CENTRE DE RECERCA MATEMÀTICA

Privacy-Providing Signatures and Their Applications

Author: Somayeh Heidarvand

Advisor: Jorge L. Villar

PhD Thesis

Privacy-Providing Signatures and Their Applications

by

Somayeh Heidarvand

In fulfillment of the requirement for the degree of Doctor of Philosophy

at the

Dept. of Applied Mathematics IV
Universitat Politècnica de Catalunya (UPC), Spain

Supervisor: ***Jorge L. Villar***

To ones I love

Privacy-Providing Signatures and Their Applications
by
Somayeh Heidarvand
In fulfillment of the requirement for the degree of Doctor of Philosophy at the Dept. of Applied Mathematics IV

Abstract

The notion of signatures is undoubtedly one of the most fundamental primitives introduced into cryptography so far. The numerous applications of signature schemes in the design of secure digital systems have made them a practical and essential part of almost all digital transactions. In this thesis we study new approaches for designing new schemes for a few types of signatures applied to some real scenarios, and specifically we discuss the notion of providing privacy for the information transferred digitally using digital signatures.

The notion of privacy for signatures is a very generic concept involving various types of signatures and up to now many different schemes have been proposed based on different situations and scenarios. Although it is possible to classify private signatures in different ways based on the applications, in this thesis we classify them into two main types, which are discussed below.

A signature is normally accompanied by two kinds of essential information: the identity of the signer and the signed message. Actually most signatures cannot be checked via the verification process if one of these two elements is not provided (except signatures with the recovery property that the message can be directly recovered from the signature). Due to this fact, in a very general setting, privacy providing signatures can be classified on the basis that they protect the identity of the signer from being revealed to ineligible parties, or they make the signed message private to pre-designated parties. Note that these two general types have their own subclasses; for example, privacy with respect to the signed message can be divided into two subclasses, based on the fact that it is the signer or the recipient of the signature who is interested in getting this privacy. *Blind* signatures are the best example of signatures belonging to the class of privacy providers for the signed message in benefit of the recipient. A good example for signatures that provide privacy with respect to the identity of the signer is a *ring* signature in which a group of signers can sign in such a way that a signature from any member of this group can be linked only to the group and not to the signer. A simple scenario of this kind of signature considers a two-member ring signature that has been used to construct a new kind of signatures called a *designated verifier* signature, which has recognized applications in practical schemes.

In this thesis, we investigate the notion of privacy for signatures belonging to

the second type, *i.e.,* those which provide privacy with respect to the signed message.

The first part of our work consists of one chapter in which one of the applications of blind signatures in on-line services and games is considered. In this way, we seek an efficient solution for a simple scenario in which a recipient can take a token from an issuer to be paid to an on-line service provider, without revealing anything about the service he intends to use.

The second part of the work considers the problem of formalizing the notion of *convertible non-transferable* signatures, which is addressed in two chapters. Here the notion of transferability refers to transferring the validity of a signature with respect to a given message by the recipient of the signature to a third party. The basic idea for constructing concrete schemes for such signatures is to build a scheme in which a signature can be valid with regard to any random message unless some piece of information is revealed. Non-transferable (sometimes also called private) signatures enable the signer or the recipient to decide who can verify the issued signature. It is possible to construct such signatures by computing a non-transferable signature using a non-transferable proof (in an interactive or non-interactive way) to prove its validity to the designated verifier. The concept of non-transferable signatures was introduced into cryptography by proposing *undeniable signatures*, in which after issuing the signature nobody can verify it with regard to the signed message without the cooperation of the signer. This restriction of verification is the point of difference between traditional signatures that are universally verifiable and *non-transferable* signatures. The best-known examples of non-transferable signatures are *undeniable signatures*, *designated confirmer signatures*, *directed signatures*, *universally designated verifier signatures*, *nominative signatures*, etc. In the first three signatures, it is the signer who is interested in making the signature recognizable only to himself or some pre-chosen party, while in the last two it is the recipient who makes the signature private to himself or some known party chosen by himself. Each of these signatures has its own natural application scenarios. Although these signatures are different in both nature and application, almost all of them are built from the same basic primitives: zero-knowledge proofs and commitments.

The last two chapters of this work are devoted to a wide discussion of two interesting notions and their applications to non-transferable signatures: *universally convertible directed signatures* and *fair exchange of signatures.*

While the second part of this thesis may appear to be independent of the first part, in the following chapters we will see that they are in fact closely related, in the sense that our proposal for on-line services implicitly uses a protocol for a fair exchange of signatures, which is studied in the last chapter.

In brief, this thesis contains a series of efficient protocols proposed to capture the notion of privacy in digital systems. The most important applications considered herein are: anonymous subscription schemes, fair exchange of signatures and directed signature schemes.

Acknowledgments

First and foremost, I would like to give my most sincere thanks to my advisor Jorge L. Villar for his confidence, patience and direction. Without him finishing this work was absolutely impossible for me.

My gratitude also goes to Centre de Recerca Matemàtica (CRM, Barcelona) that has supported my studies up to now.

I also have to give my thanks to department of applied mathematics and its members for their friendship and support.

Finally I want to show my gratitude to those teachers that during all these years have motivated me for continuing my studies and arrive to this point.

Contents

Chapter 1

Introduction

By extending the usage of Internet and digital communications to all aspects of life, from buying a train ticket to cash transactions between different locations, the problem of network transaction security has attracted much attention in recent decades, and as a consequence the science of information security and cryptography is progressing very fast in the area of computer sciences. The most important goal of cryptography is managing a secure communication in the presence of adversaries to ensure that, on one hand, the secret information belonging to honest parties will not be learnt by any malicious party and, on the other hand, no malicious party will be able to disrupt the transactions conducted between honest parties.

Before the modern era, cryptography was concerned solely with message confidentiality (*i.e.,* encryption); that is, conversion of messages from a comprehensible form into an incomprehensible one and back again, rendering it unreadable by interceptors or eavesdroppers who had no access to the key (namely, the key needed for decryption). In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs, secure computation, etc.

During the process of transferring a message to a recipient in a network, the message passes through many routers, some of which can be controlled by an adversary. The message can be stolen or changed by the adversary in order to impersonate a specific signer for malicious goals.

One of the most important primitives invented in the science of cryptography is the notion of *digital signatures*, which was devised to authenticate the messages transferred by a specific entity. Digital signatures were first proposed by Diffie and Hellman in [44], and currently are the objects of very practical applications in the design of secure schemes.

While authenticity of information can be provided by means of a traditional signature, real applications prove that in some situations this is not enough. In addition to authenticity of information, in many applications we also need some degree of privacy that traditional signatures are unable to provide. Digital conversations may include very private information for both sender and receiver. Although in a normal conversation privacy can be provided physically, in digital conversations this is not so easy to guarantee. The evident fact is that a secure (unforgeable) digital signature can be computed only by the owner of the public key. This can be used as proof that the signer accepts or believes in some statement. In some applications this property may become a problem; *i.e.,* no kind of privacy exists at all. In order to solve this problem, there must exist some efficient mechanism that can be applied to a normal signature in such a way that the resulting scheme not only provides authenticity of information, but also enables the signer or the receiver to hide the link between a message and its signature from the public, except for some authorized party that under special circumstances is able to construct such a link.

At first glance it may seem that providing so many properties for the signature is not compatible with the notion of efficiency. However, recent work on the design of *private signatures* for different applications have proved the contrary.

In this chapter we review the literature on signature schemes, as well as describing some of the most basic signatures. We also provide an introduction to our objectives and proposals for private signatures, which will be described in the subsequent chapters of this thesis.

1.1 Digital Signatures

In a ground-breaking paper in 1976, Diffie and Hellman proposed the notion of public key (also more generally known as asymmetric key) cryptography in which two different but mathematically related keys are used: a public key and a private key. A public key system is constructed in such a way that computing the private key from the public one is computationally infeasible. In the same paper, together with public key cryptography, the authors introduced the notion of digital signature schemes.

A conventional handwritten signature attached to a document is used to specify the person responsible for it. Although this is also the basic idea for a digital signature, the two schemes have some essential differences.

The standard definition for signature schemes is given by Goldwasser, Micali and Rivest [66], according to which a signature scheme is defined as follows:

Definition 1.1.1 (Signature Scheme) *A signature scheme is composed of the following three polynomial time algorithms:*

- *The key generation algorithm $\mathcal{G}$. On input 1^k, where k is the security parameter, the algorithm $\mathcal{G}$ produces a pair (PK, SK) of matching public and secret keys. Algorithm $\mathcal{G}$ is probabilistic.*
- *The signing algorithm **Sign**. Given a message m and a pair of matching public and secret keys (PK, SK), **Sign** produces a signature σ. The signing algorithm might be probabilistic, and in some schemes may also receive other inputs (for example, the identity of the receiver).*
- *The verification algorithm **Ver**. Given a signature σ on a message m and a public key PK, **Ver** tests if σ is a valid signature of m with respect to PK. The output of this algorithm is 1 for accept or 0 for reject.*

Remark 1.1.1 *In some multi-user systems it would be convenient that different signers share the same system parameters. Thus a separate **Setup** algorithm which on the input of 1^k produces the system parameters is considered. Now $\mathcal{G}$ receives those parameters as input.*

The first signature scheme was proposed by Rivest, Shamir and Adleman [105]. The security of their signature was based on the well-known RSA assumption. The RSA scheme is not secure by itself, since it is subject to existential forgery. In other words, it is easy to create a valid message-signature pair without asking the signer directly. Some other variations of this signature (like hashed RSA) have been proposed, which are secure in the *random oracle* model. Before giving a definition of *random oracle*, we define *hash functions*.

Definition 1.1.2 (Hash Function) *A hash function $h : \{0,1\}^* \to \{0,1\}^n$ is a function that maps any (possibly lengthy) message M to a small digest $h(M)$ of length n.*

"Ideally" a hash function has the following properties:

- The length of $h(M)$ is small enough to be used efficiently in the computations.

- The function should be a one-way function (*i.e.* it can be computed easily but cannot be inverted in polynomial time).
- It is difficult to find algebraic relations between the message and the hash (for example, if $M = M_1M_2$, then it is not the case that $h(M) = h(M_1)h(M_2)$, etc).
- It should be *collision free*; that is, it should be difficult to find two messages with the same hash value.

Generic methods exist for constructing collision free hash functions. One of these methods is to construct collision free hash functions from any claw-free collection of permutations. There are also some iterating methods for constructing such functions. For more details, the reader may refer to [61, Section 6]. One of the best known hash functions used in practice is MD5 [104]. However, its vulnerability to collision finding and birthday attacks has led research to seek replacements such as SHA-1.

1.1.1 The Importance of Hashing

Two major problems concerning signature schemes in public key systems require solution:

- They can be forged.
- If the message is too long then the signature will take a long time to be computed or the message will not be in the domain of signing function.

The common solution to both problems is to use a hash function applied to the message; that is, signing the message digest instead of signing the message itself. This method is known as **hash and sign paradigm**. A very well known and widely used example of this paradigm is DSA (Digital Signature Algorithm) which is a United States Federal Government standard, published in 1991. The signing algorithm usually uses SHA-1. A variant of DSA, ECDSA (Elliptic Curve Digital Signature Algorithm), was later introduced, which uses elliptic curve cryptography. The size of the public key of DSA is at least 1024 bits, whereas in ECDSA it is 160 bits, which is one of the advantages provided by using elliptic curves.

An ideal (and not real) function has been proposed, based on an idealization of a normal hash function. This is called random oracle and is now a very powerful tool, especially in proving security of signature schemes in the so-called random oracle model .

Definition 1.1.3 (Random Oracle Model (ROM)) *The so-called random oracle model introduced by Bellare and Rogaway [8] is a security model in which a hash function is assumed to behave randomly. More precisely, ROM is an ideal model of computation in which a hash function is modeled as a random oracle which assigns a random number to every possible query, so that the answers to different queries are independent uniform random variables.*

Although a random oracle does not exist in the real life, we trust that a well-designed hash function will behave like a random oracle.

The random oracle model provides efficiency and security, although not at the same level as those of standard provable security approach (*i.e.,* without random oracle). It is shown in [24] that security in the random oracle model does not imply security in the standard model.

The best examples of signatures secure in the random oracle model are Hashed-RSA [8], Fiat-shamir [52], Schnorr [107] and Hashed-Elgamal [101] signatures. There are also very short signatures in pairing-based groups [15] that are very efficient and prove to be secure in the random oracle model.

Gennaro, Halevi and Rabin [57] and Cramer and Shoup [36] proposed the first signature schemes that prove to be secure in the standard model, and are efficient enough for practical use. The security of both schemes is based on the strong RSA assumption. Extending the applications of pairings to the design of secure schemes in the standard model resulted in the introduction of signatures that are usually shorter (and in many cases more efficient) than the other proposals in the standard model. The best example is the so-called Boneh-Boyen signature [13].

1.2 Applications of Signatures

Although signatures can act as stand-alone primitives, they have found numerous applications at the heart of many schemes. Here we introduce two such applications.

By inventing the notion of public key cryptography, a new problem appeared in the cryptographic systems. The public key systems were vulnerable to the so-called *man-in-the-middle attack*, which can be illustrated as follows: a person enters a public key system to communicate with another person. To this end, she publishes her public key, thereby enabling her to send and receive messages from her companion through a secure method. However, a third person in the system can make independent connections with the other two and send messages between them, making them believe that they are talking directly to each other. To solve this problem, the notion of PKI certificate (Public Key Infrastructure certificate) was introduced enabling users of the system to combine their public key with a digital signature issued by some CA (certification authority), which shows that they indeed own the public keys they claim to. In other words, PKI certificate is a security mechanism for public keys.

Another early application introduced for signatures is the design of (anonymous) credential systems, introduced in [29] and further researched in [18] and [86]. In a credential system, a user can obtain access to a resource only by presenting a credential that demonstrates that he is authorized to do so. Some examples of paper-based credential are passports, driving licences, etc. A credential system is anonymous if it allows users to demonstrate such credentials without revealing any additional information. Digital signatures play an essential role in such systems.

1.3 Privacy Providing Signatures

Designing schemes that not only provide authenticity of information but also enable us to exercise control over our digitally transferred personal data is our main concern here. In other words, the major goal of this thesis is to investigate new and efficient solutions to the problem of providing privacy for signature schemes. However, depending on the party that benefits from the privacy, different notions and solutions can be introduced in this area. More specifically, a signature authenticates a message that can be chosen by the signer or the receiver, depending on the application. Once a signer signs a message, he issues a receipt that shows his interest or belief in some fact which cannot be denied later. In some cases this may be in conflict with the privacy of the signer, and it is preferable for the signer to sign the information in such a way that only the target recipient can verify its authenticity; this verification cannot therefore be transferred later to another party by the recipient.

In a different scenario, a receiver needs to authenticate some information by an authority but does not wish to reveal that information. There are many different scenarios and applications that could be discussed and investigated, but we confine ourselves to investigating the two very well-known scenarios described above.

Briefly, this work is divided into two parts. The first part consists of one chapter that addresses a simple protocol for *subscription schemes* using the well-known *blind signatures* in a such way that, after ending the protocol of issuing a signature for a receiver, the only information the signer has is that he has conducted some session with a specific receiver, but can never learn the message and signature that the receiver has obtained during that session. Furthermore, he cannot link that session with any other session of the system in which the issued signature is used. In this way we provide *privacy* of information in benefit of the receiver.

The remaining chapters investigate the notion of *privacy* of the signed information in benefit of the signer. In these chapters we first look deeply into the history of *non-transferable* or *restricted verifiable* signatures, the proposed schemes and their efficiency, and then propose our objectives and solutions.

1.4 Objectives and Proposals

This thesis is an investigation into two subjects: subscription schemes and non-transferable signatures. Below we briefly discuss these subjects and introduce our proposals.

1.4.1 Blind Signatures and Subscription Schemes

Subsequently to their invention, *blind signatures*, have had important applications in electronic cash, anonymous credential systems, subscription schemes, etc.. Roughly speaking, a blind signature enables a receiver to obtain a signature on some message such that the signer can only learn that the receiver has obtained a signature from him, but nothing about the message will be revealed. Here the identity of the receiver may (or may not) be known to the signer. A frequently used analogy to the blind signature is the physical act of enclosing a message in a special write-through-capable envelope, which is then signed and sealed by a signing agent. Thus, the signer does not view the message content, but a third party can later verify the signature and ensure that the signature is valid by checking the verification process of the underlying signature scheme. From this short introduction, it is clear that blind signatures are normally used to provide unlinkability, which prevents the signer from linking the blinded message that signs to a later un-blinded version it may be called upon to verify. In this case, the signer's response is first "un-blinded" prior to verification in such a way that the signature remains valid for the un-blinded message. This can be useful in schemes where user anonymity is required. A good application of blind signatures is in electronic voting where a voter needs to authenticate his vote by registering his personal information as an eligible person, and then obtains a signature for his vote from the registration authority, but does not wish his vote to be revealed.

Blind signatures play an essential role in e-cash systems. In traditional e-cash systems, the tradeoff between anonymity and fraud-detection is solved by hiding the identity of the user in the e-coin, and providing an additional triggering mechanism that opens this identity in case of double spending. Hence, fraud detection implies loss of anonymity. This appears to be a somewhat natural solution when universality of the e-coin is required (*i.e.*, the use of the coin is not determined at the time the coin is generated). However, much simpler protocols may suffice if we wish only to prevent payed tokens for accessing certain services from being over-used, even when user anonymity is perfectly preserved.

As it can be seen in Chapter 3, in order to solve the problem mentioned above in an efficient way, we propose a simple and efficient *Subscription Scheme*, that allows a set of users to anonymously pay for and request access to different services offered by a number of service providers. In our approach, the use of the token is completely determined at issuing time, yet this final aim remains hidden to the issuing authority. Moreover, in our proposal, fraud detection implies no loss of anonymity, since we make access tokens independent of the owner in a quite simple and efficient way. On the other hand, if different usages of the same token are allowed, these are fully traceable by the service providers, which depending on the situation, can be a desirable property. The proposed protocol can easily be implemented on the known systems.

1.4.2 Non-Transferable Signatures

While the notion of signatures was introduced to provide authenticity of the transferred data, linking the message and the identity (public key) of the signer by the other entities in the system can be dangerous. This may pose a problem in the case where the signature confirms some personal information that neither the signer nor the recipient wish to be revealed publicly. In these cases, the signature is designed in such a way that only the party needing the authenticated information can verify its authenticity, while the remaining entities obtain no useful information.

Let us assume that one needs to obtain a subsidy from the state because of some health problem. To do this one needs to present a signed report from a registered doctor. The problem is that such information is very private and the patient does not wish anyone except the advisory committee to verify the doctor's signature. In this case, authenticity is desirable, but it is also necessary to preserve the privacy of the issued signature with respect to the signed message. There are other scenarios in which the internal message contains some private information related to the signer. In any case, efficient mechanisms have been introduced for signatures that provide such privacy. Note that this property makes digital signatures more powerful than hand-written signatures, since the latter can hardly obtain this property. The idea of constructing signatures whose verification can be controlled by the signer or the recipient has found various applications in real scenarios.

The notion of non-transferability of signatures (in fact, non-transferability of the verification process) can be considered in two different ways: Firstly, an issued signature can be verified as a valid signature for a value related to the message, but the receiver of the signature cannot show any link between the signature and the message; secondly, it is possible to issue a signature which, after finishing the signing process, cannot be validated at all. However, in both cases it is possible to make such signatures verifiable just by some specific designated entity. In this regard, several different notions and applications have appeared for these signatures in the literature.

The history of non-transferable signatures began with the work of Chaum and Antwerpen [33] on *undeniable signatures*. In this scheme the signature on a message cannot be validated except with the help of the signer. The signer uses an interactive zero-knowledge proof to show that the issued signature can be opened with respect to the message by using the secret key of the signer[1]. This idea was later modified by Chaum himself to overcome the restrictions of undeniable signatures, and resulted in the invention of *designated confirmer signatures* [30]. A related notion was subsequently introduced by Jakobsson, Sako and Impagliazzo [76] on the basis of a new primitive called *convertible non-transferable signatures*. In almost all of the known variants of undeniable signatures, the notions of zero-knowledge proofs and commitment schemes play a very important role.

We have considered two notions in the area of convertible non-transferable signatures. The first one considers a variant of undeniable signatures called *directed signatures* [83]. This kind of signature has a close relation to *designated confirmer signatures*. Actually as its name indicates, the idea of a directed signature is to issue a signature by party A directed to another party B, and the verification can be performed only by A or B. Directed signatures were

[1]In that paper the authors gave the confirmation protocol without mentioning anything about zero-knowledge proofs, because at that time zero-knowledge proofs were not formalized.

later modified and changed to *universally convertible directed signatures*. The first proposal for this kind of signature was given in [81] and has proved to be secure in the random oracle model. However, we have discovered a slight flaw in the security proof. We have investigated a new generic construction that has proved to be secure in the standard model. Our solution, which is proposed in chapter 4, is based on *stateless commitments* that are a modification of *confirmer commitments* proposed in [92].

The second problem considered here is the problem of fair contract signing, which nowadays are highly practical, especially in e-commerce applications. The primary problem concerned the exchange of any kind of data in a network such that both parties could obtain each other's data. The idea was later specified to the problem of fair exchange of signatures. Fairness here means that both or none of the parties obtain each other's signature. This problem has a long history in cryptography, and different solutions have been given in the random oracle and standard models. Here again, the problem of privacy of the signatures shows its importance. All the protocols for fair contract signing work in at least 3 rounds, and during these communications there is the possibility that one party may try to cheat the other, not only from the point of view of fairness but also as regards privacy. This widely studied problem is known as *abuse-free contract signing*. Given the known results for optimal protocols for fair exchange of signatures (and specially contract signing), it is impossible to construct a protocol for optimistic fair exchange of signatures in less than three rounds, which makes such a protocol impossible for abuse-free fair exchange of signatures. There exist in the literature some protocols for normal optimistic fair exchange of signatures that are optimal, *i.e.,* in three rounds [99, 115], but they do not capture the abuse-freeness property. The first optimal solution to *abuse-free* contract signing protocol was given in [74] in three rounds, and works in the standard model. However, optimality applies only to the number of rounds and not to the complexity of the protocol. This solution is in principle interesting, but it cannot be applied efficiently in practice.

This problem is discussed in the last chapter, where we propose a new signature scheme based on pairings, the use of which will provide a highly efficient protocol for abuse-free contract signing that works in the standard model.

1.5 Road Map

In short, this thesis is an investigation into building some practical protocols for signature schemes achieving efficiency, privacy and provable security.

In the next chapter, we give an overview of the most important tools used in this work. In Chapter 3 we present our proposal for subscription schemes, and in Chapter 4 we investigate a solution for a practical and provably secure protocol for universally convertible directed signatures. Finally, in the last chapter we introduce a new protocol for abuse-free contract signing in the standard model.

Chapter 2

Preliminaries

In order to make this thesis self-contained and to enable the reader to find all the necessary information for checking the consistency of the proposed schemes, we provide the most basic information required in the following chapters. The basic concepts are: security notions for signature schemes, commitments, zero-knowledge proofs and bilinear maps.

2.1 Notation

- Let S be a set of elements. By the notation $s \overset{\$}{\leftarrow} S$ we mean that s has been chosen uniformly from the set S.
- Let A be an algorithm. By $y \leftarrow A(x)$, we mean that y has been obtained by running A on input x.
- In the case where a Turing Machine A gets access to an oracle O, we use the notation $y \leftarrow A^O(x)$ to express that A outputs y having the input x and getting access to the oracle O.
- A negligible function $\epsilon(k)$ is a function such that for every positive integer c, there exists an integer N_c, so that for every $x > N_c$, $|\epsilon(k)| < \frac{1}{x^c}$.
- Let b be a boolean function. The statement

  $$Pr[\{x_i \leftarrow A_i(y_i)\}_{1 \leq i \leq n} : b(x_n)] = \alpha$$

  means that the probability of $b(x_n)$ being TRUE is α, where x_n was obtained by running algorithms $A_1, .., A_n$ on inputs $y_1, ..., y_n$.
- Let A and B be two interactive Turing machines. By $A(x) \leftrightarrow B(y)$, we mean that A on input x enters into interaction with B on input y.

- Let A and B be two probabilistic algorithms. The notation $A \equiv B$ refers to the fact that for any algorithm $\mathcal{A}$ the following probability is a negligible function of k,

$$Pr[x_0 \leftarrow A(1^k),\, x_1 \leftarrow B(1^k),\, b \leftarrow \{0,1\},\, b' \leftarrow \mathcal{A}(x_b)\, :\, b = b']$$

  In the case where $\mathcal{A}$ is polynomial-time, we use the notation $A \overset{c}{\equiv} B$.

2.2 Security Definitions for Signature Schemes

Assume that $(\mathcal{G}, Sign, Ver)$ is a signature tuple defined in Def. 1.1.1. The standard notion of security for signature schemes is defined on the basis of a game between an adversary $\mathcal{A}$ and a signing oracle. A signing oracle is a virtual Turing machine which, on the input of message m, outputs $\sigma = Sign(m)$. The adversary can gain access to the signing oracle and obtain the signature for a set Q containing q_s messages chosen by himself from the message space. He then uses all the gathered information for forging a valid signature. The formal security notion for signature schemes is as follows:

Definition 2.2.1 (Secure Signature Schemes) *In order to be secure, a signature scheme must have the two following properties:*

- ***Correctness**: For any message chosen from the message space and any security parameter k:*

$$Pr[(PK, SK) \leftarrow \mathcal{G}(1^k), \sigma = Sign(SK, m) : Ver(PK, m, \sigma) = 1] = 1$$

- ***Existential Unforgeability under Chosen Message Attack**: On obtaining the public key of the system, the polynomial-time Turing machine $\mathcal{A}$ (called an adversary) gains access to the signing oracle and asks for the signatures on the adaptively chosen set Q of q_s messages. The game between $\mathcal{A}$ and the challenger can be written in the following way:*

  - ***Setup:** The challenger runs algorithm $\mathcal{G}$ to obtain a public key PK and a secret key SK. PK is given to $\mathcal{A}$.*
  - ***Queries:** Proceeding adaptively, $\mathcal{A}$ requests signatures on at most q_s messages of his choice, $M_1, ..., M_{q_s} \in \{0,1\}^*$, under PK.The challenger responds to each query with a signature σ_i =$Sign(SK, M_i)$.*
  - ***Output:** Eventually $\mathcal{A}$ outputs a pair (M, σ) and wins the game if:*

1. M is not any of $M_1, ..., M_{q_s}$, and
2. Ver$(PK, M, \sigma) = 1$.

A signature scheme is said to be existentially unforgeable if the probability of $\mathcal{A}$ winning the above game is negligible. Mathematically, for every negligible function ϵ:

$$Pr[(PK, SK) \leftarrow \mathcal{G}(1^k) \text{:} (m, \sigma) \leftarrow {\mathcal{A}_k}^{\textbf{\textit{Sign}}(.)}(1^k) \,\&\, Ver(m, \sigma) = 1 \,\&\, m \notin Q] < \epsilon$$

Remark 2.2.1 *This is the normal notion of security for signatures. Observe that the adversary cannot win the game by forging a new signature on a message that has been queried to the signing oracle. In many applications, this notion is sufficient, but in some scenarios it is not enough and thus a stronger version of this notion has been proposed.*

- ***Strong Existential Unforgeability under Chosen Message Attack***. Defined in an identical way to the notion defined above, except that a new signature (not given by the signing oracle) on a message queried to the signing oracle is also considered a valid forgery.

2.3 Commitment Schemes

Commitment schemes play an essential role in almost all constructions of modern cryptography, from zero-knowledge proofs to multi-party computation. In general, commitments can be interactive or non-interactive. Here we only consider non-interactive commitment schemes.

The concept of commitment schemes was introduced by Brassard, Chaum and Crepeau in [19] in the context of the problem of flipping a fair coin over the phone.

In a very simple sense, a commitment scheme is a 2-party protocol between a sender and a receiver by which the sender can choose a value from some set and commit to it such that he can no longer change his mind. Every commitment scheme has two phases: The *commit* phase in which the sender (or committer) commits to a value x, and a *reveal* phase in which the committer reveals the value x to the receiver.

Definition 2.3.1 (Commitment Scheme) *Considering P and V as the committer and receiver, respectively, a commitment scheme contains the following phases:*

- *Set-up phase: In this phase, an algorithm $\mathcal{G}$ takes the security parameter k and generates all the public parameters $PParam$.*
- *Committing phase: This is a polynomial-time computable function* commit *that takes $PParam$, a message x in the corresponding message space $\mathcal{M}$ and a k-bit random string r as input. To commit to a value x, P chooses a random r and computes $C \leftarrow \mathsf{commit}(x, r)$ and sends C to V.*
- *Revealing phase: To open the commitment C, P sends x and r to V. V checks that $C = \mathsf{commit}(x, r)$.*

Definition 2.3.2 (Secure Commitment Schemes) *A commitment scheme $(\mathcal{G}, \mathsf{commit})$ is secure if it has the following two properties:*

- *Computationally binding: For a polynomial time adversary $\mathcal{A}$, the following probability is negligible:*

$$Pr[PK \leftarrow \mathcal{G}(1^k) : (x_1, r_1, x_2, r_2) \leftarrow \mathcal{A}(PK),\ x_1 \neq x_2 \wedge \mathsf{commit}(x_1, r_1) = \mathsf{commit}(x_2, r_2)]$$

- *Computationally hiding: Assume that a polynomial-time adversary $\mathcal{A}$ chooses two messages x_0 and x_1 and sends them to P. P chooses a random bit $b \leftarrow \{0, 1\}$, a random r_b and computes $C_b = \mathsf{commit}(x_b, r_b)$. P sends C_b to $\mathcal{A}$. The following probability is negligibly greater than $1/2$:*

$$Pr[PK \leftarrow \mathcal{G}(1^k),\ b' \leftarrow \mathcal{A}(PK, C_b) : b' = b]$$

Remark 2.3.1 *The definition of hiding property actually requires the semantic security of a commitment scheme that is a property in common with encryption schemes. Both the binding and hiding properties can be defined in an information theoretic security scenario (i.e., the adversary has unbounded computational power). However, no commitment scheme can achieve both properties information theoretically.*

As described above, after committing to a message, there is no way to make sure of the integrity of the committed message except by running the revealing phase. For example, if the commitment for a message x is just by computing $g^x h^r$ for a random r and known group elements g and h, then from a commitment C, the receiver does not know if the committer has followed the protocol or has chosen a random group element C. In accordance with some applications, it is sometimes important that at the time of sending a commitment to a receiver, the committer proves that he can open the commitment via the revealing phase without giving any information about the committed message.

Definition 2.3.3 (Verifiable Commitments) *A verifiable commitment is a commitment* $\mathsf{commit}(.,.)$ *together with a proof* π *which shows knowledge of the committed message and the randomness such that* π *does not reveal any other information about the message.*

Proof π is normally a zero-knowledge proof of knowledge of the message and randomness. π can be an interactive zero-knowledge proof (see Section 2.4) or can be converted into a non-interactive proof by using the Fiat-Shamir paradigm [52]. Since the committer knows the witness (randomness) for opening the commitment, and since by revealing the witness any polynomial-time verifier can open the commitment, the problem of proving the integrity of the commitment is thus an NP problem, and therefore (as explained in Section 2.4) there exists an interactive zero-knowledge proof of commitment integrity.

Commitments and Signatures. The binding property of commitment schemes implies in some sense that the commitment C validates a message m because it cannot be opened in any other way, *i.e.,* any party can authenticate a message m by committing to that message and giving a proof of knowledge of opening the commitment to it. On the other hand, the fact that the commitment $C = \mathsf{commit}(m, r)$ cannot be linked to m without the knowledge of r makes them useful for hiding private information. In the final two chapters we discuss the relations between commitments and non-transferable signatures.

2.3.1 Trapdoor Commitments

A *trapdoor commitment* scheme, introduced in [19], is a commitment scheme related to a public key PK and a trapdoor key TK, such that if TK is known it is possible to open a commitment in a way indistinguishable from the original one. For those entities who do not know the trapdoor key, the commitment is a normal commitment scheme. In the public key systems where all the entities own some public and secret keys, the trapdoor key of the commitment is normally the secret key of a specific entity. In this way, the owner of the related key can open the commitment in any way he wishes. This property has some interesting applications that will be discussed later.

Definition 2.3.4 (Trapdoor Commitments) *We say that a commitment* *scheme $(\mathcal{G}, \mathsf{commit})$ is a trapdoor commitment scheme if there exists a key generation procedure $\mathcal{G}'$ such that*

- *$\mathcal{G}'(1^k)$ outputs a key pair (PK, TK).*
- *There exists an efficient algorithm A such that if $(PK, TK) \in \mathcal{G}'(1^k)$, then on the input $x_1, x_2 \in \mathcal{M}$ (the message domain), TK and a k-bit random r_1, it outputs a random k-bit r_2 such that $\mathsf{commit}(x_1, r_1) = \mathsf{commit}(x_2, r_2)$.*

The usage of trapdoor commitment is very common in those schemes that require some kind of simulation in proof of security. This is because if a simulator has been given the trapdoor as an input it can open the commitment in an effective way. In this way many adaptive attacks can be managed perfectly. The best known applications of trapdoor commitments are in the design of *concurrent zero-knowledge proofs* [38], *designated verifier signatures* [76], non-interactive zero-knowledge proofs in the common reference string model [71], chameleon signatures, certified emails, etc..

The Pedersen commitment is the best example of trapdoor commitments based on the discrete logarithm problem.

Example 2.3.1 *Assume that p is a large prime integer. Let G be a group of order p, and g be a generator of G. A party P chooses $x \in \mathbb{Z}_p{}^*$ and publishes $h = g^x$ as his public key. His trapdoor is x. Now for a commitment $C = g^{m_0} h^{r_0}$ on the message m_0 with randomness r_0, the owner of the trapdoor can open C for any other message m by computing $r = r_0 - (m - m_0)x^{-1} \bmod p$, Actually:*

$$C = g^{m_0} h^{r_0} = g^{m_0} h^{r + (m - m_0)x^{-1}} = g^{m_0} h^r g^{m - m_0} = g^m h^r$$

2.4 Zero-Knowledge Proofs

Progress in public key cryptography and the design of new schemes has led to new problems in cryptography. One of these problems concerns the design of anonymous authentication schemes in which a member of a group can gain access to some archive or private information accessible to the group without revealing his identity, *i.e.,* he has to prove his membership without revealing any other information. Another problem is that the behavior of one (probably cheating) party affects the behavior of the other (probably honest) one (*e.g.,* by behaving in a malicious way, a dishonest party can leak information from the other parties or prevent them from getting correct results). In order to solve this type of problem a new notion known as *zero-knowledge proofs* was introduced into cryptography.

One of the most fascinating applications of zero-knowledge proofs within cryptographic schemes is to enforce honest behavior while maintaining privacy. Roughly, the idea is that by using a zero-knowledge proof, a user is obliged to prove that his behavior is correct, *i.e.,* he is following the scheme instructions. Because of the *soundness* property, we know that the user must really act honestly in order to be able to provide a valid proof. Because of the *zero knowledge* property, we know that the user is not compromising the privacy of his secrets in the process of providing the proof. This application of zero-knowledge proofs was first used in the ground-breaking paper on secure multiparty computation by Goldreich, Micali, and Wigderson [64].

We start this section by first giving the definition of *interactive proofs*.

2.4.1 Interactive Proof Systems

While in the traditional mathematics a '*proof*' is a fixed sequence of self-evident statements leading to acceptance of the final result with probability 1, in cryptographic fashion we treat a different concept of proofs in which a '*proof*' is a sequence of 'commonly agreed upon' statements that make a verifier accept the proof with a reasonably high probability (usually less than 1). These are known as *randomized proofs* and form the basis of zero-knowledge proofs.

An *interactive proof* consists of a protocol between a *prover* and a *verifier* in which the prover tries to convince the verifier about the validity or invalidity of a statement. This process is interactive and can be regarded as a sequence of questions and answers. There are two requirements for proof systems; *completeness* and *soundness*. The soundness property asserts that the verification process cannot be tricked into accepting false statements. The completeness property asserts that a prover can always make the verifier accept the correct statements. Below we give a formal definition for interactive proof systems.

Definition 2.4.1 (Interactive Proof Systems (IP)) *An interactive proof system for a language L is a pair of two Turing machines (P, V) such that:*

- *Completeness: For every $x \in L$, the verifier V always accepts after interacting with the prover P on common input x.*
- *Soundness: For every $x \notin L$ and any prover P^*, the verifier rejects a probability of at least $\frac{1}{2}$.*

We stress that the soundness condition refers to all potential provers, whereas the completeness condition refers to the prescribed P. The probability of fooling the verifier can be reduced to 2^{-k} by repeating the proving process k times. Since we are working with polynomial time Turing machines, we can afford $k = poly(|x|)$, which makes the error probability *negligible*.

Remark 2.4.1 *The value $\frac{1}{2}$ is not a definitive bound for defining interactive proofs. Actually, any non-negligible value can be used as the bound, but if this value is too small then the protocol must be repeated many more times to arrive at a reasonable probability of not being fooled.*

As any NP statement has a witness ω, it is trivial that all NP statements have an interactive proof system (but without interaction and randomness) which is just sending the witness ω from the prover to the verifier.

Although we have given unbounded power to the prover (we have considered any prover P^*), in practice we can consider computationally bounded provers which result in *interactive arguments* where the soundness property holds only if the prover is polynomial-time. The formal definition is as follows.

Definition 2.4.2 (Interactive Arguments) *Is identical to Def. 2.4.1, with the difference that P^* is a polynomial-time Turing machine.*

In a cryptographic system that consists of various protocols for proving different statements, during a proving process a verifier can ask adaptive questions and gather as much information as possible. To make the systems secure, it is desirable to reduce the amount of information leaked by the verifier during the interaction. It is even more desirable to make this amount practically zero.

Zero-knowledge proofs were first conceived in 1985 by Goldwasser, Micali, and Rackoff [65] in a draft of "*The Knowledge Complexity of Interactive Proof-Systems*". While this landmark paper did not invent interactive proof systems, it did invent the IP hierarchy of interactive proof systems and introduced the concept of knowledge complexity, a measurement of the amount of knowledge about the proof transferred from the prover to the verifier. The authors also gave the first zero-knowledge proof for a concrete problem, that of deciding quadratic non-residues mod m.

To formalize the definition of gaining zero information by the verifier, we define the concept of *simulator*. The idea is that the adversary gains nothing if whatever can obtain by unrestricted adversarial behavior can also be obtained within essentially the same computational time by a benign behavior. The benign behavior is a computation based on the assertion itself.

Definition 2.4.3 (Perfect Zero-Knowledge Proof) *A prover strategy P is said to be perfect zero-knowledge over a language L, if for every probabilistic polynomial-time verifier V there exists a probabilistic polynomial-time algorithm S, such that*

$$(P, V)(x) \equiv S(x), \textit{ for every } x \in L$$

where $(P, V)(x)$ is the output of V after interacting with the prover P on common input x. S is the so-called simulator.

If we replace the equivalence relation above with $(P, V)(x) \overset{c}{\equiv} S(x)$, then we get the computational version of the definition of zero-knowledge proofs.

Remark 2.4.2 *Observe that the definition of zero-knowledge proof asserts that for different verifiers there exist different simulators, which inherently means that the simulator uses the V code in its program. This kind of simulation is called **non black-box** simulation. In the case where we use a universal simulator independent of the internal state of the verifier, we arrive at a stronger definition called **black-box** zero-knowledge proof.*

Definition 2.4.4 (Black-Box Zero-Knowledge Proof) *A prover strategy P is said to be black-box zero-knowledge over a language L if there exists a probabilistic polynomial-time algorithm S such that for every probabilistic polynomial-time verifier V,*

$$(P, V)(x) \overset{c}{\equiv} S(x), \textit{ for every } x \in L$$

2.4.2 Proofs of Knowledge

Roughly speaking, a *proof of knowledge* is an interactive proof in which the prover proves the *knowledge* of some data. Proofs of knowledge were defined by Feige, Fiat and Shamir [51] and by Tompa and Woll [114], and later redefined by Bellare and Goldreich [7].

Assume that the prover P wants to prove to V the knowledge of a witness ω for a statement x. The trivial way is just to release ω in clear, but it can also be done in such a way that the verifier at the end of the protocol is aware that P knows a witness for x and nothing else. This kind of proof is called *zero-knowledge proof of knowledge*, and these days plays an essential role in many cryptographic systems such as *anonymous identification schemes* and related areas.

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \overset{def}{=} \{\omega : (x,\omega) \in R\}$. ω is called a *witness* for x. Roughly speaking, we say that V is a verifier for knowledge of a witness ω for R with the input x, if for any prover P there exists a polynomial time algorithm E called an *extractor* such that by using V as black box can be derived ω such that $R(x,\omega)$ holds.

The formal definition given by Bellare and Goldreich is as follows:

Definition 2.4.5 (Proofs of Knowledge (PoK)) *A proof of knowledge for relation R with knowledge error function $\epsilon(.)$ is a two party protocol with a prover P and a verifier V having the following two properties:*

- *Completeness: If $(x,w) \in R$, the prover P who knows witness w for x succeeds in convincing the verifier V of his knowledge. More formally:*

$$Pr[P(x,w) \leftrightarrow V(x) \rightarrow accept] = 1$$

- *Extraction: requires that the success probability of a knowledge extractor E for extracting the witness, given oracle access to a possibly malicious prover, $\tilde{P}$, must be at least as high as the success probability of the prover $\tilde{P}$ in convincing the verifier. This Property guarantees that no prover that does not know the witness can succeed in convincing the verifier. More formally, there exists a polynomial-time machine E, given oracle access to $\tilde{P}$, such that for every $\tilde{P}$ and every x:*

$$\begin{aligned}&E^{\tilde{P}}(x) \in R(x) \cup \{\perp\} \wedge \\ &Pr[E^{\tilde{P}}(x) \in R(x)] \geq Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] - \epsilon(x).\end{aligned}$$

The knowledge error $\epsilon(x)$ is defined as the probability that the verifier V accepts while $\tilde{P}$ does not know a witness for x.

Example 2.4.1 (Schnorr Protocol) *The simplest proof of knowledge proposed in [107] is the zero-knowledge proof of knowledge of the discrete logarithm. In this way, a prover P proves to V that he knows x such that $u = g^x$ on the common input (g,u,p,G), where $g,u \in G \setminus \{1\}$ and G is a group of prime order p. The protocol is as follows:*

- *P chooses a random $r \in \mathbb{Z}_p$ and sends $v = g^r$ to V.*

- *V chooses a random challenge $c \in \mathbb{Z}_p$ and sends it to V.*
- *P sends $z = r + cx \mod p$ to V.*

V accepts if $g^z = vu^c$.

These types of three-move protocols are known as **Σ-protocols**. The probability of forging a valid response to a challenge is $1/p$ which is negligible if p is large enough.

Note that from two different transcripts (v, c_1, z_1) and (v, c_2, z_2) where $c_1 \neq c_2$, one can efficiently compute $x = (z_1 - z_2)(c_1 - c_2)^{-1} \mod p$. In this way, the extractor E that acts as a verifier, rewinding the prover with the same v, extracts x efficiently.

Example 2.4.2 (Chaum-Pedersen Protocol) *Another well-known protocol by Chaum and Pedersen [32] is the proof of equality of discrete logarithm. In this protocol, P proves to a verifier V that he knows $x \in \mathbb{Z}_p{}^*$ such that $u_1 = g_1^x$ and $u_2 = g_2^x$, on common input $(g_1, g_2, u_1, u_2, p, G_1, G_2)$, where $g_1, u_1 \in G_1 \backslash \{1\}$ and $g_2, u_2 \in G_2 \backslash \{1\}$ and G_1 and G_2 are groups of prime order p. The protocol works as follows.*

- *P chooses a random r and sends $v_1 = {g_1}^r$ and $v_2 = {g_2}^r$ to V.*
- *V chooses a random challenge c and sends it to V.*
- *P sends $z = r + cx$ to V.*

V accepts if ${g_1}^z = v_1{u_1}^c$ and ${g_2}^z = v_2{u_2}^c$.

2.4.3 Composition of Zero-Knowledge Proofs

Using interactive ZK proofs in practice usually requires running different instances of the protocol, which naturally gives rise to the question: Do zero-knowledge proofs preserve their zero-knowledgeness property even when various instances are run? There are three types of composition to be considered in practice: sequential, parallel and concurrent composition. Below we briefly explain these concepts, although the reader may refer to [60, Chapter 4] for more details.

In the case of sequential composition, the protocol is run many times and each instance of the protocol starts after the other one has finished. Sequential composition is the simplest way of composing proofs. To be of real practical value, zero-knowledge proofs must be at least sequentially composeable. Fortunately, this is the case with black-box zero-knowledge proofs, but the situation with parallel and current is more complicated.

In the parallel case, many instances of the protocol are invoked at the same time and the same pace; that is, we assume a synchronous model of communication and consider polynomially many executions that are totally synchronized, so that the ith message in all instances is sent at the same time.

Although there exist zero-knowledge proofs for some NP problems that are closed under parallel composition under standard intractability assumptions, in general zero-knowledge proofs are not closed under parallel composition (even the black-box zero-knowledge proofs).

Concurrent Zero-Knowledge Proofs. Concurrent composition generalizes both sequential and parallel composition. In this case, a polynomial number of instances of the protocol are invoked at arbitrary times and proceed at an arbitrary pace. Here we assume an asynchronous communication model.

Definition 2.4.6 (Black-Box Concurrent Zero-knowledge Proof) *A composed prover strategy P is said to be black-box concurrent zero-knowledge over a language L, if there exists a probabilistic polynomial-time algorithm S such that for every probabilistic polynomial-time verifier V, every l and common inputs $x_1, ..., x_l$*

$$(P, V)(x_1, ..., x_l) \overset{c}{\equiv} S(x_1, ..., x_l)$$

Canetti, Kilian, Petrank and Rosen in [25] showed that for achieving black-box concurrent zero-knowledge proofs, at least a logarithmic number of rounds are required. For non-black-box zero-knowledge proofs, [5] showed that concurrent zero-knowledge proofs can be achieved in constant rounds. However, the result is not efficient enough to be used practically in simple systems.

We now describe the method used by Damgaard [38] to preserve zero-knowledgeness under concurrent composition. In this paper it is shown that if one-way functions exist, then 3-round concurrent zero-knowledge *arguments* for all NP problems can be built in a model where a short auxiliary string with a prescribed distribution is available to the players. It is also shown that a wide range of known efficient proofs of knowledge using specialized assumptions can be modified to work in this model with no essential loss of efficiency.

We describe the proposed construction for Σ-protocols in [38].

Assume that we have a relation R and a Σ-protocol for R such that the transcript of three rounds of the protocol is (v, c, z). The prover and verifier get as common input x and the prover has a witness w such that $(x, w) \in R$. As the auxiliary string, we choose the public key of a trapdoor commitment scheme, pk, generated from a security parameter $k = |x|$. Following the notations for Σ-protocol, the new protocol proceeds as follows:

- On the input x and w, P computes v. He then chooses a random r and computes $C = \mathsf{commit}(v, r)$ and sends C to V.
- V chooses a random challenge c and sends it to P.
- P computes z and also opens the commitment by revealing r. He sends z, v and r to V.
- V accepts if v and r open C and also z is a correct answer to the challenge c.

Intuitively, the idea is that since the commitment scheme is perfectly hiding, the verifier can never guess v better than by guessing it randomly before the final step. In this way, a verifier cannot use his information obtained from an instance of the protocol to decide something in another instance of the protocol at the same time, because practically up to the end of the protocol the verifier has obtained nothing.

Remark 2.4.3 *Observe that if the commitment used in the protocol above is computationally binding, then the resulting protocol is an argument and not a proof. In other words the soundness property holds only if we assume that the prover cannot compute the secret key related to the public key of the commitment. Although an argument is a weaker notion than a proof, we safely use these protocols in our schemes because all our proposals work on the assumption that the entities of the system are polynomial-time. This is the case in the most practical cryptographic schemes that are normally based on hard assumptions.*

To prove the zero-knowledgeness of the scheme formally, we make a simulator S that is given the secret key of the trapdoor commitment scheme as the auxiliary string, and simulates the protocol given an arbitrary verifier V^*. The simulator works as follows:

- Generates pk with known trapdoor sk and gives x, pk to V^*.

- Computes a commitment $C = \mathsf{commit}(\alpha, \rho)$ for a random value α using the randomness ρ, sends C to V^* and gets c back.
- Runs the honest verifier simulator on input c to get an accepting conversation (v, c, z) in the original protocol. The simulator uses the trapdoor (as described in Def. 2.3.1) to compute r such that $C = \mathsf{commit}(v, r)$. It sends v, r and z to V^*.

The simulation works based on the hiding and trapdoor properties of the commitment scheme. The simulation is perfect and the simulator does not need to rewind the verifier. So the protocol is concurrently zero-knowledge. To prove that the proposed protocol is a proof of knowledge with error probability $\kappa()$, we can show that the protocol satisfies the definition when we choose $\kappa(x) = 1/q(x)$ for any $q()$. Thus, the knowledge error is smaller than the polynomial and so is negligible. The proof is highly technical and the interested reader may refer to [38].

As we need a concurrent version of *Chaum-Pedersen* proof in the following chapters, we describe here the protocol in detail.

Example 2.4.3 (Concurrent Version of Chaum-Pedersen ZK Proof) *We describe a concurrent ZK proof of knowledge, which is an instance of the construction given by Damgård in [38] applied to Chaum and Pedersen's Proof of Knowledge [32] of $x \in \mathbb{Z}_p{}^*$, such that $u_1 = g_1^x$ and $u_2 = g_2^x$, on common input $(g_1, g_2, u_1, u_2, p, G_1, G_2)$, where $g_1, u_1 \in G_1 \setminus \{1\}$ and $g_2, u_2 \in G_2 \setminus \{1\}$ and G_1 and G_2 are groups of prime order p.*

To make the protocol concurrent zero-knowledge, we make use of Pedersen's trapdoor commitments in [98]. To commit to a value $m \in \mathbb{Z}_p$, we choose a random $\alpha \in \mathbb{Z}_p$ and compute the commitment $\mathsf{commit}(m, \alpha) = g^\alpha h^m$, where g, h are two generators of a group G of prime size q. The trapdoor of the commitment is $t \in \mathbb{Z}_p$ such that $h = g^t$. Obviously, the trapdoor allows any commitment c to be opened for any m' if we also know one opening $C = \mathsf{commit}(m, \alpha)$, as $\mathsf{commit}(m', \alpha - t(m' - m)) = \mathsf{commit}(m, \alpha)$.

The concurrent version of the ZK proof works as follows: On common input $(g_1, g_2, u_1, u_2, p, G, g, h, G_1, G_2, H)$, where h is the auxiliary string and $H : G_1 \times G_2 \to \mathbb{Z}_p$ is a collision resistant hash function, the prover computes r, v_1, v_2 as in the Chaum-Pedersen proof (described in Example 2.4.2) and sends a commitment $C = \mathsf{commit}(H(v_1, v_2), \alpha)$ to the verifier. The verifier then sends a challenge c, and the prover responds by sending (α, v_1, v_2, z) to the verifier. Finally, the verifier checks the validity of the commitment and the verification equations of the Chaum-Pedersen proof.

To prove the zero-knowledgeness property, the simulator, knowing the trapdoor of the commitments, just sends a commitment $C = \mathsf{commit}(m, \alpha)$ that he can open in any possible way. Then, after receiving the challenge, the simulator computes $z, v_1 = g_1^z u_1^{-c}, v_2 = g_2^z u_2^{-c}$ and α' such that the commitment opens correctly to $m' = H(v_1, v_2)$. Hence, the verifier is allowed to choose the challenge in the whole set $\mathbb{Z}_p$, and only one round of the protocol suffices to achieve soundness with a probability of cheating equal to $1/p$.

Showing a knowledge extractor is not so trivial, but the reader may find details in [38]. Basically, after rewinding an honest prover for a polynomial number of times, with high probability one may obtain two conversations with the same v_1, v_2 but two different challenges c, c'. The extractor then computes $x = (z' - z)/(c' - c)$, where z, z' are the responses given by the prover to the challenges c, c', respectively.

2.5 Pairing-Based Cryptography

The field of pairing-based cryptography has made great progress in recent years. The most important property that pairings provide is *bilinearity*, which enriches the algebraic structure of the one-way function. As an example, a bilinear map between two groups allows for new cryptographic schemes based on the reduction of one problem in one group to a different and a usually easier problem in the other group. This idea was first proposed in [12]. In this paper the authors define the notion of *gap group*, where the decisional *Diffie-Hellman* problem is easy but the computational *Diffie-Hellman* problem is hard (Assumption 2.5.1).

Although the known implementations of pairing-based systems have been carried out on the very few known pairings, such as Tate and Weil pairings, which are defined over elliptic curves, from a theoretic point of view many schemes have been introduced based on abstract bilinear maps. One of the advantages of using bilinear maps on elliptic curves is that they usually reduce the length of the transferred data, because some of the algorithms that can break the underlying assumption in a system in sub-exponential time can only break it in exponential time when the underlying group is an elliptic curve. This fact enables us to use a lower security parameter and thus smaller groups.

2.5.1 Bilinear Maps

Consider three cyclic groups G_1, G_2 and G_T of the same prime order q. A bilinear map from $G_1 \times G_2$ to G_T is a function $e : G_1 \times G_2 \to G_T$ such that it has the following properties:

- **Bilinearity**: $\forall \{u, v, a, b\} \, : \, u \in G_1, v \in G_2, a, b \in {\mathbb{Z}_q}^*$

$$e(u^a, v^b) = e(u, v)^{ab}$$

- **Non-Degeneracy**: e is not a constant map; *i.e.*, if g_1 and g_2 are generators of G_1 and G_2 then $e(g_1, g_2)$ generates G_T.
- **Computability**: For all $u \in G_1$ and $v \in G_2$, $e(u, v)$ is efficiently computable.

Note that G_1 and G_2 can be different. However, for simplicity we work on the case that $G_1 = G_2 = G$. The commonly used classes of pairings are pairings of types 1 (where $G_1 = G_2$), 2 (where $G_1 \neq G_2$ and there exists an isomorphism $\psi : G_2 \to G_1$) and 3 (where $G_1 \neq G_2$ and no isomorphism exists $\psi : G_2 \to G_1$). The assumptions introduced in groups of the first type are called *symmetric* assumptions, while the others are called *asymmetric* assumptions. Note that an assumption holding in one case does not necessarily hold in the other.

2.5.2 Complexity

It is important for the cryptographic approach to decide which problems are hard to solve in pairing groups.

Theorem 2.5.1 *The discrete logarithm problem in G is no harder than the discrete logarithm problem in G_T.*

Proof. The idea is to reduce the solution of the DL problem in G_T to solving DL in G. Observe that to compute the discrete logarithm of $Q = P^a$ it is enough to compute the discrete logarithm of $e(P, Q)$ in base of $e(P, P)$.

Remark 2.5.1 *It is necessary to pay attention to some specific curves which make the system vulnerable to some attacks. For example, it is known that the DL problem in the divisor class group of a curve over $\mathbb{F}_q$ can be transformed into a DL problem in some finite field extension $\mathbb{F}_{q^k}$, and in the case that k is small, the DL problem will be efficiently solved. Menezes, Okamoto and Vanstone [88] showed that for supersingular curves one has $k \leq 6$. This causes cryptographers to seek suitable elliptic curves for real implementations.*

Definition 2.5.1 *Let G be a group of order q with generator g. The advantage of a probabilistic algorithm $\mathcal{A}$ in solving the decisional Diffie-Hellman problem is*

$$\mathsf{ADV}_{\mathcal{A},G}{}^{DDH} = |Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]|$$

where a, b and c are drawn from $\mathbb{Z}_q$ uniformly and the probability is taken over the choices of a, b and c and the coin flips of $\mathcal{A}$.

The DDH problem is said to be hard in G if the above advantage is negligible.

The computational version of the above definition can be similarly defined, whereby the adversary tries to compute g^{ab} from g^a and g^b.

Assumption 2.5.1 *The Decisional Diffie-Hellman(DDH) problem is easy in G. However, the computational problem can still be hard.*

Remark 2.5.2 *In the case $G_1 \neq G_2$ for special curves, DDH may remain hard, but this is only possible for the type 3 groups.*

The following assumptions, called *decisional bilinear diffie-hellman* and *decisional tripartite diffie-hellman* assumptions, are very well-known and practical in pairing groups.

Definition 2.5.2 (Bilinear Group System) *Assume that $G = \langle g \rangle$ and G_T are groups (in multiplicative notation) of prime order q and $e : G \times G \to G_T$ is an efficiently-computable non-degenerated bilinear map. Then (q, G, G_T, g, e) is called a bilinear group system.*

Assumption 2.5.2 ((Symmetric) DBDH Assumption) *Let (q, G, G_T, g, e) be a bilinear group system. The two probability distributions $D_{DBDH} = (g^\alpha, g^\beta, g^\gamma, e(g, g)^{\alpha\beta\gamma})$ and $D_{random} = (g^\alpha, g^\beta, g^\gamma, \lambda)$ for random $\alpha, \beta, \gamma \in \mathbb{Z}_q$ and random $\lambda \in G_T$ are polynomially indistinguishable.*

Assumption 2.5.3 ((Symmetric) DTDH Assumption [81]) *Let (q, G, G_T, g, e) be a bilinear group system. The two probability distributions $D_{DTDH} = (g^\alpha, g^\beta, g^\gamma, g^{\alpha\beta\gamma})$ and $D_{random} = (g^\alpha, g^\beta, g^\gamma, g^\delta)$ for random $\alpha, \beta, \gamma, \delta \in \mathbb{Z}_q$ are polynomially indistinguishable.*

Note that the DTDH assumption is a version of the DDH assumption that holds even in pairing groups, and thus makes sense in any group.

2.5.3 Applications

These days, pairings have found a large variety of applications to cryptography including signature schemes, aggregate signatures, verifiable encrypted signatures, identity based encryptions, etc. Below we mention some of the interesting applications of pairings in cryptography.

1. **Key Exchange Protocols.** Here we give a very simple example of a three party key agreement protocol due to Joux [78] which is the first positive use of pairings:

   Let (q, G, G_T, g, e) be a bilinear group system.

   - Alice picks $a \in \mathbb{Z}_q$, Bob picks $b \in \mathbb{Z}_q$ and Carol picks $c \in \mathbb{Z}_q$, all randomly.
   - Alice, Bob and Carol broadcast g^a, g^b and g^c, respectively.
   - Alice computes $e(g^b, g^c)^a = e(g, g)^{abc}$, Bob computes $e(g^a, g^c)^b = e(g, g)^{abc}$ and Carol computes $e(g^a, g^b)^c = e(g, g)^{abc}$.

   The common key is $e(g, g)^{abc}$. This is a three party version of the Diffie-Hellman key exchange protocol.

2. **BLS Signature.** Another application of the algebraic structure of pairings was introduced by Boneh, Lynn and Shacham [15] and is known as the BLS signature. Assume that (q, G, G_T, g, e) is a bilinear system. The BlS signature contains the following algorithms.

   - **Key Generation**. A user U chooses a random $x \in \mathbb{Z}_q$ and publishes $Y = g^x$ as his public key.
   - **Sign**. To sign a message m, U computes $\sigma = H(m)^x$, where H is a random oracle.
   - **Ver**. To verify a pair (m, σ), a verifier V checks that $e(g, \sigma) = e(y, H(m))$

3. **Pairing-based Non-interactive Proofs.** Another recently introduced applications of pairings is the construction of efficient schemes for NIZPs in the standard model. The first proposals for NIZPs in the standard model were introduced in [69] and [70] and consist of proof systems for proving circuit satisfiability, and thus by the Cook–Levin theorem allowing membership to be proven for every language in NP. Although the size of the common reference string and the proofs in these works are relatively small, transforming a statement into a boolean circuit causes

a considerable overhead.

Pairings have recently helped to make these very expensive schemes more efficient. The basic idea is to hide the secret value in a homomorphic commitment and use pairings for non-interactive verifications. Based on this idea, the problem was solved in [71] in such a way that there is no need to reduce the statement to a circuit, but rather to work directly in a group with a bilinear map. The authors have proposed efficient solutions based on *sub-group hiding* [69], *decisional linear* [70] and *external Diffie-Hellman* [109] assumptions.

Chapter 3

Anonymous Subscription Schemes

In the last decades, Internet transactions have found different applications specially in the transactions of money between different locations. E-cash systems were introduced in cryptography as a solution to the problem of transaction of digital money. This chapter has a deep look to a problem related to anonymity in the Internet transactions and investigates efficient solutions for it.

Anonymity in internet transactions is essential to prevent critical personal data to be inadvertently leaked to unwanted people. As an example, an eavesdropper could learn some private information about health, consumer habits or preferences of people if their identity is revealed during internet transactions. However, anonymity could be abused to make criminal acts unlinkable to individuals. To prevent such abuse, in some e-cash schemes the identity of a user can be opened under very special circumstances (e.g., double spending of an electronic ticket).

In traditional e-coins, the tradeoff between anonymity and fraud-detection (i.e., double spending or overspending) is solved by hiding the identity of the user into the coin and providing an additional triggering mechanism that opens this identity in case of double spending. Hence, fraud detection implies loss of anonymity. This seems to be a somewhat natural solution when universality of the coin is required (i.e., the use of the coin is not determined at the time the coin is generated). Double spending can only be detected by the issuer (bank). Otherwise, all merchants would have to collaborate to check for the freshness of every coin.

Nevertheless, in some real life environments (e.g., online games) the potential damage produced by a dishonest user is very limited, and it is often enough to guarantee some sort of "cloning detection" to prevent overuse of credit vouchers, without providing any identity-escrow mechanism. Indeed, this relaxation allows for simpler and more efficient payment schemes for many concrete applications.

Bearing this in mind, in this chapter we define *subscription schemes* which allow a set of users to buy access to different services. This access is payed to an issuing authority that dispenses *connection tokens*, which usage is completely determined at issuing time. More precisely, tokens are differentiated in terms of their service providers and validity period (so, time is divided into different time slots). This implies that each service provider can locally and non-interactively take control on the different tokens spent in each time slot, thus rejecting any attempt of token misuse (including overuse, incorrect service provider or incorrect time slot).

Following this approach fraud-detection does not require identification of the owner, and then no loss of anonymity is implied. This will allow for a design in which tokens are independent of any private information identifying the owner in a quite simple and efficient way.

Note that it is reasonable to expect that some information about the user identity will be learned by the issuer agency (as indeed payment is a part of the token issuing protocol). However, it is our goal that this information cannot be possibly linked either to the token itself or to the service the token is intended for. Thus, we will impose that the view of the issuing authority must be independent of the value of the issued token. As a result, no collusion of the issuer agency and one or more service providers will learn any information about the token owner.

Furthermore, payment is organized in such a way that at the end of a time slot, every service provider sends the collected tokens to the issuer to be payed for the offered service. Unused tokens can similarly be refunded to the users upon request. Thus, the subscription scheme must ensure that no collusion of users and service providers can forge new valid tokens (not issued by the agency) and they will furthermore not succeed in getting payed more than once for each issued token.

Related Work. Anonymity in commercial transactions (also known in some papers as untraceability) has been firstly introduced by Chaum in the seminal paper on blind signatures [28]. Chaum's *electronic coins* were defined as a value together with a signature from the issuing bank, which was to be withdrawn and spent by the user and subsequently deposited by the shop in the bank (thus, correctness of payment is checked on-line). In this setting, blind signature schemes are introduced as a cryptographic tool to allow the bank constructing electronic coins in such a way that he will not be able to recognize them later. In this way he will not be able to link a coin with the user that requested it or identify two payments of the same user.

Subsequent works aimed at electronic coins that could be used in an off-line setting, namely, the shop will only deposit coins every now and then, and if a client payed with the same coin twice, his identity would be revealed. Several solutions based on RSA and Schnorr signatures can be found in [31, 17, 50]. This type of work is essentially based on blind signatures which hide a certain value linked to the message they "certify" (identity of the one requesting the signature, intended future use of the signature, etc.) are called *restrictive blind signatures* and have been extensively studied (see, for instance [107, 35]).

For some applications total anonymity of electronic cash is not desirable (for instance, it could be used as an effective method for money laundering). Thus, several proposals for *partial* or *revokable anonymity* can be found in the literature (among others, see [22, 113, 77]). These schemes provide anonymity that may however be revoked by a Trusted Third Party if certain circumstances are met. Blind signatures are again one of the main building blocks of these schemes. Note that our setting here is less demanding, as we do not want to disclose the identity of fraudulent users, but only to block their allowance to a certain service.

Another related topic is that of *digital pseudonyms*, initiated by Chaum [27]. A digital pseudonym is a public verification key for a signature scheme, which corresponding private key is held by an anonymous signer. A list of pseudonymes is created by an authority which decides the uses of these pseudonymes, validity period, etc. Thus, an individual might be known to a service provider only by a pseudonym that appears in a list of acceptable clients. On the other hand, the authority will not be able to trace the use of the pseudonymes in requesting provided services. This double anonymity can however be revoked under special circumstances. This application scenario is indeed very similar to ours; however, the solution we provide here is more efficient. In particular, it requires less issuer–service provider communication: the validity of access tokens (our "pseudonymes") can be checked by the server provider with only the authorities public key (he does not need to request a list of valid pseudonyms). Moreover, it has better traceability properties, as our issuer does not know the token/pseudonym he signed.

Closer to our scenario are some recent papers which deal with how to prove group membership in an anonymous way. Some solutions in the literature have been proposed in the context of group and ring signatures (see, for instance [26, 53]). However, in that scenario no protection against double-use of access credentials has, as far as we know, been considered. Damgård, Dupont and Pedersen [39] introduced at Eurocrypt 2006 so called *unclonable group identification schemes*; which allow an honest participant to anonymously and unlinkably authenticate himself as a member of a designated group. Moreover, such scheme discloses the identity of any participant that "clones" himself and connects twice with the same keying material. In their paper, Damgård *et al.* give a generic construction for realizing the scheme, which is however rather inefficient. They also describe a concrete instantiation that employs some new zero-knowledge techniques for proving given g^x in a group of prime order, that x was chosen pseudorandomly computed from a committed secret key.; even thought the gain in efficiency is significant, still the resulting scheme is rather expensive computationally.

Subsequent work of Camenisch, Hohenberger, Kohlweiss, Lysyanskaya and Meyerovich [21] considers a slightly different goal; each participant should obtain, upon connection with an issuer/authority, enough information to connect e times to a service (anonymously and unlinkably). Again, overusing this private connection information leads to the identification of the fraudulent participant. This solution, though more practical than that of Damgård *et al.*, is still rather costly, in particular if we look at the number of operations a user has to preform each time he connects. Here we provide a quite simple subscription scheme for the same scenario in which connections are computationally very cheap for users. However, our proposal is completely traceable in a multiple access per token environment (i.e., the service provider knows which accesses correspond to the same token in a fixed time slot). Despite this drawback, our protocol suits many real life application scenarios, such as online games and online service subscriptions, and could also be applied to audience controls in metering schemes.

Our Contributions. In the following sections of this chapter, we will propose a new simple scheme for handling access policies to on-line services which achieves:

- Perfect anonymity for the user with respect to the services which has purchased (even when some service providers and the issuer collude).

- Unforgeability of tokens by a collusion of dishonest users and service providers.
- Undeniability of purchased services; valid access tokens cannot be repudiated by the issuing authority.
- Efficient management of tokens due to the independence of services and time slots.
- Efficient access to services for users.
- Very flexible access management for the service provider.

As mentioned already in the introduction, the main limitation of our scheme resides in the complete traceability of the different accesses with the same token to the same service, however in some applications this can be a desirable property.

Before introducing our solution, we describe some basic notions related to the work.

3.1 Blind Signature Schemes

Although the notion of blind signatures was first introduced by Chaum in [28], the security of blind signatures schemes was formalized in [100, 79]. Here we follow the notation and terminology of [79], however, the definition of blindness bellow is taken from [94][1].

Definition 3.1.1 (Blind Digital Signatures) *A blind signature scheme is a four-tuple* $\texttt{BlindSig} = (Signer, User, Gen, Verify)$ *where*

- Gen *and* $Verify$ *are polynomial time algorithms. Moreover:*
  - Gen*, the key generation algorithm, is a probabilistic algorithm that takes as an input an encoding of the security parameter* k *and outputs a pair* (pk, sk) *of public and secret keys.*
  - $Verify$*, the verification algorithm, is a deterministic algorithm, which on input a triplet* (pk, m, σ) *outputs one bit meaning* accept/reject*.*

[1]Basically Okamoto modified a previous definition by allowing the adversary to freely choose the public key and also to act dishonestly during `BlindSign` executions, without being forced to abort the game. Actually, Okamoto's definition is flawed and we present here the corrected version of it from [95]

- *$Signer$ and $User$ are both interactive polynomially-bounded probabilistic Turing machines, each having the following (separate) tapes: read-only input tape, write-only output tape, a read/write work tape, a read-only random tape and two communication tapes, a read-only and a write-only tape. The $User$ and $Signer$ engage in an interactive protocol for some polynomial number of rounds. At this,*
  - *$Signer$ takes as an input the key pair (pk, sk), his output will be a single bit, meaning* completed/not-completed*.*
  - *$User$ takes as an input the public key pk together with a message m (of polynomial length in the security parameter). His output will be an error message $\perp$ or a signature $\sigma(m)$.*

  It must be the case that if both $User$ and $Signer$ follow the protocol specification, then $Signer$ always outputs completed*, and the output $\sigma(m)$ $User$ is always* accepted *by $Verify$; i.e., $Verify(pk, m, \sigma(m)) = 1$.*

The following two properties must be achieved in order to consider a Blind Digital Signature scheme secure:

Definition 3.1.2 (Unforgeability) *Let $\mathcal{A}$ be a pptm (probabilistic polynomial Turing machine) adversary against a blind signature scheme* ***BlindSig*** *defined as above. Let $\mathcal{C}$ be a pptm challenger and consider the following game played by $\mathcal{A}$ and C :*

- *$\mathcal{C}$ runs the key generation algorithm $\mathcal{K}$ on input 1^k and retrieves a key pair (pk, sk), and forwards the public key pk to $\mathcal{A}$.*
- *$\mathcal{A}$ engages in L adaptive, parallel and arbitrarily interleaved interactive protocols with corresponding $\mathcal{C}$ acting as an honest $Signer$, all with input (pk, sk). At this, L is decided adaptively by $\mathcal{A}$, but it is polynomial in k. Let l be the number of the above executions which $\mathcal{C}$ accepted as valid.*
- *$\mathcal{A}$ outputs a collection of j pairs $(m_i, \sigma(m_i))$, where all messages m_i in the list are different, and so that each pair is accepted by $Verify$ on input pk.*

Then, ***BlindSig*** *is* non-forgeable *if for any probabilistic polynomial-time adversary $\mathcal{A}$, the probability, taken over coin-flips of Gen, $\mathcal{A}$ and $\mathcal{C}$, that $j > l$ is negligible in k.*

The above definition corresponds to the notion of security against "one-more" forgery considering parallel attacks from Pointcheval and Stern (see, for instance, [102]).

Definition 3.1.3 (Blindness Property) *Let $\mathcal{A}$ be a pptm adversary against a blind signature scheme* ***`BlindSig`*** *defined as above. Let $\mathcal{C}$ be a pttm challenger and consider the following game played by $\mathcal{A}$ who controls the signer and $\mathcal{C}$*

- *$\mathcal{C}$ generates the system parameters of the blind signature scheme which he forwards to $\mathcal{A}$.*
- *$\mathcal{A}$ chooses a valid[2] public key, pk_{BSig}, and two different messages m_0 and m_1 to be signed, and sends all to $\mathcal{C}$.*
- *Now $\mathcal{C}$ flips a fair coin b and starts two parallel and arbitrary interleaved instances of* ***`BlindSig`*** *on m_b and m_{1-b} with $\mathcal{A}$, notifying $\mathcal{A}$ that the former is the target one.*
- *At the end of the protocols, if $\mathcal{C}$ gets two valid blind signatures: σ_0 on m_0 and σ_1 on m_1, then $\mathcal{C}$ sends (σ_0, σ_1) to $\mathcal{A}$. Otherwise, if some of the protocols have been aborted or some of the signatures are not valid, $\mathcal{C}$ sends $\perp$ to $\mathcal{A}$.*
- *Finally, $\mathcal{A}$ ends the game by outputting a guess bit b'.*

Then the corresponding signature scheme fulfills the blindness property if the probability, taken over the choice of b, coin flips of Gen, $\mathcal{A}$ and $\mathcal{C}$ that $b = \hat{b}$ is bounded by

$$\frac{1}{2} + \varepsilon(k),$$

for some negligible function ε.

3.1.1 Hashed RSA Blind Signature

The first RSA blind signature was introduced in [27] but because the normal RSA signature is not a secure signature we use the hashed RSA to construct a blind signature scheme. We give a short description of the hashed RSA blind signature that is secure in the random oracle model.

Assume that (N, e, d) are the parameters of the RSA signature as described before. To get a blind signature on the message m, a receiver chooses a random value r relatively prime to N and computes $M = H(m)r^e$ where H is a hash function acting as a random oracle. The receiver sends M to the signer. The signer signs M as $\sigma' = M^d = H(m)^d r$. To get the signature, the receiver computes $\sigma = \sigma' r^{-1}$ and checks if $\sigma^e = H(m)$. If the equation holds, the receiver keeps σ as a valid signature on m.

[2]Here 'valid' means one of the possible outputs of `IAKeyGen`.

Because of it simplicity Hashed RSA blind signature is very practical to use in the real systems.

3.1.2 Electronic Cash

For the first time, anonymous e-cash was introduced in [31]. In this paper blind signatures have been applied to get anonymity for the payer. On the other hand they manage to provide a new mechanism that reveals the identity of the payer in the case of double spending in the scenario that the bank is off-line. Actually if the bank is supposed to be on-line, then there is no need to give this mechanism, because every time that a user pays a coin to a shop, the shop contacts the bank. In the cases that the coin is invalid or has been paid before the bank will discover it immediately. Below we describe the proposed scheme of [31] briefly. For more details, the reader can refer to the paper.

- **Setup.**The bank initially publishes an RSA modulus N such that its factorization is kept secret and $\phi(N)$ has no small odd factors. Let f and g be two two-variable collision resistant hash functions such that it is infeasible to find two inputs that map to the same point. Assume that f behaves like a random oracle and also fixing the first variable, g behaves like a one-to-one map.

  Alice has a bank account u and the bank keeps a counter v associated with it.

- **Getting an E-Coin.** To get an electronic coin, Alice finishes the following protocol with the bank:

  1. Alice chooses a_i, c_i, d_i and r_i, $1 \leq i \leq k$, independently and uniformly from the residues modulo N.
  2. Alice forms and sends to bank, k blinded candidates

     $$B_i = {r_i}^3 f(x_i, y_i) \, mod \, N, \; 1 \leq i \leq k,$$

     where

     $$x_i = g(a_i, c_i), \; y_i = g(a_i \oplus (u||(v+i)), d_i).$$

  3. The bank chooses a random subset of $\frac{k}{2}$ blinded candidate indices $R = \{i_j, \; 1 \leq i_j \leq k, \; 1 \leq j \leq k/2\}$ and transmits it to Alice.

4. Alice reveals the values a_i, c_i, d_i and r_i for all $i \in R$ and the bank checks the validity of the received information having u and v. W.L.O.G assume that $R = \{k/2+1, ..., k\}$, then bank gives

$$E = \prod_{1 \le i \le k/2} {B_i}^{1/3} \, mod \, N$$

to Alice and charges her account one dollar. The bank also increments Alice's counter v by k.

5. Alice extract the coins by computing

$$C = E^3 \prod_{1 \le i \le k/2} {r_i}^{-1} = \prod_{1 \le i \le k/2} f(x_i, y_i)^{1/3} \, mod \, N.$$

Alice reindexes the candidates in C to be lexicographic on their representation: $f(x_1, y_1) \le f(x_2, y_2) \le ... \le f(x_{k/2}, y_{k/2})$. She also increments her counter v by k.

- **Spending an E-Coin** To spend a one dollar coin to Bob, Alice and Bob proceed as follows:

  1. Alice sends C to Bob.
  2. Bob chooses a random binary string $z_1, ..., z_{k/2}$ and sends it to Alice.
  3. Alice responds to the challenge string as follows:
     1. If $z_i = 1$, then Alice sends Bob a_i, c_i and y_i.
     2. If $z_i = 0$, then Alice sends Bob $x_i, a_i \oplus (u||(v+i))$ and d_i.
  4. Bob verifies that C is well-formed and the answers fit C.
  5. Bob later sends C and Alice's responses to the bank which verifies them and credits bob's account.

  The bank stores all the received information.

In the case that Alice tries to spend a coin C twice, she will be traced with high probability: with high probability, two different shopkeepers will send complementary binary values for at least one z_i for which, B_i was in the proper form. The bank can search its records to ensure that C has not been paid before. In the case of double spending, with high probability, the bank has both a_i and $a_i \oplus (u||(v+i))$ which leads to revealing u and v that reveals the identity of Alice.

Following the basic ideas of this paper, so many proposals have been appeared in the literature that their security are based on different assumptions.

3.2 Subscription Schemes

Here we give a formal description of what we call a *Subscription Scheme.*

3.2.1 Involved Entities

Our *subscription scheme* involves different entities, modelled by probabilistic polynomial-time interactive Turing machines:

- a finite set of *service providers*, $\mathcal{SP} = \{\mathsf{SP}_1, \ldots, \mathsf{SP}_n\}$, each of them offering a concrete service managed according to their own policy. This policy must specify the duration of subscriptions to this service, using as time reference different time slots and possibly, also session identifiers distinguishing different sessions per slot. We assume this providers will never deny access upon request with a valid token.[3]
- a finite set of *users*, $\mathcal{U} = \{\mathsf{U}_1, \ldots, \mathsf{U}_n\}$, which may subscribe to any of the services above,
- an *issuing authority* IA which role is to publish and certify all information about the service providers, and dispense subscription tokens to users upon request (and payment).
- a *trusted third party* TP which will be invoked by a user in case he wants to be refunded for an unused token. This trusted party can also be used to guarantee the fairness of all paying protocols in the system. We may assume the TP is connected with each user via a private and authentic channel.

3.2.2 The Scheme

Now, the interaction between these entities is specified by the following algorithms and protocols, which define the *subscription scheme.* Here, for simplicity we assume that every token allows the user for a single access to a service. For other access policies (e.g., multiple accesses with the same token) see Section 3.6.

[3]This is quite a natural semi-honesty assumption, as it is in their own interest to gain customer loyalty. See Section 3.6 for some ways to remove this assumption.

- **Start-up algorithms:** They are only run during a set up phase, and provide all involved entities, on the input of the security parameter and possibly some other system parameters, with all the public/private key pairs needed for the scheme.
  - `IAKeyGen`. Run once by the IA; it outputs all public key/secret key pairs needed for the protocol.
  - `SPKeyGen`. Run once by each service provider SP_j; it outputs all public key/secret key pairs needed for the protocol.
  - `PublishCatalogue`. Run once by the IA; on the input of the public keys and service information of the service providers it outputs an authenticated catalogue (e.g., signed by the IA), including at least all service providers' identifiers and public keys, as well as the service descriptions and conditions of use.

- **Subscription Protocols:** We assume that the catalogue of services and the current time slot are always included as common inputs to all protocols. We also assume that all entities are supposed to be able to verify the authenticity of all public keys. Actually, only the public key of the IA needs to be certified externally. [4]
  - `VerifyToken`. Run by any party, on the input of a token x a service provider identifier SP and a time slot t, it outputs a single bit indicating the validity of x. This auxiliary algorithm will be used in the protocols described below.
  - `ObtainToken`. This protocol is run by a user U and the issuing authority IA. User's private input will include a service provider's name SP and a time slot identifier t' (not necessarily the current one). As private output, U will either receive an error message $\perp$ or a valid token to access the service offered by SP on time slot t', according to the service provider's particular access policy. To ensure this, U might execute the `VerifyToken` at some point during the protocol execution.

    Typically, an optimistic fair e-cash protocol (that is a fair exchange of signatures and will be discussed in the last chapter) is involved in this step since at this point the user pays for the service requested. This protocol requires the intervention of a Trusted Party, in order to guarantee its fairness. At this, some information about the identity

[4] Note that service providers' public keys are included in the catalogue of services, thus they are automatically certified by the IA.

of the user might be leaked, but that the IA shall get no information at all about SP or t'.

Note that the IA will always get the information corresponding to the amount payed by the user in each transaction, but we want that this is the only information he may have in order to link user identities with requested services. Bearing this in mind, in the sequel we may assume all services offered at a given time slot have the same price.

  – `AccessService`. This protocol is run by a user U and a service provider SP. User's private input includes the token, and SP's private input is sk_{SP}.

    User U requests access to the service offered by SP. He gets as output a denial or acceptance message, depending on the validity of the token, and is or not allowed into the service accordingly. As we already noted, tokens recognized as valid will be always accepted by SP. At this, the private output to SP will include some information about U's token, which, if required, could be used as a proof of service in front of the Trusted Party.

– **Payment Protocols:**

  – `Pay`. This protocol is invoked by each SP at the end of every time slot, and involves him and the IA. SP sends part of the private outputs collected after successful `AccessService` executions, including a list of the collected tokens, to the IA, to be paid for the offered service. At the end of the protocol, SP gets paid for the list of tokens and the IA keeps his private output as a receipt of payment, typically involving some function of SP's private keying material and the tokens. Eventually, IA could deny payment. Namely, whenever SP tries to execute the protocol twice in the same time slot, or if some of the tokens are invalid or have been refunded. An optimistic fair e-cash protocol is used here, and the same Trusted Party as above is used to guarantee the fairness.

  – `Refund`. A user U executes this protocol with the Trusted Party and possibly SP and IA. U's private input includes an unused token, valid for the current time slot and service provider SP. If the Trusted Party finds that the token is valid and unused, then the user gets refunded (from IA but via the Trusted Party) for his payment. Both SP and IA will get payment receipts as private output, which SP will use to reject any further attempt to use the refunded token and IA will use to prove the Third Party that the token has been already refunded. Notice that we prefer not to rely on the state of the Third Party. Unused tokens not claimed for refund by the user are on the benefit of the IA.

3.2.3 Security Model

We aim at providing the following properties:

Correctness. If all the involved entities act honestly then:

- Every service provider SP will grant access to any user U in the execution of `AccessService` within a time slot t, whenever U uses as private input a token obtained from the execution of `ObtainToken` for service provider SP and t.
- In all executions of `Pay`, IA will accept and refund all tokens collected by SP for a given time slot.

Fairness for the user U*. Recall that, by assumption, a service provider SP will deny service to U* only on input of an invalid token. An adversary corrupting all service providers, any set of users (not including the target user U*) and the IA, has only negligible probability of winning the following game: U*, who acts honestly, runs a polynomial number of instances of the protocol `ObtainToken` to get tokens for some service providers and time slots. Concurrently, U* runs a polynomial number of instances of `AccessService` with some of the service providers, and also runs `Refund` with the Trusted Party giving as private input valid tokens rejected by service providers (this can only happen in case the adversary was able to construct the same token and used it before, exhausting its validity). The adversary wins the game if for a valid token x, a service provider denies access to U* on input x, and moreover, the Trusted Party rejects U*'s execution of `Refund` against that service provider on the same token x.

Fairness for the service provider SP*. Basically, we demand that a service provider will always be paid for all services offered within a given time slot. This is formalized in the following game:

An adversary corrupting a set of users, some service providers (others than SP*) and the IA, has negligible probability of winning the following game: Some corrupt and uncorrupt users run several instances of `ObtainToken`, `AccessService` with SP*, and of `Refund` against SP*. Moreover, SP* runs several instances of `Pay` (each one at the end of a different time slot). The adversary wins the game if, impersonating the IA, he denies payment to SP^* in a `Pay` execution, and also convinces the Trusted Party that he already paid SP^* in that time slot, or that some of SP^*'s tokens are invalid or have been refunded.

Fairness for the issuing authority IA. Consider an adversary corrupting a set of users and some (possibly all) service providers. Let n_t be the number of tokens sold by the IA until the end of time slot t, and let n'_t the total number of tokens paid (directly by an execution of `Pay` or forced by the Trusted Party in `Refund`) by the IA in all time slots t' such that $t' \leq t$. Then, assuming that a polynomial number of concurrent executions of `ObtainToken`, `AccessService`, `Pay` and `Refund` on adaptively chosen inputs occur, the probability that $n'_t > n_t$ is negligible.

Essentially, fairness for the IA means that the only valid tokens in the system are the ones generated in a successful execution of `ObtainToken`, and that the IA will never pay twice for a given token. The first condition can be seen as a kind of token unforgeability, while the second requirement relies on the fairness of `Refund` and `Pay` protocols, and on the fact that tokens are bound to specific service providers and time slots.

Anonymity for user's services. Consider the following indistinguishability game between an adversary $\mathcal{A}$, corrupting all parties (i.e., the IA, all users and all service providers) in the system, and a challenger $\mathcal{C}$.

- $\mathcal{A}$ runs `Setup` and sends to $\mathcal{C}$ all the public information about the users, the service providers and the IA. During the whole game $\mathcal{A}$ may execute polynomially many instances of the `ObtainToken` and `AccessService` protocols. Notice that, in particular $\mathcal{A}$ learns the user's private output of `AccessService`.

- $\mathcal{A}$ chooses two (possibly equal) service providers' identities, SP_0 and SP_1, and two (possibly equal) user's identities, U_0 and U_1, and sends the choice to $\mathcal{C}$ along with the internal state (including all the secret information) of U_0 and U_1.

- $\mathcal{C}$ flips a fair coin $b \in \{0, 1\}$ and prepares himself to run two (possibly concurrent) instances of `ObtainToken`, one as U_0 and the other as U_1, where $\mathcal{A}$ acts as the IA. To that end, $\mathcal{C}$ marks the protocol instance corresponding to U_0 as the target one, and uses as private input (SP_b, t), where t is the only time slot considered in this game. The other instance's private input is (SP_{1-b}, t). If $\mathcal{C}$ obtains as outputs two valid tokens, we denote by x_b the one from the target instance of `ObtainToken`, and the other one by x_{1-b}.

- Once the two instances of `ObtainToken` terminate, if they were both successful $\mathcal{C}$ (concurrently) runs two instances of `AccessService`, one for token x_0 with $\mathcal{A}$ acting as SP_0, and the other for token x_1 and service provider SP_1.

  Otherwise, if $\mathcal{C}$ failed to obtain the two valid tokens (even if he got one), he does not run any instance of `AccessService`.

- Eventually, $\mathcal{A}$ ends the game outputting a bit b'.

The probability that $b' = b$ (case in which $\mathcal{A}$ wins the above game) should be non-negligibly greater that 1/2.

Although the above is only one of the many possible indistinguishability-like definitions related to the anonymity of service, it is shown in Section 3.8 that this notion implies the most general possible definition of anonymity. Namely, from the information available to the IA from `ObtainToken` instances, and to the service providers from `AccessService` instances, no polynomial time adversary can distinguish any two possible matchings between both sets of instances.

At the end of this chapter we introduce an equivalent definition for anonymity.

3.3 A Basic Subscription Scheme

The basic scheme uses a public-key encryption scheme ENC, a blind signature scheme $BSig$ and basic (general purpose) signature scheme Sig. The $BSig$ protocol is linked to an optimistic fair e-cash protocol in order to guarantee that a user gets a valid blind signature if and only if he pays to the signature issuer. This can be typically done by using the e-cash protocol to fairly send the last signer's message in the blind signing protocol. We assume that in case the user does not pay the signer then he does not receive the last message, so no blind signature is generated. Conversely, the user will not pay if the verification of the blind signature fails. To name this dedicated combination of $BSig$ and a fair e-cash protocol, we will often refer to the *modified blind signature scheme*. In the last chapter we will discuss the notion of fair exchange of signatures and provide a fast instantiation that can be applied directly to our proposal here.

Our Basic construction is explained below:

Set Up. Keys for the IA and all service providers are generated and distributed:

- Each service provider SP_j holds a key pair $(pk_{\mathsf{SP}_j}, sk_{\mathsf{SP}_j})$ for the encryption scheme ENC, and another key pair for the signature scheme Sig.
- IA generates signing keys $(pk_{\mathsf{IA}}, sk_{\mathsf{IA}})$ for $BSig$. It also signs and publishes the `catalogue`.
- Each SP maintains a list L_{SP} of accepted tokens [5]. Also, IA and each SP maintain a list of tokens payed for through `Refund` for the current time slot (denoted, respectively R_{IA} and R_{SP}).

Obtain Token. User U wants to buy access to SP's service in (a future) time slot t.

1. U generates a fresh key pair (y, s) for the basic signature scheme Sig.
2. U obtains from IA a valid[6] blind signature $\sigma = \texttt{BlindSig}(y||\mathsf{SP}||t)$ and pays for it, by means of the modified blind sign algorithm.
3. U stores the token $x = (y, \mathsf{SP}, t, \sigma)$ and s until the end of slot t.

Verify Token. Given a token $x = (y, \mathsf{SP}, t, \sigma)$, any party can verify its correctness by just verifying that σ is a valid blind signature of $m = y||\mathsf{SP}||t$.

Access Service. User U requests access to the service SP on time slot t :

1. U sends an access request message to SP.
2. SP generates a random nonce α and forwards it to U.
3. U computes $c = ENC_{\mathsf{SP}}(y||\sigma||\tilde{\sigma})$, where $\tilde{\sigma} = Sig_s(\alpha)$, and sends c to SP.
4. SP decrypts c and parses y, σ and $\tilde{\sigma}$.
5. SP checks that σ is a valid signature of $y||SP||t$ and that $\tilde{\sigma}$ is a valid signature of α with verification key y.

[5] Recall that in the above we are assuming for simplicity the access policy to be "one access per token", otherwise this lists would be configured fitting each concrete access policy.

[6] Here, we impose U does have the ability to actually check the validity of the received token, as it is explicited later in `VerifyToken`.

6. SP also checks that σ is not in the refunded token list R_{SP}.
7. SP looks at the access table for previous usages of y [7] and applies the service terms of use to decide acceptance.
8. If all checks are OK, SP allows U into the server and adds a new row $(\alpha, y, \sigma, \tilde{\sigma})$ to the access table L_{SP}.

Pay. At the end of the time slot, each SP runs the following protocol:

1. SP sends the list of collected (i.e., valid and not refunded) (y, σ) to IA.
2. IA checks whether he paid SP before in the current time slot. If not, IA checks the validity of all the items in the list for the current time slot, and that none of them have been refunded (looking them at R_{IA}), and pays SP for them via the fair e-cash protocol.
3. IA gets as a receipt SP's signature on the time slot identifier t, and keeps it until the beginning of next time slot.
4. SP resets his access table L_{SP} and the refund table R_{SP}, and enters in a `lock` state until the beginning of the next time slot.

Refund. User U asks the Trusted Party for an unused token refund.

1. U sends TP the (presumably) unused token $x = (y, \mathsf{SP}, t, \sigma)$.
2. TP checks the validity of σ and asks SP for a proof of usage or previous refund.
3. If not locked, SP checks for usages of y in table L_{SP} and sends the corresponding entry $(\alpha, \tilde{\sigma})$, if it exists. He also checks if σ is in table R_{SP} and if so, sends the corresponding refund receipt.
4. If in either case TP accepts SP's proof (or if SP in locked), then TP aborts the protocol.
5. Otherwise, TP asks IA for refund on $(y, \mathsf{SP}, t, \sigma)$.
6. If after looking at R_{IA}, IA sends a receipt of previous refund on that token, then TP aborts.
7. Otherwise, TP sends a receipt (TP's signature on `'refunded'` $t||\mathsf{SP}||\sigma$) to both SP and IA, and sends back the cash to U.
8. SP and IA add σ and the refund receipt to the corresponding refund lists R_{SP} and R_{IA}.

[7]Checking for σ would be not enough unless the blind signature is strongly unforgeable, as we need that the adversary cannot produce produce a new signature pair (m, σ), even having different signatures on m at hand.

3.4 Formal Analysis of the Proposed Scheme

Let us now argue that our generic construction fulfils the properties listed in Section 3.2.3. At this, we are assuming that the underlying blind signature scheme $BSig$ has the blindness and non-forgeability property, as defined in Section 3.1. Moreover, we assume the encryption scheme ENC to be IND-CCA secure. The basic signature scheme Sig is assumed to be existentially unforgeable under chosen message attacks. Finally, we assume the fairness of the optimistic e-cash protocols used in `ObtainToken` and `Pay`.

Correctness. It follows trivially from the correctness of the involved tools $BSig$, Sig and ENC, and the e-cash protocols.

Fairness for the user U^*. Note that the adversary will not be able to replay an eavesdropped connection message c from a previous connection, as c involves a signature of the nonce α that can only be used once. Therefore, the adversary won't succeed in a strategy of "exhausting" the usage of a token legitimately obtained by U^*.

As a result, the only case in which fairness for user U^* may be violated is that in which for a valid token x, a corrupt service provider denies access to U^* on input a legitimate c constructed from x and, moreover, the Trusted Party rejects U^*'s execution of `Refund` against that service provider on that same token x.

However, the Trusted Party rejects U^*'s execution of `Refund` only if the adversary $\mathcal{A}$ defined in section 3.2.3 shows him a valid pair $(\alpha, \tilde{\sigma})$, where α is a session identifier and $\tilde{\sigma}$ is a basic signature on α with respect to the verification key y. But this is only possible if either U^* computed $\tilde{\sigma}$ (so he indeed accessed the service) or $\mathcal{A}$ forged that signature.

Fairness for the service provider SP^*. Suppose that an honest service provider SP^* and an adversary $\mathcal{A}$ are playing the game corresponding to the present security notion, as described in section 3.2.3. Let $L_{\mathsf{SP}^*} = \{(y_k, \sigma_k, \alpha_k, \tilde{\sigma}_k)$ $\}$ be the contents of SP^*'s access table at the end of a specific time slot t. Notice that each σ_k is a valid blind signature on $m_k = y_k||\mathsf{SP}^*||t$, and all m_k are different. At the end of the time slot, SP^* runs `Pay` with the adversary, who acts as the IA, for list L_{SP^*}.

Assume that $\mathcal{A}$ cheats SP^* and denies payment. Now SP^* complains to the Trusted Party, by sending him the list L_{SP^*}. As SP^* acts honestly, the

Trusted Party is convinced about the validity of the collected tokens. Next, the Trusted Party asks $\mathcal{A}$, who acts as the IA, for both a list of receipts for tokens in L_{SP^*} which have been refunded, and a payment receipt for SP^* and current time slot. Since SP^* acts honestly, there are no unused tokens in L_{SP^*}. Hence, the only way $\mathcal{A}$ can show a refund receipt for a token in L_{SP^*} is by forging a signature on the token on behalf of the Trusted Party. Indeed, no used token can be refunded, since during the execution of `Refund`, the Trusted Party asks SP^* for a proof of usage of the token, and SP^* answers with a valid pair $(\alpha, \tilde{\sigma})$, so the Trusted Party denies refunding.

On the other hand, $\mathcal{A}$ cannot show a payment receipt for the current time slot, and thus the Trusted Party forces him to pay SP^* for all tokens in L_{SP^*}. Indeed, due to the fairness of the e-cash protocol in `Pay`, $\mathcal{A}$ can only show a payment receipt if he forged one (i.e., he forged a signature by either SP^* or the Trusted Party) or if he successfully ran `Pay` with SP^* before. But the last situation is impossible, as an honest SP^* runs `Pay` at most once per time slot.

Fairness for the issuing authority IA. Consider a successful adversary $\mathcal{A}$ who plays the game defined in Section 3.2.3. Then, we show a forger $\mathcal{F}$, who internally uses $\mathcal{A}$, winning the blind signature unforgeability game against a challenger $\mathcal{C}$, with a non-negligible probability.

Firstly, the challenger $\mathcal{C}$ generates, according to the specification of the blind signature scheme, the system parameters and the public key pk_{BSig}, and sends them to a forger $\mathcal{F}$. Next, $\mathcal{F}$ completes the public parameters of the subscription scheme (including the public key of the Trusted Party) and the public key of the (honest) IA, and sends this information to $\mathcal{A}$. Now $\mathcal{A}$ computes and sends to $\mathcal{F}$ the set of public keys of the service providers, and also a description of the corresponding services. $\mathcal{F}$ compiles and signs the catalogue of services and send it back to $\mathcal{A}$.

Now $\mathcal{A}$, acting as a (dishonest) user, concurrently runs polynomially many instances of `BlindSig` with $\mathcal{F}$ acting as the IA. $\mathcal{A}$ can also run a polynomial number of instances of the protocols `Refund` and `Pay`. Here, $\mathcal{A}$ takes the roles of both the users and the service providers, while $\mathcal{F}$ acts as both the IA and the Trusted Party.

During the game, $\mathcal{F}$ maintains a list of all valid pairs $(m_k = y_k || \mathsf{SP}_k || t_k, \sigma_k)$ of blind signatures and messages collected in all executions of `Refund` and `Pay`. As an honest IA he also maintains lists of refunded and paid tokens, and the corresponding receipts, for each service provider, which are needed in a proper execution of those protocols.

Eventually, $\mathcal{A}$ ends the game (with a non-negligible probability of having been paid for more tokens than there were bought). Finally, $\mathcal{F}$ sends $\mathcal{C}$ the list of collected message/signature pairs, and ends the game. Here we assume that $\mathcal{F}$ maintains the list in such a way that all messages in it are different, and that all signatures are valid.

Now, let us see that $\mathcal{F}$ will only pay $\mathcal{A}$ for valid tokens, and he will never pay twice for the same token. Indeed, in both protocols `Refund` and `Pay` the IA checks the validity of the token (*i.e.,* the validity of the blind signature) before paying. On the one hand, $\mathcal{F}$ maintains a list of refunded tokens, so that any repeated execution of `Refund` is rejected; and this list is also used to check for duplicates in `Pay`. Since $\mathcal{F}$ only accepts a single execution of `Pay` per service provider and time slot, and due to the collision resistance of the blind signature, no token can be paid more than once.

Finally, due to the fairness of `ObtainToken`, the only executions of `Blind-Sig` accepted by $\mathcal{F}$ come from executions of `ObtainToken` accepted by $\mathcal{F}$ (i.e., paid by $\mathcal{A}$). Hence, whenever $\mathcal{A}$ is successful, the number of executions of `BlindSig` accepted by $\mathcal{F}$ is less than the number of message/signature pairs outputted by $\mathcal{F}$, thus breaking the unforgeability of the blind signature scheme.

Anonymity for user's services. Given a successful adversary $\mathcal{A}$ against the anonymity of the subscription scheme, we show another adversary $\mathcal{B}$ who breaks the blindness of the blind signature scheme by using $\mathcal{A}$ internally. Let $\mathcal{C}$ be the challenger for $\mathcal{B}$ in the blindness game.

Firstly, $\mathcal{C}$ generates the system parameters of the blind signature scheme and gives them to $\mathcal{B}$. $\mathcal{B}$ completes the public parameters with the system parameters of the other components in the anonymous subscription system, and send them to $\mathcal{A}$. Then $\mathcal{A}$ generates the public output of the `Setup` protocol (i.e., public keys for all entities including the public key for the blind signature pk_{BSig} and the signed catalogue of services) and sends it to $\mathcal{B}$. Now, $\mathcal{A}$ selects the target identities: SP_0, SP_1 and U_0, U_1 and sends them to $\mathcal{B}$ along with the internal state of U_0 and U_1. Notice that the internal states in particular include the secret information about user's identities, needed in the e-cash protocol. After verifying the information received from $\mathcal{A}$, $\mathcal{B}$ forwards pk_{BSig} to $\mathcal{C}$. $\mathcal{B}$ also generates two key pairs for the basic signature scheme (s_0, y_0) and (s_1, y_1), and sends $m_0 = y_0||\mathsf{SP}_0||t$ and $m_1 = y_1||\mathsf{SP}_1||t$ to $\mathcal{C}$, where t is the descriptor of the current time slot.

Now $\mathcal{C}$ flips a fair coin b and starts two instances of `BlindSig` on m_b and m_{1-b}, notifying $\mathcal{B}$ that the former is the target one. For each instance, $\mathcal{B}$ executes `ObtainToken` with $\mathcal{A}$ as the IA in the following way: $\mathcal{B}$ forwards all messages corresponding to the signing protocol from $\mathcal{C}$ to $\mathcal{A}$ and from $\mathcal{A}$ to $\mathcal{C}$, and uses the corresponding identity (U_0 for the target instance, and U_1 for the other one) in the e-cash part of the protocol. $\mathcal{B}$ also informs $\mathcal{A}$ that the instance using U_0's identity is the target one.

If at the end of the protocols $\mathcal{C}$ gets two valid blind signatures: σ_0 on $m_0 = y_0||\mathsf{SP}_0||t$ and σ_1 on $m_1 = y_1||\mathsf{SP}_1||t$, then he sends (σ_0, σ_1) to $\mathcal{B}$. Otherwise, $\mathcal{C}$ sends $\perp$ to $\mathcal{B}$.

In the first case, as $\mathcal{B}$ holds valid tokens $x_0 = (y_0, \mathsf{SP}_0, t, \sigma_0)$ and $x_1 = (y_1, \mathsf{SP}_1, t, \sigma_1)$, he runs two instances of `AccessService`: one for x_0 with $\mathcal{A}$ acting as SP_0, and the other for x_1 with $\mathcal{A}$ acting as SP_1. This means that $\mathcal{A}$ receives encryptions of both $(y_0||\sigma_0||\tilde{\sigma}_0)$ and $(y_1||\sigma_1||\tilde{\sigma}_1)$, for valid nonces α_0, α_1, and valid basic signatures of them $\tilde{\sigma}_0$, $\tilde{\sigma}_1$ for verification keys y_0, y_1, respectively. In the second case, no instance of `AccessService` is executed. In both cases, $\mathcal{A}$ eventually ends the game by outputting a guess bit b', which is forwarded to $\mathcal{C}$ by $\mathcal{B}$.

It is straightforward to see that $\mathcal{B}$ perfectly simulates a challenger for $\mathcal{A}$ in the anonymity game. So $\mathcal{A}$ wins the game with a non-negligible probability, which is equal to the probability that $\mathcal{B}$ wins the blindness game.

3.5 Efficient Instances

In the previous sections a generic flexible anonymous subscription scheme has been presented. Here we go further in the efficiency analysis, roughly sketching the cost of concrete instantiations. To implement the scheme we propose using RSA blind signature that is fast and efficient for `ObtainToken` and the hashed ElGamal signature (as modified by Pointcheval and Stern [100]) as the basic general purpose signature scheme Sig used in `AccessService`. ElGamal signing requires 1 exponentiation and verification requires 3. As IND-CCA encryption scheme ENC, we choose RSA OAEP+ [112]. In terms of exponentiations, the cost of encryption and decryption is nil. Of course, corresponding encoding functions relating the underlying groups of the above schemes should be at hand.

	`ObtainToken`		`AccessService`	
	User	Issuer	User	Service Prov.
[21]	3	3	13	7
Ours	3	1	2	5

Table 3.1: Efficiency comparison between [21] and our scheme, measured in number of exponentiations.

Using this figures, we can compare our protocol with the one of [21] looking at the efficiency of the corresponding algorithms for buying tokens and connecting[8].

Their `Obtain` protocol requires 6 exponentiations (3 performed by the user and 3 by the issuer). Using RSA blind signature, the complexity of obtaining a token in our proposal is computing 3 exponentiations and 2 multiplications that is far more efficient. Although the security of this signature is proven in the random oracle model, there is no known attack against it.

Compared to ours, the protocol `Show` of [21] which is the most efficient, up to our knowledge, proposed so far — calls for 13 exponentiations from the user and 7 from the service provider, when the user connects to a service, while in our `AccessService` protocol only 2 exponentiations is computed by the user, and 5 exponentiations are performed by the service provider, what is far more efficient. This makes our protocol completely suitable in most practical scenarios.

3.6 Managing Service Access Policies and Trustness on the Service Provider

3.6.1 Multiple Accesses per Token

Our description of `AccessService` can be easily modified to provide full flexibility of the service providers policy. Multiple accesses per token can be implemented if the service provider allows more than one record per token in the access table. At this, further precautions should be taken in order to prevent replay attacks, e.g., we can add some structure to the nonce α. Namely, α may be the concatenation of a constant part α_0 and an access counter α_1. Then SP will only accept an access attempt for a signed nonce $\alpha_0||\alpha_1$, with $\alpha_1 > 0$, if a previous usage of the token shows the value $\alpha_0||\alpha_1 - 1$. It is straightforward

[8]this scheme is significantly more efficient than that of [39].

for the SP to apply a limit in the number of accesses per token based on the stored value of α_1. Actually, SP can save memory if he stores only the last usage of each token.

Also, timing information can easily be added to the access table in order to apply more complex access policies involving both the number of accesses and the total access time, or the time elapsed from the first access.

On the other hand, if the service is configured in different sessions (e.g., sub-services or groups) per time slot among which users may freely choose, then a (public) session identifier sid can be appended to the nonce α.

Obviously, in case of multiple accesses per token, the protocol `Refund` should be refined depending on the concrete policy. For instance, the Trusted Party can consider a token unused if no access to the service have been given for that token or one may impose that tokens may be refunded as long as they are not exhausted. Additionally, partial refunds (*i.e.,* refund of the estimated unused part of a token) could be considered. However, this variant has a high cost in terms of efficiency, as the `Refund` protocol (which is likely to be very costly) will presumably be executed many times.

3.6.2 Removing Trust on Service Providers

In the basic definition of `AccessService` we assumed that a service provider never denies access to the service if the user shows a valid unused token. However, dropping this assumption may make sense in settings in which client loyalty is not valuable; like services that are only required once and for which potential clients are not in touch with former users. At this, a dishonest SP could collect a valid token and deny access to the user. Then nobody can prevent SP to include this actually unused token in the `Pay` protocol. Actually, the Trusted Party should not accept any complaint from a user, since a dishonest user could complain just to be refunded on a used token.

In some settings this problem can be circumvented with a small overhead: if, for instance, the service consists of a user connected to a resource (e.g., game, multimedia streaming, chat room, ...) for a long period of time. In such scenario the user can be requested to send his token and a signature on an incremental nonce, as explained above, at a fixed and reasonable rate (say, once every minute). In the worst case, if the service provider interrupts the service then he can only prove to the Trusted Party that the user got access during one more minute than the actual access time, which is not a great deal in most applications. Moreover, a user cannot ask for refund on more than the unused time, since the SP holds a user's signature on the nonce used in the last access.

3.7 More Details on Payment Protocol

The important point relative to applying an e-cash scheme to our protocol is the problem with fairness for both the issuer and the user. Actually in our protocol, a user U takes part in an e-cash protocol to pay for a token related to some service provider. But there is no warranty if after paying to the issuer, he will get his token. This problem is related to a known subject in cryptography known as *optimistic fair exchange* (that has been widely discussed in the last chapter of this thesis) in which *fairness* means that after finishing the protocol or both of the parties get the desired value or non of them get nothing. In our protocol, the fairness warranties that finishing the protocol if the user has paid for a token he has got the token otherwise none of them has got anything. This problem has been studied in the literature and some solutions have been provided. We will give some references to some works related to this problem.

The problem of fairness is sometimes relative to the physical problems, for example if after finishing the payment protocol, the internet connection fails, how the user can prove that he has not got the token. This problem has been studied in [20] and the reader can refer to that paper to get more information.

A more serious problem is that one of the parties tries to cheat the other. Our protocol is an exchange protocol that the user sends some digital coins and the issuer sends a blind signature. Recently in [110], it has been attempted to give a solution for mixing e-cash and fair exchange protocols. The basic idea of this proposal is as follows; the user who is going to pay to some merchant, registers to a bank and asks for a certification of withdrawal of some amount of money which is a digital check. The bank gives this certification but does not charge the user's account unless the check is submitted to the bank by some merchant. After that, the user and the merchant get engage in a fair exchange protocol where, the user sends the merchant the check together with some verifiable value that defines uniquely the current exchange session and the merchant sends the user the required goods. Finishing the fair exchange protocol, the merchant refers to the bank and submits the check together with the other information . In the case that all the data is correctly computed, bank charges the user's account and deposits the merchant's account. To avoid double-spending a check by the merchant, bank uses some unique randomness and validity period in the issued check, in this way they define anonymity to be protected against the merchant and not the bank. Note that a check can be paid by the user in various sessions to various merchants without any problem. More details can be read in that paper.

3.8 Further Discussion: On the Definition of Anonymity

In this section, we give an alternative definition of anonymity that seems more general than the one given in Section 3.2.3. However, both definitions are shown to be equivalent.

Let $\mathcal{A}$ be an adversary who corrupts all service providers and IA. The information available to $\mathcal{A}$ can be summarized into two sequences: $B = (t_i, u_i)_{i=1\ldots n}$ and $S = (t'_j, sp'_j)_{j=1\ldots n'}$, where in time t_i a token have been bought by user U_{u_i} and at time t'_j a token was used for service provider $\mathsf{SP}_{sp'_j}$. We assume that both sequences are sorted in increasing times, and all time values are different. At this, the only information unavailable to $\mathcal{A}$ is which service provider corresponds to every bought token and how bought tokens and spent tokens match.

Let us summarize the unknown information as the sequences $M = (i_j)_{j=1\ldots n'}$ meaning that the token spent at time t'_j is the one bought at time t_{i_j}, and $T = (sp_i)_{i=1\ldots n}$, where the token bought at time t_i was for SP_{sp_i}. Obviously, all i_j are different, all $t_{i_j} < t'_j$, and $sp_{i_j} = sp'_j$. Notice that it could happen that $n' < n$, meaning that not all bought tokens were actually used (this actually motivates the introduction of the sequence T).

We call the pair (B, S) admissible if there exists a pair (M, T) compatible with it. Now we can give an alternative definition of anonymity. Consider the following game in which $\mathcal{A}$ generates and sends all public information about the subscription scheme and sends it to the challenger $\mathcal{C}$, who impersonates all the honest users. Next, $\mathcal{A}$ selects two different pairs (M_0, T_0) and (M_1, T_1) compatible with the same (B, S), and sends all that information to $\mathcal{C}$. Then $\mathcal{C}$ flips a fair coin $b \in \{0, 1\}$ and plays the sequences (B, S) according to (M_b, T_b) with $\mathcal{A}$, who impersonates IA and all SP. Eventually, $\mathcal{A}$ outputs a guess b' and wins the game if $b' = b$.

It can be shown that this general definition can be reduced to three basic situations:

1. $B = (t_1, u_1)$, $S = M_0 = M_1 = ()$, $T_b = (sp_b)$, i.e., there is only one token bought for SP_0 or SP_1 but unused.

2. $B = (t_1, u_1; t_2, u_2)$, $S = (t'_1, sp_1)$, $M_0 = (1)$, $M_1 = (2)$, $T_b = (sp_1, sp_1)$, i.e., there are two tokens bought for the same service provider, but only one of them has been used.

3. $B = (t_1, u_1; t_2, u_2)$, $S = (t'_1, sp_1; t'_2, sp_2)$, $M_0 = (1, 2)$, $M_1 = (2, 1)$, $T_0 = (sp_1, sp_2)$, $T_1 = (sp_2, sp_1)$, i.e., there are two tokens and both have been used.

Actually, we can define a difference bipartite graph whose vertex sets are $U = \{u_i\}_{i=1\ldots n}$ and $V = \{v_j\}_{j=1\ldots m}$ and there is an edge (u_i, v_j) if and only if $i_j = i$ in M_0 but not in M_1 or vice versa. Since all the vertices in V have degree exactly 2 and the vertices in U can have degrees 0, 1 or 2, the graph is a disjoint union of even length paths and even length cycles. Notice that the only information unknown to the adversary corresponds to the differences between the sequences M_0 and M_1 (which are represented in the difference graph) and between T_0 and T_1, which are partially redundant. Only the differences between T_0 and T_1 corresponding to tokens unused for both $b = 0$ and $b = 1$ are not contained in the difference graph, and they can be addressed by the transformation derived from situation 1.

The basic situation 3 can be seen as an elementary graph transformation (i.e., replacing edges $(a, b), (c, d)$ by $(a, d), (b, c)$), which applied iteratively can reduce any such difference graph into the disjoint union of length 2 paths (corresponding to situation 2) and length 4 cycles (corresponding to situation 3). Situation 1 can be used to change one-by-one all the service providers associated to the unused tokens. Then a standard hybrid indistinguishability argument can be applied to the resulting sequence of graphs.

Moreover, situation 3 can be reduced to situation 1 (for $sp_1 \neq sp_2$), since a distinguisher for situation 1 is enough to solve situation 3 just by using the information about the first bought token. Similarly, situation 3 can be reduced to situation 2 (for $sp_1 = sp_2$), since a distinguisher for situation 2 is enough to solve situation 3 just by using the buying information and the first token usage.

Actually, situation 3 corresponds exactly to the indistinguishability definition used in this chapter.

Chapter 4

Stateless Commitments and Universally Convertible Directed Signatures

4.1 Introduction

Although at the time of invention of digital signatures, universal verifiability was considered as a basic property, later a large variety of real-life applications showed that this property is not always desirable. An example is the design of *abuse-free contract signing protocols*, where the first signer of a contract protects his signature against being verified by any party until the end of the protocol.

Undeniable Signatures. The first digital signatures with limited verifiability were introduced by Chaum *et al.* as *undeniable signatures* [33], which cannot be verified without the cooperation of the signer[1]. After that, several variations were introduced to overcome the problem of lack of availability of the signer. *Convertible undeniable signatures* [16, 34] provide a mechanism that allows the signer to convert an undeniable signature into a universally verifiable one. *Directed signatures* [83] and *designated confirmer signatures* [30] allow a party designated by the signer to verify the signatures, without interacting with the signer. It is broadly accepted that in a designated confirmer signature scheme only the confirmer can verify (or convert) a signature [23, 67], while both the confirmer and the signer have this ability in a directed signature scheme. Delegation of the capability of verification or conversion of undeniable signatures

[1]In general, a signature that can not be verified without the cooperation of the signer is called an *invisible signature* but depending on its functionalities they have got different names in the literature

found important applications in some areas like contract signing, where a balance between the privacy of the signer and the availability of the verifier is required.

Two types of signature conversion are considered in [16]. Namely, undeniable or directed signatures can be individually converted by releasing some information specific to a particular signature (without affecting the public verifiability of other signatures), or all the signatures can be converted at once by revealing some trapdoor information. The former is referred to as *selective* conversion, while the latter is called *universal* conversion.

In [41] the first practical and provably secure universally convertible undeniable signatures were proposed. Several selective and universal convertible undeniable signature schemes have been proposed in recent years [48, 59, 72, 80, 91], but none of them are both individual and universally convertible while achieving the security requirements in the standard model. Recently other types of selective convertible schemes were proposed in [48, 82], where the conversion is related to the issuing time or to some specific events. In [73] a generic transformation from a selectively-convertible to a universally-convertible undeniable signature, secure in the standard model is given. Another generic construction is given in [47] but selective conversion is not addressed.

Most signatures with limited verifiability use the design approach “sign and encrypt”, which consists of first signing the message then encrypting the resulting signature with the public key of the confirmer. However, this approach involves very inefficient (generic) zero-knowledge proofs that the encryption contains a valid signature on a specific message, or efficient dedicated proofs that can be used only for very specific signature and encryption schemes. In [92] a different design approach “commit and sign” is used. To design an efficient scheme based on the idea of “commit and sign”, a special commitment scheme called *confirmer commitment* is introduced.

Confirmer commitments were proposed in [92] and used as a natural solution for designing designated confirmer signatures and their applications (as contract signing protocol). A confirmer commitment is a type of commitment scheme with a public key such that the only one who can verify the commitment with respect to a message m is the holder of the secret key without the knowledge of the randomness used in the commitment. The generic idea to design an efficient designated confirmer signature on the message m proposed in [92] is to hide the message in a computationally hiding confirmer commitment using the public key of the confirmer as the public key of the commitment and then signing the commitment. Later in [23] it was shown that the proposal of [92] is not secure in the multi-user system when different signers share the same confirmer. The problem is that the commitment does not hold any information about the issuer and on the other hand it is malleable. So any other signer can modify the commitment in a very simple way and sign it using his own secret key. He can use the confirmer to decide the validity of the commitment in such a way that the resulting commitment is valid if and only if the original one is valid, and this breaks easily the invisibility of the system.

In another paper Gentry, Molnar and Ramzan [58] proposed a solution for designated confirmer signatures that uses the ideas of [92] by signing a commitment of the message then encrypting the randomness used in the commitment with the public key of the confirmer. In this way the problem in the multi-user systems was solved but the scheme is expensive. It is because the encryption scheme used there is required to be IND-CCA secure and also the proof of the consistency of the encryption is not simple. A particular instance with an efficient zero-knowledge proof is also given in [58], which is secure in the standard model.

In 2007, [117] tried to repair the problem of [92] by adding a proof of knowledge of opening the commitment and then using a normal signature scheme to sign all the information. In this way they avoid using encryption schemes. This proposal is the most efficient protocol for designated verifier signatures.

Directed Signatures. Although undeniable and designated confirmer signatures have been widely studied, less attention has been paid to the case of directed signatures. The only paper addressing both universal and selective conversion for directed signatures is [81], in which an efficient scheme based on bilinear maps and secure in the random oracle model is proposed. Actually, the same paper offers the first formal definition and security model for universally convertible directed signatures. The main security properties of such a signature scheme are *unforgeability* and *invisibility*. While unforgeability is defined in the usual way, invisibility means that an adversary cannot tell apart a directed signature on an adversarially chosen message from a signature on a random message. Two levels of invisibility are defined depending on the capabilities given to the adversary. Namely, in the weak invisibility definition, the adversary can only ask for signatures on arbitrary messages and for verification of arbitrary signatures. In the strong notion it is allowed to ask for individual conversion of arbitrary signatures, or for universal conversion with respect to an arbitrary confirmer. Two proofs of invisibility are given in [81]: one for weak invisibility under the Decisional Tripartite Diffie-Hellman Assumption (DTDH), and the other for strong invisibility under a much stronger assumption. However, the first proof is not complete, as we show in the next section.

Therefore it is still an open problem to give a universally convertible directed signature scheme secure in the standard model.

Our Contributions. In this chapter we will discuss the notion of *universally convertible directed signatures* and its security definitions and propose the first protocol for this kind of signatures in the standard model. We have extended the definition of convertible directed signatures to allow the use of different conversion trapdoors for the same pair of signer/confirmer. With this extension, the signer can classify his signatures in different groups so that the opening of a specific trapdoor only converts the signatures in one group, without affecting the invisibility of the signatures in the remaining groups, even for the same signer and confirmer. In this extended model the signer can generate a new trapdoor at any time, and register it to the confirmer. As a result the confirmer issues a registration certificate to the signer, which is used as a guarantee of convertibility for the signature recipients. Every registered trapdoor is known to both the signer and the confirmer, allowing them to confirm or convert any signature corresponding to the trapdoor.

In order to address both selective and universal convertibility, we propose the notion of *stateless commitment* as a modification of the confirmer commitments introduced in [92]. Stateless commitments are dual to the trapdoor commitments. While the latter uses a trapdoor to break the binding property (*i.e.*, with the trapdoor one can open a commitment in two different ways), the former uses the trapdoor to break the hiding property. This allows the committer and the opener (trapdoor owner) be different parties. Furthermore, if the trapdoor is known to the committer, he does not need to store the randomness of a commitment to be able to open it afterwards (and this is the reason why we call it a stateless commitment). Although stateless commitments are similar to public key encryption (and actually stateless commitments can be built from some public key encryption schemes), we stress that the committed message is not properly contained in the commitment, so there is no message recovery property. The main difference between a confirmer commitment and a stateless commitment is that in the latter the opener can compute a proof of validity of the commitment for a given message, which makes the commitment verifiable for any party without affecting the hiding property of the other commitments. It is worth noticing that stateless commitments can be constructed without explicitly using an encryption scheme.

With stateless commitments at hand we give a generic construction of convertible directed signature schemes from any secure (ordinary) signature scheme. The design of the new directed signature schemes follows the approach "commit and sign", in which the directed signature is a signature on a stateless commitment to the message. Therefore, no encryption scheme is explicitly used. The invisibility property is directly implied by both the hiding of the commitment and the unforgeability of the signature. The new scheme supports multiple trapdoors, it is both selective and universal convertible, and does not rely on any random oracle. We also show two efficient instances, one based on pairings (using both DTDH and the Strong Diffie-Hellman [13] assumptions) and the other based on strong-RSA (introduced simultaneously in [6, 54]) and the Decisional Composite Residuosity [96] assumptions. To the best of our knowledge these are the first universally convertible directed signature schemes provably secure in the standard model, and they are based on standard assumptions. Furthermore, the new schemes can be easily deployed in existing platforms, and previously existing (ordinary) signing keys can be safely reused. Actually, the construction given in [58] could be seen as a particular case of ours (but in the case of designated confirmer signatures) that uses a stateless commitment constructed from a standard commitment and an IND-CCA encryption. However, we give a more flexible construction which can be easily instantiated with more efficient stateless commitments.

It is worth mentioning that our schemes require a trapdoor registration step which is not required in [81]. However, this is not a serious drawback because trapdoor registration is just an implementation of a guarantee of convertibility for the recipient of the directed signature. Furthermore, trapdoor registration gives the confirmer the knowledge of which signers are going to refer to him, and also gives him the ability to reject a specific signer by not issuing him the registration certificate. On the other hand, some adaptive attacks using multiple (related) signer keys with the same confirmer reported in [23] are automatically ruled out by the trapdoor registration.

4.2 Model and Definitions

Following the definition given in [81] a *universally convertible directed signature scheme*, DS, includes the following algorithms and sub-protocols, run by a signer A, a confirmer B and a recipient C, which syntax is briefly described below (using the notation $\mathsf{output} \leftarrow \mathsf{ProtocolName}(\mathsf{input})$, where input refers to the common input in case of multiparty protocols):

- **generation of public parameters:** $\texttt{DS.param} \leftarrow \texttt{DS.Setup}(1^k)$ is an algorithm (run by a trusted party) that takes a security parameter 1^k as input an outputs the system public parameters $\texttt{DS.param}$, which implicitly include k and are supposed to be given as input in all the above protocols.

- **key generation:** $(sk_A, pk_A) \leftarrow \texttt{DS.Signer.KeyGen}()$ is a probabilistic algorithm that outputs the secret and public keys for A, and similarly $(sk_B, pk_B) \leftarrow \texttt{DS.Confirmer.KeyGen}()$ does for B.[2]

- **key registration:** $\texttt{cert}_{pk} \leftarrow \texttt{DS.}\{\texttt{Signer,Confirmer,User}\}\texttt{.Register}(pk)$ is a protocol run by any user (with private input sk) and a "Key Registration Authority" (with some private input) to certify the public keys in the system. $\texttt{cert}_{pk} = \perp$ indicates rejection by the authority.[3] The user could be either A, B or C.

- (A, B)**-directed signature generation:** $\Sigma \leftarrow \texttt{DS.Sign}(m, sk_A, pk_B)$ is a probabilistic algorithm producing an (A, B)-directed signature on the message $m \in \{0, 1\}^*$.[4]

- (A, B)**-directed signature verification:** $v \leftarrow \texttt{DS.Signer.Verify}(\Sigma, m, sk_A, pk_B)$ and $v \leftarrow \texttt{DS.Confirmer.Verify}(\Sigma, m, pk_A, sk_B)$ are deterministic algorithms that outputs valid or $\mathsf{invalid}$ depending on whether Σ is a valid (A, B)-directed signature on m.

- (A, B)**-directed signature confirmation/denial:** Two protocols $\mathsf{DS.Signer.ConfirmOrDeny}(\Sigma, m, pk_A, pk_B)$ and $\texttt{DS.Confirmer.ConfirmOrDeny}(\Sigma, m, pk_A, pk_B)$ are protocols between a prover P (either A or B) and a verifier C. If Σ is a valid (A, B)-directed signature on m then P sends valid to C and runs a non-transferable proof of validity. Otherwise, P sends $\mathsf{invalid}$, and if Σ is well-formed (*i.e.,* it is a valid directed signature on another message $m' \neq m$) then P runs a non-transferable proof of

[2]An additional key generation algorithm could be necessary for recipient C, in order to implement some non-transferable proofs.

[3]We slightly deviate from the definition in [81] to capture the actual meaning of the key registration procedure.

[4]In practice we can assume that a nontransferable verification of Σ is executed by the signer and the recipient just after $\texttt{DS.Sign}$.

invalidity.

- **individual conversion of an (A, B)-directed signature:** Are algorithms[5] producing a universally verifiable signature on m from a valid (A, B)-directed signature on m *i.e.* $\sigma_A \leftarrow \texttt{DS.Signer.Convert}(\Sigma, m, sk_A, pk_B)$ and $\sigma_B \leftarrow \texttt{DS.Confirmer.Convert}(\Sigma, m, pk_A, sk_B)$. We call σ_A an A-converted signature, and similarly σ_B is a B-converted one.

- **verification of an A or B-converted signature:** $v \leftarrow$DS.User.Verify-Signer(σ, m, pk_A, pk_B) and $v \leftarrow \texttt{DS.User.VerifyConfirmer}(\sigma, m, pk_A, pk_B)$ are deterministic algorithms that outputs valid or invalid depending on whether σ is a valid A-converted or B-converted signature on m, respectively.

- **generation of a universal trapdoor:** $t_{AB} \leftarrow \texttt{DS.Signer.Trapdoor}(sk_A, pk_B)$ and $t_{AB} \leftarrow \texttt{DS.Confirmer.Trapdoor}(pk_A, sk_B)$ are algorithms which output t_{AB} allows any user to noninteractively verify all (A, B)-directed signatures.[6]

- **universal signature verification:** $v \leftarrow \texttt{DS.User.Verify}(\Sigma, m, pk_A, pk_B, t_{AB})$ is a deterministic algorithm that outputs valid or invalid depending on whether Σ is a valid (A, B)-directed signature on m.

A secure *universally convertible directed signature* scheme must satisfy the following properties:

- **correctness:** Assuming all parties behave honestly, all (A, B)-directed, A-converted and B-converted signatures are always accepted as valid by running verification algorithm.

- **completeness and soundness:** An honest verifier C, always accepts a proof of validity (invalidity) of a valid (invalid) (A, B)-directed signature, from an honest prover (either A or B). Moreover, an honest C will not be convinced of the validity (invalidity) of an invalid (valid) (A, B)-directed signature.

[5]Here we do not require the algorithms be deterministic.

[6]We assume that t_{AB} can be verified from pk_A and pk_B.

- **(strong) unforgeability:** A DS-signature scheme is strongly unforgeable (sEF-CMA) if the advantage of an adversary to win the following game is negligible: After getting the system parameters and the public key pk_A of the target signer A, an adversary $\mathcal{A}$ is given adaptive access to oracles implementing all protocols in the system. Namely, $\mathcal{A}$ who chooses the confirmer keys (sk_B, pk_B) has access to the following oracles implementing A's capabilities: $\texttt{Sign}(m, pk_B)$, $\texttt{ConfirmOrDeny}(\Sigma, m, pk_B)$, $\texttt{Convert}(\Sigma, m, pk_B)$ and $\texttt{Trapdoor}(pk_B)$, which output $\texttt{DS.Sign}(m, sk_A, pk_B)$, $\textsf{DS.Signer.ConfirmOrDeny}(\Sigma, m, pk_A, pk_B)$, $\texttt{DS.Signer.Convert}(\Sigma, m, sk_A, pk_B)$ and $\texttt{DS.Signer.Trapdoor}(sk_A, pk_B)$ respectively. Eventually $\mathcal{A}$ stops and outputs a forgery (m, pk_B, Σ). $\mathcal{A}$ wins if Σ is a valid (A, B)-directed signature on m such that the signing oracle never returned Σ for any query (m, pk_A, pk_B).

- **(strong) invisibility:** We say that the signatures are strongly invisible (Inv-CMA), if the advantage of an adversary to win the following game with a challenger is negligible: After getting the system parameters and the public keys pk_A and pk_B^*, the adversary $\mathcal{A}$ chooses a message $m^* \in \{0,1\}^*$ and sends it to the challenger. Then the challenger chooses a random bit $b \leftarrow \{0,1\}$ and issues $\Sigma^* \leftarrow \texttt{DS.Sign}(m^*, sk_A, pk_B^*)$ to $\mathcal{A}$ if $b = 0$, or a random Σ^* in the (A, B^*)-directed signature space otherwise. $\mathcal{A}$ ends the game by outputting a guess bit b'. $\mathcal{A}$ wins the game if $b' = b$. During the whole game $\mathcal{A}$ has access to the oracles $\texttt{Sign}(m, pk_B)$, $\texttt{ConfirmOrDeny}(\Sigma, m, pk_B)$, $\texttt{Convert}(\Sigma, m, pk_B)$ and $\texttt{Trapdoor}(pk_B)$ as explained above, for arbitrary pk_B, with the only limitation that (Σ^*, m^*, pk_B^*) is not queried to `ConfirmOrDeny` or `Convert`, and pk_B^* is not queried to `Trapdoor`.[7]

4.2.1 Multi-Trapdoor Schemes

We introduce some technical modifications to the previous definition in order to allow a signer and a confirmer to use different trapdoors. Hence the trapdoor associated to (pk_A, pk_B) is not unique and must to be registered by some mechanism. Namely we add the following subprotocol:

- **registration of a trapdoor:** $(T, t) \leftarrow \texttt{DS.TrapReg}(pk_A, pk_B)$ is a two-party protocol run by A, with private input sk_A, and B, with private input sk_B, that outputs a trapdoor t, private to A and B, and a public

[7]A weaker notion of invisibility is also considered in [81], in which $\mathcal{A}$ has no access to the oracles `Convert` and `Trapdoor`.

information T related to it. A also receives and stores a certificate C_{AB} issued by B (*e.g.*, a signature on $pk_A||T$) which allows any other user C to verify T. Both A and B stores $(pk_A, pk_B, T, C_{AB}, t)$. We also assume the existence of an algorithm `DS.TrapVer` that verifies the validity of t given the tuple (pk_A, pk_B, T, t).

We will call this new signatures (A, B, T)-directed signatures, to make explicit the dependency with the trapdoor information T. All subprotocols must be modified accordingly in such a way that T is added as an additional input whenever pk_A or pk_B is used. The trapdoor certificate tuple (pk_A, pk_B, T, C_{AB}) must be available to every user in the system (*e.g.*, attaching to each (A, B)-directed signature the information (T, C_{AB}), or making this information available on a public server).[8] As the signer A and the recipient C will only accept trapdoors T certified by the corresponding confirmer B, `Sign`, `ConfirmOrDeny`, `Convert` and `Trapdoor` will reject all unregistered trapdoors.

This modification also has some effect on the security notions. Namely, in the unforgeability game the adversary is given access to the same oracles `Sign`(m, pk_B, T), `ConfirmOrDeny`(Σ, m, pk_B, T), `Convert`(Σ, m, pk_B, T) and `Trapdoor`(pk_B, T), as well as the additional oracle `TrapReg`(pk_B, T, t, C_{AB}), which adds T as a registered trapdoor information with certificate C_{AB} for the trapdoor t. The previous oracles reject all queries corresponding to unregistered trapdoors. Now $\mathcal{A}$ wins the game if he forges a valid (A, B, T)-directed signature Σ on m (even if T has not been registered) such that the signing oracle never returned Σ for any query (m, pk_B, T).

Similarly, in the invisibility game $\mathcal{A}$ receives the target pk_A, pk_B^*, T^* and C_{AB}^*, and has access to `Sign`(m, pk_B, T), `ConfirmOrDeny`(Σ, m, pk_B, T), `Convert`(Σ, m, pk_B, T), `Trapdoor`(pk_B, T) and `TrapReg`(pk_B, T, t, C_{AB}).[9] Here another oracle `NewTrap`$()$ provides the adversary with public information $(pk_A, pk_B^*, T, C_{AB})$ of a freshly generated trapdoor (which is further considered as a registered one). Now $\mathcal{A}$ is prevented from submitting the tuple $(\Sigma^*, m^*, pk_B^*, T^*)$ to `ConfirmOrDeny` or `Convert`, and he cannot query (pk_B^*, T^*) to `Trapdoor`, where Σ^* is the target (A, B^*, T^*)-directed signature.

This extended functionality has the drawback of adding a pre-interaction between the signed and the potential confirmer. However, it is hard to believe that a practical application without such an interaction exists. Notice that B

[8]Observe that formerly any (A, B)-directed signature must be clearly associated with (pk_A, pk_B), so we are just adding more information to this association.

[9]T^* is considered as registered, so the adversary can query `Sign`, `ConfirmOrDeny` or `Convert` on T^*.

is not required to be on-line at (A, B, T)-directed signature issuing time. The main advantage of the modified scheme is that now A can decide in advance which class of (A, B, T)-directed signatures are converted at once, without affecting the invisibility of the other (A, B, T')-directed signatures.

Assuming the existence of such a pre-interaction, we can simplify the directed signature scheme by using the natural approach except that for `DS.Sign`, A and B play symmetric roles and all the secret information they need is the trapdoor t. Thus `DS.Signer.Verify`(Σ, m, sk_A, pk_B) and its confirmer version can both be replaced by a single `DS.Verify`$(\Sigma, m, pk_A, pk_B, T, t)$ and similarly with `DS.ConfirmOrDeny`, `DS.Convert` and so on.

4.3 A Comment on the First Proposal

Here we briefly explain the protocol proposed by Laguillamie, Paillier and Vergnaud in [81] and discover a flaw in their proof of weak-invisibility.

Assume that (q, G, G_T, g, e) is a bilinear group system (for simplicity we only refer to the symmetric case), where $G = \langle g \rangle$ and G_T are groups of prime order q and $e : G \times G \to G_T$ is an efficiently computable non-degenerated bilinear map. We will address only the subprotocols defining the scheme that are relevant to the proof of weak invisibility:

- DS.Signer.KeyGen: Signer A picks two random $x_1, x_2 \in \mathbb{Z}_q^*$ and publishes $X_1 = g^{x_1}$ and $X_2 = g^{x_2}$ as his public key.

- DS.Confirmer.KeyGen: Confirmer B chooses a random $y \in \mathbb{Z}_q^*$ and publishes $Y = g^y$ as his public key.

- DS.Sign: Given a message $m \in \{0, 1\}^*$ and the public key Y of the confirmer, the signer picks a random $r \in \mathbb{Z}_q^*$ and computes $U = g^r$ and $V = Y^{\frac{rx_1}{x_2 + H(m,U,Y)}}$ where H is a hash function (which is modeled in some security proofs as a random oracle). The directed signature is $\Sigma = (U, V)$.

- DS.Confirmer.Verify: Given a message m and a signature $\Sigma = (U, V)$, the confirmer checks that $e(V, g^{x_2 + H(m,U,Y)}) = e(X_1, U)^y$.

- DS.Signer.Verify: Given a message m and a signature $\Sigma = (U, V)$, the signer checks whether $e(V, g^{x_2 + H(m,U,Y)}) = e(Y, U)^{x_1}$.

Theorem 2 in [81] claims the scheme's weak-invisibility is based on the DTDH Assumption. To prove that, they consider the adversary $\mathcal{A}$ breaking the invisibility of the system with a non-negligible probability $\varepsilon_{\mathcal{A}}$, and based on it they construct another adversary $\mathcal{B}$ which solves the DTDH Problem with a non-negligible probability $\varepsilon_{\mathcal{B}}$.

Basically, $\mathcal{B}$ receives a DTDH instance description $(X_0 = g^{\alpha}, Y_0 = g^{\beta}, Z_0 = g^{\gamma}, T_0 = g^{\delta})$, then sets up the public keys of the signer and the confirmer as $X_1 = X_0$, $X_2 = g^{x_2}$, $Y = Y_0$, where x_2 is a random element in $\mathbb{Z}_q^*$, and uses the remaining part of the instance information to generate the target directed signature $\Sigma_* = (U_*, V_*)$ on the message m_* selected by $\mathcal{A}$. Namely, $\mathcal{B}$ computes $U_* = Z_0$, then he flips a coin $b \in \{0, 1\}$ and if $b = 0$ he sets $V_* = T_0^{\overline{x_2+H(m_*,U_*,Y)}^{\,1}}$, otherwise V_* is a random element in G_1. The signing queries of $\mathcal{A}$ (for the same target confirmer) are answered in a similar way. On a query m_i, $\mathcal{B}$ picks a random $s_i \in \mathbb{Z}_q^*$ and computes the directed signature $\Sigma_i = (U_i, V_i)$ as $U_i = U_*^{s_i}$ and $V_i = T_0^{\overline{x_2+H(m_i,U_i,Y)}^{\,s_i}}$. $\mathcal{B}$ ends his game by outputting 1 (meaning that he guesses $\delta = \alpha\beta\gamma$) whenever $\mathcal{A}$'s output is $b' = b$.

As claimed in [81], if $\delta = \alpha\beta\gamma$ the simulation is perfect and $\mathcal{A}$ and $\mathcal{B}$ have the same success (conditional) probability. However, when δ is a random value unrelated to α, β or γ then the the simulation is no longer perfect. Actually, $\mathcal{A}$ can behave arbitrarily and the probability that $b' = b$ can in principle be any number ranging from 0 to 1. Therefore it is wrong to directly infer that, when δ is random, $b' = b$ happens with a probability negligibly close to 1/2, unless a new reduction to some hard problem or a hybrid argument is used. One could be tempted to use independence of b and $\mathcal{A}$'s view, but it is definitely false unless no sign queries are made by $\mathcal{A}$. Notice that from a signature Σ_1 answered by $\mathcal{B}$, $b = 0$ if and only if $V_*^{s_1(x_2+H(m_*,U_*,Y))} = V_1^{x_2+H(m_1,U_1,Y)}$, where s_1 and x_2 are uniquely defined by $U_1 = U_*^{s_1}$ and $X_2 = g^{x_2}$. Actually, independence could be reached if the signing oracle is perfectly simulated even when δ is random, and thus the proof would be correct as $\mathcal{A}$ would have advantage 0 in guessing b. However, this basically means using g^{δ} only in the target signature, and not in the signing oracle simulation. And there is no obvious way to do so (mainly because the capability of generating a correct signature is related to the problem of computing $(g^r, g^{r\alpha\beta})$ given (g^{α}, g^{β}), and this is harder than directly solving the DTDH problem). In the DTDH-based instance of our construction we use a different approach: only one of the three DTDH exponents is placed into the keys, and the other two are used as (part of) the randomness of the signature, which makes the simulation straightforward.

Although there is no trivial fix for the proof, in [81] it is also given a proof of strong invisibility but based on a much stronger assumption, which instance description length grows linearly with the number of signing queries. Then, the scheme presented in [81] cannot be considered as broken.

4.4 Stateless Commitments

As it was explained in the second chapter, in normal commitment schemes, the committer stores the committed value m and the randomness r used to compute a commitment $c = \texttt{Commit}(m; r)$ so that it can be opened afterwards. Thus, the committer is supposed to maintain a storage which grows linearly with the number of committed values. In this section we introduce a new type of commitment, the *stateless commitment*, which allows the committer to be stateless. Actually, the committer keeps a secret trapdoor sk that allows him to open a commitment $c = \texttt{Commit}(m; r)$ without knowing the randomness r.

More precisely, two different parties are considered: the trapdoor owner P and the (stateless) committer C. Opening a stateless commitment $c = \texttt{Commit}(m; r)$ will mean proving (either in a transferable or a non-transferable way) that a certain message m is contained in c. The syntax of a stateless commitment is the following:

- **system setup:** $\texttt{SC.param} \leftarrow \texttt{SC.Setup}(1^\ell)$, where ℓ is the security parameter and $\texttt{SC.param}$ are the public system parameters of the commitment scheme.

- **trapdoor generation:** $(t, T) \leftarrow \texttt{SC.TrapGen}()$ is a probabilistic algorithm run by P that generates a trapdoor t and its corresponding public information T.

- **commit:** $c \leftarrow \texttt{SC.Commit}(m, T; r)$ is a probabilistic algorithm that outputs a commitment to m with respect to the trapdoor information T, using random coins r.

- **open:** $\pi \leftarrow \texttt{SC.Open}(c, m, t)$ is an algorithm that on the input of a commitment c, a trapdoor t and a message m, it outputs a validity proof π, if $c = \texttt{SC.Commit}(m, T; r)$ for some r. Otherwise, it outputs $\perp$. We also require the existence of an efficient algorithm $\texttt{SC.PubOpen}$ that computes π directly from m, T and r.

- **verify:** $v \leftarrow \texttt{SC.Verify}(c, m, T, \pi)$ is a deterministic algorithm that outputs valid or invalid depending on whether or not $c = \texttt{SC.Commit}(m, T; r)$, for some r.

- **trapdoor verification:** $v \leftarrow \texttt{SC.TrapVerify}(T, t)$ is a deterministic algorithm that outputs valid or invalid depending on whether or not a trapdoor t corresponds to T.

Observe that in the above definition the committer does not need to know the trapdoor t. Interestingly, the commitments can be opened by both a stateful committer not knowing the trapdoor, or by the trapdoor owner (who could be a different person). Furthermore, revealing the trapdoor opens all commitments at once.

The security requirements for a stateless commitment are:

- **correctness:** For every commitment computed as $c = \texttt{SC.Commit}(m, T; r)$ and every proof $\pi = \texttt{SC.Open}(c, m, t)$ it holds that $\texttt{SC.Verify}(c, m, T, \pi) = \texttt{valid}$.

- **soundness:** For every commitment c, every message m and every string π, $\texttt{SC.Verify}(c, m, T, \pi) = \texttt{valid}$ implies that there exists r such that $c = \texttt{SC.Commit}(m, T; r)$. Observe that the definition of SC.Open implies that if $\pi = \texttt{SC.Open}(c, m, t) \neq \perp$ then $\texttt{SC.Verify}(c, m, T, \pi) = \texttt{valid}$.

- **binding:** Given T it is infeasible for an adversary to compute two different m_0 and m_1 such that there exist r_0 and r_1 such that $\texttt{SC.Commit}(m_0, T; r_0) = \texttt{SC.Commit}(m_1, T; r_1)$.

- **computational hiding:** It is infeasible to guess a random $b \in \{0, 1\}$ given T and $c = \texttt{SC.Commit}(m_b, T; r)$, for two adversarially chosen m_0, m_1 with the same length.

  As usually binding can be considered as computational, statistical or perfect depending on the notion of infeasibility. Namely, it is perfect if even an unbounded adversary has success probability zero (because $\texttt{SC.Commit}(\cdot, T; \cdot)$ is injective); it is statistical if an unbounded adversary has only a negligible success probability; and it is computational if a polynomially constrained adversary has only a negligible success probability.

On the other hand, one can consider different definitions of computational hiding based on the different oracles given to the adversary. Namely, IND-CMA (for INDistinguishability of messages under Chosen Message Attack) is the definition given above, while IND-CCA (for INDistinguishability of messages under Chosen Commitment Attack) is obtained by giving the adversary access to the Open oracle on adaptively chosen message/commitment pairs. Although constructing stateless commitments with IND-CCA security is interesting, here we only need IND-CMA secure commitments.

- **smoothness:** For a random message m, $\texttt{SC.Commit}(m, T; r)$ is uniformly distributed in the commitment space.

4.4.1 Confirm-or-Deny Capabilities

Optionally, on the common input of the trapdoor information T, a commitment c and a message m, the trapdoor owner, with private input t, can be able to prove to another party V in a non-transferable way that he knows a proof π such that $\texttt{SC.Verify}(c, m, T, \pi) = \mathsf{valid}$, provided that $c = \texttt{SC.Commit}(m, T; r)$ for some r; or a proof π' that $c \neq \texttt{SC.Commit}(m, T; r)$ for all r. We call this subprotocol `SC.ConfirmOrDeny`. Notice that this protocol could imply adding more system parameters (like the description of a trapdoor commitment scheme as in the concurrent zero-knowledge proofs in [38]), and the verifier V would need some registered key. As above, we also require that both π and π' can be efficiently computed from (m, T, r), so that the knowledge of either t or r is enough to run `SC.ConfirmOrDeny`.

We refer to this extended stateless commitment scheme as a commitment scheme *with confirm-or-deny capabilities*. For this type of stateless commitments, the security notions of correctness and soundness are extended in the natural way, requiring honest verifiers to always accept proofs from honest provers, and an honest verifier accepting a confirmation (denial) proof from a dishonest prover implies the commitment is valid (invalid). But this extra requirements are directly fulfilled when a zero-knowledge proof is used in the subprotocol `SC.ConfirmOrDeny`.

4.4.2 Constructions (With Perfect Binding)

In this section we propose two generic constructions for constructing (IND-CMA) stateless commitments and then propose an efficient instantiation for each generic scheme secure in the standard model. The first instance uses the

Paillier encryption [96] and the second stateless commitment is a new commitment working in pairing-based groups that is based on DTDH assumption.

1. Actually, any IND-CPA encryption scheme can be considered by itself as a stateless commitment scheme, provided that the encryption of a random message is uniformly distributed in the ciphertext space and the encryption is randomness-recovering, *i.e.* the decryption results in both the message and the randomness. Then opening the commitment is just decrypting $c = \texttt{PKE.Enc}(m; r)$ so obtaining the randomness $\pi = r$, from which anyone can check $c = \texttt{PKE.Enc}(m; r)$.

   An efficient example of such a stateless commitment scheme is Paillier's encryption [96], where $\texttt{SC.Commit(m,N;r)} = r^N(1 + mN) \bmod N^2$, being $N = pq$ an RSA-modulus. Furthermore, confirm-or-deny capabilities can be easily added by using the zero-knowledge proofs[10] described in the Appendix. Namely, given a commitment c and a message m, ZKPoK$[r \in \mathbb{Z}_N^* \ : \ c(1 - mN) \equiv r^N \pmod{N^2}]$ is used for confirmation, while ZKPoK$\{r \in \mathbb{Z}_N^*, \gamma \in \mathbb{Z}_N \setminus \{0\} : c(1 - mN) \equiv r^N(1 + \gamma N) \pmod{N^2}\}$ is used for denial. So here $\pi = \pi' = r$.

   The `SC.Open()` protocol gives a way to "open" a single ciphertext by proving that it decrypts to a given message, without affecting the secrecy of other ciphertexts encrypted under the same public key. Actually, this functionality is in close relation to the recently introduced notion of *Public-Key Encryption with Noninteractive Opening (PKENO)* [42, 40]. However, the usual security notion for a PKENO scheme is called IND-CCPA, which takes into account chosen ciphertext and prove attacks, while we only consider here the weaker IND-CMA notion for a commitment scheme.

2. It is worth noticing that stateless commitments does not necessarily have extractability properties, *i.e.*, no efficient function computing m from $c = \texttt{SC.Commit}(m, T; r)$ is required to exist. As a consequence stateless commitment schemes not based on encryption exist. We propose the following stateless commitment scheme that is unconditionally binding, and its hiding property is based on the DTDH assumption.[11]

[10]We use the standard notation ZKPoK$[x \in X \ : \ \textit{statement}(x)]$ for a zero-knowledge proof of knowledge of $x \in X$ such that $\textit{statement}(x)$ holds.

[11]A more general version of DTDH is given in [81] for asymmetric pairing groups.

- **System Setup:** On input a security parameter 1^ℓ, a bilinear group system (q, G, G_T, g, e) is generated, such that q is an ℓ-bit prime. Let `SC.param` $= (\ell, q, G, G_T, g, e)$

- **Trapdoor Generation:** The trapdoor is generated as $(z, Z = g^z)$ where z is a random element in $\mathbb{Z}_q^*$.

- **Commit:** To commit to a message $m \in \mathbb{Z}_q$ with trapdoor information Z, choose two random elements x and y from $\mathbb{Z}_q$ and compute $c = (g^x, g^y, Z^{xy}g^m)$.

- **Open:** To open a commitment $c = (X, Y, S)$ for a message m with trapdoor z, compute the proof $\pi = (Sg^{-m})^{1/z}$, then check whether $e(X, Y) = e(\pi, g)$, and if so output π, or $\perp$ otherwise. Observe that π can be computed directly from the randomness x, y by the public algorithm $\pi = g^{xy}$.

- **Verify:** To verify a commitment $c = (X, Y, S)$ for a message m and public trapdoor Z, with a proof π one checks whether $e(X, Y) = e(\pi, g)$ and $e(Sg^{-m}, g) = e(\pi, Z)$.

- **Trapdoor Verification:** Given (z, Z) check whether $Z = g^z$.

- **Non-transferably Confirm or Deny:** To open a commitment $c = (X, Y, S)$ for a message m in a non-transferable way, the trapdoor owner computes $\pi = (Sg^{-m})^{1/z}$, then checks whether $e(X, Y) = e(\pi, g)$, and if so he outputs `valid` and proves in zero knowledge $\mathrm{ZKPoK}\{\pi \in G : e(X, Y) = e(\pi, g) \wedge e(Sg^{-m}, g) = e(\pi, Z)\}$. Otherwise, he outputs `invalid` and proves in zero knowledge $\mathrm{ZKPoK}\{\pi \in G : e(X, Y) \neq e(\pi, g) \wedge e(Sg^{-m}, g) = e(\pi, Z)\}$. In the last section of this chapter, the reader can find the details of the proof given in confirmation or denial based on the two proposals for stateless commitments.

Remark 4.4.1 *Observe that the non-transferable confirmation protocol explained above can be done in another manner, that is, a modification of Chaum-Pedersen proof of the equality of discrete logarithms of two different elements in two different groups. In this way to prove the consistency of the commitment $C = (X, Y, S)$ on the message m,* *the prover after checking the consistency by himself just needs to run ZKPoK*$\{z : Z = g^z \wedge e(Sg^{-m}, g) = e(X, Y)^z\}$.

Security analysis:

- **Correctness, Soundness and Binding:** Are straightforward.

- **Computational Hiding:** It is infeasible to guess a random $b \in \{0, 1\}$, given random X, Y, Z and $c = (X, Y, S) = \texttt{SC.Commit}(m_b, Z; x, y)$, for two adversarially chosen m_0, m_1. Actually, any adversary $\mathcal{A}$ doing so with advantage ε can be converted into another one $\mathcal{B}$, which uses $\mathcal{A}$ as a subroutine, breaking the DTDH assumption with advantage $\varepsilon/2$, nearly within the same time. Indeed, one can plug a DTDH instance (X_0, Y_0, Z_0, T_0) into the challenge commitment for $\mathcal{A}$ by setting $Z = Z_0$, $c = (X_0, Y_0, T_0 g^{m_b})$, for a random bit b. If $(X_0, Y_0, Z_0, T_0) = D_{\text{DTDH}}$ then the simulation for $\mathcal{A}$ is perfect. Otherwise, $(X_0, Y_0, Z_0, T_0) = D_{\text{random}}$ and the view of $\mathcal{A}$ is independent of b. So $\mathcal{B}$ decides the instance is D_{DTDH} whenever $\mathcal{A}$ guesses b correctly.
- **Smoothness:** Is straightforward as x, y and m are uniformly distributed random values.

4.5 Detailed Confirm/Deny Protocols

Regarding two stateless commitments considered in the previous section, the confirmation or denial protocol will be based on the following zero-knowledge proofs.

- Given a pairing system (q, G, G_T, g, e), some $h \in G \setminus \{1\}$ and $T \in G_T$, we describe a Σ-protocol for ZKPoK$[X \in G : e(h_1, X) = T]$ between a prover P and a verifier V.
  - **common input:** (q, G, G_T, g, e), $h \in G \setminus \{1\}$, $T \in G_T$.
  - **statement:** Knows $X \in G$ such that $e(h_1, X) = T$.
  - **protocol:**
    1. P sends $A = e(h, Y)$, for a random $Y \in G$, to V
    2. V sends a random challenge $c \in \mathbb{Z}_q$
    3. P responds with $S = YX^c$
    4. V checks whether $e(h, S) = AT^c$

- **knowledge extraction:** From any two accepting conversations (A, c_1, S_1) and (A, c_2, S_2) where $c_1 \neq c_2$, one can efficiently extract $X = (S_1 S_2^{-1})^{1/(c_1-c_2)}$.
- **simulatability:** Given c, pick a random $S \in G$ and compute $A = e(h, S)T^{-c}$.

• Similarly, a Σ-protocol for ZKPoK$[X \in G : e(h_1, X) = T_1 \wedge e(h_2, X) = T_2]$ follows, based on the typical AND-proof.

- **common input:** (q, G, G_T, g, e), $h_1, h_2 \in G \setminus \{1\}$, $T_1, T_2 \in G_T$.
- **statement:** Knows $X \in G$ such that $e(h_1, X) = T_1$ and $e(h_2, X) = T_2$.
- **protocol:**
  1. P sends $A_1 = e(h_1, Y_1)$ and $A_2 = e(h_2, Y_2)$, for random $Y_1, Y_2 \in G$, to V
  2. V sends a random challenge $c \in \mathbb{Z}_q$
  3. P responds with $S_1 = Y_1 X^c$ and $S_2 = Y_2 X^c$
  4. V checks whether $e(h_1, S_1) = A_1 T_1^c$ and $e(h_2, S_2) = A_2 T_2^c$
- **knowledge extraction:** From any two accepting conversations (A_1, A_2, c, S_1, S_2) and $(A_1, A_2, c', S_1', S_2')$ where $c \neq c'$ one can efficiently extract $X = (S_1 S_1'^{-1})^{1/(c-c')} = (S_2 S_2'^{-1})^{1/(c-c')}$.
- **simulatability:** Given c, pick random $S_1, S_2 \in G$ and compute $A_1 = e(h_1, S_1)T_1^{-c}$ and $A_2 = e(h_2, S_2)T_2^{-c}$.

• Now a Σ-protocol for ZKPoK$[X \in G : e(h_1, X) = T_1 \wedge e(h_2, X) \neq T_2]$ follows.

- **common input:** (q, G, G_T, g, e), $h_1, h_2 \in G \setminus \{1\}$, $T_1, T_2 \in G_T$.
- **statement:** Knows $X \in G$ such that $e(h_1, X) = T_1$ but $e(h_2, X) \neq T_2$.
- **protocol:**
  1. P sends $A_0 = (e(h_2, X)T_2^{-1})^\alpha$, $A_1 = e(h_1, Y)T_1^\gamma$ and $A_2 = e(h_2, Y)T_2^\gamma$, for random $\alpha, \gamma \in \mathbb{Z}_q^*$, and $Y \in G$, to V
  2. V sends a random challenge $c \in \mathbb{Z}_q$
  3. P responds with $S_1 = Y X^{\alpha c}$ and $s_2 = \gamma - \alpha c$
  4. V checks whether $A_0 \neq 1$, $e(h_1, S_1)T_1^{s_2} = A_1$ and $e(h_2, S_1)T_2^{s_2} = A_2 A_0^c$

  - **knowledge extraction:** From any two accepting conversations $(A_0, A_1, A_2, c, S_1, s_2)$ and $(A_0, A_1, A_2, c', S_1', s_2')$ where $c \neq c'$ one can efficiently extract $X = (S_1 {S_1'}^{-1})^{1/(s_2'-s_2)}$ and show that $e(h_2, X) = A_0^{(c-c')/(s_2'-s_2)} T_2 \neq T_2$, since $A_0 \neq 1$.
  - **simulatability:** Given c, pick random $A_0, S_1 \in G$ and $s_2 \in \mathbb{Z}_q$, and compute $A_1 = e(h_1, S_1) T_1^{s_2}$ and $A_2 = e(h_2, S_2) T_2^{s_2} A_0^{-c}$.

- Similar proofs for factoring based systems exist. For instance, given an RSA modulus $N = pq$, one can prove whether or not $z \in \mathbb{Z}_{N^2}^*$ is an N-residue, by proving the knowledge of $x \in \mathbb{Z}_N^*$ such that $z \equiv x^N \pmod N$ and $z \equiv x^N \pmod{N^2}$ or not, respectively.

  - **common input:** (N, z).
  - **statement:** Knows $x \in \mathbb{Z}_N^*$ such that $z \equiv x^N \pmod{N^2}$.
  - **protocol:**
    1. P sends $a = y^N \bmod N^2$, for a random $y \in \mathbb{Z}_N^*$, to V
    2. V sends a random challenge $c \in \mathbb{Z}_N$
    3. P responds with $s = yx^c \bmod N$
    4. V checks whether $s^N \equiv az^c \pmod{N^2}$
  - **knowledge extraction:** From any two accepting conversations (a, c, s) and (a, c', s') where $c \neq c'$ one can efficiently extract $x = (s/s')^\alpha z^\beta \bmod N$, where α, β are any solution to the equation $\alpha(c - c') + \beta N = 1$. Actually, there is no solution for α, β when $\mu = \gcd(N, c - c') \neq 1$. But then μ is a nontrivial factor of N, so P can directly compute $x = z^{1/N} \bmod N$.
  - **simulatability:** Given c, pick a random $s \in \mathbb{Z}_N^*$ and compute $a = s^N z^{-c} \bmod N^2$.

- The following proof is statistical zero-knowledge. Assume t is large enough so that 2^{-t} is a negligible probability.

  - **common input:** (N, z).
  - **statement:** Knows $x \in \mathbb{Z}_N^*$ and $\gamma \in \mathbb{Z}_N$ such that $z \equiv x^N(1 + \gamma N) \pmod{N^2}$ and $\gamma \neq 0$.
  - **protocol:**
    1. P sends $a = y^N z^\beta \bmod N^2$ and $\tau = \alpha\gamma \bmod N$, for random $\alpha \in \mathbb{Z}_N^*$, $\beta \in \mathbb{Z}_{N^2}$ and $y \in \mathbb{Z}_N^*$, to V
    2. V sends a random challenge $c \in [0, 2^t - 1]$
    3. P responds with $s_1 = yx^{\alpha c} \bmod N$, $s_2 = \beta - c\alpha$

4. V checks whether $\tau \neq 0$ and $s_1^N \equiv z^{-s_2} a(1 - c\tau N) \pmod{N^2}$

– **knowledge extraction:** From any two accepting conversations (a, τ, c, s_1, s_2) and $(a, \tau, c', s_1', s_2')$ where $c \neq c'$ one can efficiently extract $x = (s_1 {s_1'}^{-1})^\lambda z^\mu \bmod N$ and $\gamma = \lambda\tau(c - c') \bmod N$, where λ, μ fulfil the equation $\lambda(s_2 - s_2') + \mu N = 1$. If no solution for λ, μ exists then a nontrivial factor of N is revealed, so x and λ are easily extracted from z.

– **(statistical) simulatability:** Given c, pick random $\tau, s_1 \in \mathbb{Z}_N^*$, $s_2 \in \mathbb{Z}_{N^2}$ and compute $a = s_1^N z^{s_2}(1 + c\tau N) \bmod N^2$.

Following the work by Damgård [38] all these Σ-protocols can be easily converted into efficient concurrent zero knowledge arguments in the common reference string model.

4.6 A Generic Construction

In this section we describe a generic construction of a universally convertible directed signature scheme `DS` from any strongly unforgeable signature scheme `SIG` = (`SIG.Setup`, `SIG.KeyGen`, `SIG.Sign`, `SIG.Verify`) and any stateless commitment scheme `SC` with confirm-or-deny capabilities. For simplicity we only consider schemes with perfect binding, although the construction and security analysis for a more general case are similar. The new scheme supports multiple trapdoors for the same signer/confirmer pair. So we use the extended syntax given in Section 4.2 in the description of the scheme. Assume that A, B and C are the identities of the signer, the confirmer and the recipient of the signature, respectively. We assume for simplicity that the basic signature scheme used by both A and B is the same instance of SIG. The non-transferable proofs implied by the confirmation or denial protocols could imply the use of a Common Reference String or the use of designated-verifier proofs. We assume that the proof system is specified as part of the stateless commitment scheme.

- **system setup:** On input a security parameter 1^k, compute the system parameters as `DS.param` = (`SIG.param`, `SC.param`), where `SIG.param` $\leftarrow$ `SIG.Setup`(1^k) and `SC.param` $\leftarrow$ `SC.Setup`(1^k).

- **key generation:** Let $(sk_A, pk_A) \leftarrow$ `SIG.KeyGen` and $(sk_B, pk_B) \leftarrow$ `SIG.KeyGen` be the key pairs for A and B. C would need also an additional key generation algorithm for the non-transferable proofs.

- **key registration:** Any user in the system proves knowledge of its secret key to a "Key Registration Authority", who sign the corresponding secret key as a certificate.

- **registration of a trapdoor:** A and B run a two-party protocol that outputs a trapdoor t, privately to A and B, and a public information T related to it. To that end, A can run $(t, T) \leftarrow \texttt{SC.TrapGen}()$ and send (t, T) privately to B, who responds with a certificate $C_{AB} = \texttt{SIG.Sign}(sk_B, pk_A||T)$, which is verified by A.[12] Both parties store the tuple $(pk_A, pk_B, T, C_{AB}, t)$.

- **(A, B, T)-directed signature generation:** On the input of a message m, and a (registered) trapdoor information T, A computes $c = \texttt{SC.Commit}(m, T; r)$ for a random r. Then he computes $\sigma = \texttt{SIG.Sign}(pk_A, c||T||pk_B)$. The (A, B, T)-directed signature is $\Sigma = (c, \sigma)$, which is intended to be used in combination to the certificate tuple (pk_A, pk_B, T, C_{AB}). Observe that any recipient C can noninteractively prove that $\mathsf{SIG.Verify}(pk_A, \sigma, c||T||pk_B) = \mathsf{valid}$ and that (pk_A, pk_B, T, C_{AB}) is a valid certificate tuple.

- **(A, B, T)-directed signature verification:** Either A or B (say P) can non-interactively verify an (A, B, T)-directed signature $\Sigma = (c, \sigma)$ on a message m under trapdoor information T. Firstly, P checks whether $\texttt{SIG.Verify}(pk_A, \sigma, c||T||pk_B) = \texttt{valid}$. Then P looks up the certificate table for an entry $(pk_A, pk_B, T, C_{AB}, t)$ and runs $\pi = \texttt{SC.Open(c,m,t)}$. Σ is considered as valid if and only if all previous steps are fulfilled (so $\pi \neq \perp$).

- **(A, B, T)-directed signature confirmation/denial:** Assuming P (either A or B) and C checked the consistency of σ and the tuple (pk_A, pk_B, T, C_{AB}), where $\Sigma = (c, \sigma)$ is the (A, B, T)-directed signature under verification for a message m and trapdoor information T, then P runs with C the $\texttt{SC.ConfirmOrDeny}$ protocol to non-transferably prove whether $\texttt{SC.Open(c,m,t)} \neq \perp$.

- **individual conversion of an (A, B, T)-directed signature:** For a registered trapdoor information T, P (either A or B) runs all the checks

[12]In some cases, (t, T) could be the output of a secure key-agreement protocol.

in $\texttt{DS.Verify}(\Sigma, m, pk_A, pk_B, T)$, and if $\pi \neq \perp$ then P sends the converted signature (c, T, σ, π), or $\perp$ otherwise.

- **verification of a converted signature:** To verify a converted signature (c, T, σ, π) on a message m, where T is registered, C checks whether $\texttt{SIG.Verify}(pk_A, \sigma, c||T||pk_B) = \texttt{valid}$ and $\texttt{SC.Verify}(c, m, T, \pi) = \texttt{valid}$.

- **generation of a universal trapdoor:** To make all the (A, B, T)-directed signatures verifiable, A or B publishes the corresponding trapdoor t, as long as a valid tuple $(pk_A, pk_B, T, C_{AB}, t)$ exists.[13]

- **universal signature verification:** To verify an (A, B, T)-directed signature $\Sigma = (c, \sigma)$ on a message m when the trapdoor t has been revealed, check that $\texttt{SC.Verify}(c, m, t) = \texttt{valid}$ and $\texttt{SIG.Verify}(pk_A, \sigma, c||T||pk_B) = \texttt{valid}$.

Observe that the scheme can be easily adapted to a designated confirmer signature scheme in which only the verifier but not the signer can verify or convert any signature after it has been issued. To that end, the trapdoors (t, T) are generated by the designated verifier B, but only T and the certificate C_{AB} are sent to the signer.

4.7 Instantiations

To construct an efficient universally convertible signature scheme in the standard model we propose two instantiations based on the proposed stateless commitments. The first instance is based on factoring-related assumptions, and uses the Paillier encryption based stateless commitments described in Section 4.4 together with the strongly unforgeable signature by Cramer and Shoup [36]. Notice that both schemes must use independent public keys (RSA modulus) because one is used for the trapdoor information while the other is for the signatures, which must remain unforgeable even after universal conversion. The main advantage of the scheme is that it is an efficient instance secure in the standard model that can be easily implemented on existing platforms using RSA, and even reusing the existing signing keys.

[13] t can be verified from T by means of $\texttt{SC.TrapVerify}$.

The second instance is based on pairing groups and it uses our second stateless commitment based on the DTDH assumption and the strongly unforgeable signature by Boneh and Boyen [13]. Both components can share the same pairing group and the resulting instance is very efficient and leads to very short directed signatures (with an overhead of only three group elements with respect to the non-directed signature). The trapdoor information is just one group element.

4.8 Security Analysis

Correctness, completeness and soundness are straightforward from the corresponding properties of the building blocks.

4.8.1 Unforgeability

Theorem 4.8.1 *If* `SC` *is a perfect binding stateless commitment with confirm-or-deny capabilities and* `SIG` *is a strongly unforgeable signature scheme, then* `DS` $=$ (`SC`, `SIG`) *is strongly unforgeable.*

***Proof*.** Assume that there exists an adversary $\mathcal{A}$ that plays the unforgeability game described in Section 4.2 with a non-negligible success probability $\varepsilon_{\mathcal{A}}$. Then, we use $\mathcal{A}$ to build a forger $\mathbb{F}$ that breaks the strong unforgeability of the underlying signature scheme `SIG` with at least the same non-negligible probability, and roughly within the same time. A rough description of $\mathbb{F}$ is that every possible forgery (Σ, m, pk_B, T) by $\mathcal{A}$ contains a signature σ on $c||T||pk_B$ that can be used as a forgery by $\mathbb{F}$. From the binding property of the commitment scheme it is shown that $\mathbb{F}$ succeeds whenever $\mathcal{A}$ does. A detailed description of $\mathbb{F}$ follows.

- **startup:** $\mathbb{F}$ starts receiving the system parameters of the underlying signature scheme `SIG.param` and the signer's public key pk_A. Then $\mathbb{F}$ completes the system parameters `DS.param` and sends them to $\mathcal{A}$ along with pk_A. During the game $\mathbb{F}$ maintains a table with the registered trapdoors.

- **simulation of** `TrapReg`(pk_B, T, t, C_{AB})**:** $\mathbb{F}$ checks that `SC.TrapVerify` (T, t) = `valid` and `SIG.Verify`$(pk_B, C_{AB}, pk_A||T)$ = `valid`. If valid $\mathbb{F}$ stores $(pk_A, pk_B, T, C_{AB}, t)$ as a registered trapdoor (so he can later retrieve t for any registered T), and responds `valid`. Otherwise $\mathbb{F}$ responds `invalid`.

- **simulation of** $\texttt{Trapdoor}(pk_B, T)$**:** If (pk_A, pk_B, T) has not been registered then $\mathbb{F}$ responds $\perp$. Otherwise, $\mathbb{F}$ retrieves and sends the corresponding t from the table.

- **simulation of** $\texttt{Sign}(m, pk_B, T)$**:** If (pk_A, pk_B, T) has not been registered then $\mathbb{F}$ responds $\perp$. Otherwise, $\mathbb{F}$ computes $c = \texttt{SC.Commit}(m, T; r)$ for a random r and queries SIG's signing oracle to get a signature σ on $c||T||pk_B$. Then he sends the (A, B, T)-directed signature $\Sigma = (c, \sigma)$ to $\mathcal{A}$.

- **simulation of** $\texttt{ConfirmOrDeny}(\Sigma, m, pk_B, T)$ **and** $\texttt{Convert}(\Sigma, m, pk_B, T)$**:** If (pk_A, pk_B, T) has not been registered $\mathbb{F}$ responds $\perp$. Otherwise $\mathbb{F}$ retrieves t from the table and runs the corresponding protocol with $\mathcal{A}$ as in the real protocol.

- **end:** Eventually, $\mathcal{A}$ halts and outputs an (A, B, T)-forgery $\Sigma = (c, \sigma)$ on m with respect to (pk_B, T). Then $\mathbb{F}$ outputs the forgery $(\sigma, c||T||pk_B)$.

Is it easy to see that the above simulation is perfect. Furthermore, $\mathcal{A}$ succeeds whenever $\Sigma = (c, \sigma)$ is valid and different from any $\Sigma_i = (c_i, \sigma_i)$ returned by Sign oracle simulation on a query $(m_i, pk_{B_i}, T_i) = (m, pk_B, T)$. Thus $\mathbb{F}$ also succeeds. Otherwise $(\sigma, c||T||pk_B) = (\sigma_i, c_i||T_i||pk_{B_i})$, where σ_i is SIG signing oracle's answer on a query $c_i||T_i||pk_{B_i}$ made during some Sign simulation. But then $\Sigma_i = \Sigma$, $T_i = T$, $pk_{B_i} = pk_B$, and $c_i = c$. So $m_i = m$ due to the perfect binding of SC, and $\mathcal{A}$'s forgery is not valid. Thus,

$$\mathsf{Succ}_{\mathsf{DS}}^{\text{sEF-CMA}}[\mathcal{A}] \leq \mathsf{Succ}_{\mathsf{S}}^{\text{sEF-CMA}}[\mathbb{F}]$$

4.8.2 Invisibility

Theorem 4.8.2 *If SC is a smooth perfect binding stateless commitment with confirm-or-deny capabilities and SIG is a strongly unforgeable signature scheme, then DS = (SC, SIG) is strongly invisible.*

***Proof*.** Assume that there exists an adversary $\mathcal{A}$ that plays the invisibility game described in Section 4.2 with a non-negligible advantage $\varepsilon_{\mathcal{A}}$. Then, we use $\mathcal{A}$ to build both a forger $\mathbb{F}$ breaking the strong unforgeability SIG and an adversary $\mathcal{B}$ breaking the hiding property of the stateless commitment scheme SC. Both adversaries run roughly within the same time as $\mathcal{A}$ and at least one of them succeeds with a non-negligible probability. Basically $\mathbb{F}$ extracts a forgery from any signature queried by $\mathcal{A}$ to `ConfirmOrDeny` or `Convert` and not answered by `Sign` oracle, while $\mathcal{B}$ uses $\mathcal{A}$'s output bit to break the IND-CMA hiding of `SC` by putting the target commitment into $\mathcal{A}$'s challenge. We show that when $\mathcal{B}$'s simulation fails $\mathbb{F}$ can successfully forge a signature. We first describe $\mathbb{F}$:

- **startup:** During the game $\mathbb{F}$ maintains two tables, one with the registered trapdoors and the other with the issued signatures. $\mathbb{F}$ starts receiving the system parameters of the underlying signature scheme `SIG.param` and the public key pk_A of the target signer. Then $\mathbb{F}$ computes a signature key pair for the target confirmer B^* as $(sk_B^*, pk_B^*) \leftarrow \texttt{SIG.KeyGen}()$. Then he completes the system parameters `DS.param` and sends them to $\mathcal{A}$. A target trapdoor $(t^*, T^*) \leftarrow \texttt{SC.TrapGen}()$ and its certificate $C_{AB}^* \leftarrow \texttt{SIG.Sign}(sk_B^*, pk_A||T^*)$ are computed. Now $\mathbb{F}$ stores the tuple $(pk_A, pk_B^*, T^*, C_{AB}^*, t^*)$ so that T^* is considered as registered, then sends $(pk_A, pk_B^*, T^*, C_{AB}^*)$ to $\mathcal{A}$, and sets $\Sigma^* = \texttt{undefined}$ and $m^* = \texttt{undefined}$.

- **challenge generation:** Eventually $\mathcal{A}$ sends a message m^* to $\mathbb{F}$. Then $\mathbb{F}$ flips a fair coin obtaining $b \in \{0, 1\}$. Then he sets $\Sigma^* = (c^*, \sigma^*)$ and sends it to $\mathcal{A}$, where $c^* = \texttt{SC.Commit}(m^*, T^*; r^*)$ for a random r^* if $b = 0$, or a random c^* in the commitment space otherwise, and σ^* is the answer of `SIG`'s signing oracle on $c^*||T^*||pk_B^*$.

- **simulation of $\texttt{TrapReg}(pk_B, T, t, C_{AB})$:** $\mathbb{F}$ checks that $\texttt{SC.TrapVerify}(T, t) = \texttt{valid}$ and $\texttt{SIG.Verify}(pk_B, C_{AB}, pk_A||T) = \texttt{valid}$. If valid $\mathbb{F}$ stores $(pk_A, pk_B, T, C_{AB}, t)$ as a registered trapdoor and responds `valid`. Otherwise $\mathbb{F}$ responds `invalid`.

- **simulation of $\texttt{NewTrap}()$:** $\mathbb{F}$ computes $(t, T) \leftarrow \texttt{SC.TrapGen}()$ and $C_{AB} \leftarrow \texttt{SIG.Sign}(sk_B^*, pk_A||T)$. Then he stores the tuple $(pk_A, pk_B^*, T, C_{AB}, t)$ and sends $(pk_A, pk_B^*, T, C_{AB})$ to $\mathcal{A}$.

- **simulation of $\texttt{Trapdoor}(pk_B, T)$:** If (pk_A, pk_B, T) has not been registered or $T = T^*$ then $\mathbb{F}$ responds $\perp$. Otherwise, $\mathbb{F}$ sends the corresponding t from the table.

- **simulation of** $\texttt{Sign}(m, pk_B, T)$**:** If (pk_A, pk_B, T) has not been registered then $\mathbb{F}$ responds $\perp$. Otherwise, $\mathbb{F}$ computes $c = \texttt{SC.Commit}(m, T; r)$ for a random r and queries $\texttt{SIG}$'s signing oracle to get a signature σ on $c||T||pk_B$. Then he sends the (A, B, T)-directed signature $\Sigma = (c, \sigma)$ to $\mathcal{A}$, and stores the tuple (pk_A, pk_B, T, m, c, r).

- **simulation of** $\texttt{ConfirmOrDeny}(\Sigma, m, pk_B, T)$ **and** $\texttt{Convert}(\Sigma, m, pk_B, T)$**:** Let $\Sigma = (c, \sigma)$. If (pk_A, pk_B, T) has not been registered or $(\Sigma, m, pk_B, T) = (\Sigma^*, m^*, pk_B^*, T^*)$ or $\texttt{SIG.Verify}(pk_A, \sigma, c||T||pk_B) \neq \texttt{valid}$, then $\mathbb{F}$ responds $\perp$. Otherwise if $(pk_B, T) \neq (pk_B^*, T^*)$, $\mathbb{F}$ retrieves t from the table and runs the corresponding protocol with $\mathcal{A}$ as in the real protocol. If $(pk_B, T) = (pk_B^*, T^*)$ then $\mathcal{B}$ searches the table of issued signatures for an entry $(pk_A, pk_B^*, T^*, m, c, r)$ for some r. If no such entry exists then $\mathbb{F}$ outputs a forgery σ on $c||T^*||pk_B^*$ and halts. Otherwise $\mathbb{F}$ can simply use t^* in a normal execution of the corresponding protocol with $\mathcal{A}$.

- **end:** Eventually, $\mathcal{A}$ halts and outputs a bit b'. Then $\mathbb{F}$ simply aborts.

Let Forge be the event that $\mathbb{F}$ halts and outputs a forgery. Then, $\mathcal{A}$ queried $\texttt{ConfirmOrDeny}$ or $\texttt{Convert}$ on (Σ, m, pk_B^*, T^*) such that $(\Sigma, m) \neq (\Sigma^*, m^*)$, σ is a valid signature on $c||T^*||pk_B^*$, and no tuple $(pk_A, pk_B^*, T^*, m, c, r)$ for any r exists in the table of issued signatures. This means σ is a valid $\texttt{SIG}$ forgery for $\mathbb{F}$. For the sake of contradiction assume that the $\texttt{SIG}$ signing oracle answered σ on a query $c||T^*||pk_B^*$. Then either there is an entry $(pk_A, pk_B^*, T^*, m', c, r')$ in the table, and by the binding property of $\texttt{SC}$ it must be $m' = m$, or the $\texttt{SIG}$ query occurred during the challenge generation. But then $\sigma = \sigma^*$ and $c = c^*$ and by the binding property $m = m^*$. But this contradicts the fact that $(\Sigma, m) \neq (\Sigma^*, m^*)$. Therefore, $\mathsf{Succ}_{\mathsf{S}}^{\text{sEF-CMA}}[\mathbb{F}] = \mathsf{Prob}[\mathsf{Forge}]$.

Now an adversary $\mathcal{B}$ that perfectly simulates a challenger for $\mathcal{A}$ except if Forge occurs, is described below.

- **startup:** During the game $\mathcal{B}$ maintains two tables, one with the registered trapdoors and the other with the issued signatures. $\mathcal{B}$ starts receiving the system parameters of the underlying stateless commitment scheme $\texttt{SC.param}$ and the target trapdoor information T^*. $\mathcal{B}$ completes the system parameters $\texttt{DS.param}$ and sends them to $\mathcal{A}$. Then $\mathcal{B}$ computes the signature key pairs for A and B^* as $(sk_A, pk_A) \leftarrow \texttt{SIG.KeyGen}()$ and

$(sk^*_B, pk^*_B) \leftarrow \texttt{SIG.KeyGen}()$, and sends $(pk_A, pk^*_B, T^*, C^*_{AB})$ to $\mathcal{A}$, where $C^*_{AB} \leftarrow \texttt{SIG.Sign}(sk^*_B, pk_A||T^*)$. $\mathcal{B}$ stores the tuple $(pk_A, pk^*_B, T^*, C^*_{AB}, *)$ so that T^* is considered as registered, and sets $\Sigma^* = \texttt{undefined}$ and $m^* = \texttt{undefined}$.

- **challenge generation:** Eventually $\mathcal{A}$ sends a message m^* to $\mathcal{B}$. Now $\mathcal{B}$ sets $m_0 = m^*$ and picks a random m_1 with the same length as m^*. Then $\mathcal{B}$ sends (m_0, m_1) to his challenger, who responds with the target commitment c^* on m_b, where b is a random bit known to the challenger. Finally $\mathcal{B}$ computes $\sigma^* = \texttt{SIG.Sign}(sk_A, c_b||T^*||pk^*_B)$, sets $\Sigma^* = (c_b, \sigma^*)$ and sends it to $\mathcal{A}$. As the commitment is smooth, in the case $b = 1$, c^* is uniformly distributed in the commitment space.

- **simulation of `TrapReg`, `NewTrap`, `Trapdoor` and `Sign`:** Exactly as in $\mathbb{F}$ (as t^* is not required).

- **simulation of $\texttt{ConfirmOrDeny}(\Sigma, m, pk_B, T)$:** If (pk_A, pk_B, T) has not been registered or $(\Sigma, m, pk_B, T) = (\Sigma^*, m^*, pk^*_B, T^*)$ then $\mathcal{B}$ responds $\perp$. Let $\Sigma = (c, \sigma)$. If $\texttt{SIG.Verify}(pk_A, \sigma, c||T||pk_B) \neq \texttt{valid}$ then $\mathcal{B}$ outputs `invalid`. If $(pk_B, T) \neq (pk^*_B, T^*)$ then $\mathcal{B}$ retrieves t from the table and runs `SC.ConfirmOrDeny` normally. Otherwise $\mathcal{B}$ searches the table of issued signatures for an entry $(pk_A, pk^*_B, T^*, m, c, r)$ for some r and runs `SC.ConfirmOrDeny` from r. If no such entry exists, then $\mathcal{B}$ aborts.

- **simulation of $\texttt{Convert}(\Sigma, m, pk_B, T)$:** Let $\Sigma = (c, \sigma)$. If (pk_A, pk_B, T) has not been registered or $(\Sigma, m, pk_B, T) = (\Sigma^*, m^*, pk^*_B, T^*)$ or $\mathsf{SIG.Verify}(pk_A, \sigma, c||T||pk_B) \neq \texttt{valid}$ then $\mathcal{B}$ responds $\perp$. If $(pk_B, T) \neq (pk^*_B, T^*)$ he computes $\pi = \texttt{SC.Open}(c, m, t)$. If $(pk_B, T) = (pk^*_B, T^*)$ and r is known, he runs $\pi = \texttt{SC.PubOpen}(m, T, r)$ instead. Then $\mathcal{B}$ responds (c, T, σ, π) if $\pi \neq \perp$, or $\perp$ otherwise. If $(pk_B, T) = (pk^*_B, T^*)$ and r is unknown, $\mathcal{B}$ aborts.

- **end:** Eventually, $\mathcal{A}$ halts and outputs a bit b', which is forwarded by $\mathcal{B}$ to his challenger.

It is straightforward that

$$\begin{aligned}\mathsf{Prob}[\mathcal{B}\mathsf{wins}] &\geq \mathsf{Prob}[\mathcal{B}\mathsf{wins} \cap \neg\mathsf{Forge}] \\ &= \mathsf{Prob}[\mathcal{A}\mathsf{wins} \cap \neg\mathsf{Forge}].\end{aligned}$$

Thus

$$\begin{aligned}\mathsf{Adv}_{\mathsf{DS}}^{\text{Inv-CMA}}[\mathcal{A}] &= |\mathsf{Prob}[\mathcal{A}\,\mathsf{wins}] - 1/2| \\ &= |\mathsf{Prob}[\mathcal{A}\,\mathsf{wins} \cap \mathsf{Forge}] + \mathsf{Prob}[\mathcal{A}\,\mathsf{wins} \cap \neg\mathsf{Forge}] - 1/2| \\ &\leq \mathsf{Prob}[\mathsf{Forge}] + |\mathsf{Prob}[\mathcal{B}\,\mathsf{wins} \cap \neg\mathsf{Forge}] - 1/2| \\ &= \mathsf{Succ}_{\mathsf{S}}^{\text{sEF-CMA}}[\mathbb{F}] + \mathsf{Adv}_{\mathsf{SC}}^{\text{Ind-CMA}}[\mathcal{B}]\end{aligned}$$

Chapter 5

Abuse-Free Fair Exchange of Signatures in the Standard Model

5.1 Introduction

Due to the developing of the Internet and other computer networks in all aspects of life, electronic commerce has introduced the new need of being able to sign contracts remotely, especially when two parties A and B do not trust each other. A contract signing protocol is called *fair* if both parties get the desired signatures or none of them gets anything, even in the case they try to cheat each other. In traditional paper-based contract signing, the two parties sign a contract at the same place and time, so fairness can be 'physically' enforced. However, in online contract signing once one party (say B) gets the signature from the other party (say A), nobody can prevent him from immediately leaving the protocol without committing himself to the contract. Fairness in contract signing can be considered in the more general context of *fair exchange of signatures*, that is closely related to some other concepts like certified e-mail systems and non-repudiation protocols.

Various schemes for fair exchange of signatures have been proposed in the literature. The early works on this subject focused on gradual release of two signatures from two parties [62] and [37], but the proposed prtocols are very inefficient and they only provide a weak notion of fairness. Hence, most fair exchange protocols make use of a trusted third party or arbitrator TTP. Straightforward yet inefficient protocols can be defined assuming TTP is available online [10, 43]. However, the most interesting protocols are *optimistic* [1, 4], that is, TTP is not invoked in a normal protocol execution but only when one party has to complain against the other. In optimistic fair exchange (OFE) protocols the messages exchanged by the parties are commitments to the signatures, which can be opened by the trusted party in case of conflict. For instance, verifiable escrows are used in [1], while verifiable encryptions are used in [4].

The basic idea in [1] is as follows: The first signer, A, sends a verifiable escrow of his signature to the second signer, B. Then B verifies the escrow and, if it is correct then he sends a verifiable escrow of his signature back to A (in [4] a verifiable encryption is used instead). A verifies B's escrow and, if it is correct then he sends his signature to B. B verifies A's signature and if it is correct then he sends back his signature.

In [3], some different schemes have been proposed for fair exchange of signatures based on the idea that the signer signs a message m and then encrypts the signature using the public key of the trusted party in a way that the second signer can decide that the received value is a valid encryption of a signature on the message m without the help of the first signer but can not extract the signature from that value. Getting the encrypted signature and checking the validity of it, the receiver normally signs the message/contract and sends it to the signer. In the case that the interaction cuts before the second round or the first signer just leaves the protocol after getting the signature from the other party, the second signer refers to the trusted party and asks him to decrypt the partial signature and extract the signature.

Although some existing OFE protocols uses many rounds, here we will restrict ourselves to three-move protocols, in which after setting up the system a signer A sends a commitment of her signature to B, then after some verifications B sends his signature to A, and finally A opens the committed signature to B. If A does not fulfil the third move, then B can ask TTP for the opening of the committed signature. Notice that in this case, B must provide some information to TTP so that he can extract and send B's signature to A, thus guaranteing the fairness.

As well as fairness and optimism, some other desirable properties for contract signing protocols have been considered in the literature. Nevertheless, *abuse-freeness* [55] seems to be the most important and difficult property among them.

In some applications, it is very important to preserve the signers' privacy until the very end of the protocol. Therefore the following attack must be considered: Getting A's commitment to her signature, B could leave the protocol and prove to an external party C that A has committed to a signature on the contract, thus showing A's intention to sign it. Abuse-free optimistic contract signing protocols prevent this and other similar attacks. The main goal in these systems is making the transcript of an unfinished protocol instance completely simulateable by the malicious party alone, and thus non-convincing for an external party. The concept of abuse-freeness is in a very close relation to *designated verifier* signature schemes. Actually some fair exchange protocols make use of designated verifier signatures, which can be turned into universally verifiable by both the signer and TTP.

Most papers consider only a weak notion of abuse-freeness, in which a single instance of the OFE is run.

Another desirable property of an OFE is *setup-freeness*, which means that no interaction between the signers and the trusted party is required, either during setup or during the normal protocol execution. This property allows the signers to freely designate the trusted party in a system in which more than one are available. Moreover, non setup-free protocols require TTP to maintain some setup information that grows with the number of potential signers.

Related Work. Some practical fair exchange protocols have been recently introduced, but only a few of them address the abuse-free property. In 2001 an efficient protocol is proposed in [87], but abuse-freeness is not addressed. Some protocols for contract signing based on RSA are proposed in [97, 116]. However the scheme in [97] is broken and repaired in [46], where it is proved to be secure in the random oracle model. The idea of these two papers is splitting the secret exponent d into two secrets d_1 and d_2 such that $d = d_1 + d_2$, and sending d_2 to TTP. Then the first signer A gives a partial signature on the contract with the secret key d_1, along with a 5-round zero-knowledge proof of its validity (note that the related 'public' exponents e_1 and e_2 are kept secret here). Abuse-freeness is considered in [116] but only in an informal way. Actually, abuse-freeness could hold in the random oracle model, and under some computational assumption (an adversary capable to solve the DDH problem can easily break the abuse-freeness, in a single instance of the protocol). The protocol in [46] is not setup-free since the first signer must send d_2 to the trusted party before running an instance of the protocol.

In 2006, [85] proposed the first verifiably encrypted signature scheme, provably secure in the standard model. The scheme is based on Waters' signature scheme [118], and it works in the certified key model. In 2008 another proposal [75] is introduced, which works in the more general chosen key model, which means the adversary can choose its public key arbitrarily without requiring it to prove the knowledge of the corresponding secret key. Although both schemes are quite efficient, they do not address the abuse-free property. In a later work [74], the same authors define the notion of *signer-ambiguity*, which is tightly related to abuse-freeness, and they propose a new setup-free optimistic fair exchange protocol that achieves signer-ambiguity. However, the resulting scheme is far from being practical due to the computational cost of the pairing-based non-interactive zero-knowledge proofs by Groth and Sahai [71].

A more general notion of contract signing protocols in the certified key model was proposed in [45], where a generic construction in a multi-user setting based on one-way functions is provided. However, in this paper we focus on the most common scenario of two-party schemes, but we attach the identities of both signers and TTP to the contract in order to prevent trivial attacks in the multi-user setting.

Our Contributions. Our contribution is twofold. On the one hand, we propose a new signature scheme, which is a variant of Boneh-Boyen signature scheme [13], and we show it is strong existentially unforgeable under chosen message attacks in the standard model, under the strong Diffie-Hellman assumption. We also build a convertible signature scheme from it, which we call a *partial signature scheme*. The partial signatures derived from it can be verified or converted into universally verifiable signatures either by the signer or by TTP. Actually, the conversion is very efficient (compared to other schemes using verifiable encryption) since it involves just one exponentiation, and the partial signatures can be verified in zero-knowledge by either the signer or TTP by means of the efficient concurrent proof of equality of two discrete logarithms [32, 37].

On the other hand, based on the new signature schemes we propose a new contract signing protocol, which is optimistic, setup-free and abuse-free in the certified key model (*i.e.*, every party registers his public keys by proving the knowledge of the corresponding secret key). Indeed it is the verifiable knowledge of the secret key by B what makes B unable to convince a third party about the information he collects during an unfinished exchange protocol. All the above properties of our scheme hold in the standard model. To the best of our knowledge, our proposal is the most efficient abuse-free optimistic contract signing protocol in the standard model.

The efficiency of the new contract signing scheme is comparable to the existing abuse-free protocols in the random oracle model [55, 116]. The proposed scheme works in only five rounds (three rounds correspond to the three-move architecture of the exchange protocol, and the two extra rounds come from the zero-knowledge proof). The communication complexity of our protocol is very small compared to the RSA based schemes: Typically the total length of a partial signature in our scheme is about 480 bits. The ordinary signatures sent by both signers in the last rounds of our protocol have the same length as the partial signature. However, in the last round, the signer can just send one group element (160 bits) instead of a whole signature, which saves 320 bits.

5.2 Model and Definitions

In this section we restrict the definition of an optimistic contract signing protocol to the *three-move* architecture [46] described in the introduction, which is enough for our purposes. For a more general definition, see [55, 116].

A contract signing protocol is a protocol in which two parties A and B exchange two signatures σ_A, σ_B on a contract M. We assume in all the paper that A starts the protocol execution. The signatures exchanged, called *ordinary signatures*, are assumed to be existentially unforgeable[1]. Ordinary signatures correspond to a signature scheme, described with the usual subprotocols **KeyGen**, **Sign** and **Ver**, respectively denoting key generation, signing and verification.

An auxiliary subprotocol called a *partial signature scheme* is needed in the definition of the three-move protocol. It is functionally described as follows:

- **PKeyGen** is the key generation for the signer A, the recipient B and the trusted party TTP.
- **PSign**. A partial signature on a message m is computed by A. The public key of TTP is used as an input.
- **PVer** is an interactive verification protocol[2] for a partial signature ρ_A between the signer A and the recipient B, which is run immediately after **PSign**.

[1]There is no need to consider the strong unforgeability for the selected signature scheme. Actually once a signer signs a contract, it does not matter if another signature for the same contract be forged under the public key of the signer.

[2]**PVer** is defined as an interactive protocol to allow the contract signing protocol to be abuse-free.

- **Conv** is a protocol run by TTP on a partial signature ρ_A to extract the corresponding ordinary signature σ_A.

We assume that TTP can noninteractively verify the validity of a partial signature ρ_A. A three-move *optimistic contract signing* protocol can be described by the following subprotocols:

- **Setup**. On a common input of some system parameters, during this phase every signer produces, registers and publishes his signing and verification keys; and TTP produces, registers and publishes his resolving key, according to **KeyGen** and **PKeyGen**.

- **Exch**. In a first move A runs **PSig** on a contract M (known to both A and B) and sends the resulting partial signature ρ_A to B. Then A and B execute **PVer** so that B can check the validity of ρ_A. In a second move, B runs **Sig** to obtain a signature σ_B on M, and sends it back to A. In the third move, after A verifies σ_B with **Ver**, she sends a signature σ_A on M to B (computed directly with **Sig** or from the output of **Conv** on the input of ρ_A).

- **Res**. A protocol run by B and TTP, started by B when in the last move of **Exch**, B receives an invalid σ_A or nothing (*i.e.*, A left the protocol). If TTP accepts (after verifying some information provided by B, which could include ρ_A and a partial signature ρ_B on M) then he extracts σ_A and σ_B from the information received, and sends σ_A to B and σ_B to A.

Notice that we do not allow A to raise any complain to TTP. Although in a more general definition that possibility exists, in a three-move protocol either A receives nothing (an hence B does not come up with a valid signature), or A receives B's signature, either from B or from TTP.

5.2.1 Security Definitions

We briefly review the security definitions in [46, 45] except for abuse-freeness, which is not addressed there. We follow the commonly accepted definition of abuse-freeness in [55], but adapted to the case of three-move protocols.

A secure *abuse-free* optimistic contract signing protocol has to fulfil the following security requirements:

- Correctness. If all the parties behave honestly (and the network works properly) then at the end of **Exch** every signer gets a valid signature on the contract M under the public key of the other signer.

- Ambiguity. Any resolved signature (*i.e.*, resulting from **Res**) is indistinguishable from a signature generated with **Sig**.
- Security Against A. A must not be able to produce a partial signature ρ_A in such a way that convinces B about its correctness, but it cannot be transformed into a valid signature by TTP during a **Res** subprotocol execution.
- Security Against B. B must not be able to output an ordinary signature σ_A on M, even after executing **PSig**, **PVer** and **Res** for some partial signatures and messages adaptively chosen by B, with the only restriction that **Res** is not executed on any partial signature on M.
- Security against TTP. Here we assume that TTP is an honest but curious adversary. Informally, TTP is prevented from performing any meaningful computation other **Res**. More precisely, TTP must not be able to forge a valid ordinary signature σ_A on M, even after eavesdropping at some executions of **Exch** on contracts different from M.
- Abuse-freeness. B does not obtain publicly verifiable information about (honest) A signing the contract until B is also bound by the contract. Observe that A cannot abuse B in a three-move contract signing protocol, since the only information A can receive from an honest B is his ordinary signature (and this happens only when A herself is bound to the contract). Abuse-freeness is equivalent to show the impossibility for an external party to publicly verify a partial signature issued by A (providing him with the information learned by B in the **PVer** execution).

  The only known abuse-free schemes uses a very weak adversarial model in the proofs. For instance in [116] only the zero-knowledgeness of **PVer** is mentioned, while an adversary solving DDH can easily break the claimed property. Also [55] refers only to a single running instance of the contract signing protocol. Hence, attacks gaining information from (possibly concurrent) protocol instances other than the target one are not addressed. Here we also fail to prove a more powerful notion of abuse-freeness, but we provide the first proof in the standard model. So it remains as an open problem to improve the results on provably secure abuse-free contract signing schemes in both the random oracle and the standard models.

Observe that assuming the unforgeability of the ordinary signatures, the above definitions of security against A and B imply fairness. Indeed, if finally A learns a valid signature σ_B, it must come from B (in the second move) of from TTP(as a result of **Res**, which means that B also receives σ_A). But the first case means that B receives a valid ρ_A, which allows B to successfully run

Res with TTP and get σ_A. Conversely, if B gets σ_A, it must come from A (in the third move, so A also have σ_B) or from TTP(so A also received σ_B). Otherwise, B forged σ_A perhaps from ρ_A, which is not possible due to the definition of security against B.

To address the *timely termination* property (*i.e.*, the execution of a protocol instance will be terminated in a predetermined time) we assume that the communication of honest parties with TTP is reliable enough, so that an adversary can only introduce some delay in the delivery of messages (say up to certain limit Δ), but it is not allowed to delete or replace such messages. This allows us to use the deadline technique suggested by Micali in [89]. We also assume that the parties can reliably refer to TTP's local time.

An optimistic fair exchange protocol is *setup-free* if **Setup** can be run with no interaction between TTP and the signers, with common input of some trustily generated system parameters. Therefore, setup-freeness is basically depends on the properties of the **PKeyGen** protocol.

5.3 On-Line vs Off-Line Attacks

The problem of non-transferability of signatures, which in the contract signing protocols is known as abuse-freeness, can be studied in two different scenarios: abuse-freeness against on-line attacks or against off-line attacks. Different ways for providing abuse-freeness will result in different levels of security.

A very common example of online attacks describes a scenario in which two adversaries A and B collaborate to cheat an entity (here a signer). To do that the adversary A plays a game with a signer S and forwards all the information got from S to B. B also can send some information to A that can be used by A or can be forwarded to S. This kind of attacks are also known as the *man in the middle* attack.

Interactive zero-knowledge proofs are vulnerable to this attack. Actually non-transferability of these proofs holds only considering that the verifier is not in connection with the adversary at the time of interaction, however in the concurrent zero-knowledge proofs where normally a trapdoor commitment scheme is used, see [38], it is possible to resist this attack by putting the public key of the verifier in the commitment scheme[3].

[3]Here the problem of registration of keys show its importance. Actually concurrent ZKP's are also vulnerable to on-line attacks in the chosen key model because non-transferability depends on the fact that the receiver knows the secret key related to his public key

It is trivial that in the cases that a non-transferable proof of validity of a statement is given in a non-interactive way just sending some information in one round, this attack does not work. This is the case of the first proposal for *designated verifier proofs*, [76], that was proved secure in random oracle model.

5.3.1 Designated Verifier Proofs

Following the introduction of *undeniable signatures* by Chaum, another type of signatures were proposed by introducing the notion of *designated verifier proofs* in [76]. Briefly, a designated verifier signature is a signature that can be verified by a verifier designated by the signer in such a way that even by revealing his secret key, the verifier can not convince another party that the signature has been issued by that signer. In general, designated verifier proofs can be divided into two classes; those in the standard model and those in the random oracle model. However here we are interested in those proofs secure in the standard model.

- **Designated verifier proofs in the standard model**

  The notion of interactive concurrent zero-knowledge proofs was explained in section 2.4.3. Observe that in the scheme of Damgård, the prover uses a trapdoor commitment scheme to commit to his first message. The simulator works perfectly, if the trapdoor key is given to it as the auxiliary string. This gives the idea that if the trapdoor key is the secret key of the receiver, then the receiver can also simulate a transcript perfectly as the simulator does. In this way, the transcript computed by the prover is indistinguishable from a transcript computed by the receiver. This gives the non-transferable property to the proof while it still convinces the receiver of the validity of the statement. As the protocol for concurrent zero-knowledge proofs works in the standard model, this technique can be used as a generic construction to transform any Σ-protocol to a designated verifier proof in the standard model.

5.4 The New DBB Signature Scheme

Before introducing our new signature, we give a short description of Boneh-Boyen signature scheme.

Boneh-Boyen's (BB) signature scheme [13, 14]. The secret and public keys are $sk = (y, s)$ and $pk = (Y = g_2^y, S = g_2^s)$, for random $y, s \in \mathbb{Z}_p^*$. The signature on a message $m \in \mathbb{Z}_p$ is $\sigma = (r, g_1^{\frac{1}{y+m+rs}})$, for a random $r \in \mathbb{Z}_p$. The signature $\sigma = (r, b)$ is verified by checking $e(b, Y g_2^m S^r) = g_T$. Boneh-Boyen's signature scheme is (strong) existentially unforgeable under SDH (which is the asymptotic version of q-SDH, where q is bounded by a polynomial on the security parameter [14]).

Here we give a variation of the signature proposed by Boneh and Boyen, which is still strongly unforgeable under the SDH assumption. For simplicity we refer to it as DBB signature scheme (from *Double* Boneh-Boyen signature). Essentially, the new signature consists of a weak-BB signature b on a random element r, with respect to the base g_1 and signing key x, and another BB signature c on the message m with respect to the base b and signing keys y and s.

- **KeyGen**. The signer chooses four random secret elements $x, y, s, u \in \mathbb{Z}_p^*$ and publishes $pk = (X = g_2^x, Y = g_2^y, S = g_2^s, U = g_2^u)$.
- **Sign**. To sign a message $m \in \mathbb{Z}_p$, the signer chooses a random $r \in \mathbb{Z}_p$ and computes $\sigma = (r, g_1^{\frac{1}{x+r}}, g_1^{\frac{u}{(x+r)(y+m+rs)}})$.
- **Ver**. To verify a signature $\sigma = (r, b, c)$, a verifier checks that $e(b, X g_2^r) = g_T$ and $e(c, Y g_2^m S^r) = e(b, U)$.

To improve the efficiency of the scheme, we can save computing some pairings in the cost of a nonzero but negligible error probability: To verify a signature, a verifier can choose a random $\alpha < 2^t$ and check a single equality $e(b, X g_2^r U^\alpha) = g_T e(c^\alpha, Y g_2^m S^r)$. Thus the computation of one pairing is replaced by two extra exponentiations with short exponent, and the error probability in the verification is 2^{-t}.

Theorem 5.4.1 *The DBB signature scheme is strong existentially unforgeable under chosen message attack, assuming that the SDH Assumption holds.*

Proof 1 *Let $\mathcal{F}$ be a forger for the proposed scheme. We will use this forger to make another forger $\mathcal{A}$ that after at most q adaptive queries to a BB signing oracle, it outputs a valid BB message/signature pair, different from the oracle queries, with the same probability.*

After receiving the public key of the BB signature instance $(Y = g_2^y, S = g_2^s)$, $\mathcal{A}$ chooses random $x, u \in \mathbb{Z}_p^*$ and lets $X = g_2^x$, $U = g_2^u$. Then $\mathcal{A}$ sends the DBB public key $pk = (X, Y, S, U)$ for the same pairing system to $\mathcal{F}$. Every time that $\mathcal{F}$ asks for a DBB signature on a message m_i, $\mathcal{A}$ forwards the message *to the BB oracle and gets the BB signature* $(r_i, \gamma_i) = (r_i, g_1^{\frac{1}{y+m_i+r_i s}})$*. Then* $\mathcal{A}$ *computes and forwards* $(r_i, g_1^{\frac{1}{x+r_i}}, \gamma_i^{\frac{u}{x+r_i}})$ *as a DBB signature on the message* m_i *to* $\mathcal{F}$*. Eventually if* $\mathcal{F}$ *succeeds then it stops and outputs a forgery* $(r^*, b^* = g_1^{\frac{1}{x+r^*}}, c^* = g_1^{\frac{u}{(x+r^*)(y+m^*+r^*s)}})$ *on the message* m^* *such that* (m^*, r^*) *is different from all* (m_i, r_i)*. Raising* c^* *to the power of* $\frac{x+r^*}{u}$*,* $\mathcal{A}$ *will get* $g_1^{\frac{1}{y+m^*+r^*s}}$ *and outputs it as a valid BB forgery. Clearly, the success probability of* $\mathcal{F}$ *is not less than the success probability of* $\mathcal{A}$*.*

5.4.1 The Partial Signature Scheme (PDBB)

In order to get a partial signature scheme (which we call PDBB scheme), in which signatures are not universally verifiable but can only be verified by the Trusted Party or by the signer[4], we build on the previous DBB scheme, introducing the new public keys $(Z_1 = g_1^z, Z_2 = g_2^z)$ generated by the Trusted Party. The signer can convince the recipient of the signature about its validity by means of an interactive zero-knowledge proof. To make this proof non-convincing for an external party, the signer uses a concurrent version of Chaum-Pedersen proof of knowledge [32] combined to Pedersen's trapdoor commitments [98], which is an efficient zero-knowledge argument in the auxiliary string model [38]. We actually need that both the Trusted Party and the recipient register their keys by proving in zero-knowledge (*e.g.*, to a Certification Authority) the knowledge of the corresponding secret keys (respectively z and the trapdoor of the commitment), so that the recipient cannot later deny his ability to forge a validity proof.[5] The proposed partial signature scheme consists of the following algorithms protocols involving a signer A, a recipient B and the Trusted Party:

- **PKeyGen**. As in DBB signatures, A chooses $x, y, s, u \in \mathbb{Z}_p^*$ at random and publishes $pk_A = (X = g_2^x, Y = g_2^y, S = g_2^s, U = g_2^u)$. The Trusted Party chooses $z \in \mathbb{Z}_p^*$ and registers $(Z_1 = g_1^z, Z_2 = g_2^z)$ proving the knowledge of z. B chooses a random $\tau \in \mathbb{Z}_p^*$ and registers $h = g^\tau$ also proving the knowledge of τ, where $G = \langle g \rangle$ is a group of order p. (g, h) is used as the auxiliary string for the concurrent zero-knowledge proof in **PVer**, that is the public key of the trapdoor commitment. Observe that no interaction between TTP and B is required.

[4]Observe that the partial signature scheme is not a fully-fledged designated verifier scheme, since we do neither require the verifier to prove the validity of a signature in a non-transferable way, nor the signer verify any previously issued partial signature.

[5]As in most works we are only considering off-line non-transferability. For more details on how to address on-line attacks see [84].

- **PSign**. To sign a message $m \in \mathbb{Z}_p$, A chooses a random $r \in \mathbb{Z}_p$ and computes $\rho = (r, Z_1^{\frac{1}{x+r}}, g_1^{\frac{u}{(x+r)(y+m+rs)}})$.

- **PVer**. Upon reception of a partial signature $\rho = (r, d, c)$ from A, both A and B engage in an interactive verification protocol, firstly checking the equation $e(d, Xg_2^r) = e(g_1, Z_2)$, then running the concurrent zero-knowledge proof that A knows $\lambda = y + m + rs$ such that $Yg_2^m S^r = g_2^\lambda$ and $e(d, U) = e(c, Z_2)^\lambda$.

- **Conv**. The Trusted Party extracts a DBB signature $\sigma = (r, b, c)$ on m from a PDBB signature $\rho = (r, d, c)$, just by computing $b = d^{1/z}$. Observe that he can also verify the validity of ρ as $e(d, Xg_2^r) = g_T^z$ and $e(d, U) = e(c^z, Yg_2^m S^r)$.

It can be shown that the above scheme is also strong existentially unforgeable (even if z is given to the adversary) under the same assumption as DBB signature. Moreover, an *invisibility* property holds. Namely, no polynomial-time adversary can distinguish a valid PDBB signature on an adversarially chosen message m from a PDBB signature on a random message, under DTDH assumption. Due to the lack of space we decided not to include the proofs in this paper. However, these properties are implicitly used in the design of the contract signing protocol presented in the next section, and the omitted security proofs can be easily inferred from the security analysis of the contract signing protocol.

5.5 The Proposed Contract Signing Scheme

In this section we describe the proposed contract signing protocol following the description of the three-move architecture instantiated with the signature schemes presented in previous sections. The low-level algorithmic description of the protocol execution is described continuously.

In order to prevent trivial attacks in the multi-user setting[6], such as the one reported in [45], the partial and ordinary signatures will be computed on the message $m = H(M||ID_A||ID_B||ID_T||t)$ instead of the contract itself, where $H : \{0,1\}^* \to \mathbb{Z}_p$ is a collision-resistant hash function, ID_A and ID_B are the identities of both signers, ID_T is the identity of TTP, and t is the session deadline for **Exch** and **Res**. In this contract signing protocol, the ordinary signature scheme is DBB, while the partial signature scheme is PDBB, both

[6]However, we are not giving any security proof in the multi-user setting.

described in the previous section.

We assume that all the parties are given the following system parameters: a security parameter $\ell \in \mathbb{Z}$, the pairing system parameters $\mathsf{psys} = (p, G_1, G_2, G_T, g_1, g_2, e, \psi, g_T)$, where p is ℓ bits long, the group $G = \langle g \rangle$ for the trapdoor commitments, and the hash function H.

- **Setup.** Both signers A, B and TTP generate and certify their keys as specified in PDBB **PKeyGen**[7]. So we define $sk_A = (x, y, s, u, \tau)$, $pk_A = (X, Y, S, U, h)$, and similarly we have $sk_B = (x', y', s', u', \tau')$, $pk_B = (X', Y', S', U, h')$, and $sk_T = z$, $pk_T = (Z_1, Z_2)$. We recall TTP proves the knowledge of z such that $Z_2 = g_2^z$, while A and B respectively prove the knowledge of τ and τ' such that $h = g^\tau$ and $h' = g^{\tau'}$, to a certification authority.

- **Exch.** Once A and B agree on M, the trusted party identity ID_T and a timeout $t > \Delta$, both compute m and run the protocol as specified in section 5.2, using **PSig**, **PVer**, **Sig** and **Ver** in PDBB and DBB definitions. Namely, in the first move A runs **PSig** on M and sends ρ_A to B. Then A and B execute **PVer** to check the validity of ρ_A. In case B do not accept, he just leaves the protocol. In the second move, B runs **Sig** on M and sends σ_B to A. In the third move, after A verifies σ_B with **Ver**, she sends a signature σ_A on M to B (computed with **Sig** or directly from ρ_A). After the second move, if the time to the deadline t is less than Δ then B leaves **Exch** and executes **Res** with TTP. On the other hand, if A has not received any message from B or TTP at time $t + \Delta$, then she exits the protocol.

- **Res.** B refers to TTP for resolving the partial signature $\rho_A = (r, d, c)$ on M. After identifying himself, he must send t, ID_A, M together with ρ_A and his partial signature ρ_B on m to TTP. TTP non-interactively checks the validity ρ_A and ρ_B for $m = H(M||ID_A||ID_B||ID_T||t)$. If the protocol is not timed out (with respect to the deadline t) and **Res** has not been invoked before for the same m, then TTP runs **Conv** for both partial signatures and sends the resulting b to B (*i.e.*, $\sigma_A = (r, b, c)$) and σ_B to A. TTP temporarily stores the tuple (m, ID_A, ID_B, t) until time t, to prevent further executions of **Res** on the same contract.

The involving protocols for the proposed fair exchange of signature are described in details as follows:

[7]We assume both can potentially act as signers or verifiers, although A never uses τ and h.

System setup. The following objects are determined and made publicly available by all the parties in the system before running the subprotocols:

- A security parameter $\ell \in \mathbb{Z}$ and a pairing system described by $\mathsf{psys} = (p, G_1, G_2, G_T, g_1, g_2, e, \psi, g_T)$, where p is ℓ bits long, and g_1, g_2 are random generators (so one expects $\psi(g_2) \neq g_1$). We assume $G_1 \neq G_2$, but in the case of $G_1 = G_2$ the scheme works and it can be slightly simplified by letting $g_1 = g_2$ and removing the redundant element in TTP's public key.

- A group $G = \langle g \rangle$ of order p for the trapdoor commitments (it could be $G = G_1$ or G_2, or a completely different group).

- The collision resistant hash functions $H, \hat{H} : \{0,1\}^* \to \mathbb{Z}_p$, for the contract digest and the trapdoor commitments (actually, H and $\hat{H}$ could be the same function).

- We assume the existence of a string identifying every party, which is bounded to the corresponding public key by the certification authority.

Setup for signer A (and similarly for B). A chooses random $sk_A = (x, y, s, u, \tau) \in (\mathbb{Z}_p^*)^5$, and computes $pk_A = (X = g_2^x, Y = g_2^y, S = g_2^s, U = g_2^u, h = g^\tau)$. A registers her public key to a certification authority and proves in zero-knowledge her knowledge of τ such that $h = g^\tau$. (In spite of simplicity we will add a prime to the elements belonging to B.)

Setup for TTP. TTP chooses random $sk_T = z \in \mathbb{Z}_p^*$, and computes $pk_T = (Z_1 = g_1^z, Z_2 = g_2^z)$. TTP registers his public key to a certification authority and proves in zero-knowledge his knowledge of z such that $Z_2 = g_2^z$. $Z_3 = g_T^z$ can also be precomputed by TTP and included in his public key. Moreover, everybody in the system can check the consistency of pk_T by the equation $Z_3 = e(g_1, Z_2) = e(Z_1, g_2)$.

Exchange. A and B firstly agree on a contract M, the trusted party identity ID_T and a timeout $t > \Delta$. Then they compute $m = H(M||ID_A||ID_B||ID_T||t)$. A starts the protocol and they exchange messages alternatively. B normally leaves the protocol when he receives a valid signature σ_A. However, if at time $t - \Delta$, B has not got a valid signature σ_A on m, or as soon as something goes wrong in the protocol, he moves to the Resolve step described below. On the other hand A usually ends the protocol as specified below (in the third move). But if she receives a valid signature σ_B from TTP she immediately leaves the protocol. If at time $t+\Delta$, A has not received a valid signature σ_B either from B or from TTP, then she exits the protocol.

> **Move 1:** A chooses a random $r \in \mathbb{Z}_p$ and computes $\rho_A = (r, d = Z_1^{\frac{1}{x+r}}, c = g_1^{\frac{u}{(x+r)(y+m+rs)}})$ and sends it to B. She also proves her knowledge of $\lambda = y + m + rs$ such that $g_2^{\lambda} = Yg_2^m S^r$ and $e(c, Z_2)^{\lambda} = e(d, U)$. To that end, A chooses random $\gamma \in \mathbb{Z}_p^*$ and $\alpha \in \mathbb{Z}_p$, computes $v_1 = g_2^{\gamma}$, $v_2 = e(c^{\gamma}, Z_2)$, $\beta = \hat{H}(v_1, v_2)$ and $C = g^{\alpha} h'^{\beta}$. Then she sends the commitment C to B. B responds with a random challenge $f \in \mathbb{Z}_p$ and then A computes $\delta = \gamma + f\lambda$ and sends $(\alpha, v_1, v_2, \delta)$ to B. B accepts the partial signature if $C = g^{\alpha + \tau' \hat{H}(v_1, v_2)}$, $g_2^{\delta} = v_1(Yg_2^m S^r)^f$, $e(d, Xg_2^r) = e(g_1, Z_2)$ and $e(c^{\delta}, Z_2) = v_2 e(d, U)^f$. Otherwise B moves to the resolution step. Actually the two last equations can be merged into a single one, $e(d, (Xg_2^r)^{\epsilon} U^f) v_2 = Z_3^{\epsilon} e(c^{\delta}, Z_2)$, for a random $\epsilon \in \mathbb{Z}_p$ (or $\epsilon < 2^{\ell}$ for a tiny $2^{-\ell}$ error probability).
>
> **Move 2:** B computes $\sigma_B = (r', b' = g_1^{\frac{1}{x'+r'}}, c' = g_1^{\frac{u'}{(x'+r')(y'+m+r's')}})$ for a random $r' \in \mathbb{Z}_p$, and sends it to A. A checks the validity of σ_B by $e(b', X'g_2^{r'}) = g_T$ and $e(c', Y'g_2^m S'^{r'}) = e(b', U')$ (or the compressed equation $e(c', Y'g_2^m S'^{r'}) g_T^{\epsilon'} = e(b', (X'g_2^{r'})^{\epsilon'} U')$ for a random ϵ'). If the signature is invalid, A just waits for a resolution message from TTP or timeout.
>
> **Move 3:** A computes $\sigma_A = (r, b = g_1^{\frac{1}{x+r}}, c)$, sends it to B and leaves the protocol. Then B checks the validity of σ_A by using the previous ρ_A as $e(b, Z_2) = e(d, g_2)$. If the signature is valid, B leaves the protocol. Otherwise, he moves to the resolution step.

Resolution. If B has not received and accepted ρ_A then B leaves the protocol. Otherwise he executes **Res** with TTP. Actually, B sends (t, ID_A, M, ρ_A) to TTP along with $\rho_B = (r', d' = Z_1^{\frac{1}{x'+r'}}, c' = g_1^{\frac{u'}{(x'+r')(y'+m+r's')}})$ for a random $r' \in \mathbb{Z}_p$. If no previous resolution has been invoked for this protocol instance, then TTP after identifying B computes $m = H(M||ID_A||ID_B||ID_T||t)$ and checks the validity of ρ_A by the equations $e(d, Xg_2^r) = Z_3$ and $e(d, U) = e(c^z, Yg_2^m S^r)$, and similarly checks the validity of ρ_B (TTP could also compress the equations to decrease the number of pairing computations). Next TTP computes $\sigma_A = (r, b = d^{1/z}, c)$ and $\sigma_B = (r', b' = d'^{1/z}, c')$ and sends them to B and A respectively, who immediately end the protocol. TTP temporarily stores the tuple (m, ID_A, ID_B, t) until time t, to prevent further executions of **Res** on the same contract.

5.5.1 Security Analysis

Correctness and *ambiguity* are straightforward from the description of the protocol. Moreover, *setup-freeness* is implied by the noninteractive nature of **PKeyGen**.

Security Against A. If B accepts a partial signature $\rho = (r, d, c)$ on a message m, then $e(d, Xg_2^r) = e(g_1, Z_2)$ and by the soundness of the zero-knowledge argument, $e(d, U) = e(c, Z_2)^{y+m+rs}$ so $e(d, U) = e(c^z, Yg_2^m S^r)$. Thus TTP will always accept ρ as valid and he can always compute $b = d^{1/z}$, which allows B to get the valid ordinary signature $\sigma = (r, b, c)$. Notice that the only way to break the soundness of the zero-knowledge argument is by using the trapdoor information of the trapdoor commitment scheme. Indeed, an adversary doing so can be used to get two different openings of the same commitment (via rewinding), and then compute the discrete logarithm of Z_2 on base g_2.

Security Against B. As defined before, we assume that B asks A only for executions of **PSign**, **PVer** and **Conv**. Whenever B wishes to know an ordinary signature on a message m, he just asks **PSign**, then **Conv**. We consider two different adversaries F_1 and F_2 which use B as a subroutine: F_1 breaks the strong unforgeability of BB signature scheme, while F_2 directly breaks the SDH assumption. Basically F_1 extracts a BB forgery from any partial signature computed by B using a 'fresh' randomness r, while F_2 breaks SDH by tying the SDH instance description to one of the values of r used by the signing oracle simulator, provided B uses exactly that value in the forgery. In either case, a q-SDH instance is solved, since it is shown in [14] that any BB existential forger can be used to break a q-SDH instance, with the same success probability. The detailed proof is as follows:

As defined before, we assume that B asks A only for executions of **PSign**, **PVer** and **Conv**. Whenever B wishes to know an ordinary signature on a message m, he just asks **PSign**, then **Conv**. We consider two different adversaries F_1 and F_2 which use B as a subroutine: F_1 breaks the strong unforgeability of BB signature scheme, while F_2 directly breaks the SDH assumption. Basically F_1 extracts a BB forgery from any partial signature computed by B using a 'fresh' randomness r, while F_2 breaks SDH by tying the SDH instance description to one of the values of r used by the signing oracle simulator, provided B uses exactly that value in the forgery.

- **Description of** F_1**:** At startup F_1 gets the public key $(Y = g_2^y, S = g_2^s)$ of the target BB instance. Then F_1 selects random $x, z, u, \tau \in \mathbb{Z}_p^*$ and sets $pk_A = (X = g_2^x, Y, S, U = g_2^u, h = g^\tau)$ and $pk_T = (Z = g_2^z)$. F_1 also uses a knowledge extractor to learn τ' from B. Then F_1 and B exchange the public information (pk_A, pk_B, pk_T).

During the game, B asks F_1 for **PSign** on up to q adaptively chosen messages $m_i = H(M_i||ID_A||ID_B||ID_T||t_i)$ (we can consider that M_i and t_i are chosen by B and sent to F_1, although this does not affect the proof). Then for each query, F_1 asks the BB signing oracle on m_i, which outputs $(r_i, \gamma_i = g_1^{\frac{1}{y+m_i+r_i s}})$, for a random r_i. Then F_1 computes the partial signature $\rho_i = (r_i, g_1^{\frac{z}{x+r_i}}, \gamma_i^{\frac{u}{x+r_i}})$, and sends it to B. F_1 maintains a list L containing all pairs (r_i, m_i). F_1 simulates **PVer** thanks to its knowledge of the trapdoor τ' for the zero-knowledge proof.

B can also ask for conversion of some $(\bar{m}_j, \bar{\rho}_j)$, where $\bar{\rho}_j = (\bar{r}_j, \bar{d}_j, \bar{c}_j)$. In that case, F_1 checks the validity of $\bar{\rho}_j$ as a partial signature on $\bar{m}_j$ (as F_1 knows z), and rejects it if invalid. Then, F_1 checks whether $(\bar{r}_j, \bar{m}_j) \notin L$. If this is the case, then F_1 halts and outputs $(\bar{r}_j, \bar{c}_j^{\frac{x+\bar{r}_j}{u}})$ as a valid BB forgery. Otherwise F_1 sends $\bar{b}_j = \bar{d}_j^{1/z}$ to B.

Eventually, B outputs an ordinary signature $\sigma^* = (r^*, b^*, c^*)$ on m^*. If $(r^*, m^*) \notin L$, then F_1 halts and outputs the BB forgery $(r^*, (c^*)^{\frac{x+r^*}{u}})$, which is valid whenever σ^* is a valid signature. Otherwise, F_1 aborts the game.

Notice that F_1 wins the game whenever B asks for a $(\bar{r}_j, \bar{m}_j) \notin L$, or $(r^*, m^*) \notin L$ and σ^* is valid.

- **Description of** F_2**:** At startup F_2 receives $(v_0 = g_0, v_1 = g_0^x, v_2 = g_0^{x^2}, \ldots, v_q = g_0^{x^q}, X = g_2^x)$ for a pairing system $\mathsf{psys}_0 = (p, G_1, G_2, G_T, g_0, g_2, e, \psi, g_T)$. This allows F_2 being able to compute $g_0^{P(x)}$ for any polynomial P of degree at most q with known coefficients. Indeed, if $P(x) = \sum_{\ell=0}^q a_\ell x^\ell$ then $g_0^{P(x)} = \prod_{\ell=0}^q v_\ell^{a_\ell}$. F_2 sets up the exchange protocol public keys as follows: F_2 selects a list of q different random values $LR = (r_1, \ldots, r_q)$ and a random index $k \in \{1, \ldots, q\}$. Consider the polynomial $R(x) = \prod_{i=1}^q (x + r_i)$. Then he computes $g_1 = g_0^{R_k(x)}$, where $(x + r_k)R_k(x) = R(x)$ as polynomials in x. Now, F_2 generates random $\alpha, \beta, y, s, \tau \in \mathbb{Z}_p^*$ and computes $Z_2 = (X g_2^{r_k})^\alpha$, $U = Z_2^\beta$ (thus setting $z = \alpha(x + r_k)$ and $u = \beta z$) and sends $\mathsf{psys}_0 = (p, G_1, G_2, G_T, g_1, g_2, e, \psi, g_T)$, $pk_A = (X, Y = g_2^y, S = g_2^s, U, h = g^\tau)$ and $pk_T = (Z_1 = g_0^{\alpha R(x)}, Z_2)$ to B, while B sends pk_B to F_2.

Whenever B asks for a partial signature on an adaptively chosen m_i (defined as in F_1), F_2 picks the corresponding $r_i \in R$ and computes $d_i = Z_1^{\frac{1}{x+r_i}} = g_0^{\alpha R_i(x)}$ and $c_i = d_i^{\frac{\beta}{y+m_i+r_i s}}$, where $(x+r_i)R_i(x) = R(x)$ as polynomials in x. Then F_2 sends the partial signature $\rho_i = (r_i, d_i, c_i)$ to B. As above, F_2 maintains a list L containing all pairs (r_i, m_i). As F_2 knows y and s, he runs **PVer** normally.

B can also ask for conversion of some $(\bar{m}_j, \bar{\rho}_j)$, where $\bar{\rho}_j = (\bar{r}_j, \bar{d}_j, \bar{c}_j)$. In that case, F_2 checks the validity of $\bar{\rho}_j$ as a partial signature on $\bar{m}_j$ (he knows y and s), and rejects it if invalid. Otherwise, F_2 checks whether $(\bar{r}_j, \bar{m}_j) \notin L$ and, if this is the case, F_2 aborts the execution. Now $(\bar{r}_j, \bar{m}_j) = (r_i, m_i)$ for a suitable i. If $i = k$ then F_2 aborts. Otherwise, he answers B sending $\bar{b}_j = g_1^{\frac{1}{x+r_i}} = g_0^{R_{\{i,k\}}(x)}$ where $(x+r_k)(x+r_i)R_{\{i,k\}}(x) = R(x)$ as polynomials in x.

Eventually, B outputs an ordinary signature $\sigma^* = (r^*, b^*, c^*)$ on m^*. If the signature is invalid or $(r^*, m^*) \neq (r_k, m_k)$ then F_2 aborts. Otherwise, F_2 computes $b_0 = g_0^{\frac{1}{x+r_k}} = \left(b^* g_0^{-Q(x)}\right)^{\frac{1}{R_k(-r_k)}}$, where $Q(x)$ is the unique polynomial such that $R_k(x) = Q(x)(x+r_k) + R_k(-r_k)$, since $b^* = g^{\frac{1}{x+r_k}} = g_0^{\frac{R_k(x)}{x+r_k}}$. Finally, F_2 outputs the pair (r_k, b_0) thus breaking the q-SDH instance. Note that F_2 wins the game whenever all $(\bar{r}_j, \bar{m}_j) \in L$, $(r^*, m^*) = (r_k, m_k)$ and σ^* is valid.

For any adversary B let Ev be the event that all $(\bar{r}_j, \bar{m}_j) \in L$ and also $(r^*, m^*) \in L$, and let δ be its probability. Let ε be the success probability of B, and $\varepsilon_1, \varepsilon_2$ the success probability of B conditioned to Ev and $\neg Ev$, respectively. We know that $\varepsilon = \delta\varepsilon_1 + (1-\delta)\varepsilon_2$. Moreover, F_1 succeeds whenever B succeeds and $\neg Ev$ happens, while F_2 succeeds whenever B succeeds, Ev happens and $(r_k, m_k) = (r^*, m^*)$ (which reduces the conditional success probability by a factor of q)[8]. Thus, F_1 or F_2 succeed with probability $\varepsilon' \geq \frac{\delta}{q}\varepsilon_1 + (1-\delta)\varepsilon_2 \geq \frac{1}{q}\varepsilon$. In either case, a q-SDH instance is solved, since it is shown in [14] that any BB existential forger can be used to break a q-SDH

[8]Observe that B is not allowed to use as m^* any of the values $\bar{m}_j$. Thus if $(r_k, m_k) = (r^*, m^*)$ then $(\bar{r}_j, \bar{m}_j) = (r_k, m_k)$ cannot happen to F_2. Hence, the probability that F_2 wins, conditioned to Ev and a successful B, is exactly $1/q$.

instance, with the same success probability.

Security against TTP. We show that for any honest by curious TTP we can build a DBB forger F which perfectly simulates TTP's (eavesdropping) view of A and B in some contract signing protocol instances. Indeed, F starts receiving the public key (X, Y, S, U) of a DBB instance, and completes it to obtain A and B's keys $pk_A = (X, Y, S, U, h = g^\tau)$, $sk_B = (x', y', s', u', \tau')$ and $pk_B = (X', Y', S', U', h')$. Then, F extracts z and (Z_1, Z_2) from TTP(by means of the knowledge extractor for the proof of knowledge of the secret key, perhaps involving rewinding of TTP) and sends him pk_A and pk_B. Now F can trivially simulate everything for TTP except the partial signatures normally issued by A and the transcripts of **PVer**. However, a partial signature on any message m selected by F is computed by querying the DBB signing oracle to obtain σ_A, and then converting it to ρ_A by means of z. The transcripts of **PVer** are simulated by means of the trapdoor τ'. Eventually, the malicious TTP outputs a forged signature σ'_A on a new message m', which is forwarded by F thus winning the unforgeability game with the same probability as TTP(which in turn would imply that q-SDH Assumption is false).[9]

Abuse-freeness. If the transcript of **PSig**+**PVer** is simulatable by B (without interacting to A) then abuse-freeness is guaranteed. Observe that assuming that nobody can issue valid partial signatures on behalf of A, simulatability of **PSig** requires that valid and invalid partial signatures for a given message m cannot be distinguished in probabilistic polynomial time. Moreover simulatability of **PSig**+**PVer** also implies that a simulated validity proof for an invalid partial signature must be indistinguishable from a real proof of validity of a valid partial signature.

In our scheme the transcript of a successful execution **PVer** on a given (valid or invalid) PDBB signature can be easily simulated due to the zero-knowledge property of the concurrent Chaum-Pedersen proof (provided B knows his secret key, which is guaranteed by the key registration process). Indeed, to compute the simulated transcript $((C, f, \alpha, v_1, v_2, \delta)$ in the notation used in Example 2.4.3), B first chooses random challenge f and response δ of

[9]Actually, we are proving a stronger version of the security against TTP. Indeed, an honest but curious TTP cannot launch a chosen message attack, but he can only collect some message/signature pairs. However, we could think on a Trusted Party who influences the content of the contract signed by A. And this behavior must be seen as a mild chosen message attack in which the adversary cannot choose the message to be signed, but he can influence its probability distribution.

the Chaum-Pedersen proof, then computes the witness (v_1, v_2) according to the verification equations, and finally computes the commitment $C = g^{\alpha} h'^{\hat{H}(v_1,v_2)}$ for a randomness α. Observe that no secret key is involved in those computations.

Therefore, simulatability of **PSig+PVer** depends only on the invisibility of PDBB signatures, *i.e.*, no polynomial time adversary can tell apart a valid PDBB signature $\rho = (r, d, c)$ on a message m from a valid signature $\rho' = (r, d, c')$ on a random (unknown) message m', given m. We show that no external party D can distinguish ρ from ρ' given m. Actually, any such distinguisher D can be used to build an adversary F which breaks the Decisional Tripartite Diffie-Hellman Assumption (DTDH). A description of F follows:

After receiving the description $(X_0 = g_2^{x_0}, Y_0 = g_2^{y_0}, Z_0 = g_0^{z_0}, U_0 = g_2^{u_0})$ of an instance of the DTDH problem for $\mathsf{psys}_0 = (p, G_1, G_2, G_T, g_0, g_2, e, \psi, g_{T0})$ such that $\psi(g_2) = g_0$, F picks m and random $\alpha, s, \epsilon \in \mathbb{Z}_p^*$ and $r \in \mathbb{Z}_p$, and computes $\mathsf{psys} = (p, G_1, G_2, G_T, g_1 = g_0^{\epsilon}, g_2, e, \psi, g_T = g_{T0}^{\epsilon})$, $pk_A = (X = X_0 g_2^{-r}, Y = Y_0 g_2^{-m-rs}, S = g_2^s, U = U_0)$ and $pk_T = (Z_1 = \psi(X_0)^{\alpha\epsilon} = g_1^{x_0\alpha}, Z_2 = X_0^{\alpha})$ and sends $\mathsf{psys}, pk_A, pk_T$ them to D, along with m and $\rho = (r, d = g_1^{\alpha}, c = Z_0^{\epsilon})$. Clearly d is a well-formed PDBB signature, since $e(d, Xg_2^r) = e(g_1, X_0)^{\alpha} = e(g_1, Z_2)$. Moreover, ρ is valid for m if and only if $e(d, U) = e(c, Z_2)^{y+m+rs}$. But $e(d, U) = e(g_1, U_0)^{\alpha} = e(g_1, g_2^{\alpha})^{u_0}$ and $e(c, Z_2)^{y+m+rs} = e(Z_0^{\epsilon}, X_0^{\alpha})^{y_0} = e(g_1, g_2^{\alpha})^{x_0 y_0 z_0}$. So ρ is valid for m if and only if $u_0 = x_0 y_0 z_0$. Then, F forwards D's output bit and, since the simulation of D's environment is perfect, both D and F have the same advantage.

5.6 Efficiency Considerations

In this section we compare our protocol's performance compared to other existing abuse-free protocols, based on on-line/off-line computations. Comparing to non abuse-free protocols is nonsense, since achieving abuse-freeness is the most costly task.

We have provided a table for comparison of efficiency between our scheme and the abuse-free contract signing protocols [55, 116], which are proven secure in the random oracle model, and the standard-model one in [75]. The comparison is based on the number of exponentiations and pairings computed by both signers during the **Exch** protocol, and also the number of rounds and the amount of data exchanged. The number of exponentiations given in [116] differs from our estimate because they have not considered the two exponentiations included in each commitment computation. On the other hand the

	[55]	[116]	[75]	Our Protocol
Computational cost	20 exp.	15 exp.	$>$ 162 pair.	24 exp. + 6 pair.
Bytes exchanged		1220	$>$ 1600	460
Number of rounds	4	7	3	5
Security	ROM	ROM	SM	SM

Table 5.1: Efficiency comparison.

construction in [55] is a generic construction based on the so-called OR-proofs and designated verifier signatures. We analyze an instantiation based on El-Gamal cryptosystem and the designated verifier proofs in [76], which use the signature by Chaum in [33]. The number of exponentiations computed by both the prover and verifier in the proof in [76] is 10, and they need almost the same number for the encryption.

An estimation of the overall computational cost of our **Exch** results in 6 exponentiations and 2 pairings to be computed on-line by A plus 6 off-line exponentiations. On the other hand, B has to compute 8 exponentiations and 4 pairings on-line, and 4 off-line exponentiations. The total communicated data in our **Exch** is around 460 bytes (5 scalars, 6 points in G_1, one in G_2 and 1 pairing result) for the usual sizes of pairing groups (160 bit finite field and embedding degree 6), while the RSA-based protocol in [116] uses about 1220 bytes to be exchanged.

Taking into account that our protocol is secure in the standard model and that the number of rounds is only 5, our protocol is comparable with most existing practical protocols. More precisely, our protocol has better round complexity and communication complexity, while it is about 4 times slower. This shows that our scheme is not only of theoretical interest but also it is suitable for practical applications.

As for the scheme in [75], although it achieves a similar security level in only 3 rounds and in the standard model, the use of Groth-Sahai non-interactive zero-knowledge proofs increases the computational complexity beyond 162 pairing computations[10], and the parties have to exchange more than 82 group elements, or more than 1600 bytes[11]. Hence, from a practical point of view it cannot be compared to our scheme.

[10]Only the NIZK proof is counted.

[11]Not taking into account the one-time signature included in the partial signatures.

Bibliography

[1] N. Asokan, V. Shoup and M. Waidner. *Optimistic fair exchange of digtial signatures.* In EUROCRYPT 1998, LNCS 1403, PP. 591–606, 1998.

[2] N. Asokan, V. Shoup and M. Waidner. *Optimistic fair exchange of digtial signatures.* IEEE Journal on Selected Areas in Communication 18(4), PP. 593–610, 2000.

[3] G. Ateniese. *Verifiable encryption of digirtal signatures and applications.* In ACM Transaction on Information and System Security 7(1), PP. 1–20, 2004.

[4] F. Bao, R. Deng and W. Mao. *Efficint and practical fair exchange protocols with off-line TTP.* In IEEE Symposium on Security and Privacy, PP. 77–85, 1998.

[5] B. Barak. *How to go beyond the black-box simulation barrier.* In Proc. of 42rd IEEE Symposium on Foundations of Computer Science (FOCS), PP. 106–115, 2001.

[6] N. Baricand B. Pfitzmann. *Collision-free accumulators and fail-stop signature schemes without trees.* In EUROCRYPT 1997, LNCS 1233, PP.480–494, 1997.

[7] M. Bellare and O. Goldreich. *On defining proofs of knowledge.* In Proc. of CRYPTO 1992, LNCS 740, PP. 390–420, 1992.

[8] M. Bellare and P. Rogaway. *Random oracles are practical: a paradigm for designing efficient protocols.* In Proc. of the 1st CCCS, PP. 62–73, 1993.

[9] M. Bellare and P. Rogaway. *The exact security of digital signatures-How to sign with RSA and Rabin.* In EUROCRYPT 1996, LNCS 1070, PP. 399–416, 1996.

[10] M. Ben-Or, O. Goldreich, S. Micali and R. Rivest. *A fair protcol for signing contracts.* IEEE Transaction on Information Theory 36(1), PP. 40–46, 1990.

[11] M. Blum, P. Feldman and S. Micali. *Non-interactive zero-knowledge and its applications.* In Proc. of 20rd ACM Symposium on Theory of Computing (STOC), PP. 103–112, 1988.

[12] D. Boneh. *The decisional diffie-hellman problem* . In the Third Algorithmic Number Theory Symposium, LNCS 1423, PP. 48–63, 1998.

[13] D. Boneh and X. Boyen. *Short signatures without random oracles.* In EUROCRYPT 2004, LNCS 3027, PP. 56–73, 2004.

[14] D. Boneh, and X. Boyen, *Short signatures without random oracles and the SDH assumption in bilinear groups,*

In Journal of Cryptology 21(2), PP. 149–177, 2008.

[15] D. Boneh, B. Lynn and H. Shacham. *Short signatures from the weil pairings.* In ASIACRYPT 2001, LNCS 2248, PP. 514–532, 2001.

[16] J. Boyar, D. Chaum, I. Damgård and T.P. Pedersen. *Convertible undeniable signatures.* In CRYPTO 1990, LNCS 537, pp. 189-205, 1991.

[17] S. Brands. *Untraceable off-line cash in wallets with observers.* In CRYPTO 1993, LNCS 773, PP. 302–318, 1993.

[18] S. Brands. *Rethinking public key infrastructure and digital certificates-building in privacy.* Ph.D Thesis, Eindhoven Institute of Technology, Eindhoven, Netherlands, 1999.

[19] G. Brassard, D. Chaum and C. Crepeau. *Minimum disclosure proofs of knowledge.* In Journal of Computer and System Sciences (37), PP. 156–189, 1988.

[20] J. Camp, M. Harkavy, J. D. Tygar and B. Yee *Anonymous atomic transactions.* In Proc. of Second USENIX Workshop on Electronic Commerce, PP. 123–133, 1996.

[21] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya and M. Meyerovich. *How to win the clone wars: efficient periodic n-times anonymous authentication.* Cryptology ePrint Archive, Report 2006/454, http://eprint.iacr.org/, 2006.

[22] J. Camenish, U. Maurer, and M. Stadler. *Digital payment systems with passive anonymity-revoking trustees.* In Journal of Computer Security, 5(1), PP. 254–265, 1997.

[23] J. Camenisch, and M. Michels, *Confirmer signature schemes secure against adaptive adversaries.* In EUROCRYPT 2000, LNCS 1807, PP. 243–258, 2000.

[24] R. Canetti, O. Goldreich and S. Halevi. *The random oracle methodology revisited.* In Proc. of 30th Annual ACM Symposium on Theory of Computing (STOC), PP. 209–218, 1998.

[25] R. Canetti, J. Kilian, E. Petrank and A. Rosen. *Black-box concurrent zero-knowledge requires $\omega(log(n))$ rounds.* In Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC), PP. 570–579, 2001.

[26] Chi-Chao Chang and Txonelih Hwang. *Anonymous proof of membership with ring signature.* In Proc. of IEEE International Conference on Electro Information Technology, PP. 5–9, 2005.

[27] D. Chaum. *Untraceable electronic mail, return addresses and digital pseudonyms.* In Journal of Communications of the ACM 24(2), PP. 84–88, 1981.

[28] D. Chaum. *Blind signatures for untraceable payments.* In CRYPTO 1982, PP. 199–203. 1983.

[29] D. Chaum. *Security without identification: Transaction systems to make big brother obsolete.* In Communications of ACM 28(10), PP. 1030–1044, 1985.

[30] D. Chaum. *Designated confirmer signatures.* In EUROCRYPT 1994, LNCS 950, PP. 86–91, 1994.

[31] D. Chaum, A. Fiat and M. Naor. *Untraceable electronic cash.* In CRYPTO 1988, LNCS 403, PP. 319–327, 1989.

[32] D. Chaum and T. Pedersen. *Wallet databases with observers.* In CRYPTO 1992, LNCS 740, PP. 89–105, 1993.

[33] D. Chaum and H. Van Antwerpen. *Undeniable signatures.* In CRYPTO 1989, LNCS 435, PP. 212–216, 1990.

[34] D. Chaum, E. van Heijst, B. Pfitzmann. *Cryptographically strong undeniable signatures unconditionally secure for the signer.* In CRYPTO 1991, LNCS 576, PP. 470-484, 1992.

[35] X. Chen, F. Zhang, Y. Mu and W. Susilo. *Efficient provably secure restrictive partially blind signatures from bilinear pairings.* In Proc. of Financial Cryptography and Data Security, LNCS 4107, PP. 251–265, 2006.

[36] R. Cramer, V. Shoup. *Signature schemes based on the strong RSA assumption*, In ACM Transaction on Information and System Security (ACM TISSEC) 3(3), PP. 161-185, 2000.

[37] I. Damgård. *Practical and provably secure release of a secret and exchange of signatures.* In Journal of Cryptology, 8(4), PP. 201–222, 1995.

[38] I. Damgård. *Efficient concurrent zero-knowledge in the auxiliary string model.* In EUROCRYPT 2000, LNCS 1807, PP. 418–430, 2000.

[39] I. Damgård, K. Dupont and M. Pedersen. *Unclonable group identification.* In Proc. of EUROCRYPT 2006, LNCS 4004, PP. 555–572, 2006.

[40] I. Damgård, D. Hofheinz, E. Kiltz, R. Thorbek. *Public-key encryption with non-interactive opening.* In CT-RSA 2008, LNCS 4964, PP. 239-255, 2008.

[41] I. Damgård and T. Pedersen. *New convertible undeniable signature schemes.* In EUROCRYPT 1996, LNCS 1070, PP. 372–386, 1996.

[42] I. Damgård and R. Thorbek. *Non-interactive proofs for integer multiplication.* In EUROCRYPT 2007, LNCS 4515, PP. 412-429, 2007.

[43] R. Deng, L.Gong, A. Lazar and W. Wang. *Practical protocol for certified electronic mail.* In Journal of Network and System Management 4(3), PP. 279–297, 1996.

[44] W. Diffie and M. Hellman. *New directions in cryptography.* In IEEE Transaction on Information Theory 22(6), PP. 644–654, 1976.

[45] Y. Dodis, P. Lee and D. Yum. *Optimistic fair exchange in a multi-user setting* In PKC 2007, LNCS 4450, PP. 118–133, 2007.

[46] Y. Dodis and L. Reyzin. *Breaking and repairing fair exchange from PODC 2003.* In Proc. of ACM Workshop On Digital Rights and Management (DRM), PP. 47–54, 2003.

[47] L. El Aimani. *Toward a generic construction of universally convertible undeniable signatures from pairing-based signatures.* In INDOCRYPT 2008, LNCS 5365, PP. 145–157, 2008.

[48] L. El Aimani, D. Vergnaud. *Gradually convertible undeniable signatures.* In Applied Cryptography and Network Security 2007, LNCS 4521, PP. 478–496, 2007.

[49] S. Even and Y. Yakobi. *Relations among public key signature systems.* Technical Report, Computer Science Department, Technicon, Haifa, Israel, 1980.

[50] N. Ferguson *Single term off-line coins.* In EUROCRYPT 1993, LNCS 765, PP. 318–328, 1994.

[51] U. Feige, A. Fiat and A. Shamir. *Zero-knowledge proofs of identity.* In Journal of Cryptology (1), PP. 77–94, 1988.

[52] A. Fiat and A. Shamir. *How to prove yourself: practical solutions of identification and signature problems.* In CRYPTO 1986, LNCS 263, PP. 186–194, 1987.

[53] A. Fujii, G. Ohtake, G. Hanaoka and K. Ogawa. *Anonymous authentication scheme for subscription services.* Lecture Notes in Artificial Intelligence 4694, PP. 975–983, 2007.

[54] E. Fujisaki and T. Okamoto. *Statistical zero-knowledge protocols to prove modular polynomial relations.* In CRYPTO 1997, LNCS 1294, PP. 16–30, 1997.

[55] J. Garay, M. Jakobsson and P. Mackenzie. *Abuse-free optimistic contract signing.* In CRYPTO 1999, LNCS 1666, PP. 449–466, 1999.

[56] J. Garay and P. Mackenzie. *Abuse-free multi-party contract signing.* In DISC 1999, LNCS 1693, PP. 151–165, 1999.

[57] R. Gennaro, S. Halevi and T. Rabin. *Secure hash and sign signatures without random oracle.* In EUROCRYPT 1999, LNCS 1592, PP. 123–139, 1999.

[58] C. Gentry, D. Molnar and Z. Ramzan. *Efficient designated confirmer signatures without random oracles or general zero-knowledge proofs.* In ASIACRYPT 2005, LNCS 3788, PP. 662–681, 2005.

[59] R. Gennaro, T. Rabin and H. Krawczyk. *RSA-based undeniable signatures.* In Journal of Cryptology 13(4), PP. 397-416, 2000.

[60] O. Goldreich. *Foundations of cryptography, basic tools.* Cambridge University Press, Volume (1), 2001.

[61] O. Goldreich. *Foundations of cryptography, basic tools.* Cambridge University Press, Volume (2), 2004.

[62] O. Goldreich. *A simple protocols for signing contracts.* In CRYPTO 1983, PP. 133–136, 1984.

[63] O. Goldreich and Y. Oren. *Definitions and properties of zero-knowledge proof systems.* In Journal of Cryptology 7(1), PP. 1–32, 1994.

[64] O. Goldreich, S. Micali and A. Wigderson. *Proofs that yield nothing but their validity.* In Journal of the ACM 38(3), PP. 690–728, 1991.

[65] S. Goldwasser, S. Micali and C. Rackoff. *The knowledge complexity of interactive proof systems.* In Proc. of 17th Symposium on the Theory of Computation, Providence, Rhode Island, 1985.

[66] S. Goldwasser, S. Micali and R. Rivest. *A digital signature scheme secure against chosen message attacks* In SIAM Journal of Computing 17(2), PP. 281–308, 1988.

[67] S. Goldwasser and E. Waisbard. *Transformation of digital signature schemes into designated confirmer signature schemes.* In TCC 2004, LNCS 2951, PP. 77–100, 2004.

[68] S. Goldwasser, S. Micali and R. Rivest. *A paradoxical solution to the signature problem.* In Proc. of 25th FOCS, PP. 441–448, 1984.

[69] J. Groth, R. Ostrovsk and A. Sahai *Perfect non-interactive zero knowledge for NP.* In EUROCRYPT 2006, LNCS 4004, PP. 338–359, 2006.

[70] J. Groth, R. Ostrovsky and A. Sahai *Non-interactive zaps and new techniques for NIZK.* In CRYPTO 2006, LNCS 4117, PP. 97–111, 2006.

[71] J. Groth and A. Sahai *Efficient non-interactive proof systems for bilinear groups.* In EUROCRYPT 2008, LNCS 4965, PP. 415–432, 2008.

[72] X. Huang, Y. Mu, W. Susilo and W. Wu. *Provably secure pairing-based convertible undeniable signature with short signature length.* In Pairing 2007, LNCS 4575, PP. 367-391, 2007.

[73] X. Huang, Y. Mu, W. Susilo and W. Wu. *A generic construction for universally-convertible undeniable signatures.* In CANS 2007, LNCS 4856, PP. 15–33, 2007.

[74] Q. Huang, G. Yang, D. Wong and W. Susilo. *Ambigous optimistic fair exchange.* In ASIACRYPT 2008, LNCS5350, PP. 74–89, 2008.

[75] Q. Huang, G. Yang, D.S Wong and W. Susilo. *Efficient optimistic fair exchange secure in the multi-user setting and chosen key model without random oracles.* In CT-RSA 2008, LNCS 4964, PP. 106–120, 2008.

[76] M. Jakobsson, K. Sako and R. Impagliazzo. *Designated verifier proofs and their applications.* In EUROCRYPT 1996, LNCS 1070, PP. 143–154, 1996.

[77] M. Jakobsson and M. Yung. *Revokable and versatile electronic money.* In Proceedings of the 3rd CCCS, PP. 76–87, 1996.

[78] A. Joux. *A one round protocol for tripartite diffie-hellman.* In Journal of Cryptology 17, PP. 263–276, 2004.

[79] A. Juels, M. Luby, and R. Ostrovsky. *Security of blind digital signatures.* In CRYPTO 1997, LNCS 1294, PP. 150–164, 1997.

[80] K. Kurosawa and T. Takagi. *New approach for selectively convertible undeniable signature schemes.* In ASIACRYPT 2006, LNCS 4284, PP. 428-443, 2006.

[81] F. Laguillaumie, P. Paillier and D. Vergnaud. *Universally convertible directed signatures.* In ASIACRYPT 2005, LNCS 3788, PP. 682–701, 2005.

[82] F. Laguillaumie and D. Vergnaud. *Time-selective convertible undeniable signatures.* In CT-RSA 2005, LNCS 3376, PP. 154–171, 2005.

[83] C. H. Lim and P. J. Lee. *Modified Maurer-Yacobis scheme and its applications.* In Auscrypt 1992, LNCS 718, PP. 308-323, 1993.

[84] M. Liskov, and S. Micali, Online-untransferable signatures, In Proc of PKC 2008, LNCS 4939, PP. 248–267, 2008.

[85] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham and B. Waters. *Sequential aggregate signatures and multisignatures without random oracles.* In EUROCRYPT 2006, LNCS 4004, PP. 465–485, 2006.

[86] A. Lysyanskaya, R. Rivest, A. Sahai and S. Wolf. *Pseudonym systems.* In Selected Areas in Cryptography, LNCS 1758, PP. 184–199, 1999.

[87] O. Markowitch and S. Saeednia. *Optimistic fair exchange with transparent signature recovery.* In 5th International Conference on Financial Cryptography, LNCS 2339, PP. 339–350, 2001.

[88] A. Menezes, T. Okamoto and S. Vanstone. *Reducing elliptic curve logarithms to logarithms in a finite field.* In IEEE Transaction of Information Theory 39(5), PP. 1639–1646, 1993.

[89] S. Micali. *Simple and fast optimistic protocols for fair electronic exchange.* In Proc. of 22th Annual ACM symposium on principles of distributed computing(PODC), PP. 12–19, 2003.

[90] S. Micali. *Certified e-mail with invisible post offices.* Invited Presentation at RSA Conference, 1997.

[91] M. Michels, M. Stadler. *Efficient convertible undeniable signature schemes.* In SAC 1997, PP. 231–244, 1997.

[92] M. Michels, M. Stadler. *Generic constructions for secure and efficient confirmer signature schemes.* In EUROCRYPT 1998, LNCS 1403, PP. 406–421, 1998.

[93] T. Okamoto. *Designated confirmer signatures and public-key encryptions are equivalent.* In CRYPTO '94, LNCS 839, PP. 61–74, 1994.

[94] T. Okamoto. Efficient blind and partially blind signatures without random oracles. In TCC 2006, LNCS 3876, PP. 80–99, 2006.

[95] T. Okamoto. Efficient blind and partially blind signatures without random oracles. Cryptology ePrint Archive, Report 2006/102, 2006. `http://eprint.iacr.org/`.

[96] P. Paillier. *Public key cryptosystems based on composite degree residuosity classes.* In EUROCRYPT 1999 , LNCS 1592, PP. 223–238, 1999.

[97] J. Park, E. Chong, H. Siegel and I. Ray. *Constructing fair exchange protocols for e-commerce via distributed computation of RSA signatures.* In Proc. of 22th Annual ACM symposium on Principles of Distributed Computing(PODC), PP. 172–181, 2003.

[98] T. Pedersen. *Non-interactive and information-theoretic secure verifiable secret sharing.* In CRYPTO 1991, LNCS 576, PP. 129–140, 1991.

[99] B. Pfitzmann, M. Schunter and M. Waidner *Optimal efficiency of optimistic contract signing.* In the 17th Symposium on Principles of Distributed Computing (PODC), PP. 113–122, 1998.

[100] D. Pointcheval, and J. Stern. *Provably secure blind signature schemes.* In ASIACRYPT 1996, LNCS 1163, PP. 252–265, 1996.

[101] D. Pointcheval and J. Stern. *Security proofs for signature schemes.* In EUROCRYPT 1996, LNCS 1070, PP. 387-398, 1996.

[102] D. Pointcheval, and J. Stern. *Security arguments for digital signatures and blind signatures.* In Journal of Cryptology 13(3), PP. 361–396, 2000.

[103] M. Rabin. *Transaction protection by beacons.* In Journal of Computer and System Sciences 27, PP. 256–267, 1983.

[104] R. Rivest. *The MD5 Message Digest Algorithm, Request for Comments (RFC)1321.* Internet Activities Board, Internet Privacy Task Force, April 1992.

[105] R. Rivest, A. Shamir and L. Adleman. *A method for obtaining signatures and public-Key cryptosystems.* In Communications of the ACM 21(2), PP. 120–126, 1987.

[106] J. Rompel. *One-way functions are necessary and sufficient for secure signatures.* In Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC), PP. 387–394, 1990.

[107] C. P. Schnorr. *Efficint signature generation by smart cards.* In Journal of Cryptology 4(3), PP. 161–174, 1991.

[108] L. A. Schoenmakers. *An efficient electronic payment system withstanding parallel attacks.*

Technical report CS-R9522, CWI, Amsterdam, 1995.

[109] M. Scott. *Authenticated ID-based key exchange and remote log-in with insecure token and PIN number.* E-Print, http://eprint.iacr.org/2002/164.pdf, 2002.

[110] W. Shaobin, S. Na and H. Lei. *Fair e-cash payment model on credit overdraft.* In Proc. of 2006 IEEE Asia-Pacific Conference on Service Computing, 0-7695-2751-5/06.

[111] A. Shamir and Y. Tauman. *Improved online/offline signature schemes* In CRYPTO 2001, LNCS 2139, PP. 355–367, 2001.

[112] V. Shoup. *OAEP reconsidered.* In Journal of Cryptology, 15(4), PP. 223–249, 2008.

[113] S. von Solms and D. Naccache. *On blind signatures and perfect crimes.* In Journal of Computers and Security 11, PP. 581–583, 1992.

[114] M. Tompa and H. Woll. *Random self-reducibility and zero-knowledge interactive proofs of posession of information.* In 28th Annual Symposium on Foundation of Computer Science, PP. 472–482, 1987.

[115] B. Waidner and M. Waidner. *Round optimal and abuse-free optimistic multi-party contract signing.* In ICALP 2000, LNCS 1853, PP. 524–535, 2000.

[116] G. Wang. *An abuse-free fair contract signing protocol based on the RSA signature.* WWW 2005, ACM 1-59593-046-9/05/0005, 2005.

[117] G. Wang, J. Baek, D. Wang and F. Bao. *On the generic and efficient constructions of secure designated confirmer signatures.*

In PKC 2007, LNCS 4450, PP. 43–60, 2007.

[118] B. Waters. *Efficient identity-based encryption without random oracles.* In EUROCRYPT 2005, LNCS 3494, PP. 114–127, 2005

Index

CRM
CENTRE DE RECERCA MATEMÀTICA

Campus de Bellaterra, Edifici C
08193 Bellaterra, Spain
Tel.: +34 93 581 10 81
Fax: +34 93 581 22 02
crm@crm.cat
www.crm.cat