(f(x,r)), g(x, y, r) \in \mathsf{sol}_A(x)$ is computable in polynomial time,*
- *$\forall x \in L_A, \forall r > 1, \forall y \in \mathsf{sol}_B(f(x,r))$:*

$$R_B(f(x,r), y) \leq c(r) \implies R_A(x, g(x, y, r)) \leq r.$$

The PTAS-reduction is the first example of reducibility that allows the functions f and g to depend on the performance ratio.

The following lemma says that the PTAS-reduction preserves the membership in **PTAS** just like the L-reduction. The proof is almost identical to the one in Proposition 2.42.

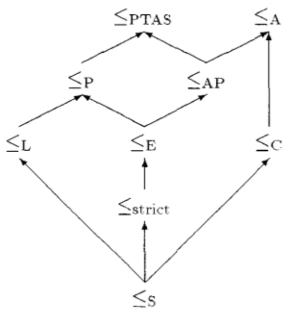

Figure 2.5: Different types of approximation-preserving reductions used in the literature. Picture from [27].

Lemma 2.45. *If $A \leq_{PTAS} B$ and $B \in \textbf{PTAS}$ then $A \in \textbf{PTAS}$.*

We have not yet defined a type of an approximation-preserving reduction which could preserve the membership in both **PTAS** and **APX**. An interesting case of a PTAS-reduction, that satisfies this property, is called AP-reduction.

Definition 2.46. *A PTAS-reduction (f, g, c) from A to B is an AP-reduction, written as $A \leq_{AP} B$, if there exists a constant α such that for any $r \in \mathbb{Q} \cap (1, \infty)$ we have $c(r) = 1 + \frac{r-1}{\alpha}$.*

We now prove the useful property of an AP-reduction mentioned earlier.

Lemma 2.47. *Suppose that $A \leq_{AP} B$. Then,*

(i) if $B \in \textbf{PTAS}$ then $A \in \textbf{PTAS}$,

(ii) if $B \in \textbf{APX}$ then $A \in \textbf{APX}$.

Proof. Clearly, $A \leq_{AP} B \implies A \leq_{PTAS} B$. Hence, the result holds by Proposition 2.42. Now, suppose that $B \in \textbf{APX}$. This means that there is an efficient algorithm which r-approximates B. Note that $A \leq_{AP} B$ implies that the function c is invertible. Let r' be the inverse of r, i.e. $c(r') = r$. Define a machine $T'(x) = g(x, T(x), r')$. Then, we have that $R_B(f(x, r), T(x)) \leq r = c(r')$ and by the third condition in the definition of a PTAS-reduction: $R_A(x, T'(x)) \leq r'$. Thus, T' r'-approximates A. □

In this brief overview we only looked at three main types of approximation-preserving reductions. We provide a graph of other types of such reductions used in the literature along with relationships between them, see Fig. 2.5.

We provide a few examples of how one can apply approximation-preserving reductions in concrete problems. Let us first focus on the MAX-kSAT defined earlier. We state the implication of the PCP Theorem [6], which is out of scope of this thesis, that says MAX-3SAT $\notin$ **PTAS** if **P** $\neq$ **NP**.

Theorem 2.48 (Corollary of the PCP Theorem)**.** *MAX-3SAT* $\notin$ ***PTAS*** *unless* $\boldsymbol{P} = \boldsymbol{NP}$.

This means that if we can reduce MAX-3SAT to some problem B (via any reduction defined in this section), then $B \notin$ PTAS unless $\mathbf{P} = \mathbf{NP}$. We now give instances of such B.

Example 2.49. We show that MAX-3SAT L-reduces to MAX-2SAT. Let ϕ be an instance of MAX-3SAT wih m clauses and denote $a_i \vee b_i \vee c_i$ to be the i-th clause for $1 \leq i \leq m$. For the i-th clause we associate the following ten new clauses:

1. a_i 2. b_i 3. c_i 4. d_i 5. $\neg a_i \vee \neg b_i$ 6. $\neg a_i \vee \neg c_i$ 7. $\neg b_i \vee \neg c_i$

8. $a_i \vee \neg d_i$ 9. $b_i \vee \neg d_i$ 10. $c_i \vee \neg d_i$.

Here, d_i is a new variable. Also, if a clause has less than 2 literals (clauses 1,2,3,4) then we introduce a new variable and add fill the clause. Let $f(\phi) = \phi'$ be the resulting instance of MAX-2SAT. One observes that if $a_i \vee b_i \vee c_i$ is true then one can select d_i such that exactly 7 of the clauses above are satisfied. For instance, if all three literals are true then clauses 1,2,3,7,8,9 are satisfied and we can just choose $d_i = 1$. We argue similarly for cases where one (resp. two) of a_i, b_i, c_i is (resp. are) true. On the other hand, if they are all false, then there is no assignment for d_i which can satisfy at least 7 out of 10 clauses. Hence, we have that $\mathsf{opt}(\phi') = 6m + \mathsf{opt}(\phi)$. Johnson [56] proved that $\mathsf{opt}(\phi) \geq \frac{m}{2}$. Thus:

$$\mathsf{opt}(\phi') = 6m + \mathsf{opt}(\phi) \leq 13\mathsf{opt}(\phi).$$

For any truth assignment $\mathbf{x}'$ for ϕ', we define $g(\phi, \mathbf{x}') = \mathbf{x}$ to be truth assignment only with respect to variables involved in ϕ. Then, by the definition of function m defined in 2.32 and the observation above, we also have that $m(\phi', \mathbf{x}') \leq m(\phi, \mathbf{x}) + 6m$. Hence,

$$\mathsf{opt}(\phi) - m(\phi, \mathbf{x}) = \mathsf{opt}(\phi') - 6m - m(\phi, \mathbf{x}) \leq \mathsf{opt}(\phi') - m(\phi', \mathbf{x}').$$

Thus, by setting $\alpha = 13$ and $\beta = 1$ we get that MAX-3SAT $\leq_L$ MAX-2SAT.

Example 2.50. Recall that in the optimisation version of the CLIQUE, we want to find a size of the largest clique in the graph. Let $r > 1$. We prove that if there exists a r-approximation algorithm for CLIQUE then there exists a r-approximation algorithm for MAX-3SAT. This means that CLIQUE $\notin$ **PTAS** unless **P**=**NP**.

Let ϕ be an instance of MAX-3SAT. Let $X = x_1, ..., x_n$ and $C = c_1, ..., c_m$ be variables and clauses of ϕ respectively. For a literal x, We write $x \in c$ meaning that the clause c contains x. Define the graph $G = (V, E)$ where

$$V = \{(x, c) | x \in c \wedge c \in C\} \text{ and } E = \{((x, c), (x', c')) | (x \neq \neg x' \wedge c \neq c'\}.$$

Let V' be any clique of G. Let us define the following truth assignment f:

$$f(x) = 1 \text{ if } (x, c) \in V'.$$

For variables x_i for which $f(x_i)$ are not yet defined, we set $f(x_i) = 0$ (and consequently, $f(\neg x_i) = 1$. We claim that f is well-defined. Suppose that $f(x) = 1$ and $f(\neg x) = 1$ for some literal x. So, (x, c) and $(\neg x, c')$ belong to V' for some $c, c' \in C$. Hence, there is an edge between them since V' is a clique. Therefore, $x \neq \neg(\neg x))$ which leads to the contradiction. One observes that f satisfies at least $|V'|$ clauses, i.e. $m(\phi, f) \geq |V'|$. The result holds in particular when we choose a clique $V' = V_{max}$ with the largest size. Now, for any subset $C' \subseteq C$ of clauses, any satisfiable truth assignment f' for C' and any $c \in C'$, we denote l_c to be any literal x of c such that $f'(c) = 1$. Then, note that the set $V' = \{(l_c, c) : c \in C'\}$ is a clique in G and $|V_{max}| \geq |V'| = |C'|$. Consequently, the size of the largest clique is equal to the largest number of satisfiable clauses. Finally, it is easy to see that all the transformations can be done in polynomial time and thus, the result holds. We observe that this example can be classified as a PTAS-reduction with $c(r) = r$.

Example 2.51. Let us again consider the CLIQUE problem. We claim that $\mathsf{CLIQUE} \notin \mathbf{APX}$ unless $\mathbf{P} = \mathbf{NP}$. Suppose there exists a r-approximation algorithm A for CLIQUE. Roughly speaking, we will first take a graph G, modify it to get a larger graph $f(G)$ and run A on $f(G)$. The approximate solution of $A(f(G))$ will turn out to be useful in finding a better approximate solution to G and hence, we find an approximation scheme. Consequently, $\mathsf{CLIQUE} \in \mathbf{PTAS}$ which leads to contradiction.

Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Define a *product graph* $G^k = (V_k, E_k)$ where $V_k = \{(v_1, ..., v_k) | v_1, ..., v_k \in V\}$ and $E_K = \{((u_1, ..., u_k), (v_1, ..., v_k)) | \forall i, u_i = v_i \vee (u_i, v_i) \in E\}$. The main observation is that if C is a clique in G then $\{(v_1, ..., v_k) | \forall i, v_i \in C\}$ is a clique in G^k of size $|C|^k$. Therefore, $\mathsf{opt}(G^k) \geq \mathsf{opt}(G)^k$. Now, suppose that C' is a clique in G^k of size $m^k < x \leq (m+1)^k$. Fix i and denote

$$n_i = |\{a \in G | \exists v_1, ..., v_{i-1}, v_{i+1}, ..., v_n, (v_1, ..., v_{i-1}, a, v_{i+1}, ..., v_n) \in C'\}|.$$

Note that we can bound the size of C' by $n_1 n_2 ... n_k$. Since $m^k < |C'|$, there exists some i such that $n_i \geq m + 1$. Therefore, the vertices counted in n_i form a clique in G since C' is a clique. Let g be the algorithm which computes such a clique of size at least $m + 1$ from C'. Hence, we have that:

$$m(G^k, A(G^k)) \leq m(G, g(A(G^k)))^k,$$

where m measures the size of a clique. Using these two observations, we get:

$$\frac{\mathsf{opt}(G)}{m(G, g(A(G)))} \leq \frac{\mathsf{opt}(G^k)}{m(G^k, A(G^k))} \leq r^{\frac{1}{k}}.$$

Let $\epsilon > 0$. If we take $k = \frac{\log r}{\log(1+\epsilon)}$, we get that $g \circ A$ $(1+\epsilon)$-approximates CLIQUE, so $\mathsf{CLIQUE} \in \mathsf{PTAS}$. This leads to a contradiction unless $\mathbf{P} = \mathbf{NP}$.

The CLIQUE problem has a property called *self-improvability.* This means that one can reduce one instance of the problem to another instance of the same problem. In the next chapter we will come across problems of similar shape and use the method called *self-reducibility.*

Chapter 3

Cryptographic Reductions

Reductions in cryptography differ from the ones in computability theory due to two main reasons. First, we required earlier that a solver has to solve the problem for all instances/inputs. We call this a worst-case scenario. Here, an attacker might not break the scheme for every input but it can break if the input given is chosen *uniformly at random.* This is called average-case scenario [14, 80]. Secondly, in the previous chapter, an algorithm either solves a problem or not. Therefore, no randomness is involved here. In cryptography, however, most of schemes generate some randomness in order to be secure. We also let the attacker generate random coins. Hence, we can only say that an attacker breaks the scheme with some probability. Thus, we say that a scheme A reduces to scheme B if existence of an attacker which breaks A with *big* probability, equal to p_A, implies the existence of an attacker which breaks B with also *big* probability, equal to p_B. Most of the time, unfortunately, one can only show that $p_A \leq L \cdot p_B$ for some L. We recall that L is a *security loss.* In general, we want $L \approx 1$ so that $p_A \approx p_B$.

In this chapter we look at reductions in cryptography and their applications in showing security of a cryptographic scheme. We first provide relevant background about standard cryptographic primitives, e.g. encryption schemes and digital signatures, as well as some attempts at formalising the notion of reduction in the literature (Section 1). Then, we propose our definitions of cryptographic reductions and apply them in concrete examples (Section 2). The following Section 3 concentrates on various specific techniques for proving reductions, namely *hybrid argument, self-reducibility* and *information-theoretic argument.* Next, we focus on *tight reductions* and secure primitives which are still secure when considered in the multi-instance setting (Section 4). Then, we focus on *meta-reduction* techniques to show that there exists no reduction (Section 5). Eventually, we consider reductions in external models such as random oracle model (Section 6).

3.1 Background

3.1.1 Notation

We denote the security parameter by $\lambda \in \mathbb{N}$ and assume that it is implicitly given to all algorithms in the unary representation 1^λ, unless stated otherwise. An algorithm here is defined as a stateless Turing Machine. Algorithms are randomised (meaning that algorithms can generate random coins) and PPT means "probabilistic polynomial time" in the (unary) security parameter λ. For a randomised algorithm $\mathcal{A}$, we denote $\mathbf{T}(\mathcal{A})$ for the worst-case expected runtime of $\mathcal{A}$, parametrized over λ. Also, we describe $(y_1, \dots) \leftarrow_{\$} \mathcal{A}(1^\lambda, x_1, \dots; r)$ as an event when $\mathcal{A}$ gets $(1^\lambda, x_1, \dots)$ as input, uses fresh random coins r and outputs $(y_1, \dots)$. We write $\mathcal{A}^B$ to denote that $\mathcal{A}$ has black-box access to algorithm B, meaning it sees only its input-output behaviour. On the other hand, $\mathcal{A}^{(\cdot)}$ means that $\mathcal{A}$ expects a black-box access to some other algorithm.

Denote $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}, \mathcal{Y} = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ as ensembles of random variables over some countable set S indexed by λ. Then, $\mathcal{X}$ and $\mathcal{Y}$ are statistically indistinguishable ($\mathcal{X} \overset{s}{\approx} \mathcal{Y}$) if $\Delta(X_\lambda, Y_\lambda) = \frac{1}{2}\sum_{s \in S} |\Pr[X = s] - \Pr[Y = s]| = \mathsf{negl}(\lambda)$. Moreover, $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable ($\mathcal{X} \overset{c}{\approx} \mathcal{Y}$) if for every PPT algorithm $\mathcal{A}$:

$$|\Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, X_\lambda)] - \Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, Y_\lambda)]| = \mathsf{negl}(\lambda).$$

3.1.2 Cryptographic Primitives

One-Way Functions. Intuitively, we say that a function is one-way (OWF) if it is easy to compute but hard to invert. Using our notation, we formalise it as follows:

Definition 3.1. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if:*

- *There exists a PPT algorithm $\mathcal{A}$ so that*

$$\Pr[f(x) \leftarrow_{\$} \mathcal{A}(1^\lambda, x) : x \leftarrow_{\$} \{0,1\}^\lambda] = 1,$$

- *For every PPT algorithm $\mathcal{A}$,*

$$\Pr[y \leftarrow_{\$} \mathcal{A}(1^\lambda, x) : x \leftarrow_{\$} \{0,1\}^\lambda, f(y) = f(x)] = \mathsf{negl}(\lambda).$$

It can be shown by simple reduction that if one-way functions exist then $\mathbf{P} \neq \mathbf{NP}$.

Most of the time, we do not deal with a single one-way function but rather with a collection of one-way functions. That is, we consider a set $\mathcal{F} = \{f\}$ of functions f that each have a finite domain, but may be parameterized over the security parameter (and other public parameters such as a fixed group). In that case, we choose f at random for the currently given security parameter (and potentially other parameters), and sample x uniformly at random from f's domain.

In the case of a family $\mathcal{F}$ of parameterized one-way functions f (with finite domain $\mathcal{D}_f$), we say that $\mathcal{F}$ is a family of one-way permutations if f is bijective and $f(\mathcal{D}_f) = \mathcal{D}_f$.

We also recall the notion of a hard-core predicate introduced by Goldreich et al. [45].

Definition 3.2. *Let $b : \{0,1\}^* \to \{0,1\}$ be a function and $u \leftarrow_\$ \{0,1\}$. Then, b is a hardcore-bit predicate for function $f : \{0,1\}^* \to \{0,1\}^*$ if*

$$(f(x), b(x)) \overset{c}{\approx} (f(x), u)$$

Goldreich et al. provide in [45] a construction of a one-way function with hard-core predicate from any given one-way function.

Lossy Trapdoor Functions. We say that a function f is a trapdoor function if it is easy to compute $f(x)$ and also easy to invert if we know some "special information" (called trapdoor) but hard to invert without trapdoor. The notion of a lossy trapdoor function was introduced by Peikert et al. [70]. A tuple of PPT algorithms $(S_{inj}, S_{loss}, F, F^{-1})$ is called a collection of $(n,k)-$lossy trapdoor functions (LTDF) if:

- S_{inj} outputs (s,t) where s is a function index and t its trapdoor, $F(s,\cdot)$ computes a deterministic injective function f over the domain $\{0,1\}^n$, and $F^{-1}(t,\cdot)$ computes f^{-1},
- S_{loss} outputs $(s,\perp)$ and $F(s,\cdot)$ computes a deterministic function f over the domain $\{0,1\}^n$ whose image has size at most 2^{n-k},
- For $(s_1,t_1) \leftarrow_\$ S_{inj}$ and $(s_2,\perp) \leftarrow_\$ S_{loss}$,

$$s_1 \overset{c}{\approx} s_2.$$

Peikert et al. also define all-but-one trapdoor functions in order to construct an IND-CCA encryption scheme. In this paper we concentrate more on LTDFs but our results can be easily generalised to the second notion.

Pseudorandom Generators. A function $G : \{0,1\}^\lambda \to \{0,1\}^{p(\lambda)}$, where $k < p(\lambda)$, is a pseudorandom generator (PRG) if given random x from $\{0,1\}^\lambda$, no PPT adversary can distinguish $G(x)$ and a random element from $\{0,1\}^{p(\lambda)}$.

Definition 3.3. *Let $G : \{0,1\}^\lambda \to \{0,1\}^{p(\lambda)}$ be a function where $\lambda < p(\lambda)$ and $p(\lambda) = \mathsf{poly}(\lambda)$, and let $x \in \{0,1\}^\lambda$ and $u \in \{0,1\}^p(\lambda)$ be uniformly* *chosen. Then, G is a pseudorandom generator if*

$$G(x) \overset{c}{\approx} u.$$

We call p the stretch factor.

Håstad et al. [48] show that one can construct a pseudorandom generator from any one-way function. Unfortunately, their security reduction has a large security loss. Since then, more efficient constructions of PRGs from OWFs with much tighter reductions have been discovered [53, 84] but they still suffer from a large reduction loss. On the other hand, Blum and Micali [18] provide a construction of a PRG from a one-way permutation which loses a factor of l.

A pseudorandom function is a function which is computationally indistinguishable from a random function. Formally, we write is as follows.

Definition 3.4. *Let $F : K \times D \to R$ be a function, where K, D, R are parameterized by λ. We say that F is a pseudorandom function (PRF) if F is efficiently computable and for any PPT algorithm $\mathcal{A}$ we have that:*

$$|\Pr_{k \leftarrow\!\!\$\, K}[1 \leftarrow\!\!\$\, \mathcal{A}^{F(k,\cdot)}] - \Pr_{f \leftarrow\!\!\$\, \{f : D \to R\}}[1 \leftarrow\!\!\$\, \mathcal{A}^{f(\cdot)}]|$$

is negligible in λ.

Pseudorandom functions can be built from pseudorandom generators using the well-known "GGM" construction by Goldreich, Goldwasser, and Micali [44].

Hashing. A family of functions $\mathcal{H} = \{h_i : A \to B\}$ is universal if for every distinct $x, x' \in A, \Pr_{h \leftarrow\!\!\$\, \mathcal{H}}[h(x) = h(x')] = 1/|B|$. Moreover, we say that $\mathcal{H}$ is pairwise independent if, for every distinct $x, x' \in A$ and every $y, y' \in B$, $\Pr_{h \leftarrow\!\!\$\, \mathcal{H}}[h(x) = y \wedge h(x') = y'] = 1/|B|^2$.

A hash function $H : \{0,1\}^k \to \{0,1\}^l$, where $k > l$, is collision resistant if for any PPT adversary $\mathcal{A}$, $\Pr[(x, y) \leftarrow\!\!\$\, \mathcal{A}(1^\lambda, h) : h(x) = h(y)] = \mathsf{negl}(\lambda)$. Similarly we can define a collection of collision-resistant hash functions.

Leftover Hash Lemma. The min-entropy of a random variable X over domain S is equal to

$$H_\infty(X) = -\log(\max_{s \in S} \Pr[X = s]).$$

We also define the notion of average min-entropy [35] for X condition on the value of Y as: $\tilde{H}_\infty(X|Y) = -\log(\mathbf{E}_{y \leftarrow Y}[2^{-H_\infty(X|Y=y)}])$. Now we recall the Leftover Hash Lemma.

Lemma 3.5. *Let n, k, l be natural numbers and X, Y be random variables such that $X \in \{0,1\}^n$ and $\tilde{H}_\infty(X|Y) \geq k$. Let $\mathcal{H}$ be a family of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^l$ and U be the uniform distribution on $\{0,1\}^l$. Then, for $h \leftarrow\!\!\$\, \mathcal{H}$, if $l \leq k - 2\log(1/\epsilon)$ then*

$$\Delta((Y, h, h(X)), (Y, h, U)) \leq \epsilon.$$

We also use the lower bound on the average min-entropy in terms of the min-entropy [35].

Lemma 3.6. *Let Y, Z be random variables such that Y takes at most 2^r possible values. Then*

$$\tilde{H}_\infty(X|(Y,Z)) \geq H_\infty(X|Z) - r.$$

3.1.3 Cryptographic Assumptions

We briefly state the most common computational and decisional problems in public-key cryptography. Let G be a cyclic group of order p where p is a λ-bit prime. Also, let $g \in G$ be a generator of G.

- Discrete Logarithm Problem (DLOG) - we say that the discrete logarithm problem is hard in G if for every PPT algorithm $\mathcal{A}$:

  $$\Pr[x \leftarrow_{\$} \mathcal{A}(g, g^x) : x \leftarrow_{\$} \mathbb{Z}_p] = \mathsf{negl}(\lambda).$$

  We also look at the n-One More Discrete Logarithm Problem (n-DL) [11] which is presented as follows. Let DL be an oracle such that given g^x, it outputs x. We denote $\mathcal{A}^{\mathsf{DL}_n}$ for an adversary which can call DL at most n times. We say that n-DL problem is hard in G if for every PPT algorithm $\mathcal{A}^{\mathsf{DL}_n}$:

  $$\Pr[(x_1, ..., x_n) \leftarrow_{\$} \mathcal{A}^{\mathsf{DL}_n}(g, g^{x_1}, ..., g^{x_n}) : x_1, ..., x_n \leftarrow_{\$} \mathbb{Z}_p] = \mathsf{negl}(\lambda).$$

- Computational Diffie-Hellman Problem (CDH) - we say that the computational Diffie-Hellman problem is hard in G if for every PPT algorithm $\mathcal{A}$:

  $$\Pr[g^{xy} \leftarrow_{\$} \mathcal{A}(g, g^x, g^y) : x, y \leftarrow_{\$} \mathbb{Z}_p] = \mathsf{negl}(\lambda).$$

- Decisional Diffie-Hellman Problem (DDH) - we say that the decisional Diffie-Hellman problem is hard in G if for $z \leftarrow_{\$} \mathbb{Z}_p$:

  $$(g, g^x, g^y, g^{xy}) \overset{c}{\approx} (g, g^x, g^y, g^z).$$

3.1.4 Public Key Schemes

Public-Key Encryption. A public-key encryption scheme for a given security parameter λ is a triple of PPT algorithms (Gen; Enc; Dec) such that:

- $\mathsf{Gen}(1^\lambda) \rightarrow (pk, sk)$ is the key generation algorithm which takes a security parameter λ and outputs a pair (pk, sk) where pk and sk are called public and secret keys respectively,
- $\mathsf{Enc}(pk, m) \rightarrow c$ is the encryption algorithm which takes a public key pk, a message m and returns a ciphertext c,

- $\mathsf{Dec}(sk, c) \to m$ is the decryption algorithm which takes a secret key sk, ciphertext c and returns a message m or $\perp$ if given ciphertext is invalid.

A public-key encryption must satisfy the correctness condition, meaning that for every message m and every security parameter λ, $\mathsf{Gen}(1^\lambda) \to (pk, sk)$ and $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = 1$.

We recall basic notions of security for public-key encryption schemes. We say that an encryption scheme $\mathcal{E} = (\mathsf{Gen}; \mathsf{Enc}; \mathsf{Dec})$ is IND-CPA secure (has indistinguishable ciphertexts under chosen plaintext attack) if there exists no PPT adversary $\mathcal{A}$ which wins the following game with $\frac{1}{2}+$ non-negligible probability:

1. Challenger C generates $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends pk to $\mathcal{A}$.
2. Adversary $\mathcal{A}$ sends messages (m_0, m_1) to C. Challenger then selects a bit b uniformly at random and returns $c = \mathsf{Enc}(pk, m_b)$.
3. At the end, $\mathcal{A}$ sends a bit b' to the challenger. Then, $\mathcal{A}$ wins if $b = b'$ and loses otherwise.

Similarly, we define IND-CCA security (indistinguishability under chosen ciphertext attack). We say that $\mathcal{E}$ is IND-CCA secure if there exists no PPT adversary $\mathcal{A}$ which wins a similar game to the one described above (with non-negligible probability), but this time $\mathcal{A}$ also has access to decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ which on input c' returns $\perp$ if $c = c'$ and $\mathsf{Dec}(sk, c')$ otherwise.

Signatures Schemes. A signature scheme consists of a tuple of PPT algorithms $(\mathsf{Gen}; \mathsf{Sign}; \mathsf{Ver})$ satisfying the following conditions:

- $\mathsf{Gen}(1^\lambda) \to (vk, sk)$ is the key generation algorithm, which takes a security parameter λ and outputs a pair (vk, sk) where vk and sk are called verification and signing keys respectively,
- $\mathsf{Sign}(sk, m) \to c$ is the signing algorithm which takes the signing key sk, a message m and returns a signature σ,
- $\mathsf{Ver}(vk, m, \sigma) \to b$ is the verification algorithm which takes the verification key vk, message m and signature σ and returns a bit b.

Any signature scheme must satisfy the correctness condition meaning that for every message m and every security parameter λ, $\mathsf{Gen}(1^\lambda) \to (vk, sk)$ and $\mathsf{Ver}(vk, m, \mathsf{Sign}(sk, m)) = 1$.

We now define a few security notions for signature schemes used in the thesis. First, we introduce existential unforgeability under a one-time chosen message attack (EUF-OTCMA). We say that the signature scheme $\mathcal{E} = (\mathsf{Gen}; \mathsf{Sign}; \mathsf{Ver})$ is EUF-OTCMA secure if there is no PPT adversary $\mathcal{A}$ which wins the following game with $\frac{1}{2}$ + non-negligible probability:

1. Challenger C generates $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends vk to $\mathcal{A}$.
2. Adversary $\mathcal{A}$ sends message m to C. Challenger then returns $\sigma = \mathsf{Sign}(sk, m)$.

3. Finally, $\mathcal{A}$ outputs a pair (m', σ') b'. Then, $\mathcal{A}$ wins if $\mathsf{Ver}(vk, m', \sigma') = 1$ and $m' \neq m$ and loses otherwise.

Unforgeable one-time signatures can be constructed from a one-way function as well as collision-resistant hash functions. We also say that a signature scheme is EUF-CMA secure (without *one-time* term) if there is no adversary which wins the following game with $\frac{1}{2}$ + non-negligible probability:

1. Challenger C generates $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends vk to $\mathcal{A}$.

2. Adversary $\mathcal{A}$ has an access to a signing oracle $\mathcal{O}$ such that, $\mathcal{A}$ can send a message m to $\mathcal{O}$ and gets back $\sigma = \mathsf{Sign}(sk, m)$.

3. Finally, $\mathcal{A}$ outputs a pair (m', σ') b'. Then, $\mathcal{A}$ wins if $\mathsf{Ver}(vk, m', \sigma') = 1$ and m' has not been queried to $\mathcal{O}$ during the whole game.

One observes that the EUF-OTCMA game is the particular case of this game with adversary calling the signing oracle only once.

We will also look at the UUF-KA (universal unforgeability against key only attack) security. We say that a signature scheme is UUF-KA secure is there is no PPT adversary which, given vk and message m, can produce a valid signature for m.

Commitment Schemes. A commitment scheme is a cryptographic primitive which provides us a way to commit to some certain statement, while keeping it hidden from others, which can be revealed later. Formally, it is a pair of algorithms Params, Commit, where Params generates parameters pm and $\mathsf{Commit}(m; r)$ takes a message m, randomness r and returns $c \leftarrow \mathsf{Commit}(m; r)$. In order to open the commitment c corresponding to the message m, release randomness r. The receiver recomputes the commitment and accepts if $c = \mathsf{Commit}(m; r)$.

There are two main notions of security for commitment schemes: binding and hiding. In this thesis we only concentrate on the latter. We say that a commitment $C = (\mathsf{Params}, \mathsf{Commit})$ is computationally binding if no polynomial-time adversary wins the following game with $\frac{1}{2}$ + non-negligible probability:

1. Challenger C generates $pm \leftarrow \mathsf{Param}(1^\lambda)$ and sends pm to $\mathcal{A}$.

2. Adversary $\mathcal{A}$ sends messages (m_0, m_1) to C. Challenger then selects a bit b uniformly at random and returns $c = \mathsf{Commit}(m; r)$ for some randomness r.

3. At the end, $\mathcal{A}$ sends a bit b' to the challenger. Then, $\mathcal{A}$ wins if $b = b'$ and loses otherwise.

A commitment scheme C is perfectly binding if no (unbounded) adversary can win this game. Formally, define $R(m, c) = \{r | c = \mathsf{Commit}(m, ; r)\}$. Then, C is perfectly hiding if for all m, m', c we have $|R(m, c)| = |R(m', c)|$.

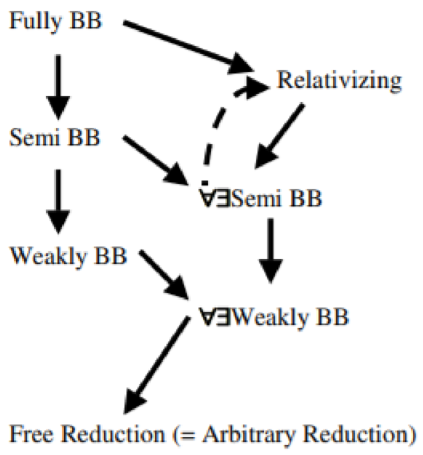

Figure 3.1: Different types of reductions defined by Reingold et al. [73].

3.1.5 Fully Black-Box Reductions

We review the framework of Reingold et al. [73] on security reductions. There are various formal definitions of reductions (such as [7, 40, 54, 73]) but in this thesis we focus on work by Reingold et al. [73] due to its simplicity. Using their notation, primitive P is a pair $\langle F_P, R_P \rangle$ where F_P is a set of functions $f : \{0,1\}^* \to \{0,1\}^*$ and R_P is a relation over pairs (f, M) for $f \in F_P$ and machine M. One can think of F_P as implementations of primitive P and R_P as security conditions on F_P. For example, if we think of P as a one-way function, the set of implementations could be a set of one-way functions. On the other hand, R_P would be the standard one-wayness game, i.e. give an image to adversary and check if it can return a correct preimage.

There is a fully black-box reduction from a primitive $P = \langle F_P, R_P \rangle$ to $Q = \langle F_Q, R_Q \rangle$ if there exist PPT machines G, S such that:

- for every implementation $f \in F_Q$, $G^f \in F_P$,
- for every implementation $f \in F_Q$ and every adversary $\mathcal{A}$, $(G^f, \mathcal{A}) \in R_P \implies (f, S^{\mathcal{A}}) \in R_Q$.

The word *black-box* comes from the fact that the machine S does not see the code of adversary. This is because of the order of quantification: S is chosen and then one can pick any $\mathcal{A}$. Hence, S can only see the input/output behaviour of adversary and hence, its view is black-box. There are many other types of reductions based on different orders of quantification. For instance, adversary $\mathcal{A}$ could be chosen before S and consequently, we can assume that S knows the code of $\mathcal{A}$. Different types of reductions defined in [73] are shown in Fig. 3.1.

As mentioned in [73], this definition of reduction does not apply to non-uniform or information-theoretic notions of security. We highlight that the definitions introduced in the literature were used mainly to show (non-)existence of reductions between certain schemes and they do not take the security loss into account.

3.2 Notion of Reductions

In this section we formalise the notion of reduction by adapting the framework of Reingold et al. (RTV) [73]. Roughly speaking, we represent security conditions as a security game instead of a set of relations. Thus, we could formally define what we mean by "breaking one primitive with about the same success as the other primitive" in terms of probabilities. At the end, we give a few examples of cryptographic primitives which satisfy our framework.

We start by stating what a primitive is and what it means for it to be secure.

Definition 3.7. *A primitive P is a tuple $\left\langle \mathbb{P}, S_P, F_P, R_P, \sigma \right\rangle$ where:*

- *$\mathbb{P}$ is a triple of sets (A, B, C) where $C \subseteq A$,*
- *F_P is a subset of $\{f : A \to B\}$,*
- *S_P is a PPT setup algorithm which sends parameters $(r_1, \ldots)$ to R_P,*
- *$R_P^{(\cdot,\cdot)}$ is a PPT security algorithm which gets parameters $(r_1, \ldots)$ from S_P.*
- *$\sigma : \mathbb{N} \to \mathbb{R}$ is a security threshold.*

We say that f is an implementation of P if $f \in F_P$.

There are three main differences between this definition and the one proposed by Reingold et al. Firstly, $\mathbb{P} = (A, B, C)$ is a triple of sets which describe the domain, co-domain and the challenge space respectively. Indeed, an implementation f is a function from A to B and the security game R_P can call f only on inputs in C. This modification enables us to characterise implementations which are defined on more abstract mathematical models (e.g. groups, rings) rather than on $\{0,1\}^*$. Secondly, R_P is now an algorithm which expects black-box access to both an implementation f and an adversary $\mathcal{A}$. One can think of R_P as a security game, e.g. one-wayness game or IND-CCA game. Here, we want to associate for each pair $(f, \mathcal{A})$ a value in $[0, 1]$ which corresponds to the probability of $\mathcal{A}$ winning the R_P game against f (see Definition 3.8). This adjustment helps us introduce the notion of a security loss. Eventually, we introduce a setup algorithm S_P which sends some values to R_P. This machine could as well send nothing or just provide fresh random coins which R_P would use in its game. However, this addition will be very useful in defining the multi-instance setting of a primitive. Informally, we can define a new security game R' which represents the security of P in the multi-user setting as follows: given parameters $(r_1, \ldots)$ from S_P, run n independent copies of R_P and send $(r_1, \ldots)$ as setup parameters to each of them. Then, R' returns a bit depending on what the n copies returned earlier. This idea is formally defined in Section 3.4

Definition 3.8. *Let $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ be a primitive and $\mathbb{P} = (A, B, C)$. Take $f \in F_P$ and any algorithm $\mathcal{A}$. We define the advantage of $\mathcal{A}$ in breaking f as*

$$\mathsf{Adv}^{\mathrm{P}}_{f,\mathcal{A}}(\lambda) := \Pr[R_P^{f|_C, \mathcal{A}} = 1] - \sigma(\lambda)$$

where the probability is defined over random coins in the system. We say that $\mathcal{A}$ $P-$breaks f if $\mathsf{Adv}^{\mathrm{P}}_{f,\mathcal{A}}(\lambda)$ is non-negligible. Primitive P is called secure if there exists an implementation f of P such that there are no PPT algorithms $\mathcal{A}$ that $P-$break f.

From the definition above one observes that we do not assign each pair $(f, \mathcal{A})$ a binary value (that would indicate, e.g., whether it satisfies a relation or not), but a probability. Therefore, the notion of a primitive from [73] is a generalisation of our definition. Indeed, any primitive in our sense can be easily transformed into a primitive from RTV definition: let $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ be a primitive in our sence and define a primitive $P' = \langle F'_P, R'_P \rangle$ such that $F'_P = F_P$ and $R'_P = \{(f, \mathcal{A}) | \mathcal{A}\ P\text{-breaks } f\}$. Then, primitives P and P' are equivalent. On the other hand, it is not clear if implication in the opposite direction holds. It is unknown if given relation set $\mathcal{R}$ we can construct a PPT algorithm R which could be equivalent to $\mathcal{R}$, meaning that $(f, \mathcal{A}) \in \mathcal{R} \iff \Pr[R^{f,\mathcal{A}} = 1]$ is not negligible. An obvious brute force solution would be to check all elements of R but this could take exponential time. Despite the fact that our definition of a primitive is less general, it allows us to spot relations between two distinct advantages and consequently, to formally define a security loss.

Using our previous definitions we formalise the notion of a fully black-box reduction.

Definition 3.9. *Let $P = \langle \mathbb{P}_1, S_P, F_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, S_Q, F_Q, R_Q, \tau \rangle$ be primitives. Then, there is a fully black-box reduction from P to Q (written as $P \hookrightarrow Q$) if there exist PPT algorithms $G^{(\cdot)}, S^{(\cdot)}$ such that:*

- *for every implementation f of Q, G^f is an implementation of P,*
- *for every implementation f of Q and every (unbounded) algorithm $\mathcal{A}$, if $\mathcal{A}$ P-breaks G^f then $S^{\mathcal{A}}$ Q-breaks f.*

We say that G is a generic construction of P from Q and S is an actual reduction.

Unsurprisingly, fully black-box reductions satisfy the transitivity property.

Lemma 3.10. *Relation $\hookrightarrow$ is transitive.*

Proof. Let P, Q, R be primitives such that $P \hookrightarrow Q$ and $Q \hookrightarrow R$ and let (G, S) and (G', S') be such reductions. Define:

$$\bar{G}^{(\cdot)} = G^{G'^{(\cdot)}} \text{ and } \bar{S}^{(\cdot)} = S'^{S^{(\cdot)}}$$

```
$G^h(1^\lambda)$
1 : if $G$ is queried on $x$:
2 :     send $x$ to $h$
3 :     get $y$ from $h$ and return $y$
```

```
$S^{\mathcal{A}}(1^\lambda)$
1 : $x \leftarrow_{\$} \{0,1\}^{2\lambda}$
2 : send $h(x)$ to $\mathcal{A}$
3 : get $x'$ from $\mathcal{A}$ and return $(x, x')$
```

Figure 3.2: PPT Algorithms G and S.

. Note that $\bar{G}$ and $\bar{S}$ run in polynomial time. Let f be an implementation of R. Then, G'^f is an implementation of S. This also implies that $G^{G'^f} = \bar{G}^f$ is an implementation of P. Thus, $\bar{G}$ is a generic construction of P from R. Now, suppose that $\mathcal{A}$ P-breaks $\bar{G}^f$ for some adversary $\mathcal{A}$. Then, $S^{\mathcal{A}}$ Q-breaks G'^f. Consequently, $\bar{S}^{\mathcal{A}}$ R-breaks f and the result holds. $\square$

We provide an extensive example of a fully black-box reduction for concrete cryptographic schemes.

Example 3.11. We show that a collision-resistant hash function is a one-way function. We first need to formalise these two primitives using our framework. Let $\mathsf{OWF} = \langle \mathbb{P}_{\mathsf{OWF}}, S_{\mathsf{OWF}}, F_{\mathsf{OWF}}, R_{\mathsf{OWF}}, 0 \rangle$ be a primitive of a one-way function. That is, $\mathbb{P}_{\mathsf{OWF}} = (A, B, C)$, F_{OWF} is a collection of functions $f : A \to B$, and R_{OWF} is a standard one-wayness game (i.e. R_{OWF} generates x uniformly at random from C, gets $f(x)$ by calling an implementation f and returns 1 only if adversary can guess the preimage of $f(x)$). For simplicity, when we write $\mathsf{OWF}_{p(\lambda)}$, where $p = \mathit{poly}(\lambda)$, we mean OWF with $\mathbb{P}_{\mathsf{OWF}} = (\{0,1\}^*, \{0,1\}^*, p(n))$ which is closer to the standard definition of a one-way function. On the other hand, define $\mathsf{CRHF} = \langle \mathbb{P}_{\mathsf{CRHF}}, S_{\mathsf{CRHF}}, F_{\mathsf{CRHF}}, R_{\mathsf{CRHF}}, 0 \rangle$ as follows: $\mathbb{P}_{\mathsf{CRHF}} = (\{0,1\}^*, \{0,1\}^\lambda, \{0,1\}^*), S_{\mathsf{CRHF}}$ returns no parameters, $F_{\mathsf{CRHF}} = \{h : \{0,1\}^* \to \{0,1\}^\lambda\}$ and R_{CRHF} is the collision resistance game, i.e. it waits until it gets (x, x') from adversary, checks if $x \neq x'$ and then calls the implementation h to check if $h(x) = h(x')$. We prove the following.

Lemma 3.12. *$\mathsf{OWF}_{2\lambda} \hookrightarrow \mathsf{CRHF}$.*

Proof. Let us define PPT algorithms $G^{\cdot}$ and $S^{\cdot}$ as in Fig. 3.2. Clearly, both G and S run in polynomial time. One can observe that G is a generic construction. Indeed, G only forwards all the queries from/to an implementation and hence, $\forall h \in F_{\mathsf{CRHF}}$, $G^h = h$. In particular, $G^h : \{0,1\}^* \to \{0,1\}^\lambda$ is a function, so $G^h \in F_{\mathsf{OWF}}$. Now, suppose that there exists an algorithm $\mathcal{A}$ which $\mathsf{OWF}_{2\lambda}$-breaks G^h. We want to prove that $S^{\mathcal{A}}$ CRHF-breaks h. Using the variables x and x' from Fig.3.2, one observes that:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{crhf}}_{h,S^{\mathcal{A}}}(\lambda) &= \Pr[R^{h,S^{\mathcal{A}}}_{\mathsf{CRHF}} = 1] \\
&= \Pr[x \neq x' \wedge h(x) = h(x')] \\
&\geq \Pr[x \neq x' \wedge h(x) = h(x') \mid |h^{-1}(h(x))| \geq 2] \cdot \Pr[|h^{-1}(h(x))| \geq 2],
\end{aligned} \tag{3.1}$$

where $h^{-1}(u) = \{v \in \{0,1\}^{2\lambda} : h(v) = u\}$. Clearly, $|h^{-1}(h(x))| \geq 1$. Note that $\Pr[x \neq x' \mid h(x) = h(x') \wedge |h^{-1}(h(x))| \geq 2] \geq \frac{1}{2}$ since adversary $\mathcal{A}$ does not know, given $h(x)$, if S chose exactly x or some other element in $h^{-1}(h(x))$ (it exists $h^{-1}(h(x)) \geq 2$). Hence, if we denote $X = |h^{-1}(h(x))|$, then we eventually have $\frac{X-1}{X} \geq 1/2$. Using this observation and the fact that $h(x)$ is generated by S with the same distribution as the challenge by R_{OWF} , we deduce that:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{crhf}}_{h,S^{\mathcal{A}}}(\lambda) &\geq \frac{1}{2}\Pr[h(x) = h(x') \mid |h^{-1}(h(x))| \geq 2] \cdot \Pr[|h^{-1}(h(x))| \geq 2] \\ &\geq \frac{1}{2}\Pr[h(x) = h(x') \mid |h^{-1}(h(x))| \geq 2] \cdot \Pr[|h^{-1}(h(x))| \geq 2] \\ &+ \frac{1}{2}\Pr[h(x) = h(x') \mid |h^{-1}(h(x))| = 1] \cdot \Pr[|h^{-1}(h(x))| = 1] \\ &- \frac{1}{2}\Pr[h(x) = h(x') \mid |h^{-1}(h(x))| = 1] \cdot \Pr[|h^{-1}(h(x))| = 1] \\ &\geq \frac{1}{2}\Pr[h(x) = h(x')] - \frac{1}{2}\Pr[|h^{-1}(h(x))| = 1] \\ &\geq \frac{1}{2}\mathsf{Adv}^{\mathsf{owf}}_{G^h,\mathcal{A}}(\lambda) - \frac{1}{2}\Pr[|h^{-1}(h(x))| = 1].\end{aligned} \tag{3.2}$$

The only thing to compute here is $\Pr[|h^{-1}(h(x))| = 1]$. Let $a_1, \ldots, a_m \in \{0,1\}^{2\lambda}$ be the bit-strings such that $|h^{-1}(h(a_i))| = 1$ for $i = 1, \ldots, m$. Clearly, $h(a_1), \ldots, h(a_m)$ are pairwise distinct. Also $\{h(a_1), \ldots, h(a_m)\} \subset \{0,1\}^{\lambda}$, and therefore $m \leq 2^{\lambda}$. Thus, $\Pr[|h^{-1}(h(x))| = 1] = \frac{m}{2^{2\lambda}} \leq \frac{2^{\lambda}}{2^{2\lambda}} = \frac{1}{2^{\lambda}}$. By substituting this result into Equation 3.2 and reordering both sides we get:

$$2\mathsf{Adv}^{\mathsf{crhf}}_{h,S^{\mathcal{A}}}(\lambda) + \frac{1}{2^{\lambda}} \geq \mathsf{Adv}^{\mathsf{owf}}_{G^h,\mathcal{A}}(\lambda),$$

which concludes that $\mathsf{OWF}_{2\lambda} \hookrightarrow \mathsf{CRHF}$. □

Computational indistinguishability is commonly used in provable security. We write $X_\lambda \overset{c}{\approx} Y_\lambda$ (see Section 3.1) for distributions X_λ, Y_λ and say that X_λ is computationally indistinguishable from Y_λ if there is no PPT algorithm $\mathcal{A}$ which can distinguish them. Note that $\overset{c}{\approx}$ is an equivalence relation, and in particular it is transitive. This key observation is a base for showing that a primitive is secure using the standard hybrid argument, game hopping techniques or the random self-reducibility property of the primitive (Section 3.4). The notion of computational indistinguishability can be derived from our definitions. Let Ω_1, Ω_2, S be finite sets and $X_\lambda : \Omega_1 \to S, Y_\lambda : \Omega_2 \to S$ be random variables (parametrized over λ). We define $f_{X,Y} : \Omega_1 \times \Omega_2 \to S \times S$ as $f_{X,Y}(u) = (X_\lambda(u), Y_\lambda(u))$. Then, the $[X_\lambda; Y_\lambda]$ primitive is a tuple $\langle \mathbb{P}, S_{X,Y}, F_{X,Y}, R_{X,Y}, \frac{1}{2} \rangle$, where $\mathbb{P} = (\Omega_1 \times \Omega_2, S \times S, \Omega_1 \times \Omega_2)$, $F_{X,Y} = \{f_{X,Y}\}$, $S_{X,Y}$ does not output anything, and $R_{X,Y}$ is the following game: generate $(u_0, u_1) \leftarrow_{\$} \Omega_1 \times \Omega_2$ and send (u_0, u_1) to an implementation $f \in F_{X,Y}$. Then, get back (v_1, v_2) from f and select $b \leftarrow_{\$} \{0,1\}$. Output v_b to an adversary $\mathcal{A}$ and eventually get back b' from $\mathcal{A}$. Then, the adversary $\mathcal{A}$ wins if $b = b'$. Using the definition of $[X_\lambda; Y_\lambda]$ we define the notion of computational indistinguishability as follows.

Definition 3.13. *Let X_λ and Y_λ be random variables over a finite set S (dependent on λ). Then, X_λ and Y_λ are computationally indistinguishable ($X_\lambda \overset{c}{\approx} Y_\lambda$) if $[X_\lambda; Y_\lambda]$ is secure.*

3.3 Selected Reduction Techniques

In this section we focus on certain techniques to derive a reduction in cryptography. We start by introducing the *hybrid argument.* Then, we look at the *self-reducibility* property which is somehow similar to self-improvability mentioned in Chapter 2. Eventually, we present some examples of reductions using tricks from information theory and combinatorics.

3.3.1 Hybrid Argument

The first technique we discuss here is called *hybrid argument* and shows the following: in order to prove that X and Y are computationally indistinguishable, one can define intermediate random variables $X_0, X_1, ..., X_n$ such that $X_0 = X$ and $X_n = Y$ and prove that $X_i \overset{c}{\approx} X_{i+1}$ for all $i = 0, ..., n-1$. This method heavily relies on the transitivity of computational indistinguishability. Let us prove this property now.

Lemma 3.14. *Relation $\overset{c}{\approx}$ is an equivalence relation.*

Proof. Let X_λ, Y_λ and Z_λ be random variables parametrized by λ. Clearly, no adversary can dinstinguish between X_λ and X_λ, so $\overset{c}{\approx}$ is reflexive. Also, it is obviously symmetric. Hence, we only need to show it is transitive. Suppose that $X_\lambda \overset{c}{\approx} Y_\lambda$ and $Y_\lambda \overset{c}{\approx} Z_\lambda$. Let $\mathcal{A}$ be any PPT algorithm and denote $\delta^{\mathcal{A}}(A, B) = |\Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, A)] - \Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, B)]|$. Then, by the triangle inequality:

$$\delta^{\mathcal{A}}(X_\lambda, Z_\lambda) \leq \delta^{\mathcal{A}}(X_\lambda, Y_\lambda) + \delta^{\mathcal{A}}(Y_\lambda, Z_\lambda).$$

But $\delta^{\mathcal{A}}(X_\lambda, Y_\lambda)$ and $\delta^{\mathcal{A}}(Y_\lambda, Z_\lambda)$ are negligible functions. Therefore, $\delta^{\mathcal{A}}(X_\lambda, Z_\lambda)$ is also negligible. □

In the similar fashion one can show that hybrid argument is indeed correct as long as the sequence of random variables X_0,...,X_n is not "too long", i.e. $n = \mathsf{poly}(\lambda)$. This because if n is sufficiently large, e.g. $n = 2^\lambda$ then using triangle inequality, we only get $\delta^{\mathcal{A}}(X, Y) \leq 2^\lambda \mathsf{negl}(\lambda)$ and this does not imply that $\delta^{\mathcal{A}}(X, Y)$ is negligible.

We now present some examples of how hybrid argument can be smartly used in deriving reductions.

Example 3.15. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a length-preserving one-way permutation, i.e. one-way function which is bijective and $\forall x \in \{0,1\}^*, |x| = |f(x)|$. Formally, we can define it similarly as in Example 3.11: OWP =

$\langle \mathbb{P}_{\mathsf{OWF}}, S_{\mathsf{OWF}}, F_{\mathsf{OWP}}, R_{\mathsf{OWF}}, 0\rangle$, where F_{OWP} now consists of length-preserving bijective functions. Let b be the hardcore bit predicate for f. Then, we claim that $G(x) = f(x)||b(x)$ is a pseudorandom generator.

We have to show $G(x) \overset{c}{\approx} u$ where $x \in \{0,1\}^\lambda$ and $u \in \{0,1\}^{\lambda+1}$. Firstly, note that $G(x) \overset{c}{\approx} f(x)||b'$ for $b \leftarrow_{\$} \{0,1\}$. This is straight from the definition of a hardcore bit predicate: given $f(x)$, no efficient adversary can distinguish $b(x)$ from uniformly random bit. On the other hand, $f(x)$ is computationally indistinguishable from $y \in \{0,1\}^\lambda$ since f is a length-preserving permutation. Hence:

$$G(x) \overset{c}{\approx} f(x)||b' \overset{c}{\approx} y||b' \overset{c}{\approx} u.$$

By the hybrid argument, we get that G is a pseudorandom generator with stretch factor $\lambda+1$. We now show we can modify this pseudorandom generator so that it stretches with arbitrary factor $n(\lambda) = \mathsf{poly}(\lambda)$.

Define $f^n(x) \coloneqq f^{n-1}(f(x))$ and $f^1(x) \coloneqq f(x)$. Similarly, we denote $b^n(x)$. Let G' be a function defined by:

$$G'(x) = b(x)||b(f(x))||...||b(f^{n-1}(x))||f^n(x).$$

We want to show that $G'(x)$ looks uniformly random for $x \leftarrow_{\$} \{0,1\}^\lambda$. Define random variables $G_0, G_1, ..., G_{n+1}$ as follows.

G_0: Select $x \in \{0,1\}^\lambda$. Then, return $G'(x)$.

G_i for $1 \leq i \leq n$: Sample $u_0, ..., u_{i-1} \leftarrow_{\$} \{0,1\}$ and $v \leftarrow_{\$} \{0,1\}^\lambda$. Then, return

$$u_0||u_1||...||u_{i-1}||b(v)||...||b(f^{n-i}(v))||f^{n-i+1}(v).$$

G_{n+1}: Select $u \in \{0,1\}^{\lambda+n}$ and return u.

We want to show that $G_0 \overset{c}{\approx} G_{n+1}$. We use the hybrid argument. First, we prove $G_0 \overset{c}{\approx} G_1$. Suppose there exists an efficient adversary $\mathcal{A}$ which can distinguish between G_0 and G_1. Consider the following adversary $\mathcal{B}$ which distinguishes $G(x)$ from random. It is first given some $x'||b'$. Then, it constructs $y' = b'||b(x')||...||b(f^{n-2}(x'))||f^{n-1}(x')$ and sends y' to $\mathcal{A}$. When $\mathcal{A}$ returns an answer $\bar{b}$, $\mathcal{B}$ also returns $\bar{b}$. Note that if $x'||b' = G(x)$ for some x, then $y' = G'(x)$, so $\mathcal{A}$ is given an instance of G_0. On the other hand, if $x'||b'$ is uniformly random, then $\mathcal{A}$ is given an instance of G_1. Hence,

$$\Pr[1 \leftarrow_{\$} \mathcal{B}(1^\lambda, G(x))|x \leftarrow_{\$} \{0,1\}^\lambda] = \Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, G'(x))|x \leftarrow_{\$} \{0,1\}^\lambda], \text{ and}$$

$$\Pr[1 \leftarrow_{\$} \mathcal{B}(1^\lambda, u)|u \leftarrow_{\$} \{0,1\}^\lambda] = \Pr[1 \leftarrow_{\$} \mathcal{A}(1^\lambda, X)|X \leftarrow_{\$} G_1].$$

If we substitute these two equations, we get that if $\mathcal{A}$ distinguishes between G_0 and G_1 then $\mathcal{B}$ distinguishes between $G(x)$ and uniformly random, which leads to contradiction.

Now, we prove $G_i \overset{c}{\approx} G_{i+1}$ for $i = 1, ..., n-1$. Suppose there is an efficient adversary $\mathcal{A}$ which distinguishes between G_i and G_{i+1}. We construct an adversary which distinguishes G from uniformly random bit-string. Algorithm $\mathcal{B}$, given $x'||b'$, first samples bits $u_0, ..., u_{i-1} \leftarrow_{\$} \{0,1\}$. Then it computes $y' = u_0||u_1||...||u_{i-1}||b'||b(x')||...||b(f^{n-i-1}(x'))||f^{n-i}(x')$ and sends y' to

$\mathcal{A}$. The key observation is if $x'||b' = G(x)$ for uniformly random x then y' is an instance of G_i. On the other hand, if $x'||b'$ is uniformly random (and consequently, both x' and b' are), then y' is an instance of G_{i+1}. Hence, in the similar fashion as in the previous proof, $\mathcal{B}$ can distinguish between G and uniformly random bit-string, which is a contradiction. Hence, $G_i \overset{c}{\approx} G_{i+1}$.

The only thing left to show is $G_n \overset{c}{\approx} G_{n+1}$. However, this clearly holds because f is a permutation. Therefore, G' is indeed a pseudorandom permutation with stretch factor $\lambda + n$. Note that we can choose n as big as we want. However, it still needs to be a polynomial in λ for the hybrid argument to work.

When defining intermediate random variables $X_0, X_1, ..., X_n$, usually X_i and X_{i+1} look very similar and differ in only one place. Most of the time, X_{i+1} is constructed just like X_i but maybe one line of code is added or modified. This technique is called in literature *game-hopping*. It is because X_i's can be represented as games and we "jump" from X_i to X_{i+1} in the standard hybrid argument.

3.3.2 Self-Reducibility

Suppose there exists a solver $\mathcal{A}$ which solves some problem P with non-negligible probability, provided that the input is given is uniformly random (average-case). How can we use $\mathcal{A}$ to construct another solver $\mathcal{B}$ which solves P for any input with the same probability as $\mathcal{A}$? This reduction is called *random self-reducibility*. We present two examples of such reductions.

Example 3.16. Let G be a group of prime order p with generator g. Suppose there exists an algorithm $\mathcal{A}$ which solves the discrete logarithm problem. Now, construct an adversary $\mathcal{B}$ as follows. Given *any* $X = g^x$, it selects $a \leftarrow_\$ \mathbb{Z}_p$ and computes $y = Xg^a = g^{x+a}$. Next, it sends y to $\mathcal{A}$ and gets back x'. Then, $\mathcal{B}$ returns $x' - a \mod p$. Note that $x + a$ has uniformly random distribution since a was chosen uniformly at random. Therefore, the probability that $\mathcal{B}$ finds the exponent x is equal to the probability that $\mathcal{A}$ breaks the discrete logarithm problem.

Example 3.17. Let again G be a group of prime order p with generator g. Suppose now there exists an algorithm $\mathcal{A}$ which solves the Decisional Diffie-Hellman problem. We construct an adversary $\mathcal{B}$ as follows. Given any DDH tuple $(g, A = g^a, B = g^b, C)$, sample $x, y \leftarrow_\$ \mathbb{Z}_p$. Then, compute $X = Ag^x = g^{a+x}$ and $Y = Bg^y = g^{b+y}$. Next, send the triple

$$(X, Y, A^x B^y C g^{xy})$$

to $\mathcal{A}$. If $\mathcal{A}$ says that $A^x B^y C g^{xy} = g^{(a+x)(b+y)}$ then return the answer "$C = g^a b$". Otherwise, return "$C = g^c$ for uniformly random $c \leftarrow_\$ \mathbb{Z}_p$".

First, note that X and Y look uniformly random. Next, if $C = g^{ab}$ then $A^x B^y C g^{xy} = g^{ax+by+ab+xy} = g^{(a+x)(b+y)}$, so $(X, Y, A^x B^y C g^{xy})$ is a valid DDH triple. On the other hand, if $C = g^c$ for some random c, then $A^x B^y C g^{xy} =$

$g^{c+(ax+by+xy)}$ which looks uniformly random. Therefore, the probability that $\mathcal{B}$ returns the correct answer is equal to the probability that $\mathcal{A}$ breaks DDH.

The main observation is that given some g^x, we can sample $a \leftarrow_\$ \mathbb{Z}_p$ and get g^{ax} (or g^{a+x}) which looks uniformly random (for $x \neq 0$). We say that, by doing that, we *rerandomise* the input.

Example 3.18. Let G be a group of λ-bit prime order p with generator g. We define U_n be the random variable sampled from uniform distribution on G^n. Also, define the following random variable X_n: select $x, y_1, ..., y_n \leftarrow_\$ \mathbb{Z}_p$ and return $(g^{xy_1}, ..., g^{xy_n})$. We claim that if DDH holds in G and $n = \mathsf{poly}(\lambda)$ then $(g^x, g^{y_1}, ..., g^{y_n}, X_n) \overset{c}{\approx} (g^x, g^{y_1}, ..., g^{y_n}, U_n)$ (or for simplicity $X_n \overset{c}{\approx} U_n$).

Suppose that there exists an efficient adversary $\mathcal{A}$ which distinguishes between X_n and U_n. We construct an adversary $S^{\mathcal{A}}$ which can win the DDH game. Firstly, S is given some $(X = g^x, Y = g^y, Z)$ as input. If $Z = 1$ and the tuple, which S was given, is a DDH triple, then $x = 0$ or $y = 0$. Hence, S can check if $g^x = 1$ or $g^y = 1$ and consequently, S can win the DDH game with big probability. Therefore, suppose that $Z \neq 1$. Then, Z is a generator of G. Also, assume that $x \neq 0$ or $y \neq 0$ since otherwise it also gets much easier (WLOG $y \neq 0$). Let S choose $a_1, ..., a_n \leftarrow_\$ \mathbb{Z}_p$. Then, S sends $(X, Y^{a_1}, Y^{a_2}, ..., Y^{a_n}, (Z^{a_1}, Z^{a_2}, ..., Z^{a_n}))$ to $\mathcal{A}$ and returns what $\mathcal{A}$ returns. Note that if (X, Y, Z) is a valid DDH tuple then S sends $(g^{x(ya_1)}, ..., g^{x(ya_n)})$. Since ya_i looks uniformly random for all i, this tuple looks like it was generated from X_n. On the other hand, if $Z \leftarrow_\$ G$ and Z is a generator of G, $(Z^{a_1}, Z^{a_2}, ..., Z^{a_n})$ looks uniformly random. Similarly as before, we deduce that if $\mathcal{A}$ can win the DDH game then $S^{\mathcal{A}}$ distinguishes between X_n and U_n.

Now, let us extend this example. Define random variables $\mathbf{X}_n$ and $\mathbf{U}_n$ as follows:

$$\mathbf{X}_n = \begin{pmatrix} g^{x_1y_1} & g^{x_1y_2} & \cdots & g^{x_1y_n} \\ \vdots & \vdots & \vdots & \vdots \\ g^{x_ny_1} & g^{x_ny_2} & \cdots & g^{x_ny_n} \end{pmatrix} \text{ and } \mathbf{U}_n = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ u_{n,1} & u_{n,2} & \cdots & u_{n,n} \end{pmatrix} \tag{3.3}$$

where $x_1, ..., x_n, y_1, ..., y_n \leftarrow_\$ \mathbb{Z}_p$ and $u_{i,j} \leftarrow_\$ G$ for all $1 \leq i, j \leq n$. Suppose that DDH holds. We prove that

$$(g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{X}_n) \overset{c}{\approx} (g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{U}_n).$$

Define a random variable $\mathbf{H}_k$, for $k = 0, ..., n$, as follows:

$$\mathbf{H}_i = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ u_{k-1,1} & u_{k-1,2} & \cdots & u_{k-1,n} \\ g^{x_ky_1} & g^{x_ky_2} & \cdots & g^{x_ky_n} \\ \vdots & \vdots & \vdots & \vdots \\ g^{x_ny_1} & g^{x_ny_2} & \cdots & g^{x_ny_n} \end{pmatrix} \tag{3.4}$$

where $u_{i,j} \leftarrow_{\$} G$ for $1 \leq i,j < k$ and $x_i, y_j \leftarrow_{\$} \mathbb{Z}_p$ for $1 \leq i \leq n$ and $1 \leq j \leq n$. Note that $\mathbf{H}_0 = \mathbf{X}_n$ and $\mathbf{H}_n = \mathbf{U}_n$. We use the hybrid argument and show that for any $k = 0, ..., n-1$, we have $(g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{H}_k) \overset{c}{\approx} (g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{H}_{k+1})$. Suppose there exists an efficient distinguisher $\mathcal{A}$ for $(\mathbf{H}_k, \mathbf{H}_{k+1})$. We construct a distinguisher $S^{\mathcal{A}}$ for (X_n, U_n). It is given some vector $(X = g^x, Y_1 = g^{y_1}, ..., Y_n = g^{y_n}, (v_1, ..., v_n))$. Then, it generates $u_{i,j} \leftarrow_{\$} G$ for $1 \leq i,j < k$ and $x_i \leftarrow_{\$} \mathbb{Z}_p$ for $1 \leq i \leq n$ and computes the matrix:

$$\mathbf{H} = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ u_{k-1,1} & u_{k-1,2} & \cdots & u_{k-1,n} \\ v_1 & v_2 & \cdots & v_n \\ {Y_1}^{x_{k+1}} & {Y_2}^{x_{k+1}} & \cdots & {Y_n}^{x_{k+1}} \\ \vdots & \vdots & \vdots & \vdots \\ {Y_1}^{x_n} & {Y_2}^{x_n} & \cdots & {Y_n}^{x_n} \end{pmatrix} \tag{3.5}$$

Then it sends $(g^{x_1}, ..., g^{x_{k-1}}, X, g^{x_{k+1}}, ..., g^{x_n}, Y_1, ..., Y_n, \mathbf{H})$ to $\mathcal{A}$ and outputs what $\mathcal{A}$ returns.

One observes that if S was given an instance of X_n then $\mathbf{H}$ is an instance of $\mathbf{X}_n$. On the other hand, if S was given uniformly random $(v_1, ..., v_n)$ then $\mathbf{H} = \mathbf{H}_{k+1}$. So, by the previous example we get that $(g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{H}_k) \overset{c}{\approx} (g^{x_1}, ..., g^{x_n}, g^{y_1}, ..., g^{y_n}, \mathbf{H}_{k+1})$. Using the hybrid argument, the result holds.

This fact will be heavily used in Section 3.4 where we use rerandomisation techniques in order to decrease the security loss of a reduction.

3.3.3 Information-Theoretic Argument

In most cases, we want prove that a certain primitive is secure against efficient adversaries. Sometimes, it might be actually easier to show security against *all* adversaries, even unbounded ones. The proofs rely more on probability and information theory. Let us illustrate it with simple example. Define a function $f(x) = 0$ for all x. Clearly, f is not one-way: if we choose a random x then an adversary can easily find a preimage of $f(x) = 0$. Indeed, if an adversary returns any y then $f(y) = 0 = f(x)$. However, no algorithm can find exact value of x with non-negligible probability. This is because an adversary $\mathcal{A}$ is only given $f(x) = 0$ and therefore $\Pr[x \leftarrow_{\$} \mathcal{A}(f(x))] = \Pr[x \leftarrow_{\$} \mathcal{A}(f(y))]$ for any y. Hence, any $\mathcal{A}$ finds x with probability $\frac{1}{|C|}$ where C is the domain of f. This example is not really useful in practice but gives a flavour of how some information-theoretic arguments look like. We provide a more involved (and more interesting) example below.

Example 3.19. Let us consider the Pedersen's commitment scheme [69]. Let $C = (\mathsf{Param}, \mathsf{Commit})$, where Param returns a description of a group G of prime order q and generators g and h, while Commit is defined by $\mathsf{Commit}(m; r) = g^m h^r$. We show that it is perfectly hiding. In order to do so, we claim that for any m, c we have $|R(m,c)| = 1$. This directly implies that for any m, m' and c, $|R(m,c)| = |R(m',c)|$. We want to find out how many r satisfy $c = g^m h^r$. Note that this is equal to

$$h^r = \frac{c}{g^m}.$$

Since h is a generator of G and $\frac{c}{g^m} \in G$, there exists exactly one $r \in \mathbb{Z}_q$ which satisfies that. Hence, $|R(m,c)| = 1$ and C is perfectly hiding.

To sum up, information-theoretic argument involves a proof that no unbounded adversary can solve some problem. It usually consists of showing that an adversary is not able to see what we do "under the hood" and consequently, prove that the advantage for any adversary is negligible.

3.4 Tight Reductions

Informally, we say that a reduction is tight if the security loss is *small* (preferably constant) and does not depend on the number of instances, queries made by adversary or the number of users. We define this notion using our framework as follows.

Definition 3.20 (C-tightness)**.** *Let $C\colon \mathbb{N} \to \mathbb{R}$ be a function. We say that a fully black-box reduction (G,S) from P to Q is C-tight (and write $P \overset{C}{\hookrightarrow} Q$) if:*

- *for every algorithm $\mathcal{A}$, $\boldsymbol{T}(S^{\mathcal{A}}) = \boldsymbol{T}(\mathcal{A}) + n(\lambda)$ for some $n \in \mathit{poly}(\lambda)$,*
- *for every implementation f of Q and every algorithm $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathrm{p}}_{G^f,\mathcal{A}}(\lambda) \le C(\lambda) \cdot \mathsf{Adv}^{\mathrm{q}}_{f,S^{\mathcal{A}}}(\lambda) + \mathit{negl}(\lambda). \tag{3.6}$$

*In particular, we say that the reduction is **fully-tight** if $C = 1$, **tight** if $C = a$ for $a \in \mathbb{N}$ and **almost-tight** if $C \in \mathsf{poly}(\lambda)$.*

The first condition for a tight reduction states that the runtime of $S^{\mathcal{A}}$ should be about the same as the runtime of the adversary $\mathcal{A}$. This prevents the reduction S from running many copies of $\mathcal{A}$. An alternative way to formalise this condition would be to use the definition of a *time-success ratio* from [48] and combine it with the security loss C. However, in this paper we do not calculate exactly the runtime of $S^{\mathcal{A}}$ [1] and thus, we omit such formalities. Further, the second condition from the definition of tight reduction assures that the success of an adversary $\mathcal{A}$ breaking the primitive is always about as large as the success of the reduction $S^{\mathcal{A}}$ breaking the other one.

We can get some simple but useful properties of tight reductions from the definition above. For example, they (again) satisfy the transitivity property.

[1]As long as it is similar to the runtime of $\mathcal{A}$.

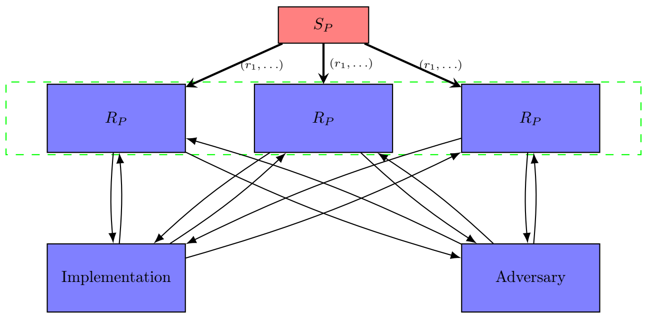

Figure 3.3: New security game for the three-instance version of primitive $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ is described as the green dashed box.

Lemma 3.21. *Let P, Q, R be primitives such that $P \overset{C}{\hookrightarrow} Q$ and $Q \overset{D}{\hookrightarrow} R$, where $C, D \in \mathsf{poly}(\lambda)$. Then, $P \overset{E}{\hookrightarrow} R$, where $E(\lambda) = C(\lambda) \cdot D(\lambda)$.*

Proof. Let (G, S) be a tight reduction from P to Q and (G', S') be a tight reduction from Q to R. Define:

$$\bar{G}^{(\cdot)} = G^{G'^{(\cdot)}}, \bar{S}^{(\cdot)} = S'^{S^{(\cdot)}}.$$

We claim that $(\bar{G}, \bar{S})$ gives a reduction from P to R.

Take $f \in F_R$. Then, G'^f is an implementation of Q. Therefore, $\bar{G}^f = G^{G'^f}$ is an implementation of P. Now, take any $f \in F_R$ and algorithm $\mathcal{A}$. Note that there are some negligible functions $\mathsf{negl}_1(\lambda), \mathsf{negl}_2(\lambda)$ and $\mathsf{negl}(\lambda)$ such that:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{p}}_{\bar{G}^f,\mathcal{A}}(\lambda) &\le C(\lambda) \cdot \mathsf{Adv}^{\mathrm{q}}_{G'^f,S^{\mathcal{A}}}(\lambda)| + \mathsf{negl}_1(\lambda) \\
&\le C(\lambda) \cdot (D(\lambda) \cdot \mathsf{Adv}^{\mathrm{r}}_{f,\bar{S}^{\mathcal{A}}}(\lambda) + \mathsf{negl}_1(\lambda)) + \mathsf{negl}_2(\lambda) \\
&= C(\lambda)D(\lambda) \cdot \mathsf{Adv}^{\mathrm{r}}_{f,\bar{S}^{\mathcal{A}}}(\lambda) + \mathsf{negl}(\lambda) \\
&= E(\lambda) \cdot \mathsf{Adv}^{\mathrm{r}}_{f,\bar{S}^{\mathcal{A}}}(\lambda) + \mathsf{negl}(\lambda)
\end{aligned} \tag{3.7}$$

by the definition of the almost-tight reduction. Therefore, we have shown that $(\bar{G}, \bar{S})$ is a $E-$tight reduction from P to R.

□

3.4.1 Multi-instance Setting

Using notions from Subsection 3.1 we define the multi-instance setting of a primitive. Informally, this means that an adversary now interacts with (polynomially) many independent copies of the security game, so it gets more information. However, the winning condition would be almost the same as in the single-instance case, e.g. returning a preimage or guessing a bit (see Fig. 3.3).

```
$\mathcal{R}_1^{f|_C,\mathcal{A}}(1^\lambda, r_1, \ldots)$

Initialise $n(\lambda)$ ind. copies of $R_P^{f|_C,\mathcal{A}}$
send $(r_1, \ldots)$ to each of them
        as the setup parameters
if one of the copies returns 1
    return 1
else return 0
```

```
$\mathcal{R}_2^{f|_C,\mathcal{A}}(1^\lambda, r_1, \ldots)$

Initialise $n(\lambda)$ ind. copies of $R_P^{f|_C,\mathcal{A}}$
send $(r_1, \ldots)$ to each of them
        as the setup parameters
if all of the copies return 1
    return 1
else return 0
```

Figure 3.4: Security algorithms for $\exists\mathsf{MI}_n(P)$ (on the left) and $\forall\mathsf{MI}_n(P)$ (on the right). They get as input the security parameter λ (in unary) and parameters $(r_1, \ldots)$ from the setup algorithm S_Q. Here, we assume that $\mathcal{R}$ has black-box access to implementation f (restricted to the domain C, where $\mathbb{P} = (A, B, C)$) and adversary $\mathcal{A}$.

We define it formally for a primitive $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ in terms of the security algorithm R_P. We, however, provide two definitions of the multi-instance setting due to the differences between computational and decisional problems.

Definition 3.22 ($\exists$-Multi-instance Setting)**.** *Let $n(\lambda)$ be a polynomial in λ and $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ be a primitive. Then, the $\exists\mathsf{MI}_n(P)$ primitive is a primitive $\langle \mathbb{P}, \mathcal{S}, \mathcal{F}, \mathcal{R}_1, \sigma \rangle$ such that $\mathcal{F} = F_P$, $\mathcal{S} = S_P$ and $\mathcal{R}_1$ is defined in Fig. 3.4 (left).*

In the similar manner we define the $\forall$-multi-instance setting.

Definition 3.23 ($\forall$-Multi-instance Setting)**.** *Let $n(\lambda)$ be a polynomial in λ and $P = \langle \mathbb{P}, S_P, F_P, R_P, \sigma \rangle$ be a primitive. Then, the $\forall\mathsf{MI}_n(P)$ primitive is a primitive $\langle \mathbb{P}, \mathcal{S}, \mathcal{F}, \mathcal{R}_2, \sigma \rangle$ such that $\mathcal{F} = F_P$, $\mathcal{S} = S_P$ and $\mathcal{R}_2$ is defined in Fig. 3.4 (right).*

Now we provide examples of cryptographic primitives in the multi-instance setting using definitions above.

Example 3.24. Let $\mathsf{OWF} = \langle \mathbb{P}_{\mathsf{OWF}}, S_{\mathsf{OWF}}, F_{\mathsf{OWF}} \rangle$ be a primitive of a one-way function defined in Example 3.11. The security game for $\exists\mathsf{MI}_n(\mathsf{OWF})$ would be as follows: it generates $x_1, \ldots, x_n$ independently, uniformly at random from C, then it gets $f(x_1), \ldots, f(x_n)$ by calling f and sends to adversary. The winning condition is that adversary returns a preimage of *one* of the values it was given i.e. x satisfying $f(x) = f(x_i)$ for some i. On the other hand, in $\forall\mathsf{MI}_n(\mathsf{OWF})$, the adversary wins if it returns preimages for *all* values $f(x_1), \ldots, f(x_n)$. In practice, the $\exists\mathsf{MI}_n(\mathsf{OWF})$ setting is more common and therefore we focus on the former case.

Example 3.25. Let us define a primitive $\mathsf{PRG} = \left\langle \mathbb{P}_{\mathsf{PRG}}, S_{\mathsf{PRG}}, F_{\mathsf{PRG}}, R_{\mathsf{PRG}}, \frac{1}{2} \right\rangle$ of a pseudorandom generator where $\mathbb{P}_{\mathsf{PRG}} = (A, B, C)$ such that $C = A$ and $|B| > |A|$. Let F_{PRG} be a collection of functions $G : A \to B$ and let S_{PRG} generate a bit $b \leftarrow_{\$} \{0, 1\}$. We also define R_{PRG}, given bit b, to generate random $x \in C$ and output the image $G(x)$ of an implementation G if $b = 0$ and uniformly random value from B otherwise. The adversary wins if it can guess the bit b. Then, the security game for $\forall \mathsf{MI}_n(\mathsf{PRG})$ would be as follows: given bit b from the setup algorithm, generate and send $G(x_1), \ldots, G(x_n)$ for some $x_1, \ldots, x_n \in A$ if $b = 0$ or n uniformly random elements from B if $b = 1$. The winning condition here is that adversary guesses the bits for *all* of the n subgames which is equivalent to guessing the bit b.

Now, consider $\exists \mathsf{MI}_n(\mathsf{PRG})$. We claim that it is not secure. Note that in this case adversary wins if it guesses the bit b for *one* of the n subgames. In other words, it has n chances to guess b. Thus, an adversary which just sends 1 to a random subgame and 0 to another one will always win one of these two games (because $b \in \{0, 1\}$) and consequently, break $\exists \mathsf{MI}_n(\mathsf{PRG})$. Therefore, we only analyse the security of $\forall \mathsf{MI}_n(\mathsf{PRG})$.

Example 3.26. We define a primitive corresponding to an IND-CPA secure public-key encryption scheme as $\mathsf{PKE} = \left\langle \mathbb{P}_{\mathsf{PKE}}, S_{\mathsf{PKE}}, F_{\mathsf{PKE}}, R_{\mathsf{PKE}}, \frac{1}{2} \right\rangle$ where, as before, S_{PKE} does the sampling $b \leftarrow_{\$} \{0, 1\}$, R_{PKE} is the IND-CPA game and F_{PKE} contains encryption schemes. Note that $\exists \mathsf{MI}_n(\mathsf{PKE})$ is not secure due to the same reasons as the $\exists$-multi instance pseudorandom generator. On the other hand, $\forall \mathsf{MI}_n(\mathsf{PKE})$ yields the definition of an encryption scheme in the multi-user setting by Bellare et al. [8]. In a similar fashion we can define IND-CCA secure PKE schemes.

One observes that one can slightly change the definition of a primitive in order to get a definition of "multi-ciphertext setting". If we give S_{PKE} black-box access to an implementation and let it also generate keys (pk, sk) instead of R_{PKE} then the security game for $\forall \mathsf{MI}_n(\mathsf{PKE})$ is indeed an IND-CPA game with many ciphertexts. However, we do not consider the multi-ciphertext security in this thesis so we omit defining it formally here.

We also introduce the notion of a primitive being *tightly extensible*, meaning that a reduction from its multi-instance setting admits the same security loss as in the single-instance case.

Definition 3.27. *Let P be a primitive and $C : \mathbb{N} \to \mathbb{R}$ be a function. Then, P is $(C, \forall)$-tightly extensible (resp. $(C, \exists)$-tightly extensible) with respect to primitive Q if:*

- $P \overset{C}{\hookrightarrow} Q$,
- $\forall n \in \mathit{poly}(\lambda)$, $\forall \mathit{MI}_n(P) \overset{C}{\hookrightarrow} Q$ *(resp.* $\exists \mathit{MI}_n(P) \overset{C}{\hookrightarrow} Q$*).*

Based on what we have already defined, we can formally state the main problem of this paper:

Problem. Suppose that $P \overset{C}{\hookrightarrow} Q$. Show that P is $(C, \exists(\text{ or } \forall))$-tightly extensible w.r.t. Q.

There are two standard approaches to show that P is tightly extensible w.r.t. Q. Namely, (i) somehow tightly reduce the multi-instance primitive to the single case, (ii) modify the former reduction and apply re-randomisation/self-reducibility techniques to eventually obtain the same security loss, or (iii) hide the factor of n in the statistical difference. In Section 4 we discuss these methods used in practical examples. When we apply (i), we use the following simple lemma.

Lemma 3.28. *Let n be a polynomial in λ and let P and Q be primitives such that $P = \langle \mathbb{P}_1, S_P, F_P, R_P, \sigma \rangle \overset{C}{\hookrightarrow} Q = \langle \mathbb{P}_2, S_Q, F_Q, R_Q, \tau \rangle$ and let (G, S) be such a reduction. Define P/Q to be the primitive $\langle \mathbb{P}_1, S_P, \mathcal{F}, R_P, \sigma \rangle$ such that $\mathcal{F} = \{G^f : f \in F_Q\}$. Then, if $\exists \mathsf{MI}_n(P) \overset{D}{\hookrightarrow} P/Q$ (resp. $\forall \mathsf{MI}_n(P) \overset{D}{\hookrightarrow} P/Q$) then $\exists \mathsf{MI}_n(P) \xhookrightarrow{C \cdot D} Q$ (resp. $\forall \mathsf{MI}_n(P) \xhookrightarrow{C \cdot D} Q$). In particular, if $D = 1$ then P is $(C, \exists)$ (resp. $(C, \forall)$)-tightly extensible w.r.t. Q.*

Proof. Using the notation above, it is easy to see that $P \overset{C}{\hookrightarrow} Q$ implies $P/Q \overset{C}{\hookrightarrow} Q$. The result holds by this simple observation and by Lemma 3.21. □

3.4.2 Tightly Extensible Primitives

We provide a few constructions of tightly extensible primitives from more general primitives. In principle, we first take a tight reduction from the single-instance primitive and see if we can extend it (in a tight way) to the multi-instance setting or otherwise, use the Lemma 3.28. In the first example, we demonstrate the use of definitions from Section 3.4, and derive formal proofs. In the later examples, however, we focus more on showing novel techniques to extend reductions to the multi-instance setting.

Example 3.29. It is well-known that signatures schemes can be constructed from one-way functions [75, 58, 47]. Concretely, we can use the Lamport construction [61] of a one-time signature scheme from one-way functions, and then extend these one-time signatures to full signatures using Merkle trees [65]. The corresponding reduction is far from tight, and not known to scale well to the multi-user setting. Here, we consider the same construction under a slightly stronger assumption (namely, collision-resistance) of the used one-way function. We will show that this stronger assumption enables a much tighter security reduction.

We continue with the Example 3.11. Recall the definition of CRHF: $\mathsf{CRHF} = \langle \mathbb{P}_{\mathsf{CRHF}}, S_{\mathsf{CRHF}}, F_{\mathsf{CRHF}}, R_{\mathsf{CRHF}}, 0 \rangle$ where $\mathbb{P}_{\mathsf{CRHF}} = (\{0,1\}^*, \{0,1\}^\lambda, \{0,1\}^*)$, S_{CRHF} returns no parameters, $F_{\mathsf{CRHF}} = \{h : \{0,1\}^* \to \{0,1\}^\lambda\}$ and R_{CRHF} is the collision resistance game, i.e. it waits until it gets (x, x') from adversary, checks if $x \neq x'$ and then calls the implementation h to check if $h(x) = h(x')$.

```
$\bar{G}^h(1^\lambda)$
1 : if $G$ is queried on $x$:
2 :     send $x$ to $h$
3 :     get $y$ from $h$
4 : return $y$
```

```
$\bar{S}^{\mathcal{A}}(1^\lambda)$
1 : $x_1, \ldots, x_n \leftarrow_\$ \{0,1\}^{2\lambda}$
2 : send $h(x_1), \ldots, h(x_n)$ to $\mathcal{A}$
3 : get $x'$ from $\mathcal{A}$, find $i$ s.t. $f(x') = f(x_i)$
4 : return $(x_i, x')$
```

Figure 3.5: PPT Algorithms G and S.

Damgård [29] showed that h, when considered as a function with domain $\{0,1\}^{\lambda+1}$, is also a one-way function. Indeed, if there exists an adversary $\mathcal{A}$ which can find preimage of $h(x)$ for uniformly random x, then we can construct adversary $S^{\mathcal{A}}$ which breaks the collision-resistance of h as follows: given h, choose random x and send $h(x)$ to $\mathcal{A}$. Let x' be the output of $\mathcal{A}$. Then, return the pair (x, x'). Note that with non-negligible probability (over x and x'), we have $x \neq x'$. Hence, adversary $S^{\mathcal{A}}$ wins the collision-resistance game and additionally, the reduction itself is clearly tight (see Example 3.11).

Damgård's argument in fact nicely extends to the multi-instance setting:

Theorem 3.30. *Let CRHF be the primitive of a collision-resistant hash function and $\mathsf{OWF}_{2\lambda}$ be the primitive of a one-way function defined in Example 3.11. Then, $\mathsf{OWF}_{2\lambda}$ is $(2, \exists)$-tightly extensible w.r.t. CRHF.*

Proof. We proved in Example 3.11 that $\mathsf{OWF}_{2\lambda} \overset{2}{\hookrightarrow} \mathsf{CRHF}$. Now, we have to show that $\mathsf{MI}_n(\mathsf{OWF}_{2\lambda}) \overset{2}{\hookrightarrow} \mathsf{CRHF}$ for any $n \in \mathsf{poly}(\lambda)$. Denote $n = n(\lambda)$. We define the reduction $(\bar{G}, \bar{S})$ as in the Fig. 3.5. One observes that $\bar{G}$ and $\bar{S}$ are both PPT algorithms, and also $\bar{G} = G$ is a generic construction of $\mathsf{MI}_n(\mathsf{OWF}_{2\lambda})$ from CRHF. Suppose that there exists an algorithm $\mathcal{A}$ which $\mathsf{OWF}_{2\lambda}$-breaks G^h for some $h \in F_{\mathsf{CRHF}}$. We extend the previous argument for many instances as follows (the probabilities are calculated over $x_1, \ldots, x_n, x'$ which are defined in Fig. 3.5):

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{crhf}}_{h,\bar{S}^{\mathcal{A}}}(\lambda) &= \Pr[R^{h,\bar{S}^{\mathcal{A}}}_{\mathsf{CRHF}} = 1] \\
&= \Pr[\exists i \text{ such that } x_i \neq x' \wedge h(x_i) = h(x')] \\
&\geq \Pr[\exists i \text{ such that } x_i \neq x' \wedge h(x_i) = h(x') \mid E] \cdot \Pr[E] \\
&\geq \Pr[(\exists i \text{ such that } h(x_i) = h(x')) \wedge F \mid E] \cdot \Pr[E],
\end{aligned} \tag{3.8}$$

where $E = \bigwedge_{i=1}^{n} |h^{-1}(h(x_i))| \geq 2$ and F represents the following event: the smallest integer $j \in \{1, 2, \ldots, n\}$, such that $h(x_j) = h(x')$ (if exists), satisfies $x_j \neq x'$. In this similar fashion as before, we have that $\Pr[F \mid \exists i \text{ such that } h(x_i) =$

$h(x') \wedge E] \geq \frac{1}{2}$. Hence,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{crhf}}_{h,\bar{S}^{\mathcal{A}}}(\lambda) &\geq \frac{1}{2}\Pr[(\exists i \text{ such that } h(x_i) = h(x')) \mid E] \cdot \Pr[E] \\ &\geq \frac{1}{2}\Pr[(\exists i \text{ such that } h(x_i) = h(x')) \mid E] \cdot \Pr[E] \\ &+ \frac{1}{2}\Pr[(\exists i \text{ such that } h(x_i) = h(x')) \mid \neg E] \cdot \Pr[\neg E] \\ &- \frac{1}{2}\Pr[(\exists i \text{ such that } h(x_i) = h(x')) \mid \neg E] \cdot \Pr[\neg E] \\ &\geq \frac{1}{2}\mathsf{Adv}^{\mathsf{mi}_{\mathrm{n}}(\mathsf{owf})}_{\bar{G}^h,\mathcal{A}}(\lambda) - \frac{1}{2}\Pr[\neg E].\end{aligned} \tag{3.9}$$

By a union bound, we compute:

$$\begin{aligned}\Pr[\neg E] &= \Pr[\bigvee_{i=1}^{n} |h^{-1}(h(x_i))| = 1] \\ &\leq \sum_{i=1}^{n}\Pr[|h^{-1}(h(x_i))| = 1] \\ &= n \cdot \Pr[|h^{-1}(h(x_1))| = 1] \\ &\leq \frac{n}{2^\lambda}.\end{aligned} \tag{3.10}$$

Eventually, by reordering both sides of Equation 3.9 we get that $\mathsf{MI}_n(\mathsf{OWF}_{2\lambda}) \overset{2}{\hookrightarrow}$ CRHF:

$$2\mathsf{Adv}^{\mathsf{crhf}}_{h,\bar{S}^{\mathcal{A}}}(\lambda) + \frac{n}{2^\lambda} \geq \mathsf{Adv}^{\mathsf{mi}_{\mathrm{n}}(\mathsf{owf})}_{\bar{G}^h,\mathcal{A}}(\lambda).$$

□

Even though this proof is not complicated, the theorem itself can be useful in constructing secure one-time signature schemes (OTS) with small security loss. Let us first define OTS using the definition of a primitive from Section 3. That is, $\mathsf{OTS} = \langle \mathbb{P}_{\mathsf{OTS}}, S_{\mathsf{OTS}}, F_{\mathsf{OTS}}, R_{\mathsf{OTS}}, 0 \rangle$, where $\mathbb{P} = (\{0,1\}^*, \{0,1\}^*, \{0,1\}^*)$, S_{OTS} does not send any global parameters, F_{OTS} is the set of all signature schemes and R_{OTS} represents the EUF-OTCMA game.

Let f be a one-way function and let n be a polynomial in λ. Consider the Lamport's one-time signature scheme [61] $(\mathsf{Gen}_L; \mathsf{Sign}_L; \mathsf{Ver}_L)$ for messages of length $n = n(\lambda)$, meaning:

- $\mathsf{Gen}_L(\lambda)$: generate $2n$ random values $x_{i,j}$ and compute $y_{i,j} = f(x_{i,j})$ for $j \in \{0,1\}, i \in \{1,\ldots,,n\}$. Then, set $vk = \{x_{1,0},\ldots,x_{n,0},x_{1,1},\ldots,x_{n,1}\}$ and $sk = \{y_{1,0},\ldots,y_{n,0},y_{1,1},\ldots,y_{n,1}\}$.
- $\mathsf{Sign}_L(m, sk)$: for message $m = m_1m_2\ldots m_n$, output $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ where $\sigma_i = x_{i,m_i}$.
- $\mathsf{Ver}_L(vk, m, \sigma)$: for signature $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ and message $m = m_1m_2\ldots m_n$, check if $f(\sigma_i) = y_{i,m_i}$ for all $i = 1,\ldots,n$. If so, return 1 and 0 otherwise.

```
$G^f(1^\lambda)$
1 : if $G$ is queried on $\mathsf{Gen}(1^\lambda)$:
2 :     run $\mathsf{Gen}_L(\lambda)$ by calling $f$
3 : if $G$ is queried on $\mathsf{Sign}(m, sk)$:
4 :     run $\mathsf{Sign}_L(m, sk)$
5 : if $G$ is queried on $\mathsf{Ver}(vk, m, \sigma)$:
6 :     run $\mathsf{Ver}_L(vk, m, \sigma)$
```

```
$S^{\mathcal{A}}(1^\lambda)$
1 : receive $y$
2 : $x_{1,0}, \ldots, x_{n,1} \leftarrow_\$ \{0,1\}^{2\lambda}$
3 : $\forall j, b, y_{j,b} = f(x_{j,b})$
4 : $l \leftarrow_\$ \{1, \ldots, n\}, b' \leftarrow_\$ \{0,1\}$
5 : $y_{l,b'} = y$
6 : $sk = (y_{1,0}, \ldots, y_{n,1})$
7 : send $sk$ to $\mathcal{A}$
8 : if $\mathcal{A}$ requests a signature on $m'$:
9 :     if $m'_l = b'$, then abort
10 :    else
11 :        return $\sigma = (x_{1,m'_1}, \ldots, x_{n,m'_n})$
12 : if $\mathcal{A}$ outputs $(m, \sigma = \sigma_1, \ldots, \sigma_n)$:
13 :    if $\mathcal{A}$ outputs a forgery at $(l, b')$ :
14 :        return $\sigma_l$
```

Figure 3.6: PPT Algorithms G and S. Here, $(\mathsf{Gen}_L; \mathsf{Sign}_L; \mathsf{Ver}_L)$ is the Lamport signature scheme.

Lamport proved that this scheme is EUF-OTCMA secure by giving a reduction to the one-wayness of f which admits the security loss of $2n$. We quickly sketch this reduction.

Proposition 3.31 (Lamport)**.** *Let $n = \mathit{poly}(\lambda)$. Then, $\mathsf{OTS} \overset{2n}{\hookrightarrow} \mathsf{OWF}_{2\lambda}$.*

Proof. Define PPT algorithms G and S as in Fig. 3.6. One observes that G represents the Lamport construction of a one-time signature scheme. In particular, G is a signature scheme, and in particular, G is a generic construction of OTS from $\mathsf{OWF}_{2\lambda}$. Now, consider $S^{\mathcal{A}}$. Note that since adversary $\mathcal{A}$ can output a valid forgery (which is different from what it gets by quering the signing oracle once), $\mathcal{A}$ can find a preimage of $y_{j,b}$ for some j, b. With probability $\frac{1}{2n}$, $j = l$ and $b = b'$. Therefore, $S^{\mathcal{A}}$ finds a preimage of $y = y_{l,b'}$ with probability $\frac{1}{2n}$ of the probability that $\mathcal{A}$ outputs a valid forgery. □

We show how to tightly reduce it to the one-wayness of f in the multi-instance setting.

Theorem 3.32. *Let $n \in \mathit{poly}(\lambda)$ and OTS and $\mathsf{OWF}_{2\lambda}$ be the primitives of a one-time signature scheme and one-way function respectively. Then, there exists a fully-tight fully black-box reduction from OTS to $\mathsf{MI}_{2n}(\mathsf{OWF}_{2\lambda})$.*

Proof. As usual, let us define PPT algorithms G and S as in Fig. 3.7. Again, we can see that G is a generic construction of OTS from $\mathsf{MI}_{2n}(\mathsf{OWF}_{2\lambda})$. Now, let us consider S and suppose that S is given $f(x_1), \ldots, f(x_{2n})$ from $R_{\mathsf{MI}_{2n}(\mathsf{OWF}_{2\lambda})}$.

```
$G^f(1^\lambda)$
1 : if $G$ is queried on $\mathsf{Gen}(1^\lambda)$:
2 :     run $\mathsf{Gen}_L(\lambda)$ by calling $f$
3 : if $G$ is queried on $\mathsf{Sign}(m, sk)$:
4 :     run $\mathsf{Sign}_L(m, sk)$
5 : if $G$ is queried on $\mathsf{Ver}(vk, m, \sigma)$:
6 :     run $\mathsf{Ver}_L(vk, m, \sigma)$
```

```
$S^{\mathcal{A}}(1^\lambda)$
1 : receive $y_1, \dots, y_{2n}$
2 : for $i = 1, \dots, 2n$
3 :     $x_{1,0}, \dots, x_{n,1} \leftarrow\$ \{0,1\}^{2\lambda}$
4 :     $\forall j, b, y_{j,b} = f(x_{j,b})$
5 :     $l_i \leftarrow\$ \{1, \dots, n\}, b_i \leftarrow\$ \{0,1\}$
6 :     $y_{l_i,b_i} = y_i$
7 :     $sk = (y_{1,0}, \dots, y_{n,1})$
8 :     send $sk$ to $\mathcal{A}$
9 :     if $\mathcal{A}$ requests a signature on $m'$:
10 :        if $m'_{l_i} = b_i$, then abort
11 :        else
12 :            return $\sigma = (x_{1,m'_1}, \dots, x_{n,m'_n})$
13 :    if $\mathcal{A}$ outputs $(m, \sigma = \sigma_1, \dots, \sigma_n)$:
14 :        if $\mathcal{A}$ outputs a forgery at $(l_i, b_i)$ :
15 :            return $\sigma_{l_i}$
```

Figure 3.7: Similarly as in Fig. 3.6, $(\mathsf{Gen}_L; \mathsf{Sign}_L; \mathsf{Ver}_L)$ is the Lamport signature scheme.

Note that S repeats the standard Lamport reduction $2n$ times and runs (at most) $2n$ copies of $\mathcal{A}$ where the i-th copy of $\mathcal{A}$ is associated with $f(x_i)$. Hence, the probability that S wins the $R_{\mathsf{MI}_{2n}(\mathsf{OWF}_{2\lambda})}$ game is equal to the probability that one of the $2n$ copies of $\mathcal{A}$ outputs a valid forgery. One observes that each copy runs independently of each other since $x_1, \dots, x_n$ were sampled independently and uniformly at random. Therefore, using the fact that the standard Lamport reduction had a security loss of $2n$ we get that:

$$\mathsf{Adv}^{\mathsf{mi_n(owf)}}_{f,S^{\mathcal{A}}}(\lambda) \geq 2n \cdot \frac{1}{2n} \mathsf{Adv}^{\mathsf{ots}}_{G^f,\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{ots}}_{G^f,\mathcal{A}}(\lambda).$$

Thus, $\mathsf{OTS} \overset{1}{\hookrightarrow} \mathsf{MI}_{2n}(\mathsf{OWF}_{2\lambda})$.

□

Combining the previous two new results we get that it is possible to construct a one-time signature scheme, which can be reduced to a collision-resistant hash function with the loss of 2. Thus, we managed to eliminate the factor n and if one wants to apply Merkle trees, the overall reduction to CRHF from a secure signature scheme would have the security loss of $O(l)$ where l is the number of signing queries.

Example 3.33. Blum and Micali [18] provided a construction of a pseudorandom generator from a one-way permutation (see Example 3.15). Let f be a one-way permutation and b be its hard-core predicate. Then, the function $G(x) = f(x)||b(x)$ is a pseudo-random generator. Now, our aim is to construct, given f and b, a tightly extensible pseudo-random generator w.r.t. some certain mathematical assumption. We find suitable f and b such that we can apply Lemma 3.28. Note that in order to do this, we need a rerandomisation property from these functions. For instance, given $b(x)$ and a value a, we should somehow be able to compute $b(ax)$. We will choose a one-way permutation on a group where the discrete logarithm problem is hard and use properties of Legendre symbol to construct a rerandomisable hard-core predicate.

Background. Let $a \in \mathbb{Z}$ be an integer and p be a prime number such that pa. We say that a is a quadratic residue mod p if there exists $x \in \mathbb{Z} \setminus \{0\}$ so that $x^2 \equiv a(\mod p)$. If there is no such x, then a is a quadratic non-residue mod p. It is a well-known fact that in $\{1, \dots, p-1\}$ there are exactly $(p-1)/2$ quadratic residues (and also $(p-1)/2$ quadratic non-residues) mod p. Clearly, the quadratic residues form a subgroup of $\mathbb{Z}_p^*$.

The Legendre symbol is defined as follows:

$$(\frac{a}{p}) = \begin{cases} 0 & \text{if } p|a \\ 1 & \text{if } a \text{ is a quadratic residue mod } p \\ -1 & \text{if } a \text{ is a quadratic non-residue mod } p \end{cases}$$

One of useful properties of the Legendre symbol is that it is homomorphic, meaning $(\frac{a}{p})(\frac{b}{p}) = (\frac{ab}{p})$ for any a, b. Moreover, by the Euler's criterion, the Legendre symbol can be computed efficiently.

The LGR Problem. We propose a new computational problem, called Legendre Problem (LGR).

Definition 3.34. *Let p be a λ-bit prime number and let G be a group of order p with generator g. We say that the Legendre Problem (LGR) is hard in G if for all PPT algorithms $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{lgr}}_{G,\mathcal{A}}(\lambda) := \Pr[b \leftarrow_{\$} \mathcal{A}(g, g^x) : x \leftarrow_{\$} \mathbb{Z}_p, b = (\frac{x}{p})] = \frac{1}{2} + \mathsf{negl}(\lambda).$$

Equivalently, define a primitive $\mathsf{LGR}_G = \langle \mathbb{P}_{\mathsf{LGR}}, S_{\mathsf{LGR}}, F_{\mathsf{LGR}}, R_{\mathsf{LGR}}, \frac{1}{2} \rangle$ such that $\mathbb{P}_{\mathsf{LGR}} = (\mathbb{Z}_p, G, \mathbb{Z}_p)$, S_{LGR} sends parameters (G, p, g) to R_P and $F_{\mathsf{LGR}} = \{f_g\}$, where f_g is defined by $a \mapsto g^a$. The security game R_{LGR} first chooses random $x \leftarrow_{\$} \mathbb{Z}_p$, sends x to an implementation $f \in F_{\mathsf{LGR}}$ of LGR_G and gets back y. Then, it outputs y to adversary $\mathcal{A}$. When it receives $b \in \{-1, 1\}$ from $\mathcal{A}$, it returns a value of the statement $(\frac{x}{p}) = b$.

One observes that the Legendre Problem is not harder than the discrete logarithm problem. Indeed, given g^x, solving the discrete logarithm problem yields

x, from which one can directly compute $(\frac{x}{p})$. A more interesting question is whether LGR is as hard as DLOG. We first show that LGR is at least as hard as DDH.

Lemma 3.35. *Let G be a group of prime order p and suppose that DDH is hard in G. Then, the Legendre Problem is also hard in G.*

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ which solves LGR with a non-negligible advantage. We construct a PPT algorithm $S^{\mathcal{A}}$ which wins the DDH game as follows. S first generates random $a, b, c \leftarrow_{\$} \mathbb{Z}_p$. Next, given g^x, g^x, g^z, where $z = xy$ or z is random, S sends g^{ax}, g^{by}, g^{cz} to $\mathcal{A}$ and gets $(\frac{ax}{p}), (\frac{by}{p}), (\frac{cz}{p})$ respectively. Then, S simply extracts $(\frac{x}{p}), (\frac{y}{p}), (\frac{z}{p})$ (e.g. by $(\frac{ax}{p})(\frac{a}{p}) = (\frac{x}{p})$) and returns the value of statement $(\frac{x}{p})(\frac{y}{p}) = (\frac{z}{p})$. Note that if $z = xy$ then this will always be true. On the other hand, if z is random, then the probability that $(\frac{z}{p}) = b$ for $b \in \{-1, 1\}$ is 1/2. All in all, S wins the DDH game with non-negligible probability. The reduction itself, however, is far from tight. □

Legendre's symbol has already been used in building pseudorandom generators [30, 64, 83]. For example, Damgård [30] applied specific subsequences of the sequence of Legendre symbols modulo a prime to obtain a pseudorandom generator. Security of such constructions, however, rely on empirical results or additional unproven conjectures.

Our Construction. We build a tightly extensible pseudorandom generator with respect to the Legendre assumption. Let G be a *densely presentable* group with generator g of prime order p, i.e. a group which satisfies the property that for $x \in \mathbb{Z}_p$, the map $x \mapsto g^x$ is a permutation (e.g. [21]). We define PRG_G to be the primitive from Example 3.25 with $\mathbb{P} = (\mathbb{Z}_p \times \{0,1\}, G, \mathbb{Z}_p \times \{0,1\})$. Then, the construction is presented as follows.

Theorem 3.36. *Let G be a densely presentable group of prime order p where the Legendre problem is hard. Denote $g \in G$ to be a generator of G. Then, PRG_G is $(2, \forall)$-tightly extensible w.r.t. LGR_G.*

Proof. Define $f : \mathbb{Z}_p \to G$ as $x \mapsto g^x$, $b : \mathbb{Z}_p \to \{0,1\}$ as $x \mapsto (1 + (\frac{x}{p}))/2$ and eventually, $F(x) \coloneqq f(x)||b(x)$. Clearly, if the Legendre Problem is hard in G then f is a one-way permutation and b is a hard-core predicate for f.

Blum and Micali [18] showed that F is a pseudorandom generator. In this case, the generic reduction is fully-tight. Thus, by Lemma 3.28 it is enough to reduce the multi-instance setting of F to the single-instance with security loss 2. Suppose that there exists an adversary $\mathcal{A}$ which can win the n-multi-instance game for the pseudorandom generator F. We construct an adversary $S^{\mathcal{A}}$ that wins a single-instance game as follows. Given $(u||v)$ $(u \in G, v \in \{0,1\})$, toss a fair coin n times. For the i-th trial, where $i = 1, \ldots, n$, if we get heads - generate random $x_i \leftarrow_{\$} \mathbb{Z}_p$ and set $y_i = F(x_i)$. On the other hand, if the coin comes out tail, we choose random $a_i \leftarrow_{\$} \mathbb{Z}_p$ and set $y_i = u^{a_i}||v_i$ where

$v_i = (1 + (\frac{a_i}{p})(2v-1))/2$. Then, send $y_1, \ldots, y_n$ to adversary $\mathcal{A}$ and eventually output what $\mathcal{A}$ returns.

Let us assume that $x \neq 0$ [2]. In order to analyse correctness of this reduction, we have to consider two cases. First, suppose that $u = g^x$ for some x. Then, $v = (1 + (\frac{x}{p}))/2$ or $v = (1 - (\frac{x}{p}))/2$.

Case 1: $v = (1+(\frac{x}{p}))/2$. Let us fix i and consider the i-th trial of flipping the coin. If it comes out heads, then $y_i = F(x_i)$ for randomly chosen x_i. Otherwise, we get that $v_i = (1 + (\frac{a_i}{p})(\frac{x}{p}))/2 = (1 + (\frac{a_i x}{p}))/2$ and thus $y_i = F(x_i)$ where $x_i = a_i x$ for uniformly random a_i. Therefore, for each i we have $y_i = F(x_i)$ and also $x_1, \ldots, x_n$ are independently, uniformly random in $\mathbb{Z}_p$.

Case 2: $v = (1 - (\frac{x}{p}))/2$. Let us denote $y_i = g^{s_i}||t_i$ where $s_i \in \mathbb{Z}_p, t \in \{0,1\}$. We need to show that for every $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$ and $\beta_1, \ldots, \beta_n \in \{0,1\}$ we have

$$\Pr[s_1 = \alpha_1, \ldots, s_n = \alpha_n, t_1 = \beta_1, \ldots, t_n = \beta_n] = \frac{1}{2^n p^n}.$$

This is the same as showing $\Pr[t_1 = \beta_1, \ldots, t_n = \beta_n | s_1 = \alpha_1, \ldots, s_n = \alpha_n] \cdot \Pr[s_1 = \alpha_1, \ldots, s_n = \alpha_n] = \frac{1}{2^n p^n}$. Note that $\Pr[s_1 = \alpha_1, \ldots, s_n = \alpha_n] = 1/p^n$ because $x \neq 0$ and $s_i = a_i x$ or $s_i = x_i$ for random a_i, x_i. Now, consider $\Pr[t_1 = \beta_1, \ldots, t_n = \beta_n | s_1 = \alpha_1, \ldots, s_n = \alpha_n]$. Clearly, this is the same as $\Pr[t_1 = \beta_1 | s_1 = \alpha_1, \ldots, s_n = \alpha_n]^n$. Firstly, assume that $\beta_1 = (1 + (\frac{\alpha_1}{p}))/2$ and let X denote the output of tossing a coin for the first time (and say H - heads, T - tails). Then, $\Pr[t_1 = \beta_1 | X = H, s_1 = \alpha_1, \ldots, s_n = \alpha_n] = 1$ because if $S^{\mathcal{A}}$ gets heads then it generates fresh $F(x_1)$ and in this case $x_1 = s_1 = \alpha_1$ so $t_1 = (1 + (\alpha_1/p))/2 = \beta_1$. On the other hand, if the coin comes out tails then we have $\alpha_i = s_i = a_i x$. Also, $t_1 = v_1 = (1 - (\frac{a_i}{p})(\frac{x}{p})/2 = (1 - (\frac{\alpha_1}{p}))/2 \neq \beta_1$ and hence $\Pr[t_1 = \beta_1 | X = T, s_1 = \alpha_1, \ldots, s_n = \alpha_n] = 0$. Consequently, we get $\Pr[t_1 = (1 + (\frac{\alpha_1}{p}))/2 | s_1 = \alpha_1, \ldots, s_n = \alpha_n] = (1+0)/2 = 1/2$. Using a similar argument it can be shown that $\Pr[t_1 = (1 - (\frac{\alpha_1}{p}))/2 | s_1 = \alpha_1, \ldots, s_n = \alpha_n] = 1/2$. Thus, $\Pr[t_1 = \beta_1, \ldots, t_n = \beta_n | s_1 = \alpha_1, \ldots, s_n = \alpha_n] = 1/2^n$ and the result holds. In particular, if $v = (1 - (\frac{x}{p}))/2$ then all the values y_i sent to $\mathcal{A}$ look independently and uniformly random.

We provide a tight fully black-box reduction from $\mathsf{MI}_n(\mathsf{PRG}_G)$ to $\mathsf{PRG}_G/\mathsf{LGR}_G$ which admits the security loss of 2. Hence, by Lemma 3.28 the result holds.

□

A rerandomisable hard-core predicate can be potentially also very useful in constructing a tightly extensible encryption scheme out of a computational problem (rather than a decisional one). However, it is not known how it can concretely be used, for example, because we do not have any information about functions related to DLOG being trapdoor one-way functions.

Example 3.37. Our aim is to construct IND-CPA secure encryption schemes in the multi-user setting from lossy trapdoor functions in a tight way. In order to do so, we introduce tightly secure LTDFs in the multi-instance setting. Let $\mathsf{LTDF} =$

[2]This occurs with an overwhelming probability.

$\langle \mathbb{P}_{\mathsf{LTDF}}, S_{\mathsf{LTDF}}, F_{\mathsf{LTDF}}, R_{\mathsf{LTDF}}, \frac{1}{2} \rangle$ be a primitive of lossy trapdoor functions, i.e. $\mathbb{P}_{\mathsf{LTDF}}$ defines the domain, codomain and challenge space, F_{LTDF} is a collection of LTDFs, S_{LTDF} provides a random bit to R_{LTDF} and R_{LTDF} runs a game where the goal is to distinguish a lossy function from an injective one. As before, let PKE be a public-key encryption scheme with its security game being IND-CPA. For exact same reasons as in PRGs or PKE cases, it is sensible only to consider $\forall\mathsf{MI}_n(\mathsf{LTDF})$ for the multi-instance setting. Also, denote DDH as a primitive representing the DDH assumption (formal definition is not required here). Peikert et al. [70] showed that:

$$\mathsf{PKE} \overset{2}{\hookrightarrow} \mathsf{LTDF} \overset{\lambda}{\hookrightarrow} \mathsf{DDH}.$$

Using these results, we show the following:

$$\forall\mathsf{MI}_n(\mathsf{PKE}) \overset{2}{\hookrightarrow} \forall\mathsf{MI}_n(\mathsf{LTDF}) \overset{1}{\hookrightarrow} \mathsf{LTDF/DDH} \overset{\lambda}{\hookrightarrow} \mathsf{DDH}. \tag{3.11}$$

Primitive $\mathsf{LTDF/DDH}$ is a construction of a lossy trapdoor function from a DDH group by Peikert et al.

Clearly, $\mathsf{LTDF} \overset{2}{\hookrightarrow} \mathsf{DDH}$ implies $\mathsf{LTDF/DDH} \overset{2}{\hookrightarrow} \mathsf{DDH}$ and so we concentrate on proving the first two reductions.

Theorem 3.38. *Let $n \in \mathsf{poly}(\lambda)$. Then, $\forall\mathsf{MI}_n(\mathsf{PKE}) \overset{2}{\hookrightarrow} \forall\mathsf{MI}_n(\mathsf{LTDF})$.*

Proof. We use the construction of Peikert et al. Let $(S_{inj}, S_{loss}, F, F^{-1})$ be a collection of (m, k)-lossy trapdoor functions. Let $\mathcal{H}$ be a family of pairwise independent hash functions from $\{0,1\}^m$ to $\{0,1\}^l$ for $l \leq k - 2\log(1/\epsilon)$ where $\epsilon = \mathsf{negl}(\lambda)$. The message space is $\{0,1\}^l$. Define the encryption scheme $\mathcal{E} = (\mathsf{Gen}; \mathsf{Enc}; \mathsf{Dec})$ where:

- $\mathsf{Gen}(1^\lambda)$ takes an injective trapdoor function $(s,t) \leftarrow_\$ S_{inj}$ and a hash function $h \leftarrow_\$ \mathcal{H}$. Then, it sets $pk = (s, h)$ and $sk = (t, h)$.
- $\mathsf{Enc}(pk, m)$ first generates random $x \leftarrow_\$ \{0,1\}^m$. Then, it sets $c_1 = F(s, x)$ and $c_2 = m \oplus h(x)$. It outputs $c = (c_1, c_2)$.
- $\mathsf{Dec}(sk, c)$ computes $x = F^{-1}(t, c_1)$ and returns $c_2 \oplus h(x)$.

Peikert et al. proved that $\mathcal{E}$ is an IND-CPA secure scheme. We show that if $P = (S_{inj}, S_{loss}, F, F^{-1})$ is a LTDF in the multi-instance setting then $\mathcal{E}$ is a IND-CPA tightly secure scheme in the multi-user setting. We do it using the same technique as Peikert et al. Consider the following variables:

- Variable X_0: choose $(s_1, t_1), \ldots, (s_n, t_n) \leftarrow_\$ S_{inj}, x_1, \ldots, x_n \leftarrow_\$ \{0,1\}^m$ and also $h_1, \ldots, h_n \in \mathcal{H}$. Then the value of X_0 is
$$(s_1, \ldots, s_n, h_1, \ldots, h_n, F(s_1, x_1), \ldots, F(s_n, x_n), h(x_1), \ldots, h(x_n)).$$
- Variable X_1: choose $(s_1, t_1), \ldots, (s_n, t_n) \leftarrow_\$ S_{loss}, x_1, \ldots, x_n \leftarrow_\$ \{0,1\}^m$ and also $h_1, \ldots, h_n \in \mathcal{H}$. Then the value of X_1 is
$$(s_1, \ldots, s_n, h_1, \ldots, h_n, F(s_1, x_1), \ldots, F(s_n, x_n), h(x_1), \ldots, h(x_n)).$$

- Variable X_2: choose $(s_1, t_1), \dots, (s_n, t_n) \leftarrow_\$ S_{loss}, x_1, \dots, x_n \leftarrow_\$ \{0,1\}^m$ and also $r_1, \dots, r_n \in \{0,1\}^l$. Then the value of X_2 is

$$(s_1, \dots, s_n, h_1, \dots, h_n, F(s_1, x_1), \dots, F(s_n, x_n), r_1, \dots, r_n).$$

- Variable X_3: choose $(s_1, t_1), \dots, (s_n, t_n) \leftarrow_\$ S_{inj}, x_1, \dots, x_n \leftarrow_\$ \{0,1\}^m$ and also $r_1, \dots, r_n \in \{0,1\}^l$. Then the value of X_3 is

$$(s_1, \dots, s_n, h_1, \dots, h_n, F(s_1, x_1), \dots, F(s_n, x_n), r_1, \dots, r_n).$$

Lemma 3.39 ([70], generalised)**.** *Let X_0, X_1, X_2, X_3 be random variables defined as above. Then, $\{X_0\} \overset{c}{\approx} \{X_1\} \overset{s}{\approx} \{X_2\} \overset{c}{\approx} \{X_3\}$.*

Proof. We first reprove this lemma for $n = 1$ and when P is a standard lossy trapdoor function and call these random variables X'_0, X'_1, X'_2, X'_3. Clearly, X'_0 and X'_1 are computationally indistinguishable since no adversary can distinguish between the lossy and injective functions. We argue similarly for X'_2 and X'_3. We only have to show that no efficient algorithm can distinguish between X'_1 and X'_2. We use the Leftover Hash Lemma (see Lemmas 3.5 and 3.6). Let s be a fixed function index generated by S_{loss}. Since $F(s, \cdot)$ has at most 2^{m-k} outputs and $x = x_1$ is chosen independently of s, we have that:

$$\tilde{H}_\infty(x|s, F(s,x)) \geq \tilde{H}_\infty(x|s) - (m-k) = k.$$

Hence, by setting $l \leq k - 2\log(1/\epsilon)$, we have that $\Delta(X'_1, X'_2) \leq \epsilon = \mathsf{negl}(\lambda)$. Thus, we are done.

Let us focus on the multi-instance case now. Note that X_0 and X_1 are computationally indistinguishable because of the multi-setting indistinguishability property of LTDFs. Identical argument works for X_2 and X_3. In order to show that X_1 and X_2 are statistically indistinguishable we use the result above for the single-instance case. Define random variables $Y_0, Y_1, ..., Y_n$ as follows.

Variable Y_i: choose $(s_1, t_1), \dots, (s_n, t_n) \leftarrow_\$ S_{loss}, x_1, \dots, x_n \leftarrow_\$ \{0,1\}^m$ and also $h_1, \dots, h_n \in \mathcal{H}, r_1, \dots, r_n \in \{0,1\}^l$. Then the value of Y_i is

$$(s_1, \dots, s_n, h_1, \dots, h_n, F(s_1, x_1), \dots, F(s_n, x_n), r_1, \dots, r_i, h(x_{i+1}), ..., h(x_n)).$$

Note that $Y_0 = X_1$ and $Y_n = X_2$. One observes that by a simple reduction one has

$$\Delta(Y_i, Y_{i+1}) \leq \Delta(X'_1, X'_2) \leq \epsilon.$$

Then, by the standard hybrid argument we get $\Delta(X_1, X_2) \leq n \cdot \epsilon$ which is still negligible.

□

One observes that, by Lemma 3.39, the encryption scheme $\mathcal{E}$ is indeed IND-CPA in the multi-user setting. Moreover, the reduction is still tight because only the statistical difference between X_1 and X_2 is dependent on the number of instances n. This completes the proof.

□

In a similar way we can define the multi-instance All-But-One LTDFs and use them to construct IND-CCA secure scheme in the multi-user setting. One could take the construction provided by Peikert et al. and extend the proof of security to many instances. This approach would give us a reduction with security loss $O(q_{\mathsf{dec}})$, the same as in the single-instance case. However, we omit the formal proof here.

Let us consider the construction of a pseudorandom generator provided by Peikert et al., i.e. define $G(x) = (h_1(F(s,x)), h_2(x))$, for $h_1, h_2 \leftarrow_{\$} \mathcal{H}$. By Lemma 3.4 in [70] and the Leftover Hash Lemma, if $(S_{inj}, S_{loss}, F, F^{-1})$ is a collection of lossy trapdoor functions then G is a pseudorandom generator. One observes that this result can be easily extended to the multi-instance setting thanks to Lemma 3.39. This example and Theorem 3.38 show that multi-instance LTDFs can be useful in constructing more general primitives in the multi-instance setting. We now present how to build such primitives from a standard assumption, namely DDH.

Constructing Tightly Extensible LTDFs. We focus on proving the second reduction in (2) which involves encrypting matrices in a way similar to ElGamal encryption scheme. Suppose we work with a group G of prime order p and generator g. For simplicity, we write $[x] = g^x$ for $x \in \mathbb{Z}_p$. We write small bold letters (e.g. $\mathbf{x}$,$\mathbf{y}$) for column vectors and capital bold letters (e.g. $\mathbf{A}$,$\mathbf{U}$) for matrices. We denote $\mathbf{A}^t$ to be the transpose of $\mathbf{A}$. For simplicity, we write $[x] = g^x$ for $x \in \mathbb{Z}_p$ and similarly $[\mathbf{x}] = ([x_1], \dots, [x_m])$ for $\mathbf{x} = (x_1, \dots, x_m)$. We use identical notation $[\mathbf{A}]$ also for matrix $\mathbf{A}$.

For a matrix $\mathbf{A}$ with at least 2 rows, we write $\mathtt{WLR}(\mathbf{A})$ for a matrix $\mathbf{A}$ without last row. Similarly, we define $\mathtt{WLC}(\mathbf{A})$ for $\mathbf{A}$ without last column. We will use the following simple observation. Let $m, n, k \geq 2$ and $\mathbf{A}, \mathbf{B}$ be $m \times n$ and $n \times k$ matrices in G respectively. Then, $\mathtt{WLR}(\mathbf{AB}) = \mathtt{WLR}(\mathbf{A})\mathbf{B}$ and $\mathtt{WLC}(\mathbf{AB}) = \mathbf{A}\mathtt{WLC}(\mathbf{B})$

We briefly recall the method for encrypting matrix $\mathbf{M} \in \mathbb{Z}_p^{m \times m}$ by Peikert et al. Firstly, we generate secret keys $\mathbf{z} = (z_1, \dots, z_m) \in \mathbb{Z}_p^m$ and set $sk = \mathbf{z}, pk = [\mathbf{z}]$. Also, denote $h_i = [z_i]$. Then, choose uniformly random $r_1, \dots, r_m \in \mathbb{Z}_p$. The encryption of $\mathbf{M}$ is a matrix $\mathbf{C} = (C_{i,j})$ where $C_{i,j} = ([r_i], [m_{i,j}][z_i]^{r_i})$. The construction of a LTDF $(S_{inj}, S_{loss}, F, F^{-1})$ from a DDH group looks as follows.

- S_{inj} first selects group parameters (G, p, g). Then, it returns $(\mathbf{C}, t)$ where $\mathbf{C}$ is a matrix encryption of the identity $\mathbf{I}$ and t consists of secret keys $\mathbf{z}$.
- S_{loss} selects group parameters (G, p, g) and returns $(\mathbf{C}, \perp)$ where $\mathbf{C}$ is a matrix encryption of zero matrix $\mathbf{0}$.

- F takes as input $(\mathbf{C}, \mathbf{x})$, where $\mathbf{C}$ is a function index and $\mathbf{x} \in \{0,1\}^m$, and returns $\mathbf{y} = \mathbf{x}\mathbf{C}$.
- F^{-1} takes as input $\mathbf{y} = ((y_{1,0}, y_{1,1}), \ldots, (y_{m,0}, y_{m,1}))$ and the trapdoor $\mathbf{z} = (z_1, \ldots, z_m)$. Then, it returns $\mathbf{x} = (x_1, \ldots, x_m)$ where $x_i = \log_g(y_{i,1}/y_{i,0}^{z_i})$ (it can be efficiently computed since x_i is a bit).

Security of this lossy trapdoor function relies heavily on the fact that the matrix encryption scheme described above gives indistinguishable ciphertexts if the DDH assumption holds [67, 70]. The key observation is that if DDH is hard in G, then for randomly chosen $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_p^m$, $[\mathbf{x}\mathbf{y}^t]$ is indistinguishable from a uniformly random chosen matrix $\mathbf{U} \leftarrow_\$ G^{m \times m}$ (see Example 3.18). We claim that this is also true for many instances, i.e. for randomly chosen $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{Z}_p^m$ where $i = 1, \ldots, n$, $([\mathbf{x}_1\mathbf{y}_1^t], \ldots, [\mathbf{x}_n\mathbf{y}_n^t])$ is indistinguishable from a uniformly random chosen matrix $([\mathbf{U}_1], \ldots, [\mathbf{U}_n])$ where $U_i \leftarrow_\$ \mathbb{Z}_p^{m \times m}$. We write it formally as follows.

Theorem 3.40. *Let n be a polynomial in λ and $m \geq 2$. Define primitive $P_m = \left\langle \mathbb{P}, S_P, F_P R_P, \frac{1}{2} \right\rangle$ where:*

- $\mathbb{P} = (\mathbb{Z}_p^m \times \mathbb{Z}_p^m, \mathbb{Z}_p^{m \times m}, \mathbb{Z}_p^m \times \mathbb{Z}_p^m)$,
- *S_P sends group information (G, p, g) to R_P,*
- *F_P contains only a function f defined by $f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}\boldsymbol{y}^t$,*
- *R_P first generates $\boldsymbol{x}, \boldsymbol{y} \leftarrow_\$ \mathbb{Z}_p^m$, calls f to get $\boldsymbol{x}\boldsymbol{y}^t$, samples $b \leftarrow_\$ \{0,1\}$ and sets $U = \boldsymbol{x}\boldsymbol{y}^t$ if $b = 0$ and $U \leftarrow_\$ \mathbb{Z}_p^{m \times m}$ if $b = 1$. Finally, it sends $([\boldsymbol{x}], [\boldsymbol{y}], [\boldsymbol{U}])$ along with (G, p, g) to adversary $\mathcal{A}$. Security game R_P returns 1 if $\mathcal{A}$ guesses the bit b.*

Then, $\forall \mathsf{MI}_n(P_m) \overset{1}{\hookrightarrow} P_m$.

Proof. Assume first that there exists a PPT algorithm $\mathcal{C}_1$, which given a triple $([\mathbf{x}], [\mathbf{y}], [\mathbf{U}])$ sent by R_P, returns another triple $([\mathbf{x}'], [\mathbf{y}'], [\mathbf{U}'])$ such that: (i) $\mathbf{y}' = \mathbf{y}$, (ii) $\mathbf{x}'$ is uniformly random, (iii) if $b = 0$ then $\mathbf{U}' = \mathbf{x}'\mathbf{y}'^t$ and if $b = 1$ then $\mathbf{U}'$ is uniformly random with probability $1 - \mathsf{negl}(\lambda)$. In a similar fashion we can define a PPT algorithm $\mathcal{C}_2$ which does the same thing as $\mathcal{C}_1$ but it fixes $\mathbf{x}$ instead of $\mathbf{y}$. Note that if $\mathcal{C}_1$ exists then clearly $\mathcal{C}_2$ also exists.

Now, suppose there exists an adversary $\mathcal{A}$ which $\forall \mathsf{MI}_n(P_m)-$breaks f. We construct an adversary $S^{\mathcal{A}}$ which P_m-breaks f as follows. Given a triple $\mathbf{v} = ([\mathbf{x}], [\mathbf{y}], [\mathbf{U}])$ from R_P, it runs n independent copies of $\mathcal{C}_1$ on input $\mathbf{v}$ and gets back outputs $\mathbf{v}_1, \ldots, \mathbf{v}_n$. Next, it runs n independent copies of $\mathcal{C}_2$ where the i-th copy of $\mathcal{C}_2$ gets as input $\mathbf{v}_i$. Then, collect the outputs $\mathbf{w}_1, \ldots, \mathbf{w}_n$ and pass them to $\mathcal{A}$. Eventually, when $\mathcal{A}$ returns a bit b', output b'. Note that this reduction is tight by the property (iii) of $\mathcal{C}_1$ and by standard hybrid argument. Therefore, what we have left is to construct an algorithm $\mathcal{C}_1$.

Consider the following algorithm for $\mathcal{C}_1$ in Fig. 3. Note that we are able to compute $[\mathbf{x}']$ and $[\mathbf{U}']$ in lines 3 and 4 even though we do not know values for

```
$\mathcal{C}_1(1^\lambda, G, p, g, [\mathbf{x}], [\mathbf{y}], [\mathbf{U}])$
1 : $\mathbf{R} \leftarrow_\$ \mathbb{Z}_p^{m \times m}$
2 : $\tilde{\mathbf{r}} \leftarrow_\$ \mathbb{Z}_p^{m}$
3 : $[\mathbf{x}'] = [\mathbf{R}\mathbf{x} + \tilde{\mathbf{r}}]$
4 : $[\mathbf{U}'] = [\mathbf{R}\mathbf{U} + \tilde{\mathbf{r}}\mathbf{y}^t]$
5 : return $([\mathbf{x}'], [\mathbf{y}], [\mathbf{U}'])$
```

Figure 3.8: PPT algorithm for $\mathcal{C}_1$.

$\mathbf{x}', \mathbf{U}'$. Clearly, property (i) is satisfied. Also, $\mathbf{x}'$ is uniformly random because of the randomness of $\tilde{\mathbf{r}}$. The most challenging part is to show (iii).

First, suppose that $b = 0$. So we have $\mathbf{U} = \mathbf{x}\mathbf{y}^t$. Hence, $\mathbf{U}' = \mathbf{R}\mathbf{U} + \tilde{\mathbf{r}}\mathbf{y}^t = \mathbf{R}\mathbf{x}\mathbf{y}^t + \tilde{\mathbf{r}}\mathbf{y}^t = (\mathbf{R}\mathbf{x} + \tilde{\mathbf{r}})\mathbf{y}^t = \mathbf{x}'\mathbf{y}^t$. Now, consider the case $b = 1$. Then, $\mathbf{U}$ is a uniformly random matrix. We want to show that $\mathbf{U}'$ is also a uniformly random matrix with overwhelming probability. Denote $\mathbf{x} = (x_1, \ldots, x_m)$ and assume that $x_m \neq 0$ (this occurs with an overwhelming probability). We slightly change the algorithm for $\mathcal{C}_1$: we first choose random $\mathbf{x}' = (x'_1, \ldots, x'_m), \tilde{\mathbf{r}} = (\tilde{r}_1, \ldots, \tilde{r}_m)$ from $\mathbb{Z}_p^m$ and $\tilde{\mathbf{R}} = (\tilde{R}_{i,j}) \in \mathbb{Z}_p^{m \times (m-1)}$. Next, we set $\mathbf{r} = (r_1, \ldots, r_m)$, where $r_i = {x_m}^{-1}(x'_i - \tilde{r}_i - \sum_{j=1}^{m-1} x_j \tilde{R}_{i,j})$, and $\mathbf{R} = \left(\tilde{\mathbf{R}} \,\middle|\, \mathbf{r}\right)$. Note that this change does not affect the input/output behaviour of $\mathcal{C}_1$. Moreover, we can rewrite $\mathbf{R}$ as:

$$\mathbf{R} = \begin{pmatrix} \tilde{r}_1 & \tilde{R}_{1,1} & \tilde{R}_{1,2} & \cdots & \tilde{R}_{1,m-1} & x'_1 \\ \tilde{r}_2 & \tilde{R}_{2,1} & \tilde{R}_{2,2} & \cdots & \tilde{R}_{2,m-1} & x'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ \tilde{r}_m & \tilde{R}_{m,1} & \tilde{R}_{m,2} & \cdots & \tilde{R}_{m,m-1} & x'_m \end{pmatrix} \begin{pmatrix} 0 & 0 & \cdots & 0 & -x_m^{-1} \\ 1 & 0 & \cdots & 0 & -x_1 x_m^{-1} \\ 0 & 1 & \cdots & 0 & -x_2 x_m^{-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -x_{m-1} x_m^{-1} \\ 0 & 0 & \cdots & 0 & x_m^{-1} \end{pmatrix} \tag{3.12}$$

Denote $\bar{\mathbf{R}}$ and $\mathbf{M}$ as the left-hand side and the right-hand side matrices respectively. Also, define $\mathbf{A} = \mathbf{M}\mathbf{U}$ and $\hat{\mathbf{R}}$ such that $\bar{\mathbf{R}} = \left(\hat{\mathbf{R}} \,\middle|\, \mathbf{x}'\right)$. Note that the first column of $\mathbf{M}$ is the (additive) inverse of the last column of $\mathbf{M}$. Consequently, we get the same property in $\mathbf{A}$. Moreover, $\texttt{WLR}(\mathbf{M})$ is clearly invertible and thus $\mathbf{U}$ being uniformly random matrix implies that $\texttt{WLR}(\mathbf{A}) = \texttt{WLR}(\mathbf{M})\mathbf{U}$ is also uniformly random. For simplicity, let us denote $\mathbf{a}$ to be the last row of $\mathbf{A}$, $\mathbf{A}_1 = \texttt{WLR}(\mathbf{A})$ and $\mathbf{A}_2$ be the matrix $\mathbf{A}_1$ without the first row (which is the inverse of the last row of $\mathbf{A}$). Then, by the observations above we can expand

$\mathbf{U}'$ as follows:

$$\begin{aligned}\mathbf{U}' &= \mathbf{R}\mathbf{U} + \tilde{\mathbf{r}}\mathbf{y}^t \\ &= \left(\left(\hat{\mathbf{R}} \,\middle|\, \mathbf{0}\right) + \left(\mathbf{0} \,\middle|\, \mathbf{x}'\right)\right)\mathbf{A} + \tilde{\mathbf{r}}\mathbf{y}^t \\ &= \hat{\mathbf{R}}\mathbf{A}_1 + \mathbf{x}'\mathbf{a}^t + \tilde{\mathbf{r}}\mathbf{y}^t \\ &= \left(\left(\tilde{\mathbf{r}} \,\middle|\, \mathbf{0}\right) + \left(\mathbf{0} \,\middle|\, \tilde{\mathbf{R}}\right)\right)\mathbf{A}_1 + \mathbf{x}'\mathbf{a}^t + \tilde{\mathbf{r}}\mathbf{y}^t \\ &= \tilde{\mathbf{r}}(\mathbf{y}^t - \mathbf{a}^t) + \tilde{\mathbf{R}}\mathbf{A}_2 + \mathbf{x}'\mathbf{a}^t.\end{aligned} \tag{3.13}$$

This is equivalent to ${\mathbf{U}'}^t = \left(\mathbf{y} - \mathbf{a} \,\middle|\, \mathbf{A}_2^t\right)\left(\tilde{\mathbf{r}} \,\middle|\, \tilde{\mathbf{R}}\right)^t + \mathbf{a}{\mathbf{x}'}^t$. Note that $\left(\mathbf{y} - \mathbf{a} \,\middle|\, \mathbf{A}_2^t\right)$ is with high probability an invertible matrix because $\mathbf{A}_1$ is uniformly random. Moreover, we chose $\left(\tilde{\mathbf{r}} \,\middle|\, \tilde{\mathbf{R}}\right)$ uniformly at random and therefore $\mathbf{U}'$ is also uniformly random (with probability $1 - \mathsf{negl}(\lambda)$).

□

Note that $\forall\mathsf{MI}_n(\mathsf{LTDF}) \overset{1}{\hookrightarrow} \mathsf{LTDF}/\mathsf{DDH}$ follows from the Theorem 3.40 applied to the construction by Peikert et al. along with the random self-reducibility of DDH. Thus, combining (3.11) with Lemma 3.28 we obtain the following.

Theorem 3.41. *LTDF is $(\lambda, \forall)$-tightly extensible w.r.t. DDH and PKE is $(2\lambda, \forall)$-tightly extensible w.r.t. DDH.*

All in all, we have provided a new way of constructing multi-user IND-CPA encryption schemes out of a DDH group using tightly extensible lossy trapdoor functions. We leave it as an open question whether it is possible to obtain tightly extensible LTDFs from different standard assumptions, such as lattices.

3.5 Reductions in External Models

In cryptography, certain schemes can be proven secure if they have access to an ideal object which cannot be instantiated in practice. A prime example is a random oracle [13]. It expects some input from a finite set X and returns a uniformly chosen element from a finite set Y. This object is an idealisation of a hash function. In many cases, we cannot prove that a scheme, which uses a hash function, is secure but we can provide a proper reduction if the hash function is ideal, i.e. random oracle. Obviously, an adversary is also given access to random oracle (as it would be given formula of the hash function).

There are many other external models used in the literature: ideal cipher model, common reference string, or even quantum random oracle, just to name a few. In this section we focus on extending our framework defined in Section 3.4 to include any external models.

Let us think about how we can define a primitive in an external model. Consider a public-key encryption scheme PKE (see Example 3.26). It is defined in the standard model only. How can we modify it to get an encryption scheme in the random oracle model PKE_{ROM}? This primitive would definitely contain all implementations of PKE and more. Note that an encryption scheme (in any model) has to define the correctness property. This leads to an idea of defining a set of implementation as functions which can win the *correctness* game. We formalise this idea as follows.

Definition 3.42. *A restricted primitive P is a tuple $\langle \mathbb{P}, M_P, S_P, C_P, R_P, \sigma \rangle$ where:*

- *$\mathbb{P}$ is a tuple of sets (A, B, C, X, Y) where $C \subseteq A$,*
- *M_P is an oracle setup machine which generates functions $O : X \to Y$ according to some probability distribution,*
- *C_P is a PPT correctness machine,*
- *S_P is a PPT setup algorithm which sends parameters $(r_1, \dots)$ to R_P,*
- *$R_P^{(\cdot,\cdot)}$ is a PPT security algorithm which gets parameters $(r_1, \dots)$ from S_P.*
- *$\sigma : \mathbb{N} \to \mathbb{R}$ is a security threshold.*

We say that $f^{(\cdot)}$ is an implementation of P if $\Pr[1 \leftarrow\!\!{\scriptstyle\$}\, C^{f^O}] = 1$, where probability is over random coins in the system, including $O \leftarrow\!\!{\scriptstyle\$}\, M_P$.

Note that M_P decides in which model the primitive is. We denote $M_P = \emptyset$ for a machine which outputs nothing. This would mean that the restricted primitive is in the standard model. One observes that if $M_P = \emptyset$ then a restricted primitive is a primitive (according to the definition in Section 3.2).

Similarly as in Section 3.2, we now define an advantage of breaking a restricted primitive.

Definition 3.43. *Let $P = \langle \mathbb{P}, M_P, S_P, C_P, R_P, \sigma \rangle$ be a restricted primitive with $\mathbb{P} = (A, B, C, X, Y)$. Take an implementation $f^{(\cdot)}$ of P and any algorithm $\mathcal{A}$. We define the advantage of $\mathcal{A}^{(\cdot)}$ in breaking f as*

$$\mathsf{Adv}^{\mathrm{P}}_{f,\mathcal{A}}(\lambda) := \Pr[R_P^{f^O|_C, \mathcal{A}^O} = 1] - \sigma(\lambda)$$

where the probability is defined over random coins in the system, including $O \leftarrow\!\!{\scriptstyle\$}\, M_P$. We say that $\mathcal{A}$ $P-$breaks $f^{(\cdot)}$ if $\mathsf{Adv}^{\mathrm{P}}_{f,\mathcal{A}}(\lambda)$ is non-negligible. Primitive P is called secure if there exists an implementation $f^{(\cdot)}$ of P such that there are no PPT algorithms $\mathcal{A}^{(\cdot)}$ that $P-$break $f^{(\cdot)}$.

We introduce a notion of a fully black-box non-programmable reduction between restricted primitives.

Definition 3.44. *Let $P = \langle \mathbb{P}_1, M, S_P, C_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, M, S_Q, C_Q, R_Q, \tau \rangle$ be restricted primitives. We say that there is a fully black-box non-programmable reduction from P to Q in M (written as $P \overset{M}{\hookrightarrow} Q$) if there exist PPT algorithms $G^{(\cdot)}, S^{(\cdot)}$ such that:*

- *for every implementation $f^{(\cdot)}$ of Q, $G^{f^{(\cdot)}}$ is an implementation of P,*
- *for every implementation $f^{(\cdot)}$ of Q and every (unbounded) algorithm $\mathcal{A}^{(\cdot)}$, if $\mathcal{A}^{(\cdot)}$ P-breaks $G^{f^{(\cdot)}}$ then $S^{\mathcal{A}^{(\cdot)}}$ Q-breaks $f^{(\cdot)}$.*

In the literature, S has access to an external oracle O instead of $\mathcal{A}$. We call this reduction *non-programmable* since we let $\mathcal{A}$ have access to O via S, meaning that if adversary wants to query O, it sends the value to S, S passes it to O and returns to $\mathcal{A}$ what it got from O. Apart from that, S does not query O at all. This is clearly equivalent to $\mathcal{A}$ having access to O, as illustrated in the definition above. There is another type of reduction called *programmable* [40], where S can simulate a random oracle on its own. However, we omit the details in this thesis.

Let M be an oracle setup machine (e.g. random oracle model). We define a notion of a $M-$extension of a primitive.

Definition 3.45. *Let $P = \langle \mathbb{P}_1, \emptyset, S_P, C_P, R_P, \sigma \rangle$ be a restricted primitive in the standard model and M be an oracle setup machine. Then, a $M-$extension $P(M)$ of P is the tuple $P(M) = \langle \mathbb{P}_1, M, S_P, C_P, R_P, \sigma \rangle$.*

We provide a simple example of a M-extension of a primitive.

Example 3.46. Let us define a restricted primitive of an IND-CPA public key-encryption scheme in the standard model. We use the definition in Example 3.26. Let $\mathsf{PKE} = \langle \mathbb{P}_{\mathsf{PKE}}, \emptyset, S_{\mathsf{PKE}}, C_{\mathsf{PKE}}, R_{\mathsf{PKE}}, \frac{1}{2} \rangle$ where, as before, S_{PKE} does the sampling $b \leftarrow_{\$} \{0,1\}$ and R_{PKE} is the IND-CPA game. Here, C_{PKE} takes a uniformly random message m and asks for encryption of m. After it gets the ciphertext c, it asks for decryption of c. After getting some m', it returns 1 if $m = m'$ and 0 otherwise. Note that an implementation of PKE is an encryption scheme in the standard model. Indeed, $\Pr[1 \leftarrow_{\$} C_{\mathsf{PKE}}^{(\mathsf{Enc},\mathsf{Dec})}] = 1$ implies that for all messages m, we have $\mathsf{Dec}(\mathsf{Enc}(m)) = m$.

Define a random oracle with finite domain X and finite co-domain Y as follows. Denote $\mathcal{O}$ to be an oracle setup machine which chooses a function $f : X \to Y$ uniformly at random. Then, the $\mathcal{O}$-extension of PKE is a restricted primitive $\mathsf{PKE}(\mathcal{O})$ which consists of encryption schemes in the random oracle model. Since the correctness machines in PKE and $\mathsf{PKE}(\mathcal{O})$ are the same, any implementation of PKE is an implementation of $\mathsf{PKE}(\mathcal{O})$.

In this section we consider the following problem: given a reduction from P to Q in the standard model, does it also work in an external model? We write it formally as a conjecture.

Conjecture 3.47. *Let P and Q be restricted primitives in the standard model and M be any oracle setup machine. Then,*

$$P \hookrightarrow Q \implies P(M) \overset{M}{\hookrightarrow} Q(M).$$

Let (G, S) be a reduction from P to Q. Intuitively, (G, S) should also be a correct reduction from $P(M)$ to $Q(M)$. However, in the M model, adversary as well as an implementation have access to the "shared state" which is an external oracle. It is not the case in the standard model. This additional ability might help an adversary to break $P(M)$ but not $Q(M)$. This would mean that the reduction does not work in M.

We present two sufficient conditions for which Conjecture 3.47 holds. The first one focuses on two certain machines being *weakly indistinguishable*. We define this term below.

Definition 3.48. *Let $\mathcal{R}_1^{(\cdot,\cdot)}, \mathcal{R}_2^{(\cdot,\cdot)}$ be machines and M be an oracle setup machine which returns an oracle O. Then, $\mathcal{R}_1$ and $\mathcal{R}_2$ are weakly indistinguishable in M if for every function $f^{(\cdot)}$ and adversary $\mathcal{A}^{(\cdot)}$:*

$$|\Pr[1 \leftarrow_{\$} \mathcal{R}_1^{f^O, \mathcal{A}^O}] - \Pr[1 \leftarrow_{\$} \mathcal{R}_2^{f^O, \mathcal{A}^O}]|$$

is negligible in a security parameter λ.

We observe that if $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalent, meaning they have the same code and input/output behaviour, then they are weakly indistinguishable. The first condition is presented as follows.

Theorem 3.49. *Let $P = \langle \mathbb{P}_1, \emptyset, S_P, C_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, \emptyset, S_Q, C_Q, R_Q, \sigma \rangle$ be restricted primitives in the standard model with the same security threshold σ and M be an oracle setup machine. Suppose that (G, S) is a fully black-box reduction from P to Q and the following hold:*

- *G is a generic construction of $P(M)$ from $Q(M)$,*
- *Let $\mathcal{P} = R_P^{G^{(\cdot)},(\cdot)}$ and $\mathcal{Q} = R_Q^{(\cdot),S^{(\cdot)}}$. Then, $\mathcal{P}$ and $\mathcal{Q}$ are weakly indistinguishable in M.*

Then, (G, S) is a fully black-box non-programmable reduction from $P(M)$ to $Q(M)$ in M.

Proof. We only need to show that for any implementation $f^{(\cdot)}$ of $Q(M)$ and every (unbounded) algorithm $\mathcal{A}^{(\cdot)}$, if $\mathcal{A}^{(\cdot)}$ $P(M)$-breaks $G^{f^{(\cdot)}}$ then $S^{\mathcal{A}^{(\cdot)}}$ $Q(M)$-breaks $f^{(\cdot)}$. Note that the condition that $\mathcal{P}$ and $\mathcal{Q}$ are weakly indistinguishable and the fact that both P and Q have the same security threshold imply that

$$|\mathsf{Adv}^{\mathrm{p}}_{G^f,\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{q}}_{f,S^{\mathcal{A}}}(\lambda)| = \mathsf{negl}(\lambda).$$

Hence, if $\mathsf{Adv}^{\mathrm{p}}_{G^f,\mathcal{A}}(\lambda)$ is non-negligible then $\mathsf{Adv}^{\mathrm{q}}_{f,S^{\mathcal{A}}}(\lambda)$ is also non-negligible. Thus, the result holds. $\square$

```
$\mathcal{P} = R_{\mathsf{OWF}}^{G^f,\mathcal{A}}$
1 :  $(R_P)$ Generate $x \leftarrow_\$ \{0,1\}^\lambda$
2 :  $(G)$ send $x$ to $f$
3 :  $(G)$ get back $f(x)$ from $f$
4 :  $(G)$ compute $z = 0^{|f(x)|}||f(x)$
5 :  $(R_P)$ send $z$ to $\mathcal{A}$
6 :  $(R_P)$ get back $x'$ from $\mathcal{A}$
7 :  $(G)$ send $x'$ to $f$
8 :  $(G)$ get back $f(x')$ from $f$
9 :  $(R_P)$ if $f(x) = f(x')$
10 : $(R_P)$     return 1 and 0 otherwise
```

```
$\mathcal{Q} = R_{\mathsf{OWF}}^{f,S^\mathcal{A}}$
1 :  $(R_Q)$ Generate $x \leftarrow_\$ \{0,1\}^\lambda$
2 :  $(R_Q)$ send $x$ to $f$
3 :  $(R_Q)$ get back $f(x)$ from $f$
4 :  $(S)$ compute $z = 0^{|f(x)|}||f(x)$
5 :  $(S)$ send $z$ to $\mathcal{A}$
6 :  $(S)$ get back $x'$ from $\mathcal{A}$
7 :  $(R_Q)$ send $x'$ to $f$
8 :  $(R_Q)$ get back $f(x')$ from $f$
9 :  $(R_Q)$ if $f(x) = f(x')$
10 : $(R_Q)$     return 1 and 0 otherwise
```

Figure 3.9: Pseudo-codes for $\mathcal{P}$ and $\mathcal{Q}$. In each line, the (X) notation means that this line is originally executed by subsystem X. We omit the interaction between the subsystems of $\mathcal{P}$ and $\mathcal{Q}$

We provide an example where Theorem 3.49 can be applied.

Example 3.50. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a length-preserving one-way function. Define a function $g(x) = 0^{|f(x)|}||f(x)$. We claim that g is also one-way. Indeed, suppose there exists an adversary $\mathcal{A}$ which finds preimage of $g(x)$. Then, a reduction $S^\mathcal{A}$, given $f(x)$, can compute $g(x) = 0^{|f(x)|}||f(x)$, send $g(x)$ to $\mathcal{A}$ and output what $\mathcal{A}$ returns. If $\mathcal{A}$ outputs x' such that $g(x) = g(x')$, this means that $f(x) = f(x')$ and consequently, S also wins the one-wayness game.

We formalise this reduction using the framework from this section. Let OWF be a restricted primitive of a one-way function in the standard model (this can be defined similarly as in Example 3.11 with a correctness machine which always returns 1). We give a reduction (G, S) from OWF to OWF as follows. Define (G, S) as:

- $G^{(\cdot)}$ having oracle access to some f: given $x \leftarrow_\$ \{0,1\}^\lambda$ from the security game, send x to f and get back $f(x)$. Then, return $0^{|f(x)|}||f(x)$.
- $S^{(\cdot)}$ having access to some $\mathcal{A}$: given y from the security game, construct $0^{|y|}||y$ and send it to $\mathcal{A}$. Then, get back x' from $\mathcal{A}$ and return x'.

Clearly, (G, S) is a fully black-box reduction from OWF to OWF by the argument from the previous paragraph. We want to check if (G, S) is still a valid reduction in an external model M. In this case, a function f has an access to an external oracle.

We want to apply Theorem 3.49. Note that the correctness machine of OWF always returns 1 so the first condition is easily satisfied. We want to prove the latter one, i.e. $\mathcal{P} = R_{\mathsf{OWF}}^{G^{(\cdot)},(\cdot)}$ and $\mathcal{Q} = R_{\mathsf{OWF}}^{(\cdot),S^{(\cdot)}}$ are weakly indistinguishable in M. We show that they are actually equivalent and have the same code.

Consequently, they have the same input/output behaviour and they are weakly indistinguishable for any M. We write down the code of $\mathcal{P}$ and $\mathcal{Q}$ in Fig. 3.9. Note that $\mathcal{P}$ and $\mathcal{Q}$ have the same code and consequently, they are equivalent. Therefore, the second condition is satisfied and by Theorem 3.49, $\mathsf{OWF}(M) \overset{M}{\hookrightarrow} \mathsf{OWF}(M)$.

We state the second sufficient condition for the Conjecture 3.47 to hold. Let us first define the notion of *simulatability.*

Definition 3.51. *Let M be an oracle setup machine and $R^{(\cdot,\cdot)}$ be an oracle machine. Then, function $f^{(\cdot)}$ is $[R, M]-$simulatable by function g if g is a function in standard model such that for all adversaries $\mathcal{A}^{(\cdot)}$, the advantage:*

$$|\Pr_{O \leftarrow\$ M}[1 \leftarrow\$ R^{f^{O},\mathcal{A}^{O}}] - \Pr_{O \leftarrow\$ M}[1 \leftarrow\$ R^{g,\mathcal{A}^{O}}]|$$

is negligible in a security parameter λ.

Note that this notion is different from being *indistinguishable* since here, adversary $\mathcal{A}$ interacts with f (or g) via R and not directly.

We show that if all implementations of $Q(M)$ are simulatable then Conjecture 3.47 is true. Formally, we state the condition as follows.

Theorem 3.52. *Let $P = \langle \mathbb{P}_1, \emptyset, S_P, C_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, \emptyset, S_Q, C_Q, R_Q, \sigma \rangle$ be restricted primitives in the standard model with the same security threshold σ and M be an oracle setup machine. Suppose that (G, S) is a fully black-box reduction from P to Q and the following hold:*

- *G is a generic construction of $P(M)$ from $Q(M)$,*
- *Let $\mathcal{P} = R_P^{G^{(\cdot)},(\cdot)}$ and $\mathcal{Q} = R_Q^{(\cdot),S^{(\cdot)}}$. Then, for every implementation f of $Q(M)$, there exists an implementation g of Q such that f is both $[\mathcal{P}, M]$ and $[\mathcal{Q}, M]$-simulatable by g.*

Then, (G, S) is a fully black-box non-programmable reduction from $P(M)$ to $Q(M)$ in M.

Proof. Just like before, the first condition for (G, S) to be a reduction is satisfied automatically by the first assumption. Now, suppose that adversary $\mathcal{A}^{(\cdot)}$ $P(M)-$breaks $G^{f^{(\cdot)}}$. By the simulatability property we have that $\mathcal{A}^{(\cdot)}$ $P(M)-$breaks G^g for some g. Note that the adversary gains no advantage of having access to oracle provided by the setup machine to break g and consequently, we can treat $\mathcal{A}^{(\cdot)}$ in the standard model. Thus, we can use the fact that (G, S) is a standard model reduction and conclude that $S^{\mathcal{A}^{(\cdot)}}$ $Q-$breaks g. Now, by adding back the oracle access to $\mathcal{A}$ and using the fact that f is $[\mathcal{Q}, \mathcal{M}]-$simulatable by g, we get that $S^{\mathcal{A}^{(\cdot)}}$ $Q(M)-$breaks $f^{(\cdot)}$, so the result holds. □

Unfortunately, not many (interesting) implementations have the simulatability property and consequently, the Theorem 3.52 does not seem to be useful in practice.

Non-uniformity. We show that Conjecture 3.47 holds for non-uniform primitives against non-uniform adversaries in the random oracle model. In order to state this theorem, we need to define non-uniformity based on our framework. We say that a primitive is non-uniform meaning that apart from the security parameter, it is also given an *advice* as input. Hence, for different advice it can have different behaviour. In case of primitives defined earlier in this section, they are not given any additional parameters apart from the security parameter. Similarly, we can define non-uniform adversaries. They are also given some additional input which could potentially help them break a scheme. We formalise the notion of uniformity below.

Definition 3.53. *A restricted non-uniform primitive P is a tuple $\langle \mathbb{P}, M_P, S_P, C_P, R_P, \sigma \rangle$ where:*

- *$\mathbb{P}$ is a tuple of sets (K, A, B, C, X, Y) where $C \subseteq A$ and K is an advice space,*
- *M_P is an oracle setup machine which generates functions $O : X \to Y$ according to some probability distribution,*
- *C_P is a PPT correctness machine,*
- *S_P is a PPT setup algorithm which sends parameters $(r_1, \ldots)$ to R_P,*
- *$R_P^{(\cdot,\cdot)}$ is a PPT security algorithm which gets parameters $(r_1, \ldots)$ from S_P.*
- *$\sigma : \mathbb{N} \to \mathbb{R}$ is a security threshold.*

An implementation of P is a function $f^{(\cdot)} : K \times A \to \{0,1\}^*$ satisfying:

$$\forall \lambda \in \mathbb{N}, \forall a_\lambda \in K, \Pr[C_P^{f_{a_\lambda}^O} = 1] = 1 - \mathsf{negl}(\lambda)$$

where O is an oracle generated by M_P and $f_x : A \to B$ is defined by $f_x(y) = f(x, y)$.

We define an advantage of a non-uniform adversary breaking a non-uniform primitive.

Definition 3.54. *Let $P = \langle \mathbb{P}, M_P, S_P, C_P, R_P, \sigma \rangle$ be a restricted non-uniform primitive with $\mathbb{P} = (K, A, B, C, X, Y)$. Take an implementation $f^{(\cdot)}$ of P and any algorithm $\mathcal{A}^{(\cdot)}$. We define the advantage of $\mathcal{A}^{(\cdot)}$ in break-*

ing $f^{(\cdot)}$ as

$$\mathsf{Adv}^{\mathrm{p}}_{f,\mathcal{A}}(\lambda, a_\lambda, b_\lambda) := \Pr[R_P^{f^O_{a_\lambda}|_C, \mathcal{A}^O(b_\lambda)} = 1] - \sigma(\lambda)$$

where the probability is defined over random coins in the system, including $O \leftarrow_\$ M_P$. We say that $\mathcal{A}$ $P-$breaks $f^{(\cdot)}$ if $\max_{a_\lambda, b_\lambda \in K} \mathsf{Adv}^{\mathrm{p}}_{f,\mathcal{A}}(\lambda, a_\lambda, b_\lambda)$ is non-negligible. Primitive P is called secure if there exists an implementation $f^{(\cdot)}$ of P such that there are no PPT algorithms $\mathcal{A}^{(\cdot)}$ that $P-$break $f^{(\cdot)}$.

Now, we are ready to introduce the notion of a reduction between non-uniform primitives.

Definition 3.55. *Let $P = \langle \mathbb{P}_1, M, S_P, C_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, M, S_Q, C_Q, R_Q, \tau \rangle$ be restricted non-uniform primitives. We say that there is a fully black-box non-programmable $\mathcal{N} \to \mathcal{N}$ (non-uniform primitives against non-uniform adversaries) reduction from P to Q in M (written as $P \underset{\mathcal{N}}{\overset{M}{\hookrightarrow}} Q$) if there exist PPT algorithms $G^{(\cdot)}, S^{(\cdot)}$ such that:*

- *for every implementation $f^{(\cdot)}$ of Q, $G^{f^{(\cdot)}}$ is an implementation of P,*
- *for every implementation $f^{(\cdot)}$ of Q and every (unbounded) algorithm $\mathcal{A}^{(\cdot)}$, if $\mathcal{A}^{(\cdot)}$ P-breaks $G^{f^{(\cdot)}}$ then $S^{\mathcal{A}^{(\cdot)}}$ Q-breaks $f^{(\cdot)}$.*

Similarly as before, we define a M-extension of a non-uniform primitive $P = \langle \mathbb{P} = (K, A, B, C, X, Y), M_P, S_P, C_P, R_P, \sigma \rangle$ as $P(M) = \langle \mathbb{P}, M, S_P, C_P, R_P, \sigma \rangle$. We also denote $\bar{P}$ to be the restricted (uniform) primitive $\langle (A, B, C, X, Y), M_P, S_P, C_P, R_P, \sigma \rangle$.

We prove that, under certain assumptions, Conjecture 3.47 holds in a non-uniform case for the random oracle model.

Theorem 3.56. *Let $P = \langle \mathbb{P}_1, \emptyset, S_P, C_P, R_P, \sigma \rangle$ and $Q = \langle \mathbb{P}_2, \emptyset, S_Q, C_Q, R_Q, \tau \rangle$ be restricted non-uniform primitives in the standard model with advice space K and $\mathcal{O}$ represents a random oracle. Suppose that (G, S) is a fully black-box $\mathcal{N} \to \mathcal{N}$ reduction from P to Q and the following hold:*

- *G is a generic construction of $P(\bar{M})$ from $Q(\bar{M})$,*
- *$\forall f, \forall \mathcal{A}, \exists \mathsf{negl}, \forall \lambda \in \mathbb{N}, \forall a_\lambda, b_\lambda \in K$:*

$$\mathsf{Adv}^{\mathrm{p}}_{G^f,\mathcal{A}}(\lambda, a_\lambda, b_\lambda) \leq \mathsf{Adv}^{\mathrm{q}}_{f,S^{\mathcal{A}}}(\lambda, a_\lambda, b_\lambda) + \mathsf{negl}(\lambda).$$

Then, (G, S) is a fully black-box non-programmable reduction from $P(\bar{\mathcal{O}})$ to $Q(\bar{\mathcal{O}})$ in M assuming PRFs exist.

Proof. Take any (uniform) implementation f of $Q(\bar{M})$ and any (uniform) adversary $\mathcal{A}$. Suppose that $\mathcal{A}$ $P(M)$-breaks G^f. Denote $\bar{f}_k$ to be a function $f|_C$ but instead of calling a random oracle, it calls a PRF on key k. Let us use the notation that for function F and G, $F \approx G$ meaning that $F - G$ is negligible. Let $O \leftarrow_\$ \mathcal{O}$. Then, by the second assumption and the PRF property we have:

$$\begin{aligned}
\Pr[R_P^{G^{f^O|_C}, \mathcal{A}^O} = 1] - \sigma(\lambda) &\approx \Pr_{k \leftarrow_\$ K}[R_P^{G^{\bar{f}_k|_C}, \mathcal{A}(k)} = 1] - \sigma(\lambda) \\
&= \frac{1}{|K|} \sum_{k \in K} (\Pr[R_P^{G^{\bar{f}_k|_C}, \mathcal{A}(k)} = 1] - \sigma(\lambda)) \\
&\leq \frac{1}{|K|} \sum_{k \in K} (\Pr[R_Q^{\bar{f}_k|_C, S^{\mathcal{A}(k)}} = 1] - \tau(\lambda) + \mathsf{negl}(\lambda)) \\
&\leq \Pr_{k \leftarrow_\$ K}[R_Q^{\bar{f}_k|_C, S^{\mathcal{A}(k)}} = 1] - \tau(\lambda) + \mathsf{negl}(\lambda) \\
&\leq \Pr[R_Q^{f^O|_C, S^{\mathcal{A}^O}} = 1] - \tau(\lambda) + \mathsf{negl}(\lambda).
\end{aligned} \tag{3.14}$$

Therefore, $S^{\mathcal{A}}$ $Q(M)$-breaks f. □

We admit that the conditions we assume look very powerful and seem to be hard to be applied in practice. All in all, we leave it as an open question whether Conjecture 3.47 holds in general or not.

3.6 Meta-reductions

Meta-reduction technique, which was introduced in [20, 68, 22], is a method of showing a non-existence of a reduction and can be understood as a "reduction against a reduction". Let (G, S) be a fully black-box reduction. Then, a meta-reduction $\mathcal{M}^S$ interacts with the reduction S by simulating adversary $\mathcal{A}$ in order to break another scheme (or the same as the one S was trying to break). Then, assuming that this scheme is secure, we have that no fully black-box reduction exists. We provide an extended example of a meta-reduction used in analysing the security of signature schemes.

Example 3.57. Let G be a group of prime order p and generator g. We start with the well-known Schnorr identification protocol [76] which asks to prove a knowledge of the discrete logarithm of a group element. It involves two parties: P (prover) and V (verifier) and they are both given $X = g^x$. Proves P wants to convince C that P knows x. The protocol has three rounds. In the first round, P sends a random $r \in \mathbb{Z}_p$ to V. Then, V sends back a *challenge* $c \leftarrow_\$ \mathbb{Z}_p$. Next, P outputs $s = r + cx$. Eventually, verifier V can check that P knows x by checking $g^s = X^c g^r$. The full protocol can be seen in Fig. 3.10.

One can modify the Schnorr's Protocol into a signature scheme using the Fiat-Shamir transformation [37]. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function (or random oracle). In this case, a signer behaves just like a prover P and

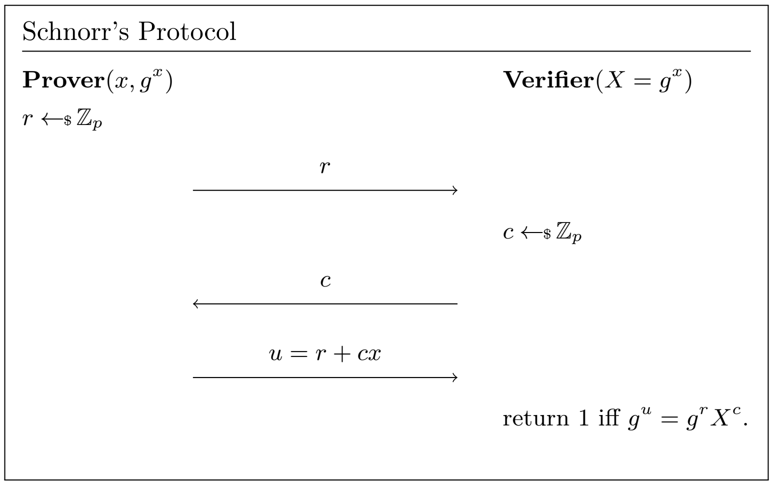
Schnorr's Protocol
Prover(x, g^x)
Verifier$(X = g^x)$
$r \leftarrow_{\$} \mathbb{Z}_p$
r
$c \leftarrow_{\$} \mathbb{Z}_p$
c
$u = r + cx$
return 1 iff $g^u = g^r X^c$.

Figure 3.10: Schnorr's Protocol.

simulates the behaviour of V by setting $c = H(m, g^r)$ where m is the message. Formally, we define the signature scheme as follows. Let $\Sigma_H = (\mathsf{Gen}; \mathsf{Sign}; \mathsf{Ver})$ be a signature scheme where:

- Gen selects at random $x \leftarrow_{\$} \mathbb{Z}_p$. The signing key is $sk = x$ and the verification key is $vk = g^x$.
- $\mathsf{Sign}(sk, m)$ first picks $r \leftarrow_{\$} \mathbb{Z}_p$, computes $R = g^x, c = H(m, R)$ and $u = r + cx$. The output is (u, c).
- $\mathsf{Ver}(vk, m, (u, c))$ returns 1 if and only if $H(m, g^u vk^{-c}) = c$.

Schnorr signatures Σ_H are widely used in many cryptographic protocols and cryptosystems, e.g. proofs of knowledge, zero-knowledge or Signed ElGamal. Pointcheval and Stern showed that if H is a random oracle then Σ_H is EUF-CMA secure as long as the discrete logarithm problem is hard [72] (and is proven using the so-called Forking Lemma [72, 12]).

We introduce the notion of an *algebraic reduction* [20]. An algorithm A is algebraic with respect to group G if A can only perform group operations on elements on G. For example, if $G = \mathbb{Z}_p^*$ is the multiplicative group of units of a finite field $\mathbb{F}_p$ then an algebraic algorithm against G cannot perform an addition operation such as $1_G + g$ for $g \in G$ (even though it is defined). One observes that algorithms in generic group model [78, 15] are clearly algebraic. Note that an algebraic reduction still can exploit properties of the underlying group G. Formally, we say that an algorithm A is algebraic w.r.t. G if there exists a PPT extractor E such that given A's input $(s, g_1, ..., g_k) \in \{0,1\}^* \times G^k$, it can recover for any $h \in G$, output by A after m steps, $\alpha_1, ...\alpha_k$ such that $h = g_1^{\alpha_1} ... g_k^{\alpha_k}$. Machine E usually has non black-box access to A, i.e. it can see the code of A.

We require that E runs in time $O(\mathsf{poly}(m, |A|))$, where $|A|$ is the size of code of A. We say that (G, S) is an algebraic reduction if S is algebraic.

Paillier and Vergnaud [68] showed that the Schnorr's signature is secure under chosen message attack. In this and the next example, we state their further results. Namely, universal and existential forgeries under any type of attack cannot be proven secure under the DLOG (or even the one-more DLOG) w.r.t. an algebraic reduction. These observations are shown using meta-reduction technique. Denote $\mathsf{UUF\text{-}KA}[\Sigma_H]$ to be primitive of Σ_H with the $\mathsf{UUF\text{-}KA}$ security game and DLOG_G to be the discrete logarithm primitive w.r.t. group G.

Theorem 3.58. *Let G be a group of prime order p and generator g. If the one-more discrete logarithm holds in G then there is no fully black-box reduction from $\mathsf{UUF\text{-}KA}[\Sigma_H]$ to DLOG_G.*

Proof. We prove it by contradiction. Let (G, S) be such a reduction. Suppose that S executes adversary $\mathcal{A}$ at most n times. We construct a meta-reduction which can break the n-DL.

Let us first define an adversary $\mathcal{A}$ which wins the $\mathsf{UUF\text{-}KA}$ game against Σ_H.

```
On input $(y, m)$:
  1. Select $R \leftarrow_\$ G$.
  2. Compute $c = H(m, R)$.
  3. Use brute-force search to find $s$ such that $g^u = Ry^c$.
  4. Return $(u, c)$.
```

Clearly, $\mathcal{A}$ is not efficient due to the brute-force approach but it creates eventually a valid signature for a message m. This means that $S^{\mathcal{A}}$ wins the discrete logarithm game with non-negligible probability. Let us define $\mathcal{A}'$ which behaves identically as $\mathcal{A}$ but is much more efficient, since it has access to the DLOG oracle. Here, `Transcript` is a global list and j is an global integer. Also, let $(R_1, ..., R_n)$ be uniformly random chosen values from G.

```
On input $(y, m)$:
  1. if $[(y, m) \mapsto (u, c)] \in$ Transcript for some signature $(u, c)$:
  2.     return $(u, c)$.
  3. Define $R = R_j$ and set $j = j + 1$.
  4. Compute $c = H(m, R)$.
  5. Ask for a discrete logarithm $u$ such that $g^u = Ry^c$.
  6. Add $[(y, m) \mapsto (u, c)]$ to Transcript.
  7. Return $(u, c)$.
```

We are ready to describe the meta-reduction $\mathcal{M}$. Firstly, $\mathcal{M}$ is given some $(R_0, ..., R_n)$ as input and wants to win the OMDL game. It takes the first group element $R_0 \in G$ and initialises $\texttt{Transcript} = \emptyset$ and $j = 1$. Then, $\mathcal{M}$ runs S on input R_0. Note that the reduction S is allowed to invoke n times the universal forger $\mathcal{A}$ with any public keys $y_i = g^{x_i}$ and messages m_i. We use the assumption that S is algebraic, meaning that when it returns a discrete logarithm k_0 of R_0 then the extractor E of S can extract values $(x_1, ..., x_n)$. Now, $\mathcal{M}$ simulates the universal forger $\mathcal{A}'$ defined above which interacts with S. During the simulation, $\mathcal{A}'$ asks for a discrete logarithm of Ry^c l times, where $1 \leq l \leq n$. For such query, $\mathcal{M}$ simply forwards Ry^c to the DL_n oracle and returns what it gets back from DL_n. Suppose that eventually S returns k_0 such that $R_0 = g^{k_0}$. Then, we can extract the values $(x_1, ..., x_n)$ by running the extractor E. Consider the list $\texttt{Transcript}$. If the j-th element of $\texttt{Transcript}$ is of form $[(g^{x_i}, m) \mapsto (u, c)]$ for some i, then $\mathcal{M}$ can compute $k_j = s - cx_i$ and k_j is indeed a discrete logarithm of R_j. At some point, $\mathcal{M}$ calculates $(k_0, k_1, ..., k_l)$. Now, for $j = l + 1$ to n, $\mathcal{M}$ can simply query DL_n on input R_j and get back k_j. Finally, $\mathcal{M}$ returns $(k_0, k_1, ..., k_n)$ and consequently, wins the one-more discrete logarithm game. Note that $\mathcal{M}$ is efficient and wins with the same probability as S solving the discrete logarithm problem. $\square$

In the similar fashion we can prove the extended version of Theorem 3.58. Let us denote $t - \mathsf{DL}_G$ to be the t-one more discrete logarithm problem.

Theorem 3.59. *Let G be a group of prime order p and generator g. If the one-more discrete logarithm holds in G then there is no fully black-box reduction from UUF-KA$[\Sigma_H]$ to $t - \mathsf{DL}_G$ for any $t \geq 0$.*

We observe that a meta-reduction $\mathcal{M}^S$ in the standard model only cares about simulating adversary interacting with the reduction S. In an external model, e.g. random oracle model, $\mathcal{M}$ also has to deal with S querying an external oracle. In case of the random oracle model, most meta-reduction in the literature consider *programmable* approach i.e. $\mathcal{M}$ simulates the random oracle queries. Usually, this comes with rewinding the adversary in order to be consistent with oracle queries. These proofs are, however, out of scope of this thesis and can be found in e.g. [40, 77, 39, 33].

Chapter 4

Conclusion

In this project we look at various ways to derive a reduction in computer science. Specifically, we show how, given a solution for one problem, we can modify it to obtain a solution to another problem. Techniques used differ between the specific areas of computer science and we considered them separately (Chapters 2 and 3).

Computability and Complexity Theory. We first focus on many-one reductions and show how they can be used to prove undecidability of a language. Simple reductions need slight modification of the output of a solver given in order to obtain another solution. More advanced ones, such as reducing 2SAT to PATH, require the knowledge of other areas of mathematics, e.g. graph theory. Later on, we move to polynomial-time reductions and defining well-known complexity classes e.g. **P** and **NP**. We present several proofs that certain problems are **NP**-complete by reducing from 3SAT (e.g. to VER-COVER or CLIQUE). These arguments require defining a graph from a 3CNF formula in a tricky way. In the last section, we give non-trivial examples of approximation-preserving reductions such as MAX-3SAT to the computational version of CLIQUE.

Cryptography. We introduce our framework of fully black-box reduction based on definitions by Reingold, Trevisan and Vadhan [70]. We then discuss a few standard reduction techniques used in the literature, namely hybrid argument, self-randomisation and information-theoretic argument.

As new results, we introduce the notion of *tight extensibility* along with a formal definition of tight fully black-box reductions. Also, we provide examples of tightly extensible primitives with respect to some more general primitives and highlight generic techniques to prove their security, namely (i) use rerandomisation property to tightly reduce the n-instance primitive to the single-instance case or (ii) modify directly the former reduction and extend it to n instances, at the same time hiding the factor n in the statistical difference. In such manners we obtain new constructions of secure signature, pseudorandom generators and public-key encryption schemes.

In the later section, we look at reductions in an external model, e.g. random oracle model. We state Conjecture 3.47 which says that a reduction in a standard model should work in any other external model. We provide a few sufficient conditions for the conjecture to hold and we also show that it is true if an additional non-uniform reduction holds.

At the end of the chapter, we discuss meta-reductions as a method of showing non-existence of a fully black-box reduction along with an extensive example involving Schnorr signature schemes.

Future work There are many areas of cryptography where we have not looked at applying tight extensibility. For example, a natural question would be if we can build tightly extensible symmetric encryption schemes or even more complicated protocols (e.g. zero-knowledge). More importantly, it is an open question whether this notion can be used in constructing more efficient and more secure primitives which could not be shown using current state-of-the-art methods.

We also leave Conjecture 3.47 as an open question. We believe that this statement is not true in general since in the external model, both adversary an implementation have access to a shared state which can somehow break the reduction but we have not yet found any concrete evidence.

Bibliography

[1] Masayuki Abe, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Tagged one-time signatures: Tight security and optimal tag size. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 312–331, Nara, Japan, February 26 – March 1, 2013. Springer, Heidelberg, Germany.

[2] Masayuki Abe, Dennis Hofheinz, Ryo Nishimaki, Miyako Ohkubo, and Jiaxin Pan. Compact structure-preserving signatures with almost tight security. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 548–580, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[3] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 2–11. IEEE, 1996.

[4] Nuttapong Attrapadung, Goichiro Hanaoka, and Shota Yamada. A framework for identity-based encryption with almost tight security. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 521–549, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[5] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Science & Business Media, 2012.

[6] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 16–25. IEEE, 1990.

[7] Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of black-box reductions, revisited. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 296–315, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.

[8] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[9] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[10] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[11] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 319–338, Grand Cayman, British West Indies, February 19–22, 2002. Springer, Heidelberg, Germany.

[12] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[13] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[14] Shai Ben-David, Benny Chor, Oded Goldreich, and Michel Luby. On the theory of average case complexity. *Journal of Computer and system Sciences*, 44(2):193–219, 1992.

[15] David Bernhard, Ngoc Khanh Nguyen, and Bogdan Warinschi. Adaptive proofs have straightline extractors (in the random oracle model). In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17:*

15th International Conference on Applied Cryptography and Network Security, volume 10355 of *Lecture Notes in Computer Science*, pages 336–353, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[16] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340, Graz, Austria, March 15–17, 2006. Springer, Heidelberg, Germany.

[17] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 408–425, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[18] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd Annual Symposium on Foundations of Computer Science*, pages 112–117, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[19] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[20] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[21] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 320–329, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[22] Daniel R. L. Brown. Breaking RSA may be as difficult as factoring. *Journal of Cryptology*, 29(1):220–241, January 2016.

[23] Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 435–460, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[24] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[25] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[26] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[27] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Computational Complexity, 1997. Proceedings., Twelfth Annual IEEE Conference on (Formerly: Structure in Complexity Theory Conference)*, pages 262–273. IEEE, 1997.

[28] Pierluigi Crescenzi, Viggo Kann, Riccardo Silvestri, and Luca Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28(5):1759–1782, 1999.

[29] Ivan Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology – EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216, Amsterdam, The Netherlands, April 13–15, 1988. Springer, Heidelberg, Germany.

[30] Ivan Damgård. On the randomness of legendre and jacobi sequences. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 163–172, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

[31] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[32] Martin Davis. *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions.* Courier Corporation, 2004.

[33] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[34] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[35] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[36] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. Mathematical logic, 2nd edn. undergraduate texts in mathematics, 1994.

[37] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[38] Marc Fischlin and Roger Fischlin. Efficient non-malleable commitment schemes. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 413–431, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[39] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 444–460, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[40] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[41] Romain Gay, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Tightly CCA-secure encryption without pairings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 1–27, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[42] Romain Gay, Dennis Hofheinz, and Lisa Kohl. Kurosawa-desmedt meets tight security. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in*

Computer Science, pages 133–160, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[43] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[44] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[45] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[46] Junqing Gong, Jie Chen, Xiaolei Dong, Zhenfu Cao, and Shaohua Tang. Extended nested dual system groups, revisited. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 133–163, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany.

[47] Iftach Haitner, Thomas Holenstein, Omer Reingold, Salil P. Vadhan, and Hoeteck Wee. Universal one-way hash functions via inaccessible entropy. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 616–637, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[48] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[49] Dennis Hofheinz. Algebraic partitioning: Fully compact and (almost) tightly secure cryptography. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 251–281, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[50] Dennis Hofheinz. Adaptive partitioning. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 489–518, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[51] Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 590–607, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[52] Dennis Hofheinz, Jessica Koch, and Christoph Striecks. Identity-based encryption with (almost) tight security in the multi-instance, multi-ciphertext setting. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 799–822, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

[53] Thomas Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 443–461, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[54] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[55] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 8–26, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

[56] David Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[57] Richard Manning Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[58] Jonathan Katz and Chiu-Yuen Koo. On constructing universal one-way hash functions from arbitrary one-way functions. Cryptology ePrint Archive, Report 2005/328, 2005. http://eprint.iacr.org/2005/328.

[59] Stephen Cole Kleene. General recursive functions of natural numbers. *Mathematische annalen*, 112(1):727–742, 1936.

[60] Stephen Cole Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1943.

[61] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[62] Benoît Libert, Marc Joye, Moti Yung, and Thomas Peters. Concise multi-challenge CCA-secure encryption and signatures with almost tight security. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 1–21, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[63] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Compactly hiding linear spans - tightly secure constant-size simulation-sound QA-NIZK proofs and applications. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 681–707, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[64] Christian Mauduit and András Sárközy. On finite pseudorandom binary sequences i: Measure of pseudorandomness, the legendre symbol. *Acta Arithmetica*, 82(4):365–377, 1997.

[65] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[66] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[67] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of psuedo-random functions. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 170–181, 1995.

[68] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany.

[69] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[70] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. Cryptology ePrint Archive, Report 2007/279, 2007. http://eprint.iacr.org/2007/279.

[71] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[72] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[73] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[74] Hartley Rogers and H Rogers. *Theory of recursive functions and effective computability*, volume 5. McGraw-Hill New York, 1967.

[75] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

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

[77] Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[78] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[79] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[80] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[81] Robert I Soare. *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets.* Springer Science & Business Media, 1999.

[82] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[83] Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum algorithms for some hidden shift problems. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 489–498, Baltimore, MD, USA, January 12–14, 2003. ACM-SIAM.

[84] Yu Yu, Xiangxue Li, and Jian Weng. Pseudorandom generators from regular one-way functions: New constructions with improved parameters. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 261–279, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.