Special Signature Schemes and Key Agreement Protocols

Caroline J. Kudla

Technical Report
RHUL–MA–2006–8
6 October 2006

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Special Signature Schemes and Key Agreement Protocols

Caroline J. Kudla

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
Department of Mathematics
Royal Holloway, University of London
2006

Declaration

These doctoral studies were conducted under the supervision of Kenneth G. Paterson.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Caroline J. Kudla
January, 2006

Acknowledgements

My sincere thanks go to my supervisor, Kenny Paterson, whose dedication has been outstanding, and who has given generously of his time to guide and encourage me in my work over the last 3 years. His invaluable comments, constructive criticism and unwavering support have been crucial in helping me to achieve my goals and develop as a researcher. I would also like to thank my advisor, Chris Mitchell, for his encouragement in my various endeavors.

I would like to express my deepest gratitude to HP Labs in Bristol, whose generous financial support has enabled me to pursue my studies. HP also permitted me to work from their labs, allowing me insight into the world of industrial research and providing me with invaluable experience. In particular, I would like to thank Keith Harrison for all his encouragement and advice during my time at HP, and Liqun Chen for her guidance and support. It has been great working with and learning from you both.

I would also like to thank my friends and family, near and far, for all their encouragement over the last 3 years. I have finished studying - at last! Finally, I want to thank my beloved husband, Guillaume. He has been fantastic in every way, and I couldn't have done this without his constant love and support.

Abstract

This thesis is divided into two distinct parts. The first part of the thesis explores various deniable signature schemes and their applications. Such schemes do not bind a unique public key to a message, but rather specify a set of entities that could have created the signature, so each entity involved in the signature can deny having generated it. The main deniable signature schemes we examine are ring signature schemes.

Ring signatures can be used to construct designated verifier signature schemes, which are closely related to designated verifier proof systems. We provide previously lacking formal definitions and security models for designated verifier proofs and signatures and examine their relationship to undeniable signature schemes.

Ring signature schemes also have applications in the context of fair exchange of signatures. We introduce the notion of concurrent signatures, which can be constructed using ring signatures, and which provide a "near solution" to the problem of fair exchange. Concurrent signatures are more efficient than traditional solutions for fair exchange at the cost of some of the security guaranteed by traditional solutions.

The second part of the thesis is concerned with the security of two-party key agreement protocols. It has traditionally been difficult to prove that a key agreement protocol satisfies a formal definition of security. A modular approach to constructing provably secure key agreement protocols was proposed, but the approach generally results in less efficient protocols.

We examine the relationships between various well-known models of security and introduce a modular approach to the construction of proofs of security for key agreement protocols in such security models. Our approach simplifies the proof process, enabling us to provide proofs of security for several efficient key agreement protocols in the literature that were previously unproven.

Contents

Chapter 1

Introduction

1.1 Contributions

This thesis is divided into two distinct parts. The first part of the thesis explores various non-standard digital signature schemes. The signature schemes we examine differ from standard digital signature schemes in the sense that they do not bind a unique public key to a message. Rather, the signature schemes that we consider specify a set of entities that could have created the signature. Such a signature therefore binds the set of public keys to the message being signed, and no single entity (or public key) is implicated as the signer.

We say that such signatures are *deniable* since the identity of the signer is "hidden" in a group of possible signers, and therefore each entity involved in the signature can deny having generated it. We examine various types of deniable signature schemes and ways in which they may be used.

The most common deniable signature schemes (i.e. signature schemes satisfying the above properties) are *ring signature schemes*, and indeed all the other deniable signature schemes that we consider can be derived from ring signature schemes.

The original motivation for ring signature schemes was to be able to leak information without being held accountable. An entity could sign some declaration in such a way that he is not implicated by the signature, but rather a group of entities of his choice (including the signer) is implicated. In this way, the real signer is able to "hide" his involvement within a group of possible signers, and can therefore deny any involvement in producing the signature.

Although this is an interesting application, ring signature schemes also possess some additional properties in the two-party case. In this case the ring signature could have been produced by either of two parties but where it is infeasible for a third party to determine

which of the two possible signers generated the signature. Both possible signers can deny having produced the signature to a third party, but the two parties involved both know who created the signature. The signer knows that he created the signature, and the other party (the non-signer) knows that that he did not create the signature and can therefore uniquely identify the signer.

Two party ring signature schemes can be used to construct *designated verifier signature schemes*. The goal of designated verifier signature schemes is for a signer to prove to a specific verifier (called a designated verifier) that they have signed a message. However the verifier should not be able to prove this to a third party. Two-party ring signature schemes exactly meet this requirement, although there are also other ways to construct designated verifier signatures.

We also examine the related notion of *designated verifier proof systems*. Here the goal is not to sign a message, but rather to prove the validity of some statement to a designated verifier. A designated verifier proof could have been created by the prover or the designated verifier, so the verifier cannot transfer the proof to a third party since he himself could have generated it. Designated verifier proofs are usually used in the context of *undeniable signature schemes*, so we consider them in this context as well.

The last type of deniable signature schemes that we consider are introduced in the context of fair exchange of signatures. The goal of a fair exchange protocol is to allow two entities to exchange signatures in a *fair* way. By this we mean that by engaging in a protocol, either each party obtains the other's signature, or neither party does. It should not be possible for one party to terminate the protocol at some stage leaving the other party committed when they themselves are not.

Existing techniques for solving this problem either involved a timed release of signatures or the use of a special purpose trusted third party. Timed release of signatures is highly interactive and cannot always guarantee complete fairness, and in practice, suitable trusted third parties can be difficult to find. We therefore introduce the notion of *concurrent signature schemes* which provide an efficient (partial) solution to the problem of fair exchange of signatures that does not require a trusted third party, but which forfeits some of the security guarantees normally expected from a full fair exchange solution.

In order to construct a concurrent signature scheme, we require a special type of deniable signature, which we call a *non-transferable signature*. Non-transferable signatures can be constructed using ring signature schemes of a specific form.

The second part of the thesis is concerned with the study of key agreement protocols.

Since the seminal paper by Diffie and Hellman [50] which allowed two parties to generate a shared secret key without having any previous shared secret, the study of key agreement protocols using asymmetric techniques has been very fertile.

It is difficult to design secure cryptographic primitives, and particularly so for key agreement protocols. The literature contains many attempts at constructing secure key agreement protocols, many of which turned out to be insecure.

Unlike many other cryptographic primitives such as encryption and digital signature schemes, the correct definition of a secure key agreement protocol is still debated. Furthermore, assuming a suitable definition of security has been found, constructing a proof that the concrete key agreement protocol concerned meets the required definition of security is usually very difficult.

The later chapters of this thesis consider various well-known models of security (and corresponding security definitions) for key agreement protocols. We examine what security guarantees the various definitions in fact provide, and we analyze what concrete properties various security definitions demand.

The identification of certain necessary requirements for a protocol to be secure allows us to develop a modular method by which a certain class of key agreement protocols may be proven secure. If protocols cannot be proven secure in their current form, our modular technique often identifies the cause of the problem, and in many cases the problem is easily addressed by making some small modifications to the protocol. The proof techniques developed greatly simplify the proof process and we are able to provide proofs of security for various previously unproven protocols.

1.2 Overall Structure

Part I In Chapter 2, we cover the nomenclature and definitions that are relevant for the subsequent chapters in Part I of the thesis. Here we define the concept of a digital signature scheme and security for digital signatures. We also introduce the topic of provable security, random oracles, and proof techniques for the signature schemes in later chapters.

In Chapter 3, we introduce ring signature schemes. We give a formal definition of ring signature schemes and their security, and we present an efficient concrete ring signature scheme.

In Chapter 4, we introduce the notions of non-interactive designated verifier (NIDV) proof systems and undeniable signature schemes (which use NIDV proofs). We provide

formal definitions for these schemes and their security. We go on to provide a concrete undeniable signature scheme using NIDV proofs. We then introduce the notion of designated verifier signature schemes and present formal definitions for these schemes and their security. We show that secure two-party ring signatures are in fact also secure DV signatures, and we present two concrete designated verifier signature schemes (one of which is a ring signature scheme, and the other of which is derived from an NIDV proof), and we provide proofs of security for these concrete schemes.

In Chapter 5, we introduce concurrent signature schemes, giving formal definitions of concurrent signature schemes and their security. We go on to show how a specific two party ring signature scheme can be used to construct a concurrent signature scheme. We provide a proof of security for the resulting concrete scheme.

Part II In Chapters 6 and 7 we introduce the nomenclature and relevant background for the second part of the thesis which is concerned with key agreement protocols. We also introduce the topic of provable security for key agreement protocols as well as various security models and definitions of security for key agreement protocols.

In Chapter 8 we analyze certain of the security models presented in Chapter 7, and present our modular techniques for constructing proofs of security for key agreement protocols in these security models. We then consider various key agreement protocols in the literature which lack proofs of security (or complete proofs of security) and use our techniques to construct full proofs of security for these protocols in appropriate security models.

1.3 Publications

This thesis contains certain material that was previously published with L. Chen [39, 40], material that was published with K.G. Paterson [73, 74] as well as material that was published with L. Chen and K.G. Paterson [41].

The content of [74] forms a basis for Chapter 4, the content of [41] forms a basis for Chapter 5, and the content of [73] forms a basis for the content of Chapter 8. In addition, a protocol appearing in [39, 40] is also presented in Chapter 8.

Part I

Special Signature Schemes

Chapter 2

Preliminary Topics

2.1 Mathematical Background

2.1.1 Complexity Theory

The goal of complexity theory is to provide mechanisms by which computational problems may be classified in terms of the resources required to solve them. The resources measured are usually time, and occasionally storage space. We now define some of the terminology required.

An *algorithm* is a computational procedure which takes a variable input and terminates with some output. If an algorithm follows the same execution path each time it is executed with the same input, then we say that the algorithm is *deterministic*. By contrast, a *randomized* algorithm's execution path differs each time it is executed on the same input since its decisions rely on a supply of random bits.

The *running time* of an algorithm on a particular input is the number of steps or primitive operations executed before the algorithm terminates.

The *worst-case running time* of an algorithm is an upper bound on the running time of an algorithm for any input. This is usually expressed as a function of the input size.

The *expected running time* of an algorithm is the average running time of an algorithm over all inputs of a specific size. This is usually expressed as a function of the input size.

Since the exact running time of an algorithm is often difficult to derive, we often rely on approximations of the running time. In particular, we often refer to the order of the asymptotic upper bound of the running time of an algorithm using the big-O notation.

Definition 2.1 For two functions $f(l)$ and $g(l)$, we say that $f(l) = O(g(l))$ if there exists a positive constant c and a positive integer l_c such that for all $l > l_c$, $0 \leq f(l) \leq cg(l)$.

Two frequently used concepts in cryptology are the notions of *negligible functions* and *polynomial time algorithms*. We define these as follows.

Definition 2.2 A function $\epsilon(l)$ is *negligible* in parameter l if for all $c \geq 0$, there exists an integer $l_c > 0$ such that for all $l > l_c$, $\epsilon(l) < l^{-c}$.

Definition 2.3 A *polynomial time algorithm* is an algorithm whose worst-case running time is of the form $O(l^c)$ where l is the input size and c is a constant. An algorithm whose running time cannot be bounded in this way is called a *non-polynomial time algorithm*.

In general, we regard polynomial time algorithms as being *efficient*, and non-polynomial time algorithms as being *inefficient*. We call a problem that cannot be solved in polynomial time *intractable* or *infeasible*.

The complexity of algorithms is always measured with respect to some parameter l. For cryptographic algorithms, we call this parameter the *security parameter*. By manipulating the size of this parameter, we can change the key lengths and group sizes of the cryptographic scheme, and this in turn affects the security of the scheme.

2.1.2 Abstract Algebra

We let the set of natural number be denoted by $\mathbb{N}$, and the set of integers be denoted by $\mathbb{Z}$. For any positive integer n, we denote the ring of integers modulo n by $\mathbb{Z}_n = \{0, 1, .., n-1\}$. We denote the group of units of $\mathbb{Z}_n$ (that is, elements relatively prime to n and therefore having an inverse under multiplication) by $\mathbb{Z}_n^*$, and we denote by $\phi(n)$ the number of integers in $\{1, .., n\}$ that are relatively prime to n. The function ϕ is called the *Euler totient function*.

We recall that if n is prime, then $\phi(n)$=$n-1$, and if $n = p.q$ where p and q are relatively prime, then $\phi(n) = \phi(p).\phi(q)$.

Let G be a group with binary operation $*$.

Definition 2.4 The number of elements in group G is denoted by $|G|$ and is called the *order* of G. Group G is called *finite* if $|G|$ is finite.

Definition 2.5 A group G is called *cyclic* if there exists an element $g \in G$ such that for each element $a \in G$, there exists an integer i such that $a = g^i$. Such an element g is called a *generator* of G.

Theorem 2.1 [106] If p is prime, then $\mathbb{Z}_p^*$ is a cyclic group.

If H is a non-empty subset of group G and H itself is a group, then we call H a *subgroup* of G. If $a \in G$, then the set of all powers of a forms a cyclic subgroup of G and is denoted by $\langle a \rangle$. If $\langle a \rangle = G$, then a is a generator of G. The *order* of $a \in G$ is defined to be the least positive integer t such that $a^t = 1$. So if the order of a is t, then $|\langle a \rangle| = t$. If no such integer t exists then the order of a is defined to be ∞.

Theorem 2.2 (Lagrange) If G is a multiplicative group of order n, and $a \in G$, then the order of a divides n.

Corollary 2.3 If G is a multiplicative group, and $a \in G$ has order t, then the order of a^k is $t/gcd(t, k)$.

Theorem 2.4 [106] If p is prime, then an integer $\alpha \in \mathbb{Z}_p^*$ is a generator of $\mathbb{Z}_p^*$ if and only if $\alpha^{\phi(p)/q} \neq 1 \bmod p$ for each prime divisor q of $\phi(p)$ and therefore $\mathbb{Z}_p^*$ has $\phi(\phi(p)) = \phi(p-1)$ generators.

Theorem 2.4 provides us with an efficient method for testing whether a given element is indeed a generator for $\mathbb{Z}_p^*$ if we know the factorization of $p-1$. For example, if $p = 2q+1$, where p and q are prime, then $\phi(p) = p - 1 = 2q$, and $\mathbb{Z}_p^*$ has $\phi(2q) = \phi(2).\phi(q) = q - 1$ generators. If α is a generator of $\mathbb{Z}_p^*$, then by Corollary 2.3, α^2 has order q and therefore generates the subgroup of $\mathbb{Z}_p^*$ of order q.

2.1.3 The Discrete Logarithm and Diffie-Hellman Problems

We define here the computational problems on which the security of our cryptographic schemes are based. We start by defining the discrete logarithm problem.

Definition 2.6 We say that an algorithm ParamGen is a *discrete logarithm (DL) parameter generator* if, for input a security parameter $l \geq 1$, ParamGen runs in polynomial time and generates a prime q as well as the description of a finite cyclic group G of prime order q. ParamGen outputs the description of group G.

We assume that the description of the group G contains the group order q. We also assume that the group operations can be performed efficiently, group elements can efficiently be sampled with uniform distribution, and that group membership and equality of group members can be tested efficiently.

Definition 2.7 Let G be a group of prime order q which was output by $\mathsf{ParamGen}(l)$, and let g be a random generator of G. Given a random element $a \in G$, the *Discrete Logarithm Problem (DLP)* in G is to find a $w \in \mathbb{Z}_q$ such that $a = g^w$.

Such a w is called the *discrete logarithm* of a in G, also denoted $DL(a, g)$. We say that an algorithm A has advantage ϵ in solving the DLP in G if

$$Pr[A(a, g) = DL(a, g)] \geq \epsilon$$

This probability is measured over the random choices of $a, g \in G$ and the random inputs of A, if any.

The DL assumption states that no polynomial time algorithm A has non-negligible advantage (in l) in solving the DLP for G generated by $\mathsf{ParamGen}(l)$.

The Diffie-Hellman problem [87] is closely related to the DLP and is the basis for the security of many cryptographic primitives.

Definition 2.8 Let G be a group of prime order q which was output by $\mathsf{ParamGen}(l)$, and let g be a random generator of G. Given g and the random elements g^a and g^b in G, the *Computational Diffie-Hellman Problem (CDHP)* in G is to find $g^{ab} \in \mathbb{G}$.

We say that an algorithm A has advantage ϵ in solving the CDHP in G if

$$Pr[A(g^a, g^b) = g^{ab}] \geq \epsilon$$

This probability is measured over the random choices of $g, g^a, g^b \in G$ and the random inputs of A, if any.

The CDH assumption states that no polynomial time algorithm A has non-negligible advantage (in l) in solving the CDHP for G generated by $\mathsf{ParamGen}(l)$.

If we suppose that the DLP can be efficiently solved, then one can solve the CDHP as follows. Given g and the random elements g^a and g^b in G, find a from g^a by solving the DLP, and then compute $(g^b)^a = g^{ab}$. Whether the two problems are in fact equivalent remains unresolved in general, although they are known to be equivalent in certain circumstances [20, 42, 45, 48, 80, 82].

Closely related to the CDHP are the decisional Diffie-Hellman and Gap Diffie-Hellman problems.

Definition 2.9 Let G be a group of prime order q which was output by $\mathsf{ParamGen}(l)$, and let g be a random generator of G. Given g and the random elements g^a, g^b and g^c in G, the *Decisional Diffie-Hellman Problem (DDHP)* in G is to determine if $g^{ab} = g^c$.

Definition 2.10 Let G be a group of prime order q which was output by $\mathsf{ParamGen}(l)$. The *Gap Diffie-Hellman Problem (GDHP)* in G is to solve the CDHP in G given a hypothetical polynomial time subroutine, or *oracle*, that solves the DDHP in G.

2.1.4 Additional Notation

We now define some additional notation and terminology that is used in this thesis.

Definition 2.11 Let a, b be integers. Then a *divides* b if there exists an integer c such that $b = ac$. If a divides b, then this is denoted by $a|b$.

For any group G, we defined $|G|$ to be the order (or size) of the group. Similarly, if X is any set, then $|X|$ denotes the number of elements in X, or the size of X.

We let $\{0,1\}^*$ denote the set of all bitstrings of unspecified length. For any string $s \in \{0,1\}^*$, we denote the bitlength of s by $|s|$ and we denote the concatenation of strings s and t by $s\|t$. If s and t are not strings, then $s\|t$ denotes the concatenation of the bit representation of s and t in the appropriate endian.

2.2 Public Key Cryptography

2.2.1 Basic Terminology and Cryptographic Goals

We start by defining some basic terminology which we require when discussing cryptography and its goals. An *entity* (or party) is someone or something which sends, receives or manipulates information. It may be a person, or it may be some computing device.

When two parties communicate, we refer to the *channel* as the means of conveying information from one entity to another. We call the legitimate transmitter of information the *sender*, and the intended recipient of the information the *receiver*.

The objective of cryptography is to provide security for information using mathematical techniques. This objective is usually broken down into four key goals:

- **Confidentiality** which is concerned with keeping data secret from all but those authorized to access it.
- **Data integrity** which is concerned with being able to detect the manipulation of data by unauthorized entities.
- **Authentication** which is related to identification, of both entities and data. It is therefore subdivided into two classes: entity authentication, which is concerned with

verifying the identity of entities, and data origin authentication, which is concerned with verifying the origin of data (which implicity provides data integrity as well).

- **Non-repudiation** which is concerned with preventing entities from denying previous commitments or actions.

We now define some of the mechanisms used to achieve security goals or objectives.

Definition 2.12 A *(cryptographic) scheme* is a set of algorithms used to provide some cryptographic service. For example, a signature scheme may be used to provide authentication.

Definition 2.13 A *(cryptographic) protocol* is a distributed algorithm defined by a sequence of steps specifying the actions required by two or more entities to achieve a specific security objective.

Technically, all schemes can be called protocols, however we will reserve the term protocol for mechanisms that specifically demand interaction (or communication) between parties.

A *cryptographic primitive* is a basic tool used to provide some security functionality. For example, a signature scheme may be used as a cryptographic primitive within some protocol.

2.2.2 Key Infrastructures

The study of cryptography can be divided into the areas of symmetric cryptography and asymmetric cryptography. Usually, entities that wish to securely communicate using cryptographic techniques have to first share a secret key. The security of the communications depends on this key, and the key has to be kept secret from all other entities for as long as the communication is to remain secure.

A fundamental problem in symmetric key cryptography is that the secret keys must be distributed between the relevant entities before secure communication can commence. In 1976, Diffie and Hellman [50] revolutionized cryptography by providing a solution to this problem via the notion of asymmetric cryptography, also known as *Public Key Cryptography (PKC)*.

In a public key setting, each entity I has two distinct keys, a public key PK_I, and a private key SK_I, which the entity can generate locally. The private key must be kept

secret, but the public key can be widely distributed without compromising the secrecy of the private key.

The fundamental idea of PKC is that, in order to securely communicate with an entity, one need only know the public key of that entity. However before using a public key, one must be entirely convinced that the public key does in fact belong to the correct entity. So PKC transforms the problem of securely distributing secret keys into the problem of distributing authentic public keys.

A security infrastructure designed to distribute and subsequently manage public keys is called a *public key infrastructure (PKI)*. The traditional method of distributing public keys in an authentic manner is to use a trusted authority (TA). The most common approach is for an entity to register their public key with the TA, who authenticates the entity and attests to the authenticity of the public key by issuing a digital certificate binding the entity's identity with their public key. Such a TA is called a *certification authority (CA)*.

An alternative approach to managing public keys is enabled by the use of identity-based cryptography [18, 100]. In this environment, an entity's public key is derived directly from an entity's identity (or identifying information, e.g. their email address), eliminating the requirement for a certificate binding the entity's identity to their public key. Although certificates are no longer required, an entity must still contact a TA in order to obtain their private key. The TA must authenticate the entity, generate the private key from the given public key (or identity) and securely deliver the private key to the authenticated entity. Such a TA is often referred to as a *Key Generation Centre* (KGC).

For the remainder of this thesis, we will assume that the public keys being used are authenticated using some PKI.

2.2.3 Digital Signature Schemes

We now introduce the concept of a digital signature scheme, which is one of the most important cryptographic primitives enabled by PKC. Digital signature schemes are the focus of the first part of this thesis, and are fundamental in providing various cryptographic services such as entity authentication, data origin authentication, data integrity, and non-repudiation.

Informally, digital signature schemes provide a means by which entities can bind their identity (or public key) to a piece of information (usually referred to as a message).

A digital signature scheme is defined via the following algorithms [87]:

Setup(l): which takes as input security parameter l and outputs a set of parameters $params$.

KeyGen($params$): which takes as input the public parameters $params$ and outputs a public key PK and a private key SK.

Sign(m, SK): which takes as input a message m and a private key SK and produces a signature σ for the message m.

Verify(m, PK, σ): which takes as input a message m, a public key PK and a signature σ, and outputs either *accept* or *reject*.

Suppose that an entity I has public and private key pair $\langle PK_I, SK_I \rangle$. We say that entity I *signs* a message m if I runs Sign on input (m, SK_I) to produce a signature σ_I. We then call σ_I entity I's signature on message m.

If Verify outputs *accept* when run on input a message m, a public key PK and a signature σ, then we say that σ is a *valid* signature for entity I on message m.

Informally, we require the following properties from a digital signature scheme:

1. Signatures produced by the Sign algorithm should be valid (that is, accepted by Verify).

2. It should be computationally infeasible for any entity other than I to produce a valid signature for I on any message m.

In particular, the second property is one of the requirements for a digital signature scheme to provide non-repudiation. We will provide a more formal definition of security for digital signature schemes after having introduced the topic of provable security.

2.3 Provable Security

It is very difficult to design secure cryptographic schemes. This is illustrated by the number of cryptographic schemes that have been proposed over time and in which flaws have subsequently been discovered. These flaws may be due to new attacks which were not previously known, or simply due to inadequate security analysis on the part of the scheme's designers. It is therefore crucial to rigorously analyze a scheme for possible security flaws before it is implemented and used in practice.

Traditionally, a cryptographic scheme was analyzed by constructing convincing arguments that a scheme was immune to the best currently known attack methods because

the resources required were greater than those of any reasonable attacker. Such analysis is called *heuristic analysis* and schemes that survive such analysis are said to have *heuristic security*.

However heuristic security is only a measure of security against currently known attacks. It gives little assurance that a scheme is in fact secure since it cannot guarantee that no previously undiscovered attack cannot compromise the scheme's security.

In 1984, Goldwasser and Micali [61] introduced the paradigm of *provable security* and lead the way for far more rigorous treatments of cryptographic schemes by developing precise definitions and appropriate "models" of security for various cryptographic primitives. Goldwasser, Micali and Rivest [63] were the first to formalize a notion of security for digital signature schemes. They also presented a scheme that satisfied their definition under reasonable assumptions.

In order to analyze cryptographic primitives, we introduce some useful terminology. An *adversary* or *attacker* of a cryptographic scheme is an entity which tries to defeat the intended security objective of the scheme. A *passive adversary* is one which only monitors communication channels. An *active adversary* is one which attempts to delete, add, or in some way modify the transmissions on a channel. When reasoning about cryptographic schemes under attack, the entities involved in such schemes, as well as the attacker(s) are modelled as interactive *Turing machines*, which can be seen as abstractions of modern computers. In general, these Turing machines are *probabilistic*, meaning that they have access to a supply of random bits.

Giving a precise definition of security is an important step when analyzing the security of a cryptographic scheme. Firstly, the objectives of the scheme need to be clearly understood. Then we need to define what it means for the adversary to *break* the scheme, that is, what the adversary's goal is in attacking the scheme. Finally, we need to define the adversarial model, or the resources to which the adversary has access when attacking the scheme.

Once an appropriate definition of security has been formulated, a mathematical *proof of security* for a given scheme can then be derived. This usually takes the form of a *reduction*: assuming that a successful adversary of the scheme exists, then one shows that such an adversary can be used to solve some underlying problem. If this problem is believed to be intractable, then no such adversary can exist, and the scheme is assumed to be secure.

We note that a proof of security (for an appropriate definition of security) does not guarantee security in an unconditional sense. Rather, it clearly identifies one or more

underlying assumptions (usually related to a well-studied problem) which must be violated in order to subvert the security of the scheme. In addition, the security of a scheme is usually only defined within certain bounds. For example, an attacker is usually assumed to have clearly defined and polynomially limited resources. If these bounds are breached, then the proof of security becomes inapplicable.

In the context of digital signature schemes, the goal of the adversary is to *forge* signatures. However there are different criteria for what it means to break a signature scheme.

1. **Total break:** An adversary is able to produce signatures on arbitrary messages as if he were the true signer. This is akin to recovering the private key of the signer.

2. **Selective forgery:** An adversary is able to produce a valid signature for a message of his choice.

3. **Existential forgery:** An adversary is able to create a valid signature for at least one message. In this case the adversary may have no control over the message corresponding to the signature obtained.

An adversary may launch different types of attacks against a digital signature scheme depending on the resources to which he has access. We distinguish between the following types of attack:

1. **Key only attack:** An adversary knows only the signer's public key.

2. **Known message attack:** An adversary has access to signatures on messages known to the adversary, but not chosen by him.

3. **Chosen message attack:** An adversary has access to signatures on a chosen list of messages before attempting to break the signature scheme. This attack is *non-adaptive* since the adversary must choose the list of messages to be signed before any of the signatures are obtained.

4. **Adaptive chosen message attack:** An adversary is able to use the signer as an *oracle*, meaning that the adversary may request signatures on messages of his choice, and these message may depend on the signer's public key, on previous messages or on previous signatures obtained.

The designers of digital signature schemes would like the security of their schemes to be as strong as possible. This means that the scheme should resist even the weakest types of

forgery against an adversary with the most resources. If this is the case, then the scheme will certainly be secure against more serious forgeries by adversaries with more limited resources. It is therefore common to evaluate the security of a digital signature scheme by its resistance to existential forgeries under an adaptive chosen message attack.

We now present a formal definition for the security of a digital signature scheme.

2.3.1 Security Definition for Digital Signatures

Security for digital signature schemes is defined via the following two notions.

Correctness

A digital signature scheme is *correct* if, for any σ produced by running Sign on message m and private key SK (where PK is the corresponding public key), then $\mathsf{Verify}(m, PK, \sigma)$ outputs *accept*.

Unforgeability

We define the strongest notion of unforgeability for a digital signature scheme, namely existential unforgeability under a chosen message attack [63]. This is defined via the following game between a challenger C and an adversary E.

Initialization: C runs Setup on security parameter l to generate the public parameters *params*. C also runs KeyGen to obtain a public key PK and a private key SK. C gives the parameters *params* and the public key PK to E.

Sign: E may request a signature on any message m. C computes σ=$\mathsf{Sign}(m, SK)$, and gives σ to E.

Output: E outputs a signature σ^* and a message m^*. E wins the game if $\mathsf{Verify}(m^*, PK, \sigma^*)$ outputs *accept*, and no previous **Sign** query was made on m^*.

Definition 2.14 We say that a digital signature scheme is *existentially unforgeable under an adaptive chosen message attack* if the probability of success of any polynomially bounded adversary in the above game is negligible (as a function of the security parameter l).

We notice that it is possible to obtain a slightly stronger definition of unforgeability by extending the model presented above. In the unforgeability game presented above,

we make the restriction (in the output conditions) that the adversary cannot have made any previous **Sign** query on m^*. In the extended model, we could relax these conditions by making the restriction that the signature σ^* cannot previously have been output by a **Sign** query on m^*. This modification means that an adversary could win the extended unforgeability game by requesting signatures on m^*, and finally outputting a *new* signature on m^* which is different from those output from previous **Sign** queries on m^*. In the extended model, this is considered to be a valid attack on the signature scheme.

If the digital signature scheme is deterministic, then such an attack is impossible, so in this case, the two models of unforgeability are equivalent. However if the digital signature scheme is non-deterministic, then there may be many valid signatures for each message, and such an attack may be possible.

Although the extended model captures these additional attacks and therefore results in a slightly stronger definition of unforgeability, we do not consider such attacks to be useful since the attack does not commit the signer to any message to which he is not already committed. We therefore choose to model the unforgeability of our signature schemes in Chapters 3 to 5 in the model presented above. An additional advantage of using the simpler model of unforgeability is that we are able to directly apply certain useful proof techniques (e.g. the Forking Lemma, which is presented in Section 2.3.4), which in turn enable simpler proofs of security.

2.3.2 Cryptographic Hash Functions

Cryptographic hash functions (hereafter simply referred to as hash functions) play a fundamental role in cryptography. Hash functions create a short "fingerprint" of the input data, and if the data is altered, in general the fingerprint will no longer be valid. In this way, hash functions can be used to provide assurance of data integrity, and therefore play a vital role in the construction of most signature schemes, as well as many other cryptographic schemes and protocols.

Definition 2.15 A hash function H is an efficient algorithm that maps an input x of arbitrary finite bitlength (i.e. $x \in \{0,1\}^*$) to an output $H(x)$ in a finite set D.

Since a hash function is a many-to-one function, the existence of *collisions* (that is, pairs of inputs which map to the same output) is unavoidable. However for cryptographic use, we require that such collisions are computationally difficult to find. The following notions informally describe different levels of security for hash functions [87].

Preimage resistance: For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., given any y for which no corresponding input is known, it is infeasible to find any preimage x such that $H(x) = y$.

2nd preimage resistance: It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, it is infeasible to find a 2nd-preimage $x' \neq x$ such that $H(x') = H(x)$.

Collision resistance: It is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $H(x) = H(x')$. (Note that here there is free choice of both inputs.)

The output set D of H is commonly the set of all strings of some fixed bitlength n, i.e. $H : \{0,1\}^* \rightarrow \{0,1\}^n$, and in this case, H acts as a compression function. However if the output space D has a more complex structure (e.g. an algebraic group), then H may be relatively complex, requiring intermediate mapping functions and deterministic encoding operations. For example, in a particular application H may need to map arbitrary strings to elements of a group G. Suppose G is a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q and let g be a generator of G. Such a hash function may be constructed by using a standard hash function to map the input to a bitstring representing an integer, reducing that integer modulo p and then raising the result to the power $(p-1)/q$ modulo p.

In practice, it is very difficult to design secure hash functions, and recent work has shown that many common hash functions are weaker than previously thought [15, 113, 114, 115, 116].

2.3.3 The Random Oracle Model

Hash functions play an important role in the design of many cryptographic schemes, and therefore also have a major influence over the security of such schemes. Many cryptographic primitives have efficient designs using hash functions, but in general it is very difficult to obtain security arguments (or proofs of security) for such schemes.

If a hash function H is well designed, then it should be infeasible to compute $H(x)$ without evaluating H on x. This should be the case even if many other hash values $H(x_1), H(x_2), ..$ have been computed. Bellare and Rogaway [11] therefore advocated an idealized model for hash functions, which attempts to capture the concept of an ideal hash

function. This model is commonly referred to as the *random oracle model*, and involves modelling hash functions as random functions.

In the random oracle model, hash functions are replaced by *random oracles*, which output truly random values for each new input. Obviously, if the same input is submitted to the random oracle twice, then identical outputs should be obtained. It is argued that proofs of security in this model attest to the security of the overall scheme as long as the hash function used has no weakness.

Proofs of security in the random oracle model are often far easier to construct than proofs in the standard model (i.e. without random oracles), and we find that schemes proven secure in the standard model tend to be less efficient than schemes that employ hash functions and which can be proven secure in the random oracle model. The random oracle model has therefore become a very popular tool in the construction of security proofs for a variety of schemes.

Recent works [7, 60, 81, 90] have demonstrated that for certain schemes, proofs of security in the random oracle model do not translate into security for the actual scheme when the random oracle is instantiated by a hash function. However most of the examples given use hash functions in a completely unnatural way, and it is unclear whether proofs of security in the random oracle model are adequate in general. Despite doubts that have been cast over the use of random oracles, proofs of security in the random oracle model are still widely accepted.

2.3.4 Rewinding Oracles and the Forking Lemma

An example of a proof technique enabled by the random oracle model, we present a result by Pointcheval and Stern [96] known as the *Forking Lemma*. This lemma is a useful tool in constructing proofs of unforgeability for a large class of digital signature schemes.

Informally, the Forking Lemma applies to signature schemes that make use of a hash function, which will be modelled as a random oracle. We assume that adversary E is a polynomial time Turing machine which interacts with a random oracle and the challenger of the unforgeability game of Section 2.3.1, and runs with some random tape which it uses as its source of randomness.

The rationale for the Forking Lemma is that, if E has non-negligible probability η of forging a signature σ in the unforgeability game, then by "rewinding" E and running E again on the same random tape, but with a different random oracle, we can obtain a second (related) forgery with non-negligible probability.

There are two restrictions which apply when using the Forking Lemma. Firstly, the model of unforgeability used should be equivalent to the one presented in Section 2.3.1. Secondly, the digital signature scheme should be of a particular *form*, namely on input a message M, the signature scheme should produce a signature of the form (r_1, h, r_2) where r_1 takes its value randomly from a large set, h is the hash of M and r_1, and r_2 depends only on r_1, M and h. In particular, no value r_1 can appear with probability greater than $2/2^l$ where l is the security parameter. We call such a digital signature scheme a *generic digital signature scheme*.

If the two conditions above are met, then the Forking Lemma is as follows.

Lemma 2.5 (The Forking Lemma [96]): Suppose E is a polynomial time Turing machine with input only public data which produces, in time τ and with probability $\eta \geq 10(\mu_s + 1)(\mu_s + \mu)/2^l$ (where l is a security parameter) a valid generic signature (M, r_1, h, r_2). Here μ is the number of random oracle (hash) queries made by E, and μ_s is the number of signature queries made by E. Suppose also that triples r_1, M, r_2 are simulatable with indistinguishable probability distribution without knowledge of the secret key. Then there exists an algorithm A, which controls E and replaces E's interaction with the signing oracle and random oracle by a simulation, and which produces two valid generic signatures (M, r_1, h, r_2) and (M, r_1, h', r_2') such that $h \neq h'$ in expected time at most $\tau' = 120686\mu_s\tau/\eta$.

Since E may need to be "rewound" multiple times before outputting an appropriate second forgery, we find that the expected time for E to produce two related forgeries is much larger than τ. Details of the proof of the Forking Lemma can be found in [95, 96].

By examining the proof of the Forking Lemma in [96], we notice that the value r_1 of a generic signature scheme does not need to be chosen randomly from a large set, but must be indistinguishable from a value chosen randomly from a large set. For instance, r_1 may depend on some public parameters (or even M) together with some value chosen randomly from a large set. In addition, the computation of r_2 may also depend on values other than r_1, M and h (e.g. r_2 may also depend on public parameters or public keys). It is also not necessary for r_2 to be uniquely determined by r_1, M and h.

Since none of the above modifications to the definition of a generic signature scheme affect the proof of the Forking Lemma, we will include these modifications into the definition of a generic signature scheme and we will use this more general definition of generic signature schemes in the rest of the thesis.

2.3.5 The Non-Generic Forking Lemma

In Chapters 4 and 5, we will need to apply a modified version of the Forking Lemma which we introduce now. Since we will need to apply the forking lemma to signature schemes that do not exactly match the format required, we present a modified version of the forking lemma for signature schemes which take as input an additional parameter, which we denote T. The value T simply acts as a placeholder value, and has no real affect on the proof of the forking lemma, but will be used in the security proofs of certain non-standard digital signature schemes in later chapters.

This modified version of the Forking Lemma applies to digital signature schemes which on input a message M and a value T produce signatures of the form (r_1, h, r_2). Here r_1 randomly distributed in a large set, h is the hash of M and r_1, and r_2 depends on r_1, M and h. As before we require that no value r_1 can appear with probability greater than $2/2^l$ where l is the security parameter. We call such digital signature schemes *non-generic digital signature schemes*.

The unforgeability model for non-generic (NG) signature schemes is identical to the unforgeability model for generic signature schemes (presented in Section 2.3.1) except that **Sign** queries take an additional value T as input. The analogous output conditions require a valid signature for some M and T and forbid an adversary from making previous **Sign** queries on M, T. We call the resulting unforgeability model the NG signature unforgeability model.

As in the more general definition of a generic signature scheme, the value r_1 of an NG signature scheme does not need to be chosen randomly from a large set, but must be indistinguishable from a value chosen randomly from a large set. For instance, r_1 may depend on some public parameters (or even M and T) together with some value chosen randomly from a large set. We also explicitly allow the computation of r_2 in a non-generic signature scheme to depend on values other than r_1, M and h (e.g. r_2 may also depend on public parameters or public keys). We also do not require r_2 to be uniquely determined by r_1, M and h.

Lemma 2.6 (The NG Forking Lemma): Suppose E is a polynomial time Turing machine with input only public data which produces, in time τ and with probability $\eta \geq 10(\mu_s+1)(\mu_s+\mu)/2^l$ (where l is a security parameter) a valid NG signature (M, T, r_1, h, r_2) (in the NG unforgeability model). Here μ is the number of random oracle (hash) queries made by E, and μ_s is the number of signature queries made by E. Suppose also that

tuples r_1, M, T, r_2 are simulatable with indistinguishable probability distribution without knowledge of the secret key. Then there exists an algorithm A, which controls E and replaces E's interaction with the signing oracle and random oracle by a simulation, and which produces two valid NG signatures (M, T, r_1, h, r_2) and (M, T', r_1, h', r_2') such that $h \neq h'$ in expected time at most $\tau' = 120686\mu_s\tau/\eta$.

Proof:

The proof of this lemma follows almost exactly the proof of Lemma 2.5 (Theorem 3 in [96]), which is broken up into two steps. As in [96], we first require the following lemma which deals with adversaries in a no-message attack (Theorem 1 in [96]).

Lemma 2.7 (The No-Message Attack NG Forking Lemma): Suppose E is a polynomial time Turing machine with input only public data, and suppose that E produces, in time τ and with probability $\eta \geq 7\mu/2^l$ (where l is a security parameter) a valid NG signature (M, T, r_1, h, r_2), where μ is the number of random oracle (hash) queries made by E. Then there exists an algorithm A, which controls E and which produces two valid NG signatures (M, T, r_1, h, r_2) and (M, T', r_1, h', r_2') such that $h \neq h'$ in expected time at most $\tau' = 84480\mu\tau/\eta$.

Proof: This lemma is proven in exactly the same way as Theorem 1 in [96] since the changes in the structure of the signature (i.e. the differences between generic and non-generic signatures) have no effect on the proof in [96]. □

As in the proof of Theorems 1 and 3 in [96], we consider an algorithm B such that B executes E and responds to E's random oracle queries. However in addition, in a chosen-message attack, B must also answer E's signature queries on input a message M and a value T. However since tuples r_1, M, T, r_2 are simulatable with indistinguishable probability distribution without knowledge of the secret key, B is able to adequately answer these queries.

The rest of the proof is exactly the same as the proof of Theorem 3 in [96] since once again the structure of the non-generic signatures and **Sign** queries have no effect on the proof, and the risk of random oracle collisions is exactly the same as before if no value r_1 can appear with probability greater that $2/2^l$ where l is the security parameter.

□

We will require the NG version of the Forking Lemma to prove the security of certain non-standard signature schemes in Chapters 4 and 5. As long as the signature scheme can be rewritten in the appropriate form and the model of unforgeability (when rewritten

in the appropriate form) is equivalent to the one presented in Section 2.3.1 (or the NG version of this model), then we may apply the appropriate version of the Forking Lemma.

Chapter 3

Ring Signature Schemes

3.1 Introduction

The notion of *ring signatures* was first formalized by Rivest *et al.* [97]. A ring signature specifies a set (or ring) of possible signers without revealing which member actually produced the signature. Ring signatures require no co-ordination between the ring members: any member can choose any set of possible signers that includes himself and can sign any message using his private key and the public keys of the other ring members.

Rivest *et al.* proposed various applications for ring signatures. A verifier of a particular ring signature should be convinced that a member of the ring created the signature, but cannot determine which of the ring members is the actual signer. Ring signatures therefore provide an elegant and efficient way to leak information or secrets in an anonymous way.

Ring signatures can also be used as a building block to create group signature schemes [38]. In fact, ring signature schemes had previously been generated for this purpose [26, 38], but the distinct notion of ring signatures was only distilled in [97].

Since it is impossible to distinguish which ring member actually produced a particular ring signature, we find that the actual signer (and in fact all the ring members) can deny having produced the signature. Ring signatures therefore do not have the property of non-repudiation.

3.2 Preliminaries

We call a set of possible signers a *ring*. We call the ring member who actually produced a specific signature the *signer* and each of the other ring members are called *non-signers*.

3.3 Ring Signature Definitions

We now define a (1 out of n) ring signature scheme. We assume that each member I of the ring possesses a public and private key pair (PK_I, SK_I) from some digital signature scheme, and that each member is associated with their public key via some PKI or certificate. This public key defines this member's signature scheme and specifies his verification key. We denote the corresponding private key as SK_I.

Definition 3.1 A ring signature scheme is defined via the following algorithms:

RingSign**:** (m, R, sig, SK_{sig}) which produces a ring signature σ for the message m, given the list of public keys $R = \{PK_1, PK_2, .., PK_n\}$ which constitute the ring, and the private key SK_{sig} of the member in position sig in R (who is the signer).

RingVerify**:** (m, R, σ) which takes as input a message m, a list of public keys R, and a signature σ, and outputs either *accept* or *reject*.

We note that in addition to the above algorithms each member I will have a key generation algorithm (and possibly some setup algorithm) to set up the public and private key pair (PK_I, SK_I). However these algorithms are specific to each member and must be run before an entity can be a ring member, so we do not consider them as part of the ring signature definition.

We also note that the size of the ring (n) may vary for each ring signature produced. In other words, the number of public keys taken as input by RingSign may vary each time the algorithm is run.

3.3.1 Related Work

We note that a ring signature scheme is *set-up free*, meaning that the signer does not require any assistance or co-ordination with other ring members in order to generate a ring signature. He only needs to know their public keys. This also means that the public keys of the various ring members need not be of the same type. Such a ring signature scheme is called *separable*.

Some ring signature schemes (e.g. [2, Appendix A]) require that the public keys of all ring members be related. For example, they may all require the same public parameters, and in this case, explicit initialization algorithms would be required. For such schemes, a Setup algorithm would be required to generate the necessary public parameters from

a security parameter, and a KeyGen algorithm would be required to generate public and private keys from the public parameters. The public parameters would also be taken as input to RingSign and RingVerify. Such a ring signature scheme is called **non-separable** and can offer some efficiency advantages over separable ring signature schemes.

The concept of ring signatures has also been extended to the identity-based setting by Zhang and Kim [117] and also by Herranz and Sáez [66]. Bresson *et al.* [25] introduce the concept of threshold ring signature schemes, where instead of a single signer being able to create a ring signature for a ring of size n, a threshold number of signers (say $t \leq n$) must cooperate to create a ring signature.

In [65], Herranz and Sáez formalize the notion of generic ring signature schemes, and provide some techniques for proving the security of such schemes. Informally, they provide a version of the Forking Lemma [96] adapted for ring signatures and use this to analyze the security of certain schemes.

3.4 Security Model

The security of ring signature schemes is defined via the following notions.

3.4.1 Correctness

A ring signature scheme is *correct* if, for any m, σ where σ was output by RingSign on input m, a list of public keys $R = \{PK_1, PK_2, .., PK_n\}$ and secret key SK_{sig} ($sig \in \{1, .., n\}$), then RingVerify(m, R, σ) outputs *accept*.

3.4.2 Anonymity

Anonymity ensures that it is impossible to distinguish which member of a ring has generated a given signature, and is defined via the following game between a challenger C and an adversary E.

Initialization: C firstly generates a set $S = \{(PK_i, SK_i)\}_{i=1}^{n(l)}$ of public and private keys using key generation algorithms of its choosing, where n is a polynomial function of the security parameter l, and gives this set to E. The set of public keys is set to be $\mathcal{R} = \{PK_i\}_{i=1}^{n(l)}$.

Challenge: E finally produces a message m, and a ring $R \subseteq \mathcal{R}$. C chooses some PK_{sig} at random from R, computes σ=RingSign(m, R, sig, SK_{sig}), and gives σ to E.

Output: E outputs some $PK_j \in R$. E wins the game if $PK_{sig} = PK_j$.

Definition 3.2 We say that a ring signature scheme is *anonymous* if no polynomially bounded adversary has advantage non-negligibly greater than $1/|R|$ of winning in the above game. We say that a ring signature scheme is *perfectly anonymous* if all adversaries have probability exactly $1/|R|$ of winning the above game.

3.4.3 Unforgeability

Unforgeability ensures that no-one other than a ring member can create a valid ring signature for that ring. Unforgeability is defined via the following game between a challenger C and an adversary E.

Initialization: C firstly generates a set $S = \{(PK_i, SK_i)\}_{i=1}^{n(l)}$ of public and private keys using key generation algorithms of its choosing, where n is a polynomial function of the security parameter l, and the list of public keys $\mathcal{R} = \{PK_i\}_{i=1}^{n(l)}$ is given to E.

RingSign Queries: E may request a signature on any message m, for any ring $R \subseteq \mathcal{R}$, with any public key $PK_{sig} \in R$. C computes σ=$\mathsf{RingSign}(m, R, sig, SK_{sig})$, and gives σ to E.

Corrupt Queries: E may request the private key SK_i corresponding to any public key PK_i.

Output: E finally outputs a message m^*, a ring $R^* \subseteq \mathcal{R}$, and a signature σ^*. E wins the game if $\mathsf{RingVerify}(m^*, R^*, \sigma^*)$ outputs *accept*, no public key $PK \in R^*$ has been corrupted, and no previous **RingSign** query was made on m^* and R^*.

Definition 3.3 We say that a ring signature scheme is *unforgeable* if the probability of success of any polynomially bounded adversary in the above game is negligible.

3.4.4 Notes on the Security Definitions for Ring Signatures

Anonymity: This definition of anonymity is very strong since the adversary knows the private keys of all members of the ring. We present this definition since it appears to be the strongest definition of anonymity, and the concrete ring signature scheme that we present later in this chapter is able to meet this definition.

Many papers in the literature do not rigorously define what they mean by anonymity, and those that do often propose weaker notions of anonymity. Bender *et al.* [14]

give a good overview of different definitions although they do not consider the definition above. Weaker definitions restrict the adversary's access to the private keys of members of the ring R.

Unforgeability: Our definition of unforgeability is also stronger than usual. In particular, we allow the adversary to make **Corrupt** queries on all public keys except those involved in the forged ring signature. We also allow the adversary to stipulate which private key it wishes to be used when making a **RingSign** query even though, if the ring signature scheme is anonymous, it should make no difference to the adversary which private key is used to generate the ring signature.

We include these additional adversarial resources in our model of unforgeability in order to make our security definitions as strong as possible, and also to ensure that our models of security for ring signatures are compatible with models of security for other cryptographic schemes presented in later chapters.

3.5 A Concrete Scheme

Since we only require a non-separable ring signature scheme in later chapters, we present the non-separable ring signature scheme of [2, Appendix A] which is defined by the following algorithms.

Setup: For some security parameter l, let p and q be large primes, where $q|(p-1)$. Let G be a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q and let g be a generator of G. We also assume that $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a cryptographic hash function. The public parameters are $params = \langle p, q, g, H \rangle$.

KeyGen: This algorithm takes as input public parameters $params = \langle p, q, g, H \rangle$. A private key x is chosen at random from $\mathbb{Z}_q^*$, and the corresponding public key is computed as $X = g^x \bmod p$.

RingSign: Given a message m, a list of public keys $R = \{X_1, X_2, .., X_n\}$, and a private key x_{sig} which corresponds to a public key X_{sig} in position sig of R, select random $t, h_i \in \mathbb{Z}_q$ for $i = 1, .., n$, $i \neq sig$, and compute

$$\begin{aligned} z &= g^t \Pi_{i=1, i \neq sig}^{n} X_i^{h_i} \bmod p \\ h &= H(X_1, .., X_n, m, z) \\ h_{sig} &= h - (h_1 + .. + h_{sig-1} + h_{sig+1} + .. + h_n) \bmod q \\ s &= t - x_{sig} \cdot h_{sig} \bmod q \end{aligned}$$

The signature is $\sigma = \langle s, h_1, .., h_n \rangle$.

RingVerify: Given a message m, a list of public keys $R = \{X_1, X_2, .., X_n\}$, and a signature $\sigma = \langle s, h_1, .., h_n \rangle$, compute

$$\begin{aligned} h &= H(X_1, .., X_n, m, g^s X_1^{h_1} \cdots X_n^{h_n} \bmod p) \\ h' &= \textstyle\sum_{i=1}^{n} h_i \bmod q. \end{aligned}$$

If $h = h'$ then output *accept*, otherwise output *reject*.

We notice that the 1-party version of the ring signature scheme above is in fact equivalent to the Schnorr signature scheme [99]. The Schnorr signature scheme was proven to be unforgeable in [96] (using the Forking Lemma presented in Section 2.3.4), and we will show that the unforgeability of this ring signature scheme in fact relies on the unforgeability of the Schnorr signature scheme.

3.6 Security of the Concrete Scheme

Proofs of security for the above scheme are omitted in [2]. However work in later chapters relies on the security of this scheme, so we now present security results for the above ring signature scheme.

Theorem 3.1 The ring signature scheme presented in Section 3.5 is *perfectly anonymous* assuming that H is a random oracle.

Proof: Consider any ring signature $\sigma = \langle s, h_1, .., h_n \rangle$ produced by RingSign on input a message m, a list of public keys $R = \{X_1, X_2, .., X_n\}$, and a private key x_{sig} which corresponds to a public key X_{sig} in position sig of R.

Each h_i for $i \neq sig$ is selected randomly from $\mathbb{Z}_q$, so these values are distributed uniformly at random in $\mathbb{Z}_q$. Now h_{sig} depends on an output from H, but since H is a random oracle, the outputs of H are distributed uniformly at random over G, and therefore h_{sig} is uniformly distributed over $\mathbb{Z}_q$. Since s depends on randomly selected t, s also has uniform distribution on $\mathbb{Z}_q$.

The h_i values and s are not independent since for each valid signature the following equation holds:

$$\sum_{i=1}^{n} h_i \bmod q = H(X_1, .., X_n, m, g^s X_1^{h_1} \cdots X_n^{h_n} \bmod p)$$

However this equation holds for all valid signatures and the inputs all have equal distributions, so any two valid signatures generated by RingSign on R and m are indistinguishable, irrespective of which private key was used in each case. Therefore no adversary has advantage greater than $\frac{1}{|R|}$ of winning the anonymity game of Section 3.4.2. □

We note that if we assume that the outputs of H are only computationally indistinguishable from random, then our concrete ring signature is *anonymous* (but not perfectly anonymous).

Theorem 3.2 The ring signature scheme presented in Section 3.5 is *unforgeable*, assuming the unforgeability of the Schnorr signature scheme [99].

Proof:

Suppose that H is a random oracle and there exists an algorithm E that makes at most μ queries to the random oracle H, at most μ_s **RingSign** queries, and wins the unforgeability game of Section 3.4.3 in time at most τ with non-negligible probability η in the security parameter l. We now show that there exists an algorithm B that uses E, and which has a non-negligible chance of forging a Schnorr signature.

Unforgeability for a Schnorr signature scheme is defined in the same way as unforgeability for a standard digital signature scheme in Section 2.3.1. Since B attempts to forge a Schnorr signature, we assume that B in turn interacts with a challenger C in the unforgeability game of Section 2.3.1. Since the Schnorr signature scheme is essentially the same as the 1-party version of our ring signature scheme, the algorithms Setup and KeyGen for the two schemes are identical. C therefore initializes the unforgeability game in the same way as a challenger in the unforgeability game for ring signatures, except that C only generates a single public and private key pair $\langle X, x\rangle$. C gives B the public key X and the public parameters $\langle p, q, g\rangle$, and access to the random oracle H.

C will also answer B's **Sign** queries on any message M, and will respond with a Schnorr signature $\sigma' = \langle s', h'\rangle$ where $h' = H(M, g^{s'}X^{h'})$. We note that the message M may be of any form and of any length.

B in turn initializes an unforgeability game (for ring signatures) for E. B gives the public parameters $params = \langle p, q, g\rangle$ to E. B also sets the number of participants to be $n(l)$, where n is a polynomial function of the security parameter l, and picks a random $j \in \{1, .., n(l)\}$. B generates a set $S = \{(X_i, x_i)\}_{i=1, i\neq j}^{n(l)}$ of public and private keys using KeyGen. B sets public key $X_j = X$, and the list of public keys $\mathcal{R} = \{X_i\}_{i=1}^{n(l)}$ is given to E. B now simulates a challenger for E by simulating all the queries which E can make as follows:

***H*-Queries:** B passes all of E's random oracle queries to C and returns C's response to E.

RingSign Queries: E may request a signature on any message m, for any ring R where $R = \{X_1, .., X_k\} \subseteq \mathcal{R}$, and with any public key $X_{sig} \in R$. If $X_j \notin R$ then B computes σ=RingSign(m, R, sig, x_{sig}) and outputs σ. The running of RingSign requires a single call to the random oracle H.

If $X_j \in R$ then B sets message $M = X_1, .., X_k, m$, makes a **Sign** query on M to C, and receives a response $\sigma' = \langle s', h' \rangle$. For each $X_i \in R$ where $X_i \neq X_j$, B picks a random value $h_i \in Z_q$. B computes $h_j = h' - \sum_{i \neq j} h_i \bmod q$ and $s = s' - \sum_{i \neq j} x_i h_i \bmod q$. B sets $\sigma = \langle s, h_1, .., h_k \rangle$ and outputs σ.

Corrupt Queries: E can make a Corrupt query on any public key X_i. If $X_i = X_j$, then B aborts and terminates E. Otherwise B returns the appropriate private key x_i.

Output: On termination, with non-negligible probability E outputs a signature $\sigma^* = \langle s^*, h_1^*, .., h_k^* \rangle$, a ring $R^* = \{X_1^*, .., X_k^*\} \subseteq \mathcal{R}$ and a message m^*, where RingVerify(m^*, R^*, σ^*) outputs *accept*, and no previous **RingSign** query was made on m^* and R^*.

If $X_j \notin R^*$ then B aborts. Otherwise we assume that $X_j \in R^*$, which occurs with probability $|R^*|/n$ (and in this case, B would not have had to abort on any **Corrupt** query since E cannot have corrupted any public key in R^*).

B computes $s'' = s^* + \sum_{i \neq j} x_i^* h_i^* \bmod q$ and sets $h'' = h_j^*$. B outputs $\sigma'' = \langle s'', h'' \rangle$ as a forgery to C on message $M = X_1^*, .., X_k^*, m^*$. Since σ^* is a valid ring signature, σ'' is a valid Schnorr signature, and since E made no previous **RingSign** query on m^* and R^*, B made no previous **Sign** query on M.

Therefore B has forged a Schnorr signature in time τ and with non-negligible probability $\eta' = \eta |R^*|/n$, making at most μ_s **Sign** queries and at most $\mu + \mu_s$ random oracle queries.

This contradicts the unforgeability of the Schnorr signature scheme which was proven in [96]. □

Chapter 4

Non-interactive Designated Verifier Proofs and Their Applications

4.1 Introduction

Undeniable signatures were first presented in 1989 by Chaum and van Antwerpen [37], and were designed to have the property that signatures could be freely distributed, but were not self-authenticating. More precisely, undeniable signatures should not be verifiable without the cooperation of the signer. However, any party wrongly accused of having produced the signature can deny his involvement. The true signer may prove his authorship of an undeniable signature by running a confirmation protocol with a verifier, and a falsely implicated signer may deny his involvement by running a denial protocol with a verifier.

The confirmation (and possibly also the denial) proofs must be *non-transferable*, meaning that they must be convincing only to the intended recipient. A verifier should not be able to pass on (or transfer) the proof to other parties, since otherwise the signature can be verified without the cooperation of the signer. Obviously, only the true signer should be able to successfully complete a confirmation protocol. Moreover the true signer should be unable to successfully complete a denial protocol for any of his signatures. Therefore the true signer cannot deny having produced his signatures.

The confirmation and denial protocols for the undeniable signature scheme of [37] were made zero-knowledge in [35], and this goes some way to ensuring that the signer has control over who can verify an undeniable signature. However, as was pointed out in [49], even though the confirmation protocol is zero-knowledge, the signer may still not always be able to control who is able to verify the validity of a signature if a group of verifiers

cooperate. The undeniable signature scheme in [37] is also vulnerable to a blackmailing attack [68]. Jakobsson *et al.* [70] provided a solution to such attacks, called *designation of verifiers*, to ensure that only a specified verifier can confirm an undeniable signature.

Informally, a designated verifier (DV) proof is a proof of correctness of some "statement" that either the prover or some designated verifier could have produced. If the prover created the proof, then the "statement" is correct. However a designated verifier can simulate a valid proof without a correct "statement". Therefore since a third party cannot determine whether a given DV proof was generated by the prover or by the designated verifier, they cannot determine whether the corresponding "statement" is correct or not.

A DV proof should convince the designated verifier of the correctness of the "statement" since the designated verifier knows that he did not create the proof himself, and therefore the proof must have been generated by the prover. However no other party will be convinced of the validity of the proof since the designated verifier could have simulated it. In the context of undeniable signatures, a DV proof can be used to convince (only) the designated verifier of the validity of the undeniable signature. We refer to undeniable signature schemes that use DV proofs as their confirmation and denial proofs as DV undeniable signatures.

Although the authors of [70] did not give a formal definition of DV proofs, they provided concrete examples of such proofs. The construction of DV proofs in [70] used trapdoor commitment schemes [24]. It was also shown there that the DV proofs could be made non-interactive. Such proofs are called NIDV proofs and can in turn be used to construct NIDV undeniable signatures. However a flaw in the concrete NIDV proof of [70] was recently discovered by Wang [112], whereby a cheating signer can create a "non-standard" undeniable signature which the signer can prove valid via the NIDV confirmation proof and later deny via the denial proof. Wang proposed two ways to repair the scheme of [70], but did not offer any proofs of security.

For normal undeniable signature schemes, unforgeability and invisibility (or anonymity) are usually considered to be the key notions of security. As for the security of confirmation and denial proofs, the literature suggests that most authors have been content to simply prove that they are zero-knowledge and sound. This may suffice for the case where zero-knowledge confirmation and denial proofs are used, but it is unclear whether these notions of security are satisfactory for NIDV confirmation and denial proofs.

A concept related to DV proofs is what is commonly referred to as DV signatures [75, 77, 98]. DV signature schemes and DV proofs are commonly confused in the liter-

ature, although they are quite distinct notions. Informally, a DV signature can be seen as an NIDV proof, but instead of proving the validity of some statement (e.g. that a certain undeniable signature is valid), it is simply associated with some message. DV signatures retain the property of non-transferability (that we find with DV proofs) since a DV signature could have been created by the signer or by the designated verifier.

Some authors have proposed formal definitions [75, 77] and security notions [77] for DV signatures. It has also been noted that in fact secure two-party ring signatures satisfy the definition of DV signatures and that the security requirements for DV signatures and ring signatures are essentially the same [75, 98].

In Section 4.2 we present a formal definition for NIDV proof systems which we believe are of independent interest. Our formal definition is compatible with the concrete scheme of [70]. In Section 4.3 we propose a formal model of security for NIDV proof systems.

In Sections 4.4 and 4.5 we present a formal definition for NIDV undeniable signature schemes as well as a security model which models the security of both the core signature scheme and the NIDV confirmation and denial proof systems with which it is composed. Essentially, two NIDV proof systems are required to construct an NIDV undeniable signature scheme. The NIDV proof systems are for complementary languages, one providing confirmation proofs and the other denial proofs for the core signature scheme.

We go on to repair the NIDV proofs of [70], producing secure NIDV proof systems suited to combination with the full domain hash variant of Chaum's undeniable signature scheme [35, 37]. The NIDV proofs we obtain are actually a little shorter than those in [70]. Our work confirms that one of the fixes proposed by Wang [112] does indeed repair the NIDV proofs of [70]. The result is a concrete and efficient NIDV undeniable signature scheme.

For completeness, in Sections 4.8 and 4.9 we provide a formal definition of DV signatures as well as a security model for DV signatures which closely follows our model of security for ring signatures in Section 3.4.

We also discuss how DV signature schemes can be constructed from both 2-party ring signature schemes and NIDV proofs, and we give a concrete example demonstrating how one of our concrete NIDV proofs may be modified to create a DV signature scheme. The chapter concludes with some open problems and ideas for further extensions of our work.

Our work in this chapter represents the first time that a formal security model for NIDV undeniable signatures has been developed. The model does not require the signature scheme to be randomized. It is also a multi-party model, reflecting the fact that designated

verifier proofs naturally involve more than one party (and therefore involve more than one secret keys), and that a party may play different roles at different times. For example, an entity A may generate NIDV proofs for some designated verifier B, and at the same time be a recipient of an NIDV proof (i.e. play the role of designated verifier) from B. It is therefore important to model such interaction and interchanging of roles in the security model.

We consider NIDV proofs and NIDV undeniable signatures separately. One reason for doing so is that, in any application, undeniable signatures may exist independently of proofs. For example, a prover may generate a signature as a commitment to a message but only later provide an NIDV proof of its correctness, or a prover may generate many proofs for different designated verifiers on the same signature. A second reason is that NIDV proofs may also be useful in contexts other than undeniable signatures. One possible application of NIDV proofs is to deniable proofs of knowledge, in particular, proofs of knowledge of a private key. For example, when registering a public key with certification authority C, A could demonstrate knowledge of the appropriate private key by presenting C with an NIDV proof of knowledge of the private key of A.

4.1.1 Related Work and Notions

As we mentioned before, there is some confusion in the literature between DV proofs, NIDV proofs and DV signatures, and these terms are often used interchangeably. In much of the literature on DV signature schemes [75, 77, 98], the authors ascribe the term DV signatures to [70] and describe the concrete NIDV undeniable signature scheme of [70] as a DV signature scheme. In fact, this terminology was *never* used in [70]. Despite this incorrect association, the works of [75, 77, 98] are in fact concerned with DV signatures.

Although most authors do acknowledge that unforgeability and non-transferability are necessary properties for a secure DV signature scheme, most authors do not formalize these notions. Lipmaa *et al.* [77] are the first to examine the security properties of DV signature schemes more formally. In their work, they formalize and extend the attack of Wang [112] on [70] to DV signatures. They then propose two additional security properties which are required for secure DV signatures, namely non-delegatability and non-disavowability. In Section 4.9.4 we contrast our model of security with theirs and discuss how the notions of non-delegatability and non-disavowability relate to our security definitions.

Another related notion is that of strong designated verifier (SDV) proofs and signatures [70, 98, 109], which provide stronger security guarantees than DV signatures and NIDV

proofs. SDV signatures provide similar properties to DV signatures except that only the designated verifier is able to verify the signatures produced, since the verification algorithm makes use of the private key of the designated verifier. By contrast, DV signatures (and NIDV proofs) are universally verifiable, but only convincing to the designated verifier.

Another notion is that of universal designated verifier (UDV) signatures [75, 104, 105]. Although at first glance these appear to be similar to NIDV undeniable signatures, they are quite different. UDV signature schemes produce signatures which are universally verifiable. However any party in possession of a valid UDV signature on message M from signer S can provide (to any verifier of their choice) a designated verifier proof that they possess a valid UDV signature on M by S. In comparison, NIDV undeniable signatures are not universally verifiable, and only the signer is able to produce designated verifier proofs of a signature's validity.

4.2 NIDV Proofs

We now present a formal definition for NIDV proof systems. This formal definition was previously lacking in [70] but our definitions are compatible with the concrete scheme of [70].

Unless explicitly stated, we will denote the public key of a participant I by X_I, and the private key of I by x_I. P will in general represent a prover, and V a verifier.

A non-interactive designated verifier (NIDV) proof system is defined with respect to some family of languages $\mathcal{L}$. The goal of an NIDV proof system is to prove the membership of elements e in a language $L \in \mathcal{L}$. An NIDV proof system consists of the following algorithms:

$\mathsf{Setup}(l)$**:** A probabilistic algorithm which takes a security parameter l as input and returns the system parameters *params* and a description of a family of languages $\mathcal{L}$. Amongst the public parameters *params* are descriptions of the following spaces: a public key space $\mathcal{PK}$, a private key space $\mathcal{SK}$, an element space $\mathcal{E}$ and a proof space $\mathcal{P}$. $\mathcal{E}$ contains all elements $e \in L$ for all languages $L \in \mathcal{L}$.

$\mathsf{KeyGen}(params)$**:** A probabilistic algorithm which takes as input the public parameters *params* and returns a key pair (x, X) where $x \in \mathcal{SK}$ is a private key and $X \in \mathcal{PK}$ is the corresponding public key.

$\mathsf{PGen}(X_P, X_V, x_P, e)$**:** A (possibly probabilistic) proof generation algorithm which takes

as input $X_P, X_V \in \mathcal{PK}$, $X_P \neq X_V$, $x_P \in \mathcal{SK}$, and $e \in \mathcal{E}$, and produces an NIDV proof $\pi \in \mathcal{P}$ for e.

PVerify(X_P, X_V, e, π): A verification algorithm which takes as input $X_P, X_V \in \mathcal{PK}$, $X_P \neq X_V$, $e \in \mathcal{E}$ and $\pi \in \mathcal{P}$, and outputs *accept* or *reject*.

Note that we parameterize the languages $L \in \mathcal{L}$ by public keys, and for any public key $X \in \mathcal{PK}$, $L(X) \subseteq \mathcal{E}$. This parametrization will be needed for the proof systems required for use with undeniable signatures, but is not necessary in general.

4.3 Security for NIDV Proofs

We say that an NIDV proof system is secure if it satisfies the notions of correctness, non-transferability and soundness. These are defined as follows.

4.3.1 Correctness

An NIDV proof system is *correct* if when PGen is run on any input $\langle X_P, X_V, x_P, e\rangle$ where $X_P, X_V \in \mathcal{PK}, X_P \neq X_V$, $x_P \in \mathcal{SK}, e \in L(X_P)$, and outputs some $\pi \in \mathcal{P}$, then PVerify on input $\langle X_P, X_V, e, \pi\rangle$ outputs *accept*.

4.3.2 Non-transferability

We say that an NIDV proof system is *non-transferable* if there exists a polynomial time algorithm FakePGen that on input a tuple $\langle X_P, X_V, x_V, e'\rangle$, where $X_P, X_V \in \mathcal{PK}, X_P \neq X_V, x_V \in \mathcal{SK}, e' \in \mathcal{E}$, but where e' is not necessarily in $L(X_P)$, produces a proof $\pi' \in \mathcal{P}$ such that $\langle X_P, X_V, e', \pi'\rangle$ is accepted by PVerify. In addition, if π is produced by PGen when run on inputs $\langle X_P, X_V, x_P, e\rangle$ where $e \in L(X_P)$, then if e and e' are indistinguishable, then the distributions of proofs π' and π must be polynomially indistinguishable.

4.3.3 Soundness

Soundness of an NIDV proof system is defined via the following game between a challenger C and an adversary E:

Initialize: C firstly runs Setup for a given security parameter l to obtain the public parameters *params*, and a description of a family of languages $\mathcal{L}$. C runs KeyGen to generate the public and private keys X_I and x_I for each participant, where the number of participants is bounded by n, where n is a polynomial function of l. We

define the set of all participants' public keys to be $\mathcal{X}$. E is given $params, \mathcal{L}$ and $\mathcal{X}$ while C retains the private keys.

E can make the following types of query to the challenger C:

EGen Queries: E can request an element of a particular language (or its complement). On input a public key X_I and a bit b (and possibly some *seed*), if $b = 1$ then C outputs an element $e \in L(X_I)$, otherwise C outputs an element $e \notin L(X_I)$.

PGen Queries: E can request an NIDV proof for input $\langle X_P, X_V, e \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$ and $e \in \mathcal{E}$. C runs $\mathsf{PGen}(X_P, X_V, x_P, e)$ to produce an NIDV proof $\pi \in \mathcal{P}$. If $\mathsf{PVerify}(X_P, X_V, e, \pi)$ outputs *accept* then C outputs π. Otherwise C outputs *invalid*.

FakePGen Queries: E can request a fake NIDV proof on input $\langle X_P, X_V, e' \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, and $e' \in \mathcal{E}$. C runs $\mathsf{FakePGen}(X_P, X_V, x_V, e')$ to produce an NIDV proof $\pi' \in \mathcal{P}$. C outputs π'.

Corrupt Queries: E can request the private key corresponding to any public key $X_I \in \mathcal{X}$. C outputs the corresponding private key x_I.

Output: Finally E outputs $\langle X_P^*, X_V^*, e^*, \pi^* \rangle$, where $X_P^*, X_V^* \in \mathcal{X}$, $X_P^* \neq X_V^*$, X_V^* is uncorrupted, $e^* \in \mathcal{E}$ and $\pi^* \in \mathcal{P}$. E wins if $\langle X_P^*, X_V^*, e^*, \pi^* \rangle$ is accepted by $\mathsf{PVerify}$, no **FakePGen** query was made on $\langle X_P^*, X_V^*, e^* \rangle$, and either:

1. X_P^* is uncorrupted and no **PGen** query was made on $\langle X_P^*, X_V^*, e^* \rangle$, or
2. $e^* \notin L(X_P^*)$.

Definition 4.1 We say that an NIDV proof system is *sound* if the probability of success of any polynomially bounded adversary in the above game is negligible (as a function of the security parameter l).

4.3.4 Notes on the Security Definitions for NIDV Proof Systems

Non-transferability: The existence of $\mathsf{FakePGen}$ that can be run by V to create an NIDV proof for any element e (not necessarily in L) ensures that no-one besides V will be convinced by an NIDV proof for e.

We note that we do not require the elements e and e' to be indistinguishable for non-transferability; rather only the proofs π and π' are required to be indistinguishable.

If an outside party can already distinguish elements in $L(X_P)$ from elements not in $L(X_P)$, then they have no need of NIDV proofs for $L(X_P)$. However an NIDV proof for an element e should not give any extra information regarding e to parties other than the designated verifier.

Soundness: The soundness definition guarantees that if an uncorrupted verifier receives a valid NIDV proof, then it was created using the private key x_P and $e \in L(X_P)$. So a prover cannot cheat. Soundness also guarantees that no-one other than P can convince an uncorrupted designated verifier that $e \in L(X_P)$. In the context of undeniable signatures, this means that a verifier must cooperate with the real signer in order to verify an undeniable signature since no-one other than the real signer is able to produce an NIDV proof of a valid undeniable signature that will be accepted by an uncorrupted designated verifier.

Although not immediately obvious, the soundness definition also guarantees that if PGen is run on $\langle X_P, X_V, x_P, e\rangle$ where $e \notin L(X_P)$, and outputs some $\pi \in \mathcal{P}$, then PVerify on input $\langle X_P, X_V, e, \pi\rangle$ outputs *reject*. If this is not the case, then E could generate an element $e \notin L(X_P)$, generate a proof π using PGen which is accepted by PVerify, and output π. E would win the game since $e \notin L(X_P)$. This observation, together with the correctness property, means that an entity with public key X_P can determine whether an element e is in $L(X_P)$ or not by running PGen on $\langle X_P, X_V, x_P, e\rangle$ for some X_V and then PVerify. PVerify outputs *accept* if and only if $e \in L(X_P)$. This is in fact used when answering **PGen** queries.

We note that **EGen** queries allow E to stipulate some "seed" to be used. This is necessary in the context of undeniable signatures where **EGen** queries correspond to signature queries, and the message to be signed corresponds to the seed.

The existence of FakePGen from Section 4.3.2 also enables C to answer **FakePGen** queries. We consider it important to model such queries since an adversary may have access to such "fake" NIDV proofs that are produced by dishonest verifiers using FakePGen.

The model for soundness is multiparty, reflecting the fact that NIDV proofs naturally involve more than one party, and that a party may play different roles at different times.

4.4 NIDV Undeniable Signatures

As mentioned earlier, the main application of NIDV proofs has historically been in undeniable signatures, even though the current security models for undeniable signatures (in particular the definitions of soundness) do not to support NIDV proofs. We now present a formal definition for NIDV undeniable signature schemes.

Definition 4.2 An NIDV undeniable signature scheme consists of a core signature scheme as well as NIDV confirmation and denial proof systems. The core signature scheme consists of the following algorithms:

$\mathsf{Setup}(l)$**:** A probabilistic algorithm which takes a security parameter l as input and returns the system parameters *params*. Amongst the public parameters are descriptions of the following spaces: a public key space $\mathcal{PK}$, a private key space $\mathcal{SK}$, a message space $\mathcal{M}$ and a signature space $\mathcal{S}$.

$\mathsf{KeyGen}(params)$**:** A probabilistic algorithm which takes as input the public parameters *params* and returns a key pair (x, X) where $x \in \mathcal{SK}$ and $X \in \mathcal{PK}$.

$\mathsf{USign}(x, m)$**:** A (possibly probabilistic) signature generation algorithm which on input $x \in \mathcal{SK}, m \in \mathcal{M}$, produces an undeniable signature $\sigma \in \mathcal{S}$.

The core signature scheme defines a language $L(X)$ for each public key X, where $L(X) = \{(m, \sigma) : \sigma = \mathsf{USign}(x, m)\}$. In other words, $L(X)$ is the language of all possible valid message and signature pairs for public key X and $\overline{L(X)}$ is the language of all invalid message and signature pairs for public key X. The family of languages $\mathcal{L}$ is defined as $\mathcal{L} = \{L(X) : X \in \mathcal{PK}\}$ and $\overline{\mathcal{L}}$ is defined as $\overline{\mathcal{L}} = \{\overline{L(X)} : X \in \mathcal{PK}\}$.

The families of languages $\mathcal{L}$ and $\overline{\mathcal{L}}$ parameterize the confirmation and denial proofs. The confirmation proof is an NIDV proof system $\mathcal{C}$ for $\mathcal{L}$, and the denial proof is an NIDV proof system $\mathcal{D}$ for $\overline{\mathcal{L}}$. The setup algorithms for $\mathcal{C}$ and $\mathcal{D}$ use the public key space $\mathcal{PK}$, the private key space $\mathcal{SK}$, and set the element space $\mathcal{E}$ to be $\mathcal{M} \times \mathcal{S}$. The proof spaces for $\mathcal{C}$ and $\mathcal{D}$ are denoted $\mathcal{P}_C$ and $\mathcal{P}_D$ respectively. The following algorithms then make up the confirmation and denial proofs.

$\mathsf{ConfGen}(X_P, X_V, x_P, m, \sigma)$**:** A confirmation proof generation algorithm which, on input $X_P, X_V \in \mathcal{PK}$, $X_P \neq X_V$, $x_P \in \mathcal{SK}$, $(m, \sigma) \in \mathcal{E}$ runs PGen of $\mathcal{C}$ on $\langle X_P, X_V, x_P, (m, \sigma)\rangle$.

ConfVerify$(X_P, X_V, m, \sigma, \pi_C)$**:** A confirmation proof verification algorithm which, on input $X_P, X_V \in \mathcal{PK}$, $X_P \neq X_V$, $(m, \sigma) \in \mathcal{E}$ and $\pi_C \in \mathcal{P}_C$ runs PVerify of $\mathcal{C}$ on $\langle X_P, X_V, (m, \sigma), \pi_C \rangle$.

DenyGen$(X_P, X_V, x_P, m, \sigma)$**:** A denial proof generation algorithm which, on input $X_P, X_V \in \mathcal{PK}$, $X_P \neq X_V$, $x_P \in \mathcal{SK}$, $(m, \sigma) \in \mathcal{E}$ runs PGen of $\mathcal{D}$ on $\langle X_P, X_V, x_P, (m, \sigma) \rangle$.

DenyVerify$(X_P, X_V, m, \sigma, \pi_D)$**:** A denial proof verification algorithm which, on input X_P, $X_V \in \mathcal{PK}$, $X_P \neq X_V$, $(m, \sigma) \in \mathcal{E}$ and $\pi_D \in \mathcal{P}_C$ runs PVerify of $\mathcal{D}$ on $\langle X_P, X_V, (m, \sigma), \pi_D \rangle$.

4.5 Security for NIDV Undeniable Signatures

In analyzing the security of NIDV undeniable signatures, we consider the security of the confirmation and denial proofs being used, and their composition with the core signature scheme.

Definition 4.3 The confirmation (denial) proof of an NIDV undeniable signature is *secure* if $\mathcal{C}$ (respectively $\mathcal{D}$) is a secure NIDV proof system for $\mathcal{L}$ (respectively $\overline{\mathcal{L}}$). That is, $\mathcal{C}$ (respectively $\mathcal{D}$) is correct, non-transferable and sound.

4.5.1 The Security of the Core Signature Scheme

The security of the core signature scheme is defined via the notions of unforgeability and invisibility.

4.5.1.1 Unforgeability

Unforgeability of an NIDV undeniable signature scheme is defined via the following game between a challenger C and an adversary E:

Initialize: C firstly runs Setup for a given security parameter l to obtain the public parameters *params*. C runs KeyGen to generate the public and private keys X_I and x_I for each participant, where the number of participants is bounded by n, where n is a polynomial function of l. We define the set of all participants' public keys to be $\mathcal{X}$. E is given *params* and $\mathcal{X}$ while C retains the private keys.

E can make the following queries to the challenger C:

USign Queries: E can request an undeniable signature for input $\langle X_I, m\rangle$ where $X_I \in \mathcal{X}$, and $m \in \mathcal{M}$. C takes the private key x_I corresponding to X_I and runs $\mathsf{USign}(x_I, m)$ to produce $\sigma \in \mathcal{S}$, where x_I is the private key corresponding to X_I. C outputs σ.

Conf/Deny Queries: E can request a confirmation or denial proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. C takes the private key x_P corresponding to X_P and proceeds as follows. C runs $\mathsf{ConfGen}(X_P, X_V, x_P, m, \sigma)$ to produce an NIDV proof $\pi_C \in \mathcal{P}_C$. If $\mathsf{ConfVerify}(X_P, X_V, m, \sigma, \pi_C)$ returns *accept*, then C outputs π_C. Otherwise C runs $\mathsf{DenyGen}(X_P, X_V, x_P, m, \sigma)$ to produce an NIDV proof $\pi_D \in \mathcal{P}_D$ which it outputs. C informs E which case has occurred.

FakeConf Queries: E can request a fake confirmation proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. C takes the private key x_V corresponding to X_V and runs $\mathsf{FakePGen}$ of the NIDV proof system $\mathcal{C}$ on $\langle X_P, X_V, x_V, (m, \sigma)\rangle$ to produce an NIDV proof $\pi_C \in \mathcal{P}_C$, which C outputs.

FakeDeny Queries: E can request a fake denial proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. C takes the private key x_V corresponding to X_V and runs $\mathsf{FakePGen}$ of the NIDV proof system $\mathcal{D}$ on $\langle X_P, X_V, x_V, (m, \sigma)\rangle$ to produce an NIDV proof $\pi_D \in \mathcal{P}_D$, which C outputs.

Corrupt Queries: E can request the private key corresponding to any public key $X_I \in \mathcal{X}$. C outputs the corresponding private key x_I.

Output: Finally E produces $X_I^* \in \mathcal{X}$, $m^* \in \mathcal{M}$ and $\sigma^* \in \mathcal{S}$, where X_I^* is uncorrupted and no **USign** query was previously made on $\langle X_I^*, m^*\rangle$. E wins the game if $(m^*, \sigma^*) \in L(X_I^*)$.

Definition 4.4 We say that an NIDV undeniable signature scheme is *unforgeable* if the probability of success of any polynomially bounded adversary in the above game is negligible in l.

4.5.1.2 Invisibility

Invisibility of an NIDV undeniable signature scheme is defined via the following game between a challenger C and an adversary E:

Initialize: This is as in the Unforgeability game above.

Phase 1: The adversary can make **USign**, **Conf/Deny**, **FakeConf**, **FakeDeny** and **Corrupt** queries, and these are all answered as in the Unforgeability game.

Challenge: E produces $m^* \in \mathcal{M}$, $X_I^* \in \mathcal{X}$, where X_I^* is uncorrupted. In addition, if the USign algorithm is deterministic, then E should not have previously made a **USign** query on $\langle X_I^*, m^* \rangle$ in Phase 1. C chooses a random bit β and if $\beta = 0$, C sets $\sigma^* = r$ where r is randomly chosen from $\mathcal{S}$, otherwise C sets $\sigma^* = \mathsf{USign}(x_I^*, m^*)$. C gives σ^* to E.

Phase 2: Again E can make queries as in Phase 1, except that E cannot corrupt X_I^* and E cannot make a **Conf/Deny** query on $\langle X_I^*, X_V, m^*, \sigma^* \rangle$ for any X_V. If the signature algorithm USign is deterministic, E is also forbidden from making a **USign** query on $\langle X_I^*, m^* \rangle$.

Output: Finally E outputs a bit β' and wins the game if $\beta' = \beta$.

Definition 4.5 We say that an NIDV undeniable signature scheme is *invisible* if the difference between the success probability of any polynomially bounded adversary and $1/2$ in the above game is negligible in l.

Definition 4.6 We say that an NIDV undeniable signature scheme is *secure* if the confirmation and denial NIDV proofs systems $\mathcal{C}$ and $\mathcal{D}$ with which it is composed are secure, and the core signature scheme is unforgeable and invisible.

4.5.2 Notes on the Security Definitions for Undeniable Signatures

Correctness: Although we do not explicitly define correctness for NIDV undeniable signatures, correctness is ensured by the correctness of proof systems $\mathcal{C}$ and $\mathcal{D}$.

Unforgeability: Our model of unforgeability differs from security models for normal undeniable signatures in two main ways. Firstly it is multiparty due to the multiparty nature of the NIDV confirmation and denial proofs. Secondly, we allow the adversary to make **FakeConf** and **FakeDeny** queries. We consider these to be necessary since an adversary may conceivably have access to such "fake" proofs produced by dishonest designated verifiers.

In answering **Conf/Deny** queries, C can determine whether a given message and signature pair (m, σ) is in the language $L(X_P)$ or not by generating an NIDV proof that $(m, \sigma) \in L(X_P)$ using $\mathsf{ConfGen}$. The resulting proof will be accepted by $\mathsf{ConfVerify}$ if and only if $(m, \sigma) \in L(X_P)$. This is guaranteed by the soundness of the

underlying NIDV proof. Alternatively, if USign is deterministic, then C could run USign to generate the unique signature σ' on m and compare this to σ. In this case $(m, \sigma) \in L(X_P)$ if and only if $\sigma = \sigma'$.

Invisibility: We use the notion of invisibility in our model of security rather than anonymity. Anonymity for NIDV undeniable signatures, which could be defined in a similar way to [55], captures the notion that an adversary cannot determine which of two possible signers created a given undeniable signature. Analogous results to those in [55] would show that our definition of invisibility implies anonymity for NIDV undeniable signatures and so is the stronger notion. Moreover, we feel that the stronger definition of invisibility is appropriate for NIDV undeniable signatures since we model the existence of fake NIDV proofs and their corresponding (possibly fake) signatures. This means that random (invalid) signatures (with corresponding NIDV proofs) can appear in the adversary's game, and these should be indistinguishable from true signatures (and their corresponding NIDV proofs). Therefore true signatures should be indistinguishable from random strings, which corresponds to the stronger notion of invisibility.

Determinism: Our definitions of unforgeability and invisibility encompass both deterministic and non-deterministic undeniable signatures. In either case, signers can identify their own valid signatures using ConfGen and ConfVerify.

We note that in the deterministic case, invisibility actually implies unforgeability, since an adversary who can forge signatures can trivially win the invisibility game. However for randomized signatures, these properties are distinct. We keep the properties distinct when proving the security of a deterministic undeniable signature scheme later in the paper because it simplifies the proof process. We first prove unforgeability of our concrete scheme and then use this result to simplify the proof of invisibility, thus avoiding a single, highly complex proof.

However it should be noted that deterministic schemes have distinct weaknesses, and that in practice, non-deterministic undeniable signature schemes should always be used. Although the scheme presented in the next section is deterministic, it can easily be made non-deterministic by adding a random salt into the hash function and distributing the random salt with the signature.

Overall security: We argue that the definition of security that we present for NIDV undeniable signatures does indeed capture the security properties that we require

for NIDV undeniable signatures.

Unforgeability guarantees that no-one can forge undeniable signatures without the appropriate private key, and invisibility guarantees that no-one can verify a given undeniable signature (or associate it with a certain public key) without the assistance of the signer.

When interacting with the signer, the soundness of the confirmation proof ensures that it is convincing. In other words, if the confirmation proof is valid, then the undeniable signature was indeed created by the signer. However the non-transferability of the confirmation proof guarantees that the verifier cannot prove the authorship of the undeniable signature to a third party.

When falsely accused of creating a given undeniable signature, the denial proof allows a party to deny having created an undeniable signature. The soundness of the denial proof ensures that it is convincing. In other words, if the denial proof is valid, then the undeniable signature was not created by the accused party. However the non-transferability of the denial proof guarantees that the verifier cannot prove this to a third party.

4.6 A Concrete NIDV Undeniable Signature Scheme

We present the full domain hash variant of the undeniable signature scheme of Chaum [35] with NIDV confirmation and denial proofs.

4.6.1 The Core Signature

$\mathsf{Setup}(l)$**:** For some security parameter l, let p and q be large primes, where $q|(p-1)$. Let G be the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q and let g be a generator of G. We also assume that $H_1 : \{0,1\}^* \to G$ is a cryptographic hash function. We set $\mathcal{PK} = \mathcal{S} = G$, $\mathcal{M} = \{0,1\}^*$ and $\mathcal{SK} = \mathbb{Z}_q^*$. The public parameters $params$ are $\langle p, q, g, H_1 \rangle$ as well as descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}$ and $\mathcal{S}$.

$\mathsf{KeyGen}(params)$**:** To set up a user's public and private keys, the private key x is chosen at random from $\mathbb{Z}_q^*$, and the corresponding public key is $X = g^x \bmod p$.

$\mathsf{USign}(x_I, m)$**:** On input $x_I \in \mathbb{Z}_q$, and $m \in \{0,1\}^*$, compute $\sigma = H_1(m)^{x_I} \bmod p$, and output σ.

4.6.2 The Confirmation and Denial Proofs

The USign algorithm defines a language $L(X_I) = \{(m, \sigma) : \sigma = \mathsf{USign}(x_I, m)\}$ for each public key X_I. For the above signature scheme, we can write $L(X_I) = \{(m, \sigma) : DL(\sigma, H_1(m)) = DL(x_I, g) \text{ in } G\}$. In other words, $L(X_I)$ is the language of all possible message and signature pairs (m, σ) where the discrete logarithm of σ to the base $H_1(m)$ equals the discrete logarithm of X_I to the base g modulo p. The family of languages $\mathcal{L}$ is defined as $\mathcal{L} = \{L(X_I) : X_I \in G\}$.

We can now define confirmation and denial proofs with respect to the languages $L(X_I)$.

Confirmation proof: The confirmation proof requires a secure NIDV proof system $\mathcal{C}$ for $\mathcal{L}$. Informally, $\mathcal{C}$ must prove the equality of two discrete logarithms (EDL).

Denial proof: The denial proof requires a secure NIDV proof system $\mathcal{D}$ for $\overline{\mathcal{L}}$. Informally, $\mathcal{D}$ must prove the inequality of two discrete logarithms (IDL).

4.6.3 A Concrete NIDV EDL Proof System

The NIDV proof we present is a modification of the scheme of Jakobsson et al. [70]. The modification repairs a flaw in the original proof, which was discovered by Wang [112].

Since our NIDV EDL proof will be used with the above undeniable signature scheme, the Setup algorithm will be identical to that in Section 4.6.1 except that in addition we require another cryptographic hash function $H_2 : \{0,1\}^* \to \mathbb{Z}_q$, and we define the spaces $\mathcal{E} = \mathcal{M} \times \mathcal{S}$ and $\mathcal{P} = \mathbb{Z}_q^4$. KeyGen will be exactly as in Section 4.6.1. The family of languages will be defined by the USign algorithm of the concrete scheme as described above in Section 4.6.2. We still need to define the PGen and $\mathsf{PVerify}$ algorithms.

NIDV EDL $\mathsf{PGen}(X_P, X_V, x_P, m, \sigma)$**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_P \in \mathcal{SK}$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$, the algorithm picks random $w, r, t \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^t \bmod p \\ D &= H_1(m)^t \bmod p \\ h &= H_2(c, G, D, m, \sigma, X_P, X_V) \\ d &= t - x_P(h + w) \bmod q \end{aligned}$$

The algorithm outputs $\pi = \langle w, r, h, d \rangle$.

NIDV EDL PVerify$(X_P, X_V, m, \sigma, \pi)$**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, message $m \in \mathcal{M}$, signature $\sigma \in \mathcal{S}$, and proof $\pi = \langle w, r, h, d \rangle \in \mathcal{P}$, the algorithm computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^d X_P^{(h+w)} \bmod p \\ D &= H_1(m)^d \sigma^{(h+w)} \bmod p \end{aligned}$$

and verifies that $h = H_2(c, G, D, m, \sigma, X_P, X_V)$. If the last equation holds, then the algorithm outputs *accept*, otherwise it outputs *reject*.

Comparison to the scheme of Jakobsson et al.

The main difference is that we include the values σ, X_P and X_V in the input of H_2. Our proof π also has one less element than the NIDV EDL proof of [70].

4.6.4 A Concrete NIDV IDL Proof System

Our denial proof is an NIDV version of the proof of inequality of discrete logarithms in [28].

Our Setup and KeyGen algorithms are as above in Section 4.6.3 for the NIDV EDL proof scheme except that now $\mathcal{P} = G \times \mathbb{Z}_q^4$.

NIDV IDL PGen$(X_P, X_V, x_P, m, \sigma)$**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_P \in \mathcal{SK}$, $m \in \mathcal{M}$, and $\sigma \in \mathcal{S}$, the algorithm picks random $t \in \mathbb{Z}_q$ and computes $C = (\frac{H_1(m)^{x_P}}{\sigma})^t \bmod p$.

The algorithm then constructs a designated verifier proof to demonstrate knowledge of some α and β such that $C = H_1(m)^\alpha \sigma^{-\beta} \bmod p$ and $1 = g^\alpha {X_P}^{-\beta} \bmod p$. The algorithm sets $\alpha = x_P t \bmod q$ and $\beta = t$, picks random $w, r, r_1, r_2 \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} C &= H_1(m)^\alpha \sigma^{-\beta} \bmod p \\ c &= g^w X_V^r \bmod p \\ G &= g^{r_1} (X_P)^{-r_2} \bmod p \\ D &= H_1(m)^{r_1} \sigma^{-r_2} \bmod p \\ h &= H_2(C, c, G, D, m, \sigma, X_P, X_V) \\ d_1 &= r_1 - \alpha(h + w) \bmod q \\ d_2 &= r_2 - \beta(h + w) \bmod q \end{aligned}$$

The algorithm outputs $\pi = \langle C, w, r, h, d_1, d_2 \rangle$ as the NIDV proof to verifier V that $\mathrm{DL}(\sigma, H_1(m)) \neq \mathrm{DL}(X_P, g)$.

NIDV IDL PVerify$(X_P, X_V, m, \sigma, \pi)$**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $m \in \mathcal{M}$, $\sigma \in \mathcal{S}$, and $\pi = \langle C, w, r, h, d_1, d_2 \rangle \in \mathcal{P}$. The algorithm first checks that $C \neq 1$. If $C = 1$, then the algorithm outputs *reject* and halts. Otherwise it computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^{d_1} X_P^{-d_2} \bmod p \\ D &= C^{h+w} H_1(m)^{d_1} \sigma^{-d_2} \bmod p \end{aligned}$$

and verifies that $h = H_2(C, c, G, D, m, \sigma, X_P, X_V)$. If the last equation holds, then the algorithm outputs *accept*, otherwise it outputs *reject*.

4.7 Security of the Concrete Scheme

4.7.1 Security of the NIDV EDL proof system

Theorem 4.1 The NIDV EDL proof system of Section 4.6.3 is *correct*.

Proof: It is trivial to verify that if NIDV EDL PGen is run on input $\langle X_P, X_V, x_P, m, \sigma \rangle$ where $(m, \sigma) \in L(X_P)$ and produces a proof $\pi = \langle w, r, h, d \rangle$, then NIDV EDL PVerify on input $\langle X_P, X_V, m, \sigma, \pi \rangle$ will output *accept*. □

Theorem 4.2 The NIDV EDL proof system of Section 4.6.3 is *non-transferable*.

Proof: We define algorithm NIDV EDL FakePGen as follows. On input $\langle X_P, X_V, x_V, m, \sigma' \rangle$, where $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_V \in \mathcal{SK}$, $m \in \mathcal{M}$ and $\sigma' \in \mathcal{E}$, NIDV EDL FakePGen produces proof $\pi' \in \mathcal{P}$ as follows:

NIDV EDL FakePGen chooses random $d', \alpha, \beta \in \mathbb{Z}_q$ and calculates:

$$\begin{aligned} c' &= g^\alpha \bmod p \\ G' &= g^{d'} X_P^{-\beta} \bmod p \\ D' &= H_1(m)^d (\sigma')^{-\beta} \bmod p \\ h' &= H_2(c', G', D', m, \sigma', X_P, X_V) \\ w' &= \beta - h' \bmod q \\ r' &= (\alpha - w') x_V^{-1} \bmod q \end{aligned}$$

NIDV EDL FakePGen outputs $\pi' = \langle w', r', h', d' \rangle$. It is easy to check that $\langle X_P, X_V, m, \sigma', \pi' \rangle$ will be accepted by NIDV EDL PVerify. We now show that π' is indistinguishable from any $\pi = \langle w, r, h, d \rangle$ produced by running NIDV EDL PGen on input $\langle X_P, X_V, x_P, m, \sigma \rangle$.

Examining the proof π' output by NIDV EDL FakePGen and a proof π produced by running NIDV EDL PGen on input $\langle X_P, X_V, x_P, m, \sigma \rangle$, we find that:

- Since r is chosen randomly from $\mathbb{Z}_q$, and r' depends on the random value α, r and r' are uniformly distributed in $\mathbb{Z}_q$, and therefore indistinguishable.
- Since w' depends on the random value $\beta \in \mathbb{Z}_q$ and w is chosen randomly from $\mathbb{Z}_q$, w and w' are uniformly distributed in $\mathbb{Z}_q$ and therefore indistinguishable.
- Since d' is chosen randomly from $\mathbb{Z}_q$ and d depends on the random value $t \in \mathbb{Z}_q$, d and d' are uniformly distributed in $\mathbb{Z}_q$ and therefore indistinguishable.

In proof π (respectively π'), w, r and d (respectively w', r' and d') are independent since each depends on a randomly chosen value (or is randomly chosen itself). Now h and h' are both outputs from the same hash function on indistinguishable inputs (if (m, σ) and (m, σ') are indistinguishable), therefore h and h' are indistinguishable, and the distributions of π and π' are indistinguishable. □

In order to analyze the soundness of our NIDV EDL proof, we first need to introduce a related non-generic (NG) signature scheme. We recall that generic signature schemes are simply digital signature schemes (as defined in Section 2.2.3) that take a certain form (which is described in Section 2.3.4). In addition, we recall from Section 2.3.5 that NG signature schemes are identical to generic signature schemes except that instead of generating signatures on a message M, NG signature schemes take an additional value T as input when generating or verifying a signature.

Our concrete NG signature scheme is defined as follows.

- The Setup and KeyGen algorithms are identical to those of the concrete NIDV EDL scheme except that the hash function H_1 is not required.
- The Sign algorithm for some public key $X \in \mathcal{PK}$ takes as input a message M where M may be of one of two forms:
    1. $M = m, \sigma, X, X_V$ where $(m, \sigma) \in L(X)$ and $X_V \in \mathcal{PK}$. The signing algorithm also has as inputs a value $T \in G$ and the private key $x \in \mathcal{SK}$ corresponding to

X. The algorithm runs in an identical way to NIDV EDL $\mathsf{PGen}(X, X_V, x, m, \sigma)$ except that T replaces $H_1(m)$ in producing a proof $\pi = \langle w, r, h, d\rangle$. The algorithm sets $r_2 = \langle w, r, d\rangle$ and $r_1 = g^w X_V^r \bmod p, g^d X_P^{h+w} \bmod p, T^d \sigma^{h+w} \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2\rangle$.

2. $M = m, \sigma, X_P, X$ where $m \in \mathcal{M}$, $\sigma \in \mathcal{S}$ and $X_P \in \mathcal{PK}$. The signing algorithm also has as inputs a value $T \in G$ and the private key $x \in \mathcal{SK}$ corresponding to X. The algorithm runs in an identical way to NIDV EDL $\mathsf{FakePGen}(X_P, X, x, m, \sigma)$ except that T replaces $H_1(m)$ in producing a proof $\pi = \langle w, r, h, d\rangle$. The algorithm sets $r_2 = \langle w, r, d\rangle$ and $r_1 = g^w X_V^r \bmod p, g^d X_P^{h+w} \bmod p, T^d \sigma^{h+w} \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2\rangle$.

- The Verify algorithm on input $M = m, \sigma, X_P, X_V$, a value $T \in G$ and a signature $\sigma_{NG} = \langle r_1, h, r_2\rangle$ where $X_P = X$ or $X_V = X$ and $r_2 = \langle w, r, d\rangle$, sets $\pi = \langle w, r, h, d\rangle$ and runs in an identical way to NIDV EDL $\mathsf{PVerify}(X_P, X_V, m, \sigma, \pi)$ except that T replaces $H_1(m)$ in the verification process.

Security for NG signature schemes is defined in the same way as security for digital signature schemes (in Section 2.3.1) except that the model is adapted to accommodate the additional value T. We refer to the above scheme as the NIDV EDL NG signature scheme.

Theorem 4.3 The NIDV EDL proof system is *sound* in the random oracle model assuming the hardness of the discrete logarithm problem in G.

Proof: We suppose that H_1 and H_2 are random oracles and there exists a polynomial time algorithm E that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_p **PGen** and μ_f **FakePGen** queries, and wins the soundness game of Section 4.3.3 in time τ with non-negligible probability η' (in security parameter l) where the number of participants is bounded by ρ. We assume that $\eta' > 10\rho(\mu_s + 1)(\mu_s + \mu_2)/2^l$ where $\mu_s = \mu_p + \mu_f$.

In Step 1 of the proof, we show how E can be used to construct an algorithm B that makes at most μ_2 queries to the random oracle H_2, at most μ_s **Sign** queries to its challenger C, and wins the NG version of the unforgeability game of Section 2.3.1 for the NIDV EDL NG signature scheme in time at most τ with non-negligible probability $\eta = \eta'/\rho$ where $\eta > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

In Step 2 of the proof, we then replace C with an algorithm C' that uses B to solve

the discrete logarithm problem in G. Step 2 of the proof will make use of Lemma 2.6, the NG Forking Lemma.

Step 1 We will show that there exists an algorithm B that uses E to forge an NIDV EDL NG signature with non-negligible probability when interacting with a challenger C in the NG version of the unforgeability game of Section 2.3.1.

The challenger C initializes the NG unforgeability game for B and gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$, and access to the random oracle H_2.

B can make $\boldsymbol{H_2}$ as well as **Sign** queries on any message $M = m, \sigma, X_P, X_V$ (where $m \in \mathcal{M}$, $\sigma' \in \mathcal{S}$, $X_P, X_V \in \mathcal{PK}$, and $X_P = X$ or $X_V = X$), and a value $T \in G$. B must eventually output a message M^*, a value T^*, and an NIDV EDL NG signature $\sigma^*_{NG} = \langle r^*_1, h^*, r^*_2\rangle$. B wins the game if $\langle M^*, T^*, \sigma^*_{NG}\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*, T^*.

In order to win the above game, B in turn simulates an NIDV soundness game for E.
NIDV Soundness Simulation:

B gives the parameters $\langle g, p, q\rangle$ and the descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$ to E. B generates a set of participants U, where $|U| = \rho(l)$ and ρ is a polynomial function of the security parameter l. For some random participant J, B sets $X_J = X$, and for each $I \neq J$, B runs KeyGen to generate a private key x_I and public key X_I. We define the set of all participants' public keys to be $\mathcal{X}$. E is given $\mathcal{X}$.

B now simulates the challenger by simulating all the queries which E can make as follows:

$\boldsymbol{H_1}$-Queries: E can query the random oracle H_1 at any time. B simulates the random oracle by keeping a list L_{H_1} of tuples $\langle str_i, r_i\rangle$. When the oracle is queried with an input $str \in \{0,1\}^*$, B responds as follows:

1. If the string str is already in L_{H_1} in the tuple $\langle str = str_i, r_i\rangle$, then B outputs $g^{r_i} \bmod p$.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs $g^r \bmod p$ and adds $\langle str, r\rangle$ to L_{H_1}.

$\boldsymbol{H_2}$-Queries: E can query any string str on the H_2 oracle. If $str = r_1, M$ where $r_1 \in G^3$ and $M = m, \sigma, X_P, X_V$ where $m \in \mathcal{M}$, $\sigma \in \mathcal{S}$, $X_P, X_V \in \mathcal{X}$ and $X_V \neq X_P$, then B

first queries m on H_1. B then simulates the H_2 oracle by passing all H_2 queries to C and passing C's response back to E.

EGen Queries: E can request an element of a particular language $L(X_I)$ (or its complement). On input a public key X_I, a bit b and message $m \in \mathcal{M}$, B proceeds as follows. If $X_I \neq X_J$ then B runs $\mathsf{USign}(x_I, m)$ to produce a signature $\sigma \in \mathcal{S}$ such that $(m, \sigma) \in L(X_I)$. If $X_I = X_J$ then B queries m on the H_1 oracle and receives some g^{r_i} as response. B retrieves the value r_i from L_{H_1} and sets $\sigma = X_I^{r_i} \bmod p$. If $b = 1$ then B outputs $\langle m, \sigma \rangle$, otherwise B picks a random $\sigma' \in \mathcal{S}$ where $\sigma' \neq \sigma$ and outputs $\langle m, \sigma' \rangle$. Since USign is deterministic, if $\sigma' \neq \sigma$, then $(m, \sigma') \notin L(X_I)$.

PGen Queries: E can request an NIDV EDL proof for input $\langle X_P, X_V, m, \sigma \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$.

B queries m on H_1 and receives some $H_1(m) = g^{r_i}$. B retrieves the value r_i from L_{H_1} and computes $\sigma' = X_P^{r_i} \bmod p$. If $\sigma' \neq \sigma$ then B outputs *invalid*.

If $\sigma' = \sigma$ and $X_P \neq X_J$, then B runs $\mathsf{NIDV\ EDL\ PGen}(X_P, X_V, x_P, m, \sigma)$ to produce a proof $\pi = \langle w, r, h, d \rangle$, and B outputs π to E.

If $\sigma' = \sigma$ and $X_P = X_J$, then B sets $M = m, \sigma, X_P, X_V$ and $T = H_1(m)$ and makes a **Sign** query to C on M and T. C responds with an NIDV EDL NG signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $r_2 = \langle w, r, d \rangle$. B sets $\pi = \langle w, r, h, d \rangle$ and outputs π to E.

FakePGen Queries: E can request a fake NIDV EDL proof on input $\langle X_P, X_V, m, \sigma' \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$, and $\sigma' \in \mathcal{S}$.

If $X_V \neq X_J$ then B runs $\mathsf{NIDV\ EDL\ FakePGen}(X_P, X_V, x_V, m, \sigma)$ to produce a proof $\pi = \langle w, r, h, d \rangle$, and B outputs π to E.

If $X_V = X_J$ then B sets $M = m, \sigma, X_P, X_V$ and $T = H_1(m)$ and makes a **Sign** query to C on M and T. C responds with an NIDV EDL NG signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $r_2 = \langle w, r, d \rangle$. B sets $\pi = \langle w, r, h, d \rangle$ and outputs π to E.

Corrupt Queries: E can request the private key corresponding to any public key $X_I \in \mathcal{X}$. If $X_I = X_J$, then B aborts and terminates E. Otherwise B returns the appropriate private key x_I.

Output: Finally E outputs $\langle X_P^*, X_V^*, m^*, \sigma^*, \pi^* \rangle$, where $X_P^*, X_V^* \in \mathcal{X}$, $X_P^* \neq X_V^*$, X_V^* is uncorrupted, $m^* \in \mathcal{M}$, $\sigma^* \in \mathcal{S}$ and $\pi^* \in \mathcal{P}$. E wins if $\langle X_P^*, X_V^*, m^*, \sigma^*, \pi^* \rangle$ is accepted by $\mathsf{PVerify}$, no **FakePGen** query was made on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$, and either:

1. X_P^* is uncorrupted and no **PGen** query was made on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$, or
2. $(m^*, \sigma^*) \notin L(X_P^*)$.

B takes E's output π^* where $\pi^* = \langle w^*, r^*, h^*, d^* \rangle$, and sets $M^* = m^*, \sigma^*, X_P^*, X_V^*$, $T^* = H_1(m^*)$ and $\sigma_{NG}^* = \langle r_1^*, h^*, r_2^* \rangle$ where

$$r_1^* = g^{w^*} {X_V^*}^{r^*} \bmod p, g^{d^*} {X_P^*}^{h^*+w^*} \bmod p, {T^*}^{d^*} {\sigma^*}^{h^*+w^*} \bmod p$$

and $r_2^* = \langle w^*, r^*, d^* \rangle$.

B outputs M^*, T^* and σ_{NG}^* to C. If E satisfied output condition 1, then E never made any **FakePGen** or **PGen** queries on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$. If E satisfied output condition 2 then E never made any **FakePGen** queries on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$ and B would have output *invalid* for any **PGen** queries on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$ E had made. In either case, B never made any **Sign** queries on $M^* = m^*, \sigma^*, X_P^*, X_V^*$ and $T^* = H_1(m^*)$.

The only way that E could detect an inconsistency in the game is if B aborts or if B incorrectly answers any **PGen** query. We therefore have to check that B's simulation of these queries is indistinguishable from that output by a real challenger.

We notice that any **PGen** query on $\langle X_P, X_V, m, \sigma \rangle$ will output *invalid* if and only if $(m, \sigma) \notin L(X_P)$. We therefore need to check that if algorithm NIDV EDL PGen is run on $\langle X_P, X_V, m, \sigma \rangle$ to produce a proof π, then $\langle X_P, X_V, m, \sigma, \pi \rangle$ will be accepted by NIDV EDL PVerify if and only if $(m, \sigma) \in L(X_P)$.

The correctness of the NIDV EDL scheme guarantees that if $(m, \sigma) \in L(X_P)$ then NIDV EDL PVerify$(X_P, X_V, m, \sigma, \pi)$ will output *accept*. It is also trivial to verify that if $(m, \sigma) \notin L(X_P)$ then NIDV EDL PVerify$(X_P, X_V, m, \sigma, \pi)$ will output *reject*. We leave the details to the reader. Therefore E can detect no inconsistency in B's simulation of an NIDV soundness game unless B aborts.

The probability that B does not have to abort, E wins the game, and that either $X_V^* = X_J$, or $X_P^* = X_J$, is η'/ρ, which is non-negligible in security parameter l if η is. In this case, B would not have had to abort, and B wins the NG unforgeability game with probability $\eta = \eta'/\rho$ making at most μ_2 queries to the random oracle H_2, and at most $\mu_s = \mu_p + \mu_f$ **Sign** queries to C.

Step 2 We now define an algorithm C' that replaces B's challenger C and uses B to solve the discrete logarithm problem in G. C' will simulate the random oracle H_2 and the challenger in an NG unforgeability game with B. C''s goal is to solve the discrete

logarithm problem on input $\langle g, X, p, q\rangle$, that is to find $x \in \mathbb{Z}_q$ such that $g^x = X \bmod p$, where g is of prime order q modulo prime p and generates group G.

NG Unforgeability Simulation:

C' initializes the NG unforgeability game for B as follows. C' gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$ as they are defined in NIDV EDL Setup, and access to the random oracle H_2.

C' now simulates all the queries which B can make as follows:

H_2-Queries: B can query the random oracle H_2 at any time. C' simulates the random oracle by keeping a list L_{H_2} of tuples $\langle str_i, r_i\rangle$. When the oracle is queried with an input $str \in \{0,1\}^*$, C' responds as follows:

1. If the string str is already in L_{H_2} in the tuple $\langle str = str_i, r_i\rangle$, then B outputs r_i.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs r and adds $\langle str, r\rangle$ to L_{H_2}.

Sign Queries: C' will also answer **Sign** queries made by B. All of B's queries are on messages $M = m, \sigma, X_P, X_V$ where $X_P = X$ or $X_V = X$, and values $T \in G$. C' picks random $w, r, h \in \mathbb{Z}_q$ and computes

$$r_1 = g^w X_V^r \bmod p, g^d X_P^{h+w} \bmod p, T^d \sigma^{h+w} \bmod p.$$

C' constructs the string $str = r_1, M$, and adds the tuple $\langle str, h\rangle$ to L_{H_2}. C' then sets $r_2 = \langle w, r, d\rangle$ and $\sigma_{NG} = \langle r_1, h, r_2\rangle$ which it outputs to B.

Output: Finally B should output a message $M^* = m^*, \sigma^*, X_P^*, X_V^*$ where $X_P^* = X$ or $X_V^* = X$, a value T^* and an NIDV EDL NG signature $\sigma_{NG}^* = \langle r_1^*, h^*, r_2^*\rangle$. B wins the game if $\langle M^*, T^*, \sigma_{NG}^*\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*, T^*.

Since H_2 is a random oracle, C' can simulate NIDV EDL NG signatures that are indistinguishable from true NIDV EDL NG signatures, so B can detect no inconsistency in the game.

From Step 1, we know that B that makes at most μ_2 queries to the random oracle H_2, at most $\mu_s = \mu_p + \mu_f$ **Sign** queries, and wins the above game in time at most τ with non-negligible probability $\eta = \eta'/\rho$ where $\eta > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

By Lemma 2.6, (the NG Forking Lemma), C' can rewind B (and therefore also E) with the same random coins and repeat its simulation with a different random oracle H_2 so that B outputs another NG signature $\sigma_{NG} = \langle r_1^*, h, r_2 \rangle$ on $M^* = m^*, \sigma^*, X_P^*, X_V^*$, together with a value T, where $h \neq h^*$, $r_2 = \langle w, r, d \rangle$ and

$$r_1^* = g^w {X_V^*}^r \bmod p, g^d {X_P^*}^{h+w} \bmod p, T^d {\sigma^*}^{h+w} \bmod p.$$

However we know that in Step 1, B was constructed in such a way that $T^* = H_1(m^*)$ and $T = H_1(m^*)$ where H_1 is simulated by B. In each case B is run on the same random coins and therefore simulates H_1 in the same way until the H_2 query on $r_1^*, M^* = m^*, \sigma^*, X_P^*, X_V^*$ is made. Therefore since B will always query m^* on H_1 before making the corresponding H_2 query on r_1^*, M^*, we know that $T^* = T$.

We therefore obtain the equations

$$g^{w^*} {X_V^*}^{r^*} = g^w {X_V^*}^r \bmod p \quad (4.1)$$

$$g^{d^*} {X_P^*}^{(h^*+w^*)} = g^d {X_P^*}^{(h+w)} \bmod p \quad (4.2)$$

$${T^*}^{d^*} {\sigma^*}^{(h^*+w^*)} = {T^*}^d {\sigma^*}^{(h+w)} \bmod p. \quad (4.3)$$

We now need to distinguish between two cases, depending on whether E wins the NIDV soundness game (simulated by B) by satisfying output condition 1 or 2. If E wins the game by satisfying condition 1, then we say that E is a Type-1 adversary, otherwise we say that E is a Type-2 adversary.

Case 1. We suppose that E is a Type-1 adversary. In this case, X_P^* and X_V^* were both uncorrupted in the NIDV soundness game, and we could have that $X_P^* = X$ or that $X_V^* = X$. We now consider two subcases, depending on whether $r^* \neq r$ or $r^* = r$. If $r^* \neq r$ then if $X_V^* = X$ then C' can solve (4.1) for the discrete logarithm of $X_V^* = X$. The probability that $X_V^* = X$ is 1/2 since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$. If $r^* = r$, then $w^* = w$. If $X_P^* = X$ then since $h^* \neq h$ we have that $h^* + w^* \neq h + w$, so C' can solve (4.2) for the discrete logarithm of $X_P^* = X$. The probability that $X_P^* = X$ is 1/2 since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$.

Case 2. We suppose that E is a Type-2 adversary. In this case, we know that X_V^* is uncorrupted and that $(m^*, \sigma^*) \notin L(X_P^*)$. Since X_V^* is uncorrupted we could have that $X_V^* = X$. As in Case 1, if $r^* \neq r$ and $X_V^* = X$ then C' can solve (4.1) for the discrete

logarithm of $X_V^* = X$. The probability that $X_V^* = X$ is at least 1/2 since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$. If $r^* = r$, then $w^* = w$. But since $h^* \neq h$ we can rewrite equations (4.2) and (4.3) as $X_P^* = g^{\frac{d-d^*}{h^*-h}}$ and $\sigma^* = T^{*\frac{d-d^*}{h^*-h}}$. But this contradicts our assumption that $(m^*, \sigma^*) \notin L(X_P^*)$.

In cases 1 and 2, since the NG Forking Lemma produces a second appropriate signature with expected time at most $\tau' = 120686\mu_s\tau$, we find that C' can solve the discrete logarithm problem in time at most τ' and with probability at least $\eta/2$. If η' is non-negligible, then η is also non-negligible, and this contradicts the hardness of the discrete logarithm problem. Therefore no algorithm B as defined in Step 1 can have non-negligible probability of winning the NG unforgeability game, and in turn no polynomially bounded adversary E can have non-negligible probability of winning the soundness game of Section 4.3.3. □

4.7.2 Security of the NIDV IDL proof system

Theorem 4.4 The NIDV IDL proof of Section 4.6.4 is *correct*.

Proof: It is trivial to verify that if NIDV IDL PGen is run on input $\langle X_P, X_V, x_P, m, \sigma\rangle$ where $(m, \sigma) \notin L(X_P)$ and produces a proof $\pi = \langle C, w, r, h, d_1, d_2\rangle$, then NIDV IDL PVerify on input $\langle X_P, X_V, m, \sigma, \pi\rangle$ will output *accept*. □

Theorem 4.5 The NIDV IDL proof of Section 4.6.4 is *non-transferable*.

Proof: We define algorithm NIDV IDL FakePGen as follows. On input $\langle X_P, X_V, x_V, m, \sigma'\rangle$, where $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_V \in \mathcal{SK}$, $m \in \mathcal{M}$ and $\sigma' \in \mathcal{E}$, NIDV IDL FakePGen produces proof $\pi' \in \mathcal{P}$ as follows:

NIDV IDL FakePGen chooses random $y' \in \mathbb{Z}_q^*$, and checks that $H_1(m)^{y'} \neq \sigma'$. If $H_1(m)^{y'} = \sigma'$ then a new y' is chosen. The algorithm then chooses random $s', t', u', d_1', d_2' \in \mathbb{Z}_q$ and calculates:

$$\begin{aligned}
C' &= \left(\frac{H_1(m)^{y'}}{\sigma'}\right)^{t'} \bmod p \\
G' &= g^{d_1'} X_P^{-d_2'} \bmod p \\
D' &= C^{s'} H_1(m)^{d_1'} (\sigma')^{-d_2'} \bmod p \\
c' &= g^{u'} \bmod p \\
h' &= H_2(C', c', G', D', m, \sigma', X_P, X_V)
\end{aligned}$$

$$\begin{aligned} w' &= s' - h' \bmod q \\ r' &= (u' - s' + h')x_V^{-1} \bmod q \end{aligned}$$

NIDV IDL FakePGen outputs $\pi' = \langle C', w', r', h', d_1', d_2' \rangle$. It is easy to check that $\langle X_P, X_V, m, \sigma', \pi' \rangle$ will be accepted by NIDV IDL PVerify. We now show that π' is indistinguishable from any $\pi = \langle C, w, r, h, d_1, d_2 \rangle$ produced by running NIDV IDL PGen on input $\langle X_P, X_V, x_P, m, \sigma \rangle$.

Examining proofs π' and π we find that:

- C and C' are both computed as some power of $H_1(m)$ multiplied by $\sigma^{-1} \bmod p$, all to some random power in $\mathbb{Z}_q^*$, and are therefore indistinguishable.
- Since w is chosen randomly from $\mathbb{Z}_q^*$, and w' depends on the random value $s' \in \mathbb{Z}_q^*$, w and w' are uniformly distributed in $\mathbb{Z}_q^*$ and therefore indistinguishable.
- Since r is chosen randomly from $\mathbb{Z}_q^*$, and r' depends on the random value $u' \in \mathbb{Z}_q^*$, r and r' are uniformly distributed in $\mathbb{Z}_q^*$ and therefore indistinguishable.
- Since d_1 depends on the random value $r_1 \in \mathbb{Z}_q^*$, and d_1' is chosen randomly from $\mathbb{Z}_q^*$, d_1 and d_1' are uniformly distributed in $\mathbb{Z}_q^*$ and therefore indistinguishable.
- Since d_2 depends on the random value $r_2 \in \mathbb{Z}_q^*$, and d_2' is chosen randomly from $\mathbb{Z}_q^*$, d_2 and d_2' are uniformly distributed in $\mathbb{Z}_q^*$ and therefore indistinguishable.

In proof π (similarly π'), C, w, r, d_1 and d_2 (similarly $C' w', r', d_1'$ and d_2') are independent since each depends on a randomly chosen value (or is randomly chosen itself). Now h and h' are both outputs from the same hash function on indistinguishable inputs (if (m, σ) and (m, σ') are indistinguishable), therefore h and h' are indistinguishable, and the distributions of π and π' are indistinguishable. □

As for the NIDV EDL proof, in order to analyze the soundness of the NIDV IDL proof, we first need to introduce a related non-generic (NG) signature scheme, which we refer to as the NIDV IDL NG signature scheme. Our concrete NIDV IDL NG signature scheme is defined as follows.

- The Setup and KeyGen algorithms are identical to those of the concrete NIDV IDL scheme except that the hash function H_1 is not required.
- The Sign algorithm for some public key $X \in \mathcal{PK}$ takes as input a message M where M may be of one of two forms:

1. $M = m, \sigma, X, X_V$ where $(m, \sigma) \notin L(X)$ and $X_V \in \mathcal{PK}$. The signing algorithm also takes as input a value $T \in G$ and the private key $x \in \mathcal{SK}$ corresponding to X. The algorithm runs in an identical way to NIDV IDL $\mathsf{PGen}(X, X_V, x, m, \sigma)$ except that T replaces $H_1(m)$ in producing a proof $\pi = \langle C, w, r, h, d_1, d_2 \rangle$. The algorithm sets $r_2 = \langle C, w, r, d_1, d_2 \rangle$ and $r_1 = C, g^w X_V^r \bmod p, g^{d_1} X_P^{-d_2} \bmod p, C^{h+w} T^{d_1} \sigma^{-d_2} \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2 \rangle$.
2. $M = m, \sigma, X_P, X$ where $m \in \mathcal{M}$, $\sigma \in \mathcal{S}$ and $X_P \in \mathcal{PK}$. The signing algorithm also takes as input a value $T \in G$ and the private key $x \in \mathcal{SK}$ corresponding to X. The algorithm runs in an identical way to NIDV IDL $\mathsf{FakePGen}(X_P, X, x, m, \sigma)$ except that T replaces $H_1(m)$ in producing a proof $\pi = \langle C, w, r, h, d_1, d_2 \rangle$. The algorithm sets $r_2 = \langle C, w, r, d_1, d_2 \rangle$ and $r_1 = C, g^w X_V^r \bmod p, g^{d_1} X_P^{-d_2} \bmod p, C^{h+w} T^{d_1} \sigma^{-d_2} \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2 \rangle$.

- The Verify algorithm on input $M = m, \sigma, X_P, X_V$, a value $T \in G$ and a signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $X_P = X$ or $X_V = X$ and $r_2 = \langle C, w, r, d_1, d_2 \rangle$, sets $\pi = \langle C, w, r, h, d_1, d_2 \rangle$ and runs in an identical way to NIDV IDL $\mathsf{PVerify}(X_P, X_V, m, \sigma, \pi)$ except that T replaces $H_1(m)$ in the verification process.

Theorem 4.6 The NIDV IDL proof of Section 4.6.4 is *sound* in the random oracle model assuming the hardness of the discrete logarithm problem in G.

Proof:

The proof of this theorem is similar to the proof of Theorem 4.3, so we highlight the areas of the proof that differ, and where the proof is highly duplicated, we leave the details to the reader.

As before, we suppose that H_1 and H_2 are random oracles and there exists an algorithm E that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_p **PGen** and μ_f **FakePGen** queries, and wins the soundness game of Section 4.3.3 in time at most τ with non-negligible probability η' (in security parameter l) where the number of participants is bounded by ρ and $\eta' > 10\rho(\mu_p + \mu_f + 1)(\mu_p + \mu_f + \mu_2)/2^l$.

As in the proof of Theorem 4.3 we divide the proof into two steps. In Step 1, we show how E can be used to construct an algorithm B that makes at most μ_2 queries to the random oracle H_2, at most $\mu_s = \mu_p + \mu_f$ **Sign** queries to its challenger C, and wins the NG version of the unforgeability game of Section 2.3.1 for the NIDV IDL NG signature scheme in time at most τ with non-negligible probability $\eta = \eta'/\rho > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

In Step 2, we then replace C with an algorithm C' that uses B to solve the discrete logarithm problem in G. Step 2 of the proof will make use of Lemma 2.6, the NG Forking Lemma.

Step 1 We will show that there exists an algorithm B that uses E to forge an NIDV IDL NG signature with non-negligible probability when interacting with a challenger C in the NG version of the unforgeability game of Section 2.3.1.

The challenger C initializes the NG unforgeability game for B and gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$, and access to the random oracle H_2.

B can make $\boldsymbol{H_2}$ as well as **Sign** queries on any message $M = m, \sigma, X_P, X_V$ (where $m \in \mathcal{M}$, $\sigma' \in \mathcal{S}$, $X_P, X_V \in \mathcal{PK}$, and $X_P = X$ or $X_V = X$), and a value $T \in G$. B must eventually output a message M^*, a value T^*, and an NIDV IDL NG signature $\sigma^*_{NG} = \langle r^*_1, h^*, r^*_2\rangle$. B wins the game if $\langle M^*, T^*, \sigma^*_{NG}\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*, T^*.

In order to win the above game, B in turn simulates an NIDV soundness game for E.

NIDV Soundness Simulation:

B initializes the game for E exactly as in the proof of Theorem 4.3, and answers the $\boldsymbol{H_1}, \boldsymbol{H_2}$ and **Corrupt** queries exactly as before. The other queries are answered as follows.

EGen Queries: B answers these queries as in the proof of Theorem 4.3, except that in this case if $b = 0$ then B outputs a valid signature, otherwise B picks a random invalid signature and outputs this.

PGen Queries: E can request an NIDV EDL proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$.

B queries m on H_1 and receives some $H_1(m) = g^{r_i}$. B retrieves the value r_i from L_{H_1} and computes $\sigma' = X_P^{r_i} \bmod p$. If $\sigma' = \sigma$ then B outputs *invalid*.

If $\sigma' = \sigma$ and $X_P \neq X_J$, then B runs $\mathsf{NIDV\ IDL\ PGen}(X_P, X_V, x_P, m, \sigma)$ to produce a proof $\pi = \langle C, w, r, h, d_1, d_2\rangle$, and B outputs π to E.

If $\sigma' = \sigma$ and $X_P = X_J$, then B sets $M = m, \sigma, X_P, X_V$ and $T = H_1(m)$ and makes a **Sign** query to C on M and T. C responds with an NIDV IDL NG signature $\sigma_{NG} = \langle r_1, h, r_2\rangle$ where $r_2 = \langle C, w, r, d_1, d_2\rangle$. B sets $\pi = \langle C, w, r, h, d_1, d_2\rangle$ and outputs π to E.

FakePGen Queries: E can request a fake NIDV proof on input $\langle X_P, X_V, m, \sigma' \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$, and $\sigma' \in \mathcal{S}$.

If $X_V \neq X_J$ then B runs NIDV IDL FakePGen$(X_P, X_V, x_V, m, \sigma)$ to produce a proof $\pi = \langle C, w, r, h, d_1, d_2 \rangle$, and B outputs π to E.

If $X_V = X_J$ then B sets $M = m, \sigma, X_P, X_V$ and $T = H_1(m)$ and makes a **Sign** query to C on M and T. C responds with an NIDV IDL NG signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $r_2 = \langle C, w, r, d_1, d_2 \rangle$. B sets $\pi = \langle C, w, r, h, d_1, d_2 \rangle$ and outputs π to E.

Output: Finally E outputs $\langle X_P^*, X_V^*, m^*, \sigma^*, \pi^* \rangle$, where $X_P^*, X_V^* \in \mathcal{X}$, $X_P^* \neq X_V^*$, X_V^* is uncorrupted, $m^* \in \mathcal{M}$, $\sigma^* \in \mathcal{S}$ and $\pi^* \in \mathcal{P}$. E wins if $\langle X_P^*, X_V^*, m^*, \sigma^*, \pi^* \rangle$ is accepted by NIDV IDL PVerify, no **FakePGen** query was made on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$, and either:

1. X_P^* is uncorrupted and no **PGen** query was made on $\langle X_P^*, X_V^*, m^*, \sigma^* \rangle$, or
2. $(m^*, \sigma^*) \in L(X_P^*)$.

B takes E's output π^* where $\pi^* = \langle C^*, w^*, r^*, h^*, d_1^*, d_2^* \rangle$, and sets $M^* = m^*, \sigma^*, X_P^*, X_V^*$, $T^* = H_1(m^*)$ and $\sigma_{NG}^* = \langle r_1^*, h^*, r_2^* \rangle$ where

$$r_1^* = C^*, g^{w^*} {X_V^*}^{r^*} \bmod p, g^{d_1^*} {X_P^*}^{-d_2^*} \bmod p, {C^*}^{h^* + w^*} {T^*}^{d_1^*} {\sigma^*}^{-d_2^*} \bmod p$$

and $r_2^* = \langle C^*, w^*, r^*, d_1^*, d_2^* \rangle$.

B outputs M^*, T^* and σ_{NG}^* to C. The rest of Step 1 follows in the same way as in Step 1 of the proof of Theorem 4.3.

Once again we find that the probability that B does not have to abort, E wins the game, and that either $X_V^* = X_J$, or $X_P^* = X_J$, is η'/ρ, which is non-negligible in security parameter l if η is. In this case, B would not have had to abort, and B wins the NG unforgeability game with probability $\eta = \eta'/\rho$ making at most μ_2 queries to the random oracle H_2, and at most $\mu_s = \mu_p + \mu_f$ **Sign** queries to C.

Step 2 We now define an algorithm C' that replaces B's challenger C and uses B to solve the discrete logarithm problem in G. As in the proof of Theorem 4.3, C' will simulate the random oracle H_2 and the challenger in an NG unforgeability game with B. C''s goal is to solve the discrete logarithm problem on input $\langle g, X, p, q \rangle$, that is to find $x \in \mathbb{Z}_q$ such that $g^x = X \bmod p$, where g is of prime order q modulo prime p and generates group G.

NG Unforgeability Simulation:

The challenger C' initializes the NG unforgeability game for B exactly as before, giving B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$, and access to the random oracle H_2.

C' now simulates the challenger exactly as before except for **Sign** queries which C' simulates as follows.

Sign Queries: C' will also answer **Sign** queries made by B. All of B's queries are on messages $M = m, \sigma, X_P, X_V$ where $X_P = X$ or $X_V = X$, and values $T \in G$.

C' picks random $t, w, r, h, d_1, d_2 \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} C &= \left(\frac{X_P^{r_i}}{\sigma}\right)^t \bmod p \\ r_1 &= C, g^w X_V^r \bmod p, g^{d_1} X_P^{-d_2} \bmod p, C^{h+w} t^{d_1} \sigma^{-d_2} \bmod p. \end{aligned}$$

C' constructs the string $str = r_1, M$, and adds the tuple $\langle str, h\rangle$ to L_{H_2}. $r_2 = \langle C, w, r, d_1, d_2\rangle$ and $\sigma_{NG} = \langle r_1, h, r_2\rangle$ which it outputs to B.

Since H_2 is a random oracle, C' can simulate NIDV IDL NG signatures that are indistinguishable from true NIDV IDL NG signatures, so B can detect no inconsistency in the game.

From Step 1, we know that B that makes at most μ_2 queries to the random oracle H_2, at most $\mu_s = \mu_p + \mu_f$ **Sign** queries, and wins the above game in time at most τ with non-negligible probability $\eta = \eta'/\rho > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

By Lemma 2.6, (the NG Forking Lemma), C' can rewind B (and therefore also E) with the same random coins and repeat its simulation with a different random oracle H_2 so that B outputs another NG signature $\sigma_{NG} = \langle r_1^*, h, r_2\rangle$ on $M^* = m^*, \sigma^*, X_P^*, X_V^*$, together with a value T, where $h \neq h^*$, $r_2 = \langle C^*, w, r, d_1, d_2\rangle$ and

$$r_1^* = C^*, g^w {X_V^*}^r \bmod p, g^{d_1} {X_P^*}^{-d_2} \bmod p, {}^*C^{h+w} T^{d_1} {\sigma^*}^{-d_2} \bmod p.$$

By the same argument as in the proof of Theorem 4.3, we know that $T^* = T$, and we obtain the equations

$$g^{w^*} {X_V^*}^{r^*} = g^w {X_V^*}^r \bmod p \tag{4.4}$$

$$g^{d_1^*} {X_P^*}^{-d_2^*} = g^{d_1} {X_P^*}^{-d_2} \bmod p \tag{4.5}$$

$$C^{*h^*+w^*} {T^*}^{d_1^*} {\sigma^*}^{-d_2^*} = C^{*h+w} {T^*}^{d_1} {\sigma^*}^{-d_2} \bmod p. \tag{4.6}$$

Once again, we distinguish between two cases, depending on whether E wins the NIDV soundness game (simulated by B) by satisfying output condition 1 or 2. If E wins the game by satisfying condition 1, then we say that E is a Type-1 adversary, otherwise we say that E is a Type-2 adversary.

Case 1. We suppose that E is a Type-1 adversary. In this case, X_P^* and X_V^* were both uncorrupted in the NIDV soundness game, and we could have that $X_P^* = X$ or that $X_V^* = X$. Now if $r^* \neq r$ then if $X_V^* = X$ then C' can solve (4.4) for the discrete logarithm of $X_V^* = X$. The probability that $X_V^* = X$ is $1/2$ since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$. If $d_2 \neq d_2^*$ and $X_P^* = X_J$ then C' can solve (4.5) for the discrete logarithm of $X_P^* = X$. The probability that $X_P^* = X$ is $1/2$ since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$. Alternatively, if $r^* = r$ and $d_2 = d_2^*$, then $w = w^*$ and $d_1 = d_1^*$. However this is impossible since $h \neq h^*$.

Case 2. We suppose that E is a Type-2 adversary. In this case, we know that X_V^* is uncorrupted and that $(m^*, \sigma^*) \in L(X_P^*)$. As in Case 1, if $r^* \neq r$ and $X_V^* = X_J$ then C' can solve (4.4) for the discrete logarithm of $X_V^* = X$. The probability that $X_V^* = X$ is at least $1/2$ since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X$ or $X_P^* = X$. If $r = r^*$, then $w = w^*$, and therefore $h + w \neq h^* + w^*$ (since $h \neq h^*$). By equation (4.6) we find that $d_1 \neq d_1^*$ or $d_2 \neq d_2^*$, and by equation (4.4), we find that $d_1 \neq d_1^*$ and $d_2 \neq d_2^*$. We can therefore rewrite equations (4.5) and (4.6) as

$$X_P^* = g^{\frac{d_1 - d_1^*}{d_2 - d_2^*}}$$

$$\sigma^* = (T^*)^{\frac{d_1 - d_1^*}{d_2 - d_2^*}} C^{*\frac{h + w - h^* - w^*}{d_2 - d_2^*}}$$

Since $C^* \neq 1$ and C^* is raised to a non-zero power, this contradicts our assumption that $(m^*, \sigma^*) \in L(X_P^*)$.

In cases 1 and 2, since the NG Forking Lemma produces a second appropriate signature with expected time at most $\tau' = 120686\mu_s\tau$, we find that C' can solve the discrete logarithm problem in time at most τ' and with probability at least $\eta/2$. If η' is non-negligible, then η is also non-negligible, and this contradicts the hardness of the discrete logarithm problem. Therefore no algorithm B as defined in Step 1 can have non-negligible probability of winning the NG unforgeability game, and in turn no polynomially bounded adversary E can have non-negligible probability of winning the soundness game of Section 4.3.3.

□

4.7.3 Application to the core signature scheme

Since our NIDV EDL and IDL proof systems are secure, they can be composed with our concrete scheme to form secure NIDV confirmation and denial proofs for the NIDV undeniable signature scheme. All that remains for the whole NIDV undeniable signature scheme to be secure is to show that the core signature scheme satisfies the unforgeability and invisibility properties.

Theorem 4.7 The core signature scheme of Section 4.6.1 is *unforgeable* in the random oracle model assuming the hardness of the Computational Diffie-Hellman problem in G.

The proof of Theorem 4.7 is similar to the proof of unforgeability in [92] (corrected in [91]), although we use a slightly different security model and therefore provide our own proof of security.

Proof: We suppose that H_1 and H_2 are random oracles, and suppose there exists an algorithm E in a game with at most ρ participants that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_s **USign** queries, and at most μ_c **Conf/Deny** queries, and wins the unforgeability game of Section 4.5.1.1 in time at most τ with probability at least η, where η is non-negligible in security parameter l.

We show how to construct an algorithm B that uses E to solve the computational Diffie-Hellman problem. B will simulate the random oracles and the challenger C in a game with E. B's goal is to solve the computational Diffie-Hellman problem on input $\langle g, g^a, g^b, p, q\rangle$, that is to find $g^{ab} \in \mathbb{Z}_q$, where g is of prime order q modulo prime p.

Simulation: B initializes the game using Setup and the parameters g, p and q. B gives the parameters $\langle g, p, q\rangle$ and the descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$ to E. B generates a set of participants U, where $|U| = \rho(l)$ and ρ is a polynomial function of the security parameter l. For some random participant J, B sets $X_J = g^a \bmod p$, and for each $I \neq J$, B runs KeyGen to generate a private key x_I and public key X_I. We define the set of all participants' public keys to be $\mathcal{X}$. E is given $\mathcal{X}$. In addition, B randomly chooses bit $\beta \in \{0, 1\}$ and an integer $k \in \{1, .., \mu_1\}$.

***H_1*-Queries:** B simulates the random oracle by maintaining a list L_{H_1} of tuples $\langle str_i, r_i\rangle$.

When H_1 is queried with an input $str \in \{0, 1\}^*$, B responds as follows:

1. If the query str is already in L_{H_1} then it must be contained in some tuple $\langle str = str_i, r_i \rangle$. If this is the kth tuple on the list, then B outputs $(g^b)^{r_i} \bmod p$. Otherwise B outputs $g^{r_i} \bmod p$.
2. If str is not already in L_{H_1}, B selects a random $r \in \mathbb{Z}_q$. If str is the kth distinct H_1 query then B outputs $(g^b)^r \bmod p$ and adds $\langle str, r \rangle$ to L_{H_1}. Otherwise B outputs $g^r \bmod p$ and adds $\langle str, r \rangle$ to L_{H_1}.

H_2-Queries: B simulates the H_2 oracle in the same way as the H_2 oracle in the proof of Theorem 4.3.

USign Queries: E can request an undeniable signature for input $\langle X_I, m \rangle$ where $X_I \in \mathcal{X}$, and $m \in \mathcal{M}$. If $X_I \neq X_J$ then B runs $\mathsf{USign}(x_I, m)$ to produce a signature $\sigma \in \mathcal{S}$. If $X_I = X_J$ then B queries m on the H_1 oracle and receives some response g^{r_i}. If the tuple on L_{H_1} containing m is the kth tuple, then B aborts. Otherwise B retrieves the value r_i from the tuple containing m on L_{H_1} and computes $\sigma = X_I^{r_i} \bmod p$. B outputs σ.

Conf/Deny Queries: E can request a confirmation or denial proof for input $\langle X_P, X_V, m, \sigma \rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. B keeps a list L_{CD} of all distinct **Conf/Deny** queries that E makes.

Case 1 If $X_P \neq X_J$ then C takes the private key x_P corresponding to X_P and proceeds as follows. C runs $\mathsf{ConfGen}(X_P, X_V, x_P, m, \sigma)$ to produce an NIDV proof $\pi_C \in \mathcal{P}_C$. If $\mathsf{ConfVerify}(X_P, X_V, m, \sigma, \pi_C)$ returns *accept*, then C outputs π_C. Otherwise C runs $\mathsf{DenyGen}(X_P, X_V, x_P, m, \sigma)$ to produce an NIDV proof $\pi_D \in \mathcal{P}_D$ which it outputs.

Case 2 If $X_P = X_J$ then B queries m on the H_1 oracle. If the tuple $\langle m, r_i \rangle$ on L_{H_1} containing m is not the kth tuple, B retrieves the value r_i from the tuple on L_{H_1} and computes $\sigma' = X_I^{r_i} \bmod p$.

Case 2a If $\sigma' = \sigma$ then B simulates $\mathsf{ConfGen}$ by simulating an NIDV EDL proof as follows. B picks random $w, r, t, h \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^d X_P^{(h+w)} \bmod p \\ D &= H_1(m)^d \sigma^{(h+w)} \bmod p \end{aligned}$$

If the H_2 oracle has previously been queried on input $c, G, D, m, \sigma, X_P, X_V$, then B

starts again by picking new w, r, t, h. Otherwise B sets $str = c, G, D, m, \sigma, X_P, X_V$, adds the tuple $\langle str, h\rangle$ to L_{H_2}, and outputs $\pi_C = \langle w, r, h, d\rangle$.

Case 2b If $X_P = X_J$, and $\sigma' \neq \sigma$ or $\langle m, r_i\rangle$ is the kth tuple on L_{H_1}, then B proceeds as follows.

If $\langle m, r_i\rangle$ is the kth tuple on L_{H_1}, then B selects some random value $y \in \mathbb{Z}_q^*$ and computes $\sigma' = g^{br_iy} = H_1(m)^y \bmod p$. If $\sigma' = \sigma$, then B starts again by picking a new y.

Now B has some value $\sigma' \neq \sigma$ (generated at the beginning of Case 2 or in Case 2b) and B simulates DenyGen by simulating an NIDV IDL proof as follows. B picks random $t, w, r, h, d_1, d_2 \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} C &= (\frac{\sigma'}{\sigma})^t \bmod p \\ c &= g^w X_V^r \bmod p \\ G &= g^{d_1} X_P^{-d_2} \bmod p \\ D &= C^{h+w} H_1(m)^{d_1} \sigma^{-d_2} \bmod p \end{aligned}$$

If the H_2 oracle has previously been queried on input $C, c, G, D, m, \sigma, X_P, X_V$, then B starts the DenyGen simulation again by picking new t, w, r, h, d_1, d_2. Otherwise B sets $str = C, c, G, D, m, \sigma, X_P, X_V$, adds the tuple $\langle str, h\rangle$ to L_{H_2}, and outputs $\pi_D = \langle C, w, r, h, d_1, d_2\rangle$.

A denial proof may be incorrect if $X_P = X_J$ and $\langle m, r_i\rangle$ is the kth tuple on L_{H_1}. However we deal with this possible inconsistency later in the proof.

FakeConf Queries: E can request a fake confirmation proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. If $X_V \neq X_J$ then B runs FakePGen of the NIDV EDL proof system on input $\langle X_P, X_V, x_V, m, \sigma\rangle$ to generate a fake NIDV EDL proof π'_C which B outputs. If $X_V = X_J$ then B simulates an NIDV EDL proof as follows. B picks random $w, r, t, h \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^d X_P^{(h+w)} \bmod p \\ D &= H_1(m)^d \sigma^{(h+w)} \bmod p \end{aligned}$$

If the H_2 oracle has previously been queried on input $c, G, D, m, \sigma, X_P, X_V$, then B starts again by picking new w, r, t, h. Otherwise B sets $str = c, G, D, m, \sigma, X_P, X_V$, adds the tuple $\langle str, h\rangle$ to L_{H_2}, and outputs $\pi'_C = \langle w, r, h, d\rangle$.

FakeDeny Queries: E can request a fake denial proof for input $\langle X_P, X_V, m, \sigma\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$, $m \in \mathcal{M}$ and $\sigma \in \mathcal{S}$. If $X_V \neq X_J$ then B runs FakePGen of the NIDV IDL proof system on input $\langle X_P, X_V, x_V, m, \sigma\rangle$ to generate a fake NIDV IDL proof π'_D which B outputs. If $X_P = X_J$ then B simulates an NIDV IDL proof as follows. B picks random $y, t, w, r, h, d_1, d_2 \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} C &= \left(\frac{H_1(m)^y}{\sigma}\right)^t \bmod p \\ c &= g^w X_V^r \bmod p \\ G &= g^{d_1} X_P^{-d_2} \bmod p \\ D &= C^{h+w} H_1(m)^{d_1} \sigma^{-d_2} \bmod p \end{aligned}$$

If $C = 1$ or H_2 has previously been queried on input $C, c, G, D, m, \sigma, X_P, X_V$, then B starts again by picking new y, t, w, r, h, d_1, d_2. Otherwise B sets $str = C, c, G, D, m, \sigma,$
X_P, X_V, adds the tuple $\langle str, h\rangle$ to L_{H_2}, and outputs $\pi'_D = \langle C, w, r, h, d_1, d_2\rangle$.

Corrupt Queries: E can request the private key corresponding to any public key $X_I \in \mathcal{X}$. If $X_I = X_J$, then B aborts and terminates E. Otherwise B returns the appropriate private key x_I.

Output: Finally E produces $X_I^* \in \mathcal{X}$, $m^* \in \mathcal{M}$ and $\sigma^* \in \mathcal{S}$, where X_I^* is uncorrupted and no **USign** query was previously made on $\langle X_I^*, m^*\rangle$. E wins the game if $(m^*, \sigma^*) \in L(X_I^*)$.

We know that E produces a valid forgery on some message m with probability at least η and in time at most τ. Unless E queries the random oracle H_1 on m at some point in the game, then E's advantage is negligible. Since E's probability of winning the game is non-negligible, we assume that E queries H_1 on m at some point during the game. Therefore the probability that E produces a valid forgery σ' for public key X_J and on message $m = m'$ where m' is in the kth tuple (say $\langle m', r\rangle$) on L_{H_1}, is at least $\eta/(\mu_1\rho)$.

From now on we assume that E has produced a valid forgery σ' for X_J on m' (this occurs with non-negligible probability at least $\eta/(\mu_1\rho)$ and in time τ).

If $\beta = 0$ then B outputs $(\sigma^*)^{r^{-1}}$ where the tuple $\langle m^* = m', r\rangle$ on L_{H_1} is the kth tuple. If $\beta = 1$ then B randomly picks a **Conf/Deny** query made by E of the form $\langle m', \sigma\rangle$ and outputs $\sigma^{r^{-1}}$ where $\langle m', r\rangle$ is the kth tuple on L_{H_1}. If no such **Conf/Deny** query was made by E, then B proceeds as if $\beta = 0$.

If at any stage in the game, E made a **Conf/Deny** query on input $\langle X_J, X_V, m', \sigma' \rangle$ for any $X_V \in \mathcal{X}$, we say that a critical **Conf/Deny** query occurred. If a critical **Conf/Deny** query occurred, then B would have responded incorrectly, and from this point onwards E's behaviour is undefined. If no critical **Conf/Deny** query occurred, then no inconsistency could have arisen in the way B answered the **Conf/Deny** queries, so E's view of the game would be as in a real game.

Case 1: We assume that no critical **Conf/Deny** query occurred, and therefore no inconsistency in the way B responded to **Conf/Deny** queries could have occurred. In this case, with probability at least $\eta/(\mu_1\rho)$, E's output would have been $\langle X_I^*, m^*, \sigma^* \rangle = \langle X_J, m', \sigma' \rangle$. In addition, E would not have made a **USign** query on X_J, m', and would not have made a **Corrupt** query on X_J, so B would have not have aborted during the simulation.

In this case, B solves the CDH problem if $\beta = 0$ and $\langle X_I^*, m^*, \sigma^* \rangle = \langle X_J, m', \sigma' \rangle$, which occurs with probability at least $\eta/(2\mu_1\rho)$ and in time τ.

Case 2: We assume that at some point a critical **Conf/Deny** query occurred, resulting in an inconsistency in the way B responded to **Conf/Deny** queries arose. From this point onwards E's behavior is undefined, and we can say nothing about whether E's output $\langle m^*, \sigma^* \rangle$ is valid or not. However we do know that in this case, the valid message signature pair $\langle m', \sigma' \rangle$ for X_J was queried on **Conf/Deny**.

In this case, B solves the CDH problem if $\beta = 1$, E has produced a valid forgery σ' for X_J on m', and the random **Conf/Deny** query $\langle m', \sigma \rangle$ is in fact the critical query $\langle m', \sigma' \rangle$ on X_J, which occurs with probability at least $\eta/(2\mu_1\rho\mu_c)$ and in time τ. The only problem is that if an inconsistency arises, then E may not terminate within time τ. Therefore if E does not terminate in time τ, then B terminates E and assumes that a critical query did occur (and so proceeds as if $\beta = 0$).

From cases 1 and 2, we can see that B solves the CDH problem in time τ with probability at least

$$\gamma = \min\{\eta/(2\mu_1\rho), \eta/(2\mu_1\rho\mu_c)\} = \eta/(2\mu_1\rho\mu_c).$$

This is non-negligible, contradicting the hardness of the CDH problem.

□

Theorem 4.8 The core signature scheme of Section 4.6.1 is *invisible* in the random oracle model assuming the hardness of the Decision Diffie-Hellman problem in G.

Proof: We suppose that H_1 and H_2 are random oracles, and suppose there exists an algorithm E in a game with at most ρ participants that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_s **USign** queries, and wins the invisibility game of Section 4.5.1.2 in time at most τ with probability at least η, where η is non-negligible in security parameter l.

We show how to construct an algorithm B that uses E to solve the Decision Diffie-Hellman problem. B will simulate the random oracles and the challenger C in a game with E. B's goal is to solve the Decision Diffie-Hellman problem on input $\langle g, g^a, g^b, g^c, p, q\rangle$, that is to decide whether $ab = c \bmod q$, where g is of prime order q modulo prime p. If $ab = c$ then B should output 1 otherwise B should output 0.

Simulation: B initializes the game as in the proof of Theorem 4.7, except that in this case B is not required to pick a random bit b and integer k.

Phase 1: B simulates the random oracles H_1 and H_2 and answers all **USign**, **Conf/Deny**, **FakeConf**, **FakeDeny** and **Corrupt** queries exactly as in the proof of Theorem 4.7. Once again, B may answer a **Conf/Deny** query incorrectly, but we deal with this possible inconsistency later in the proof.

Challenge: Finally, E outputs $\langle X_I^*, m^*\rangle$, where X_I^* is uncorrupted and E made no **USign** query on $\langle X_I^*, m^*\rangle$ in Phase 1. B queries m^* on H_1 and retrieves the tuple $\langle m^*, r\rangle$ on L_{H_1}. If $\langle m^*, r\rangle$ is not the kth tuple on L_{H_1}, or $X_I^* \neq X_J$, then B aborts. Otherwise B outputs $\sigma^* = (g^c)^r = g^{cr} \bmod p$.

Phase 2: E can continue to make **USign**, **Conf/Deny**, **FakeConf**, **FakeDeny** and **Corrupt** queries, and these are answered as in Phase 1. But E cannot make a **Conf/Deny** query on $\langle X_I^*, X_V, m^*, \sigma^*\rangle$ for any $X_V \in \mathcal{X}$, and E cannot make a **USign** query on $\langle X_I^*, m^*\rangle$.

Output: Finally E outputs a bit β.

B outputs β as its solution to the DDH problem on $\langle g, g^a, g^b, g^c, p, q\rangle$.

Unless E queries the random oracle H_1 on m^* at some point in the game, then E's advantage is negligible. Since E's probability of winning the game is non-negligible, we assume that E queries H_1 on m^* at some point during the game.

We note that if $ab = c \bmod q$ then σ^* is a valid signature for X_I^*, and if $ab \neq c \bmod q$, then σ^* is simply a random element from the signature space. Therefore B correctly

answers E's challenge.

We note that an inconsistency in B's simulation of the game could arise if B answers a **Conf/Deny** query on input $\langle X_I^*, X_V, m^*, \sigma \rangle$ where $X_I^* = X_J$, the tuple $\langle m^*, r \rangle$ is the kth tuple on L_{H_1}, and $(m^*, \sigma) \in L(X_I^*)$. In this case B responds incorrectly with a denial (NIDV IDL) proof. But in this case no previous **USign** query could have been made on $\langle X_I^*, m^* \rangle$, since otherwise B would have aborted. So if $(m^*, \sigma) \in L(X_I^*)$, then E has successfully forged a signature on m. But by Theorem 4.7, the NIDV undeniable signature scheme is unforgeable. Therefore such an inconsistency only arises with negligible probability.

The probability that B does not abort is therefore at least $1/(\rho\mu_1)$, so B solves the DDH problem in time τ and with probability $\eta/(\rho\mu_1)$ which is non-negligible, contradicting the hardness of the DDH problem. □

Theorem 4.9 The NIDV undeniable signature scheme of Section 4.6.1 is *secure*.

Proof: This result follows immediately from the security of the NIDV EDL (confirmation) proof and NIDV IDL (denial) proof with which the core signature scheme is composed, as well as Theorems 4.7 and 4.8. □

Alternative constructions for NIDV EDL and IDL proofs may be possible. For example, it may be the case that the techniques of [46] could yield more general constructions of such NIDV proofs, although we believe that such general constructions are unlikely to be more efficient than the concrete examples presented here.

4.8 DV Signatures

We present a formal definition for DV signature schemes. The definition is a mixture of the definition of NIDV proofs from Section 4.2 and the definition of two-party ring signature schemes from Section 3.3.

Definition 4.7 A DV signature scheme is defined via the following algorithms:

Setup(l)**:** A probabilistic algorithm which takes a security parameter l as input and returns the system parameters *params*. Amongst the public parameters are descriptions of the following spaces: a public key space $\mathcal{PK}$, a private key space $\mathcal{SK}$, a message space $\mathcal{M}$ and a signature space $\mathcal{S}$.

KeyGen($params$): A probabilistic algorithm which which takes as input the system parameters $params$ and outputs a public key $X \in \mathcal{PK}$ and a corresponding private key $x \in \mathcal{SK}$.

DVSign(X_S, X_V, x_S, m): A (possibly probabilistic) signature generation algorithm which takes as input the signer's public key $X_S \in \mathcal{PK}$, and corresponding private key $x_S \in \mathcal{SK}$, the designated verifier's public key $X_V \in \mathcal{PK}$, $X_V \neq X_S$, and a message $m \in \mathcal{M}$, and outputs a signature $\sigma \in \mathcal{S}$.

DVVerify(X_S, X_V, m, σ): A verification algorithm which takes as input the signer's public key $X_S \in \mathcal{PK}$, the designated verifier's public key $X_V \in \mathcal{PK}$, $X_S \neq X_V$, a signature $\sigma \in \mathcal{S}$ and a message $m \in \mathcal{M}$, and outputs either *accept* or *reject*.

We note that some definitions of DV signatures [77, 98] include an algorithm Simulate which describes how a designated verifier may simulate a DV signature produced by a signer. However, as for our definition of NIDV proofs, we require the existence of such an algorithm in the definition of non-transferability, and therefore consider this algorithm to be necessary for the scheme's security, rather than part of the scheme's formal definition.

4.9 Security for DV Signatures

As for the formal definition of a DV signature scheme, we derive the notions of security for DV signatures from the security notions of NIDV proofs presented in Section 4.3 and the security notions for ring signatures presented in Section 3.4. We say that a DV signature scheme is secure if it satisfies the notions of correctness, non-transferability and unforgeability. These are defined as follows.

4.9.1 Correctness

A DV signature scheme is *correct* if when DVSign is run on any input $\langle X_S, X_V, x_S, m\rangle$ and outputs a signature σ, then DVVerify on input $\langle X_S, X_V, m, \sigma\rangle$ outputs *accept*.

4.9.2 Non-transferability

We say that a DV signature scheme is *non-transferable* if there exists a polynomial time algorithm FakeDVSign that on input a tuple $\langle X_S, X_V, x_V, m\rangle$, where X_S is the public key of the signer, $X_V \neq X_S$ is the public key of the designated verifier, x_V is the private key of the designated verifier, and m is a message, produces a signature σ' such that

$\langle X_S, X_V, m, \sigma' \rangle$ is accepted by DVVerify and the distributions of signatures σ' generated by FakeDVSign are polynomially indistinguishable from those of signatures σ produced by DVSign when run on input $\langle X_S, X_V, x_S, m \rangle$, even if the private keys x_S and x_V are known.

4.9.3 Unforgeability

Unforgeability of a DV signature scheme is defined via the following game between a challenger C and an adversary E:

Initialize: C firstly runs Setup for a given security parameter l to obtain the public parameters *params*. C runs KeyGen to generate the public and private keys X_i and x_i for each participant, where the number of participants is bounded by n, where n is a polynomial function of l. We define the set of all participants' public keys to be $\mathcal{X}$. E is given *params* and $\mathcal{X}$ while C retains the private keys.

E can make the following queries to the challenger C:

DVSign Queries: E can request a DV signature on input $\langle X_S, X_V, m \rangle$ where $X_S, X_V \in \mathcal{X}$, $X_V \neq X_S$, and $m \in \mathcal{M}$. C runs DVSign(X_S, X_V, x_S, m) to produce a signature σ which it outputs.

FakeDVSign Queries: E can request a fake DV signature on input $\langle X_S, X_V, m \rangle$ where $X_S, X_V \in \mathcal{X}$, $X_V \neq X_S$, and $m \in \mathcal{M}$. C runs FakeDVSign as defined in the non-transferability game of Section 4.9.2 on input $\langle X_S, X_V, x_V, m \rangle$ to produce a signature σ' which it outputs.

Corrupt Queries: E can request the private key corresponding to any public key $X_i \in \mathcal{X}$. C outputs the corresponding private key x_i.

Output: Finally E outputs $\langle X_S^*, X_V^*, m^*, \sigma^* \rangle$, where $X_S^*, X_V^* \in \mathcal{X}$, $X_V^* \neq X_S^*$, $m^* \in \mathcal{M}$ and $\sigma^* \in \mathcal{S}$. E wins the game if $\langle X_S^*, X_V^*, m^*, \sigma^* \rangle$ is accepted by DVVerify, no **DVSign** query was made on $\langle X_S^*, X_V^*, m^* \rangle$, no **FakeDVSign** query was made on $\langle X_S^*, X_V^*, m^* \rangle$, and neither X_S^* nor X_V^* have been corrupted.

Definition 4.8 We say that a DV signature scheme is *unforgeable* if the probability of success of any polynomially bounded adversary in the above game is negligible (as a function of the security parameter l).

4.9.4 Notes on the Security Definitions for DV Signatures

Non-transferability: As we mentioned earlier, the algorithm FakeDVSign corresponds to the Simulate algorithm that other authors choose to include as part of the formal definition of DV signatures. We choose to define such an algorithm as part of the security definition of non-transferability. We note that our definition of non-transferability does not restrict a distinguisher's access to the private keys x_S and x_V, so the signatures should be indistinguishable even with knowledge of the private keys.

Non-transferability of DV signatures is similar to the anonymity property of ring signatures although there are some subtle differences. The definition of ring signatures automatically implies that all ring members can create valid ring signatures, and anonymity simply guarantees that these signatures are indistinguishable, hence concealing the identity of the true signer. However the definition of a DV signature scheme does not guarantee the existence of a FakeDVSign algorithm. Instead, this is guaranteed by the definition of non-transferability. In addition, non-transferability guarantees indistinguishability between signatures created using FakeDVSign and signatures created using DVSign.

FakeDVSign queries: The existence of FakeDVSign from Section 4.9.2 enables the challenger in the game of unforgeability to answer **FakeDVSign** queries. We consider it important to model such queries for NIDV undeniable signatures (as in the case of NIDV proofs) since an adversary may have access to such "fake" DV signatures that are produced by dishonest verifiers using FakeDVSign.

4.9.5 Comparison to Other Work

The only other work that contains formal security definitions for DV signatures is by Lipmaa *et al.* [77]. Therefore we compare our security model to theirs.

Lipmaa *et al.* also define the notions of correctness, non-transferability and unforgeability. These definitions are similar to ours although their models are not fully multiparty since they only model a single signer interacting with a single designated verifier. By contrast, we model a group of interacting participants where each participant can assume the role of signer or designated verifier at different times.

As in many papers on DV signatures, Lipmaa *et al.* include a Simulate algorithm as part of their definition. Their definition of non-transferability is therefore only concerned with

indistinguishability of signatures generated by DVSign and Simulate. However their model of unforgeability does grant the adversary access to signatures generated by Simulate, whereas we permit the adversary to make **FakeDVSign** queries.

Lipmaa *et al.* go on to discuss two other security properties for DV signatures, which they call *non-delegatability* and *non-disavowability*.

Informally, a DV signature is delegatable if a signer S can, without disclosing his private key x_S, delegate her signing rights to another entity. For example, S may be able to disclose some function $f(x_S)$ from which it is infeasible to determine x_S, but with which an entity can create valid DV signatures for S. This is a valid concern for all signature schemes since if a signature is delegatable, a verifier can no longer be sure that the signature was created by the real signer. However it is unclear how one would model such an attack. Lipmaa *et al.* propose a definition for non-delegatability which involves black-box knowledge extractors, although they do not clarify exactly what access the adversary has to private keys (or functions of private keys). Moreover, in their proof of non-delegatability for a concrete scheme, the adversary appears to have the same oracle access and the same objective as in the unforgeability game, implying that unforgeability is in fact equivalent to non-delegatability. We do not consider non-delegatability in our model of security.

Lipmaa *et al.* call a DV signature scheme non-disavowable if neither the signer nor the designated verifier can prove to a third party (even if they cooperate) which of them really created a DV signature. Our definition of non-transferability implies non-disavowability since we require that signatures created by DVSign and FakeDVSign are indistinguishable, even if the private keys of the signer and designated verifier are known.

4.10 Concrete DV Signature Schemes

4.10.1 DV Signatures from Ring Signatures

As mentioned earlier, a DV signature scheme may be constructed from a 2-party ring signature scheme. We refer the reader back to Chapter 3 where we defined ring signature schemes and their security. We assume that we have a non-separable ring signature scheme (e.g. the concrete scheme presented in Section 3.5) which has explicit Setup and KeyGen algorithms.

Theorem 4.10 A secure DV signature scheme may be constructed from a secure non-separable 2-party ring signature scheme.

Proof: We provide a sketch of the proof of this theorem. We first show how a DV signature scheme can be constructed from a secure 2-party ring signature scheme. We then discuss how the security of the ring signature scheme ensures that the resulting DV signature scheme is also secure.

The DV signature Setup and KeyGen algorithms are identical to those of the ring signature scheme. Suppose we have a signer S and a designated verifier V with public and private key pairs $\langle X_S, x_S \rangle$ and $\langle X_V, x_V \rangle$ respectively. We relabel the key pairs of S and V as $\langle X_1, x_1 \rangle$ and $\langle X_2, x_2 \rangle$ respectively, and set the ring $R = \{X_1, X_2\}$. Now DVSign and DVVerify are defined as follows.

- $\mathsf{DVSign}(X_S, X_V, x_S, m)$ is defined by running $\mathsf{RingSign}(m, R, 1, x_1)$, and
- $\mathsf{DVVerify}(X_S, X_V, m, \sigma)$ is defined by running $\mathsf{RingVerify}(m, R, \sigma)$.

It is easy to see that the correctness of the ring signature scheme implies correctness of the DV signature scheme. In order to show that the DV signature scheme is non-transferable, we must define a suitable FakeDVSign. This may be done by letting FakeDVSign be defined as running $\mathsf{RingSign}(m, R, 2, x_2)$. It is clear that signatures generated by FakeDVSign will be accepted by DVVerify. It is now easy to verify that the anonymity of the underlying ring signature scheme ensures that the DV signature scheme is non-transferable. It remains to show that the DV scheme is unforgeable.

We notice that the unforgeability games for ring signatures and DV signatures are identical except for the **FakeDVSign** queries in the game for DV signatures. However since **FakeDVSign** queries are answered by running FakeDVSign, such queries correspond exactly to **RingSign** queries for R with $sig = 2$ which are allowed in the unforgeability game for ring signatures. Therefore all queries that an adversary in the DV signature unforgeability game could ask correspond to queries in the ring signature unforgeability game, so the unforgeability of the ring signature scheme implies unforgeability of the DV signature scheme.

□

We note that it is also possible to construct a secure DV signature scheme from a separable 2-party ring signature scheme, and in this case the DV signature scheme is also separable.

4.10.2 DV Signatures from NIDV Undeniable Signatures

It is also possible to construct a DV signature scheme from an NIDV undeniable signature scheme. In this case, the DV signature Setup and KeyGen algorithms are identical to those of the NIDV undeniable signature scheme. Suppose we have a signer S and a designated verifier V with public and private key pairs $\langle X_S, x_S \rangle$ and $\langle X_V, x_V \rangle$ respectively. Now DVSign and DVVerify are defined as follows.

- $\mathsf{DVSign}(X_S, X_V, x_S, m)$ is defined by running $\mathsf{USign}(x_S, m)$ to produce an undeniable signature σ, and then $\mathsf{ConfGen}(X_S, X_V, x_S, m, \sigma)$ to produce an NIDV proof π. The output is $\langle \sigma, \pi \rangle$
- $\mathsf{DVVerify}(X_S, X_V, m, \langle \sigma, \pi \rangle)$ is defined by running $\mathsf{ConfVerify}(X_S, X_V, m, \sigma, \pi)$.

As in Section 4.10.1, correctness is easy to verify. It can also be shown that the DV signature scheme is unforgeable if the NIDV undeniable signature scheme is secure. However the DV signature scheme *does not* satisfy the definition of non-transferability. This is because, given a DV signature $\langle \sigma, \pi \rangle$ output by the scheme defined above, σ is an undeniable signature for the signer with public key X_S, so the signer can distinguish this from a random element in the signature space.

We could weaken the definition of non-transferability for DV signatures to say that the distributions of signatures produced by FakeDVSign should be polynomially indistinguishable from the distributions of signatures produced by DVSign *without knowledge of the signer's private key.* In this case, by letting FakeDVSign be defined by running the FakePGen algorithm corresponding to ConfGen, it is possible to show that the DV signature scheme defined above satisfies the weaker notion of non-transferability.

4.10.3 A Concrete DV Signature Scheme from an NIDV EDL Proof

Instead of trying to construct a DV signature scheme directly from an NIDV undeniable signature scheme, it may be possible to modify the NIDV proof system itself to obtain a DV signature scheme that is fully secure.

As an example, we now show how the NIDV EDL proof of Section 4.6.3 can be converted into a DV signature scheme. The DV signature scheme is defined by the following algorithms.

$\mathsf{Setup}(l)$: For some security parameter l, let p and q be large primes, where $q|(p-1)$. Let G be the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q and let g be a generator of

G. We also assume that $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a cryptographic hash function. We set $\mathcal{PK} = G$, $\mathcal{SK} = \mathbb{Z}_q^*$, $\mathcal{M} = \{0,1\}^*$ and $\mathcal{S} = \mathbb{Z}_q^4$. The public parameters *params* include $\langle p, q, g, H \rangle$ as well as the descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}$ and $\mathcal{S}$.

KeyGen(*params*)**:** To set up a user I's public and private keys, the private key x_I is chosen at random from $\mathbb{Z}_q^*$, and the public key is $X_I = g^{x_I} \bmod p$.

DVSign(X_P, X_V, x_P, m)**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_P \in \mathcal{SK}$, and $m \in \mathcal{M}$, the algorithm picks random $w, r, t \in \mathbb{Z}_q$ and computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^t \bmod p \\ h &= H_2(c, G, m, X_P, X_V) \\ d &= t - x_P(h + w) \bmod q \end{aligned}$$

The algorithm outputs $\sigma = \langle w, r, h, d \rangle$.

DVVerify(X_P, X_V, m, σ)**:** On input $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, message $m \in \mathcal{M}$, and signature $\sigma = \langle w, r, h, d \rangle \in \mathcal{S}$, the algorithm computes:

$$\begin{aligned} c &= g^w X_V^r \bmod p \\ G &= g^d X_P^{(h+w)} \bmod p \end{aligned}$$

and verifies that $h = H_2(c, G, m, X_P, X_V)$. If the last equation holds, then the algorithm outputs *accept*, otherwise it outputs *reject*.

4.10.4 Security of our Concrete DV Signature Scheme

We now present some security results for the concrete DV signature scheme. Most of the security results follow from the security results for the NIDV EDL proof in Section 4.7.1 with some minor modifications.

Theorem 4.11 The DV signature scheme of Section 4.10.3 is *correct*.

Proof: It is trivial to verify that signatures generated by DVSign will be accepted by DVVerify.

□

Theorem 4.12 The DV signature scheme of Section 4.10.3 is *non-transferable*.

Proof: FakeDVSign is defined in an almost identical way to the way NIDV EDL FakePGen is defined in the proof of Theorem 4.2. On input $\langle X_P, X_V, x_V, m\rangle$, where $X_P, X_V \in \mathcal{PK}$, $X_V \neq X_P$, $x_V \in \mathcal{SK}$, $m \in \mathcal{M}$, FakeDVSign produces a $\sigma' \in \mathcal{S}$ as follows.

FakeDVSign chooses random $d', \alpha', \beta' \in \mathbb{Z}_q$ and calculates:

$$\begin{aligned} c' &= g^{\alpha'} \bmod p \\ G' &= g^{d'} X_P^{-\beta'} \bmod p \\ h' &= H_2(c', G', m, X_P, X_V) \\ w' &= \beta' - h' \bmod q \\ r' &= (\alpha' - w')x_V^{-1} \bmod q \end{aligned}$$

FakeDVSign outputs $\sigma' = \langle w', r', h', d'\rangle$. As in the proof of Theorem 4.2, it is easy to verify that that $\langle X_P, X_V, m, \sigma'\rangle$ will be accepted by DVVerify and that σ' is indistinguishable from any σ produced by running DVSign on input $\langle X_P, X_V, x_P, m\rangle$. □

In order to analyze the unforgeability of our DV signature scheme, we first need to introduce a related generic signature scheme, which we call the GDV signature scheme. Our concrete GDV signature scheme is defined as follows.

- The Setup and KeyGen algorithms are identical to those of the concrete DV signature scheme.
- The Sign algorithm for a public key $X \in \mathcal{PK}$ takes as input a message $M = m, X_P, X_V$ where $m \in \mathcal{M}$, $X_V \in \mathcal{PK}$, $X_P = X$ or $X_V = X$, and the private key $x \in \mathcal{SK}$ corresponding to X. If $X_P = X$ then the algorithm runs in an identical way to DVSign(X, X_V, x, m) to produce a DV signature $\sigma = \langle w, r, h, d\rangle$. If $X_V = X$ then the algorithm runs in an identical way to FakeDVSign(X_P, X, x, m) to produce a DV signature $\sigma = \langle w, r, h, d\rangle$. The algorithm sets $r_2 = \langle w, r, d\rangle$ and $r_1 = g^w X_V^r \bmod p, g^d X_P^{h+w} \bmod p$, and outputs $\sigma_G = \langle r_1, h, r_2\rangle$.
- The Verify algorithm on input $M = m, \sigma, X_P, X_V$, where $X_P = X$ or $X_V = X$, and a signature $\sigma_G = \langle r_1, h, r_2\rangle$ where $r_2 = \langle w, r, d\rangle$, sets $\sigma = \langle w, r, h, d\rangle$ and runs in an identical way to DVVerify$(X_P, X_V, x_P, m, \sigma)$.

Theorem 4.13 The DV signature scheme of Section 4.10.3 is *unforgeable* in the random oracle model assuming the hardness of the discrete logarithm problem in G.

Proof:

We suppose that H_2 is a random oracle and there exists a polynomial time algorithm E that makes at most μ_2 queries to the random oracle H_2, at most μ_s **PGen** and **FakePGen** queries, and wins the unforgeability game of Section 4.9.3 in time at most τ with non-negligible probability η' (in security parameter l) where the number of participants is bounded by ρ and $\eta' > 5\rho(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

As in the proof of Theorem 4.3, we divide the proof into two steps. In Step 1, we show how E can be used to construct an algorithm B that makes at most μ_2 queries to the random oracle H_2, at most μ_s **Sign** queries to its challenger C, and wins the unforgeability game of Section 2.3.1 for the GDV signature scheme in time at most τ with non-negligible probability $\eta = 2\eta'/\rho > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

In Step 2, we then replace C with an algorithm C' that uses B to solve the discrete logarithm problem in G. Step 2 of the proof will make use of Lemma 2.5, the Forking Lemma.

Step 1 We will show that there exists an algorithm B that uses E to forge a GDV signature with non-negligible probability when interacting with a challenger C in the unforgeability game of Section 2.3.1.

The challenger C initializes the unforgeability game for B and gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}, \mathcal{P}$, and access to the random oracle H_2.

B can make $\boldsymbol{H_2}$ as well as **Sign** queries on any message $M = m, \sigma, X_P, X_V$ (where $m \in \mathcal{M}$, $\sigma' \in \mathcal{S}$, $X_P, X_V \in \mathcal{PK}$ and $X_P = X$ or $X_V = X$). B must eventually output a message M^* and a GDV signature $\sigma_G^* = \langle r_1^*, h^*, r_2^*\rangle$. B wins the game if $\langle M^*, \sigma_G^*\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*.

In order to win the above game, B in turn simulates an DV signature unforgeability game for E.

DV Signature Unforgeability Simulation:

B gives the parameters $\langle g, p, q\rangle$ and the descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}$ to E. B generates a set of participants U, where $|U| = \rho(l)$ and ρ is a polynomial function of the security parameter l. For some random participant J, B sets $X_J = X$, and for each $I \neq J$, B runs KeyGen to generate a private key x_I and public key X_I. We define the set of all participants' public keys to be $\mathcal{X}$. E is given $\mathcal{X}$.

B now simulates the challenger by simulating all the queries which E can make as follows:

H_2-Queries: E can query any string str on the H_2 oracle. B simulates the H_2 oracle by passing all H_2 queries to C and passing C's response back to E.

DVSign Queries: E can request a DV signature for input $\langle X_P, X_V, m\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$ and $m \in \mathcal{M}$.

If $X_P = X_J$ then B sets $M = m, X_P, X_V$ and makes a **Sign** query to C on M. C responds with a generic signature $\sigma_G = \langle r_1, h, r_2\rangle$ where $r_2 = \langle w, r, d\rangle$. B sets $\sigma = \langle w, r, h, d\rangle$ and outputs σ to E.

If $X_P \neq X_J$ then B runs $\mathsf{DVSign}(X_P, X_V, x_P, m)$ to produce a DV signature σ which B outputs to E.

FakeDVSign Queries: E can request a fake DV signature on input $\langle X_P, X_V, m\rangle$ where $X_P, X_V \in \mathcal{X}$, $X_V \neq X_P$ and $m \in \mathcal{M}$.

If $X_V = X_J$ then B sets $M = m, X_P, X_V$ and makes a **Sign** query to C on M. C responds with a generic signature $\sigma_G = \langle r_1, h, r_2\rangle$ where $r_2 = \langle w, r, d\rangle$. B sets $\sigma = \langle w, r, h, d\rangle$ and outputs σ to E.

If $X_V \neq X_J$ then B runs $\mathsf{FakeDVSign}(X_P, X_V, x_V, m)$ to produce a DV signature σ which B outputs to E.

Corrupt Queries: E can request the private key corresponding to any public key $X_I \in \mathcal{X}$. If $X_I = X_J$, then B aborts and terminates E. Otherwise B returns the appropriate private key x_I.

Output: On termination, E outputs $\langle X_P^*, X_V^*, m^*, \sigma^*\rangle$, where $X_P^*, X_V^* \in \mathcal{X}$, $X_V^* \neq X_P^*$, $m^* \in \mathcal{M}$, $\sigma^* \in \mathcal{S}$, and X_P^* and X_V^* are uncorrupted. E wins if $\langle X_P^*, X_V^*, m^*, \sigma^*\rangle$ is accepted by $\mathsf{DVVerify}$, no **DVSign** query was made on $\langle X_P^*, X_V^*, m^*\rangle$ and no **FakeDVSign** query was made on $\langle X_P^*, X_V^*, m^*\rangle$.

B takes E's output σ^* where $\sigma^* = \langle w^*, r^*, h^*, d^*\rangle$, and sets $M^* = m^*, X_P^*, X_V^*$ and $\sigma_G^* = \langle r_1^*, h^*, r_2^*\rangle$ where

$$r_1^* = g^{w^*} {X_V^*}^{r^*} \bmod p, g^{d^*} {X_P^*}^{h^*+w^*} \bmod p$$

and $r_2^* = \langle w^*, r^*, d^*\rangle$. B outputs M^* and σ_G^* to C.

With probability at least $2/\rho$ one of the public keys X_P^* or X_V^* is equal to X_J and no **Corrupt** query was made on X_J. In this case, B did not abort, and if E wins the DV signature unforgeability game, then B wins the generic signature unforgeability game. So

B wins the generic signature unforgeability game, where X_P^* or X_V^* equals X_J, and X_J is uncorrupted, with non-negligible probability $\eta \geq 2\eta'/\rho$ where $\eta \geq 10(\mu_s+1)(\mu_s+\mu_2)/2^l$ and in time τ, with at most μ_2 queries to the random oracle H_2 and μ_s **Sign** queries.

Step 2 We now replace B's challenger C with an algorithm C' that uses B to solve the discrete logarithm problem in G. C' will simulate the random oracle H_2 and the challenger in an unforgeability game for the GDV signature scheme with B. C''s goal is to solve the discrete logarithm problem on input $\langle g, X, p, q\rangle$, that is to find $x \in \mathbb{Z}_q$ such that $g^x = X \bmod p$, where g is of prime order q modulo prime p and generates group G.

Generic Signature Unforgeability Simulation:

The challenger C' initializes the unforgeability game for B as follows. C' gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{S}$, and access to the random oracle H_2.

C' now simulates the challenger by simulating all the queries which B can make as follows:

***H_2*-Queries:** B can query the random oracle H_2 at any time. C' simulates the random oracle by keeping a list L_{H_2} of tuples $\langle str_i, r_i\rangle$. When the oracle is queried with an input $str \in \{0,1\}^*$, C' responds as follows:

1. If the string str is already in L_{H_2} in the tuple $\langle str = str_i, r_i\rangle$, then B outputs r_i.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs r and adds $\langle str, r\rangle$ to L_{H_2}.

Sign Queries: C' will also answer B's **Sign** queries on any message $M = m, X_P, X_V$ where $X_P = X$ or $X_V = X$. C' picks random $w, r, t, h \in \mathbb{Z}_q$ and computes

$$r_1 = g^w X_V^r \bmod p, g^d X_P^{h+w} \bmod p.$$

C' constructs the string $str = r_1, M$, and adds the tuple $\langle str, h\rangle$ to L_{H_2}. C' then sets $r_2 = \langle w, r, d\rangle$ and $\sigma_G = \langle r_1, h, r_2\rangle$ which it outputs to B.

Output: Finally B should output a message $M^* = m^*, X_P^*, X_V^*$ where $X_P^* = X$ or $X_V^* = X$ and a GDV signature $\sigma_G^* = \langle r_1^*, h^*, r_2^*\rangle$. B wins the game if $\langle M^*, \sigma_G^*\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*.

Since H_2 is a random oracle, C' can simulate GDV signatures that are indistinguishable from true GDV signatures, so B can detect no inconsistency in the game.

From Step 1, we know that B that makes at most μ_2 queries to the random oracle H_2, at most μ_s **Sign** queries, and wins the above game in time at most τ with non-negligible probability $\eta = 2\eta'/\rho > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

By Lemma 2.5 of Section 2.3.4 (the Forking Lemma), C' can rewind B (and therefore also E) with the same random coins and repeat its simulation with a different random oracle H_2 so that B outputs another GDV signature $\sigma_G = \langle r_1^*, h, r_2 \rangle$ on $M^* = m^*, X_P^*, X_V^*$, where $h \neq h^*$, $r_2 = \langle w, r, d \rangle$ and

$$r_1^* = g^w {X_V^*}^r \bmod p, g^d {X_P^*}^{h+w} \bmod p.$$

We therefore obtain the equations

$$g^{w^*} {X_V^*}^{r^*} = g^w {X_V^*}^r \bmod p \quad (4.7)$$

$$g^{d^*} {X_P^*}^{(h^*+w^*)} = g^d {X_P^*}^{(h+w)} \bmod p \quad (4.8)$$

Now if $r^* \neq r$ and $X_V^* = X$ then C' can solve (4.7) for the discrete logarithm of $X_V^* = X$. The probability that $X_V^* = X$ is 1/2 since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X_J$ or $X_P^* = X_J$. If $r^* = r$, then $w^* = w$, and since $h^* \neq h$ we have that $h^* + w^* \neq h + w$. So if $X_P^* = X$ then C' can solve (4.8) for the discrete logarithm of $X_P^* = X$. The probability that $X_P^* = X$ is 1/2 since $X_J = X$ was chosen randomly from $\mathcal{X}$ and we know that $X_V^* = X_J$ or $X_P^* = X_J$.

Since the Forking Lemma produces a second appropriate signature with expected time at most $\tau' = 120686\mu_s\tau$, we find that C' can solve the discrete logarithm problem in time at most τ' and with probability at least $\eta/2$. If η' is non-negligible, then η is also non-negligible, and this contradicts the hardness of the discrete logarithm problem. Therefore no such polynomially bounded adversary B can have non-negligible probability of winning the generic signature unforgeability game, and in turn no polynomially bounded adversary E can have non-negligible probability of winning the unforgeability game of Section 4.9.3.

□

4.11 Conclusions and Open Problems

We have presented models of security for NIDV proof systems, NIDV undeniable signatures and DV signatures and argued that NIDV proofs can have applications outside of the context of undeniable signatures such as in deniable proofs of knowledge or possession. We then repaired the original NIDV undeniable signature scheme of [70], producing a

concrete scheme that is efficient and proven secure. We then showed how secure DV signature schemes can be constructed from secure 2-party ring signature schemes, and gave an example of how a secure DV signature scheme can be constructed from an NIDV proof, using our concrete NIDV EDL proof as an example.

In future work, it would be interesting to investigate how to extend our model to include strong designated verifier proofs [98, 109, 70]. It would also be interesting to provide models of security for NIDV versions of confirmer signatures [22, 36, 47, 27] and other signature schemes closely related to undeniable signatures.

Chapter 5

Concurrent Signatures

5.1 Introduction

The problem of fair exchange of signatures is a fundamental and well-studied problem in cryptography, with potential application in a wide range of scenarios in which the parties involved are mutually distrustful. Ideally, we would like the exchange of signatures to be done in a *fair* way, so that by engaging in a protocol, either each party obtains the other's signature, or neither party does. It should not be possible for one party to terminate the protocol at some stage leaving the other party committed when they themselves are not.

The literature contains essentially two different approaches to solving the problem of fair exchange of signatures.

Early work on solving the problem was based on the idea of timed release or timed fair exchange of signatures [21, 53, 58]. Here, the two parties sign their respective messages and exchange their signatures "little-by-little" using a protocol. Typically, such protocols are highly interactive with many message flows. Moreover, one party, say B, may often be at an advantage in that he sometimes has (at least) one more bit of A's signature than she has of B's. This may not be a significant issue if the computing power of the two parties are roughly equivalent. But if B has superior computing resources, this may put him at a significant advantage since he may terminate the protocol early and use his resources to compute the remainder of A's signature, while it may be infeasible for A to do the same. Even if the fairness of such protocols could be guaranteed, they may still be too interactive for many applications. See [57] for further details and references for such protocols.

An alternative approach to solving the problem of fair exchange of signatures involves the use of a (semi-trusted) third party or arbitrator T who can be called upon to handle disputes between signers. The idea is that A registers her public key with T in a one-

time registration, and thereafter may perform many fair exchanges with other entities. To take part in a fair exchange with B, A creates a partial signature which she sends to B. Entity B can be convinced that the partial signature is valid (perhaps via a protocol interaction with A) and that T can extract a full, binding signature from the partial signature. However, the partial signature on its own is not binding for A. B then fulfils his commitment by sending A his signature, and if valid, A releases the full version of her signature to B. The protocol is fair since if B does not sign, then A's partial signature is worthless to B, and if B does sign but A refuses to release her full signature then B can obtain it from T. The third party is only required in case of dispute; for this reason, protocols of this type are commonly referred to as optimistic fair exchange protocols. See [4, 5, 19, 6, 28, 52, 56, 94] for further details of such schemes.

The main problem with such an approach is the requirement for a dispute-resolving third party with functions beyond those required of a normal Certification Authority. In general, appropriate third parties may not be available.

It is our belief that the *full* power of fair exchange is not necessary in many application scenarios. We therefore introduce a somewhat weaker concept, which we name *concurrent signatures*. The cost of concurrent signatures is that they do not provide the full security guarantees of a fair exchange protocol. Their benefit is that they have none of the disadvantages of previous solutions: they do not require a special trusted third party[1], and they do not rely on a computational balance between the parties. Moreover, our concrete realization is computationally and bandwidth efficient. Informally, concurrent signatures appear to be as close to fair exchange as it's possible to get whilst staying truly practical and not relying on special third parties.

We introduce the notion of *concurrent signatures* and *concurrent signature protocols*. In a concurrent signature protocol, two parties A and B interact without the help of a third party to sign (possibly identical) messages m_A and m_B in such a way that both A and B become publicly committed to their respective messages at the same moment in time (i.e. concurrently). This moment is determined by one of the parties through the release of an extra piece of information k which we call a *keystone*. Before the keystone's release, neither party is publicly committed through their signatures, while after this point, both are. In fact, from a third party's point of view, before the keystone is released, either party could have been responsible for producing both signatures, so these initial signatures are non-transferable (i.e. not binding for the signer).

[1] Our concurrent signatures will still require a conventional CA for the distribution of public keys, but not a trusted third party with any other special functions.

We note that the party who controls the keystone k has a degree of extra power: it controls the timing of the keystone release, to whom it is released and indeed whether the keystone is released at all. Upon receipt of B's signature σ_B, A might privately show σ_B and k to a third party C and gain some advantage from doing so. This is the main feature that distinguishes concurrent signatures from fair exchange schemes. In a fair exchange scheme, each signer A should either have recourse to a third party to release the other party B's signature or be assured that the B cannot compute A's signature significantly more easily than A can compute B's. With concurrent signatures, only when A releases the keystone do both signatures become simultaneously binding (i.e. both signers become publicly committed by their signatures), and there is no guarantee that A will do so. However, in the real world, there are often existing mechanisms that can naturally be used to guarantee that B will receive the keystone should his signature be used. These existing mechanisms can provide a more natural dispute resolution process than reliance on a special trusted party. We argue that concurrent signatures are suited to any fair exchange application where:

- There is no sense in A withholding the keystone because she needs it to obtain a service from B. For example, suppose B sells computers. A signs a payment instruction to pay B the price of a computer, and B signs that he authorizes her to pick one up from the depot (B's signature may be thought of as a receipt). Now A can withhold the keystone, but as soon as she tries to pick up her computer, the depot will ask for a copy of B's signature authorizing her to collect one. In this way B can obtain the keystone which validates A's payment instruction. In this example, the application itself forces the delivery of the keystone to B.

- There is no possibility of A keeping B's signature private in the long term. For example, consider the routine "four corner" credit card payment model. Here C may be A's acquiring bank, and B's signature may represent a payment to A that A must channel via C to obtain payment. Bank C would then communicate with B's issuing bank D to obtain payment against B's signature and D could ensure that B's signature, complete with keystone, reaches B (perhaps via a credit card statement). As soon as B has the keystone, A becomes bound to her signature. In this application, the back-end banking system provides a mechanism by which keystones would reach B if A were to withhold them.

- There is a single third party C who verifies both A and B's signature. Now, if A tries to present B's signature along with k to C whilst withholding k from B, B will be

able to present A's signature to C and have it verified. As an application, consider the (perhaps somewhat artificial) scenario where A and B are two politicians from different parties who want to form a coalition to jointly release a piece of information M to the press C in such a way that neither of them is identified as being the sole signatory to the release. Concurrent signatures seem just right for this task. In this case A and B both produce initial (non-transferable) signatures on the same message M. Here the keystone is not necessarily returned to B, but it does reach the third party C to whom B wishes to show A's signature.

We also consider an example where concurrent signatures provide a novel solution to an old problem: that of fair tendering of contracts (our signatures can also be used in a similar way in auction applications). Suppose that A has a bridge-building contract that she wishes to put out to tender, and suppose companies B and C wish to put in proposals to win the contract and build the bridge. This process is sometimes open to abuse by A since she can privately show B's signed proposal to C to enable C to better B's proposal. Using concurrent signatures, B would sign his proposal to build the bridge for an amount X, but keep the keystone private. If A wishes to accept the proposal, she returns a payment instruction to pay B amount X. She knows that if B attempts to collect the payment, then A will obtain the keystone through the banking system. But A may also wish to examine C's proposal before deciding which to accept. However there is no advantage for A to show B's signature to C since at this point B's signature is non-transferable and so C will not be convinced of anything at all by seeing it. After all, A may have created the signature herself in an attempt to get a better proposal from C. We see that the tendering process is therefore immune to abuse of this kind by A. We note that this example makes use of the non-transferability of our signatures prior to the keystone release, and although the solution can be realized by using standard fair exchange protocols, such protocols do not appear to previously have been suggested for this purpose.

Our schemes are not abuse-free in the sense of [6, 56], since the party A who holds the keystone can always determine whether to complete or abort the exchange of signatures, and can demonstrate this by showing an outside party C the signature from B with the keystone before revealing the keystone to B. However the above example shows that abuse can be addressed by our schemes in certain applications.

5.1.1 Technical Approach

We briefly explain how concurrent signature schemes can be built using (two-party) ring signature schemes, which were defined in Section 3.3.

A ring signature has the property that it could have been produced by either of two parties. The *anonymity* property of the underlying ring signature scheme also ensures that it is infeasible to determine which of the two possible signers created the signature. These properties allow both possible signers to deny having produced a specific NT signature. However, we note that if A creates an NT signature which either A or B could have created, and sends this to B, then B is convinced of the authorship of the signature since he knows that he did not create it himself. However B cannot transfer this conviction and prove A's involvement to a third party since he could have created the signature himself. The same situation applies when the roles of A and B are reversed.

When generating a two-party ring signature, a party A will usually choose some random bits f to combine with a party B's public key. A will then use her private key to complete the signature. Now, if the value f was not chosen randomly but rather was generated from some preimage k, then A can demonstrate authorship of the signature by revealing the preimage k of f. The ring signature alone is not binding for A, but the ring signature together with the preimage k constitutes a binding signature for A. We use this concept to construct concurrent signatures.

We begin by taking a two-party ring signature scheme of a specific form (i.e. one where randomness is chosen for combination the public keys of each non-signer) and using it to construct what we call a *non-transferable signature scheme* (NT signature scheme). NT signature schemes are almost identical to ring signature schemes, and are formally defined as part of a concurrent signature scheme in the sequel.

The general idea is that a party A generates an NT signature σ_A, using B's public key and a preimage k to generate the value f. The value k is called the *keystone* and is kept private by A. A sends σ_A to party B.

Party B can verify that A created the signature σ_A but cannot demonstrate this to a third party. Now B generates his own NT signature σ_B using A's public key and the same value f, and sends σ_B to A.

Now A can verify that B generated σ_B, but as long as k remains secret, neither party can demonstrate authorship to a third party. However if A releases the keystone k, then any third party can be convinced of the authorship of both signatures, since both

signatures use the value f for which k is a preimage. Thus the pairs $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ amount to a simultaneously binding pair of signatures on A and B's messages. We call these pairs *concurrent* signatures.

We note that Rivest *et al.* in their pioneering work on ring signatures [97] considered the situation in which an anonymous signer A wants to have the option of later proving her authorship of a ring signature. Their solution was to choose the bits h_B pseudo-randomly and later to reveal the seed used to generate h_B. Here we use the same trick for a new purpose: to ensure that either both or neither of the parties can be identified as signers of messages.

An NT signature scheme is almost identical to a two-party ring signature scheme. The main difference is that an NT signature scheme needs to be of a specific structure in order to accommodate a keystone, and the random bits h_B are taken as input to the signature algorithm instead of being chosen randomly within the algorithm.

We point out that any suitable NT signature scheme can be used to produce a concurrent signature protocol. We base our concrete scheme on the non-separable ring signature scheme of Section 3.5, although alternate concurrent signature schemes may be constructed from other suitable ring signature schemes such as the short ring signature scheme of [19]. It may also be possible to construct a concurrent signature scheme from designated verifier signatures presented in Chapter 4. However these are often not of the correct form unless constructed from a 2-party ring signature scheme.

We give generic definitions of concurrent signature schemes and concurrent signature protocols, define a suitably powerful multi-party adversarial model for this setting, and give a formal definition of what it means for such schemes and protocols to be secure.

5.1.2 Published Work

An earlier version of this work appears in [41] and forms the basis for this chapter. However the nomenclature in this chapter differs slightly from the published work. In particular, what we refer to as non-transferable (NT) signatures in this chapter are referred to as *ambiguous signatures* in [41]. This is to avoid confusion between NT signatures and the security property of anonymity for ring signature schemes. We rename the function that transforms a keystone k into a suitable h value. In [41] this function was called KGen, but to avoid confusion with the key generation algorithm KeyGen, we now refer to this function as $\mathsf{KCommit}$. In [41], the outputs of the function KGen were called keystone footprints. Here we rename refer to the outputs of $\mathsf{KCommit}$ as keystone footprints.

5.2 Formal Definitions

5.2.1 Concurrent Signature Algorithms

We now give a more formal definition of concurrent signature schemes and the protocol for exchanging concurrent signatures.

Definition 5.1 A concurrent signature scheme is a digital signature scheme comprised of the following algorithms:

Setup**:** A probabilistic algorithm that on input a security parameter l, outputs the public parameters including descriptions of: the public key space $\mathcal{PK}$, the private key space $\mathcal{SK}$, the message space $\mathcal{M}$, the signature space $\mathcal{S}$, the keystone space $\mathcal{K}$, the keystone footprint space $\mathcal{F}$, and a function $\mathsf{KCommit} : \mathcal{K} \rightarrow \mathcal{F}$.

KeyGen**:** A probabilistic algorithm which takes as input the public parameters and outputs a public key $X \in \mathcal{PK}$ and a corresponding private key $x \in \mathcal{SK}$.

NTSign**:** A probabilistic algorithm that on inputs $\langle X_i, X_j, x_i, h_j, m\rangle$, where $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$, $x_i \in \mathcal{SK}$ is the private key corresponding to X_i, $h_j \in \mathcal{F}$, and $m \in \mathcal{M}$, outputs a signature $\sigma = \langle s, h_i, h_j\rangle$ on m, where $s \in \mathcal{S}$, $h_i, h_j \in \mathcal{F}$.

NTVerify**:** An algorithm which takes as input $S = \langle \sigma, X_i, X_j, m\rangle$, where $\sigma = \langle s, h_i, h_j\rangle$, $s \in \mathcal{S}$, $h_i, h_j \in \mathcal{F}$, $X_i, X_j \in \mathcal{PK}$, and $m \in \mathcal{M}$, outputs *accept* or *reject*.

CSVerify**:** An algorithm which takes as input $\langle k, kpos, S\rangle$ where $k \in \mathcal{K}$ is a keystone, $kpos \in \{1, 2\}$, and S is of the form $S = \langle \sigma, X_i, X_j, m\rangle$, where $\sigma = \langle s, h_i, h_j\rangle$ with $s \in \mathcal{S}$, $h_i, h_j \in \mathcal{F}$, $X_i, X_j \in \mathcal{PK}$, and $m \in \mathcal{M}$. If $kpos = 1$ then the algorithm checks if $\mathsf{KCommit}(k) = h_i$, and if not, it terminates with output *reject*. If $kpos = 2$ then the algorithm checks if $\mathsf{KCommit}(k) = h_j$, and if not, it terminates with output *reject*. The algorithm then runs $\mathsf{NTVerify}(S)$ (in which case the output of CSVerify is just that of NTVerify).

We call a signature σ that is output by NTSign an *NT signature*, and if $\mathsf{NTVerify}(\sigma, X_i, X_j, m)$ returns *accept*, then we say that σ is a *valid NT signature* on m for X_i and X_j.

An NT signature σ on message m for X_i and X_j, together with a keystone k is called a *concurrent signature*. The value *kpos* determines the position of the keystone footprint within the NT signature. Therefore if $\mathsf{CSVerify}(k, 2, S = \langle \sigma, X_i, X_j, m\rangle)$ where

$\sigma = \langle s, h_i, h_j \rangle$ returns *accept*, then we say that the pair $\langle k, \sigma \rangle$ is a *valid concurrent signature* on m for X_i. Similarly, if $\mathsf{CSVerify}(k, 1, S = \langle \sigma, X_i, X_j, m \rangle)$ where $\sigma = \langle s, h_i, h_j \rangle$ returns *accept*, then we say that the pair $\langle k, \sigma \rangle$ is a valid concurrent signature on m for X_j.

5.2.2 Concurrent Signature Protocol

We now describe a concurrent signature protocol between two parties A and B. Since one party needs to create the keystone and send the first NT signature, we call this party the *initial signer*. A party who responds to this initial signature by creating another NT signature with the same keystone footprint we call a *matching signer*. Without loss of generality, we assume A to be the initial signer, and B the matching signer. From here on, we will use subscripts A and B to indicate initial signer A and matching signer B. The signature protocol works as follows:

We assume that Setup has been run to determine the public parameters, and A and B have run KeyGen to determine their public and private keys. We assume that A's public and private keys are X_A and x_A, and B's public and private keys are X_B and x_B.

Step 1: A picks a random keystone $k \in \mathcal{K}$, and computes $f = \mathsf{KCommit}(k)$. A picks a message $m_A \in \mathcal{M}$ to sign and then computes her NT signature as

$$\sigma_A = \langle s_A, h_A, f \rangle = \mathsf{NTSign}(X_A, X_B, x_A, f, m_A),$$

and sends this to B.

Step 2: Upon receiving A's NT signature σ_A, B verifies the signature by checking that $\mathsf{NTVerify}(\langle s_A, h_A, f \rangle, X_A, X_B, m_A)$ returns *accept*. If not then B aborts, otherwise B picks a message $m_B \in \mathcal{M}$ to sign and computes his NT signature as

$$\sigma_B = \langle s_B, h_B, f \rangle = \mathsf{NTSign}(X_B, X_A, x_B, f, m_B)$$

and sends this back to A. Note that B uses the same value f in his signature as A did to produce σ_A.

Step 3: Upon receiving B's signature σ_B, A verifies that $\mathsf{NTVerify}(\langle s_B, h_B, f \rangle, X_B, X_A, m_B)$ returns *accept*, where f is the same keystone footprint as A used in Step 1. If not then A aborts, otherwise A sends keystone k to B.

Note that inputs $\langle k, 2, S_A\rangle$ and $\langle k, 2, S_B\rangle$ will now both be accepted by $\mathsf{CSVerify}$, where $S_A = \langle\langle s_A, h_A, f\rangle, X_A, X_B, m_A\rangle$ and $S_B = \langle\langle s_B, h_B, f\rangle, X_B, X_A, m_B\rangle$.

By following the concurrent signature protocol, the value *kpos* will only ever be set to 2. However there may be other ways to produce a valid NT signature, where the keystone corresponds to the first h value, and these signatures should still be considered valid. For example, if the NT signature scheme is constructed from a ring signature scheme, then a signer may choose to reverse the order of the public keys and h values, and could produce a concurrent signature where the value *kpos* is 1.

5.3 Formal Security Model

Concurrent signatures naturally involve more than one party, and any party may interact with many other parties and in different roles. Our security model is therefore multiparty and assumes a system with a number of different participants that is polynomial in the security parameter l.

We say that a concurrent signature scheme is secure if it satisfies the notions of correctness, non-transferability, unforgeability and fairness. These are defined as follows.

5.3.1 Correctness

Definition 5.2 We say that a concurrent signature scheme is *correct* if the following conditions hold.

If $\sigma = \langle s, h_i, f\rangle = \mathsf{NTSign}(X_i, X_j, x_i, f, m)$, and $S = \langle\sigma, X_i, X_j, m\rangle$, then $\mathsf{NTVerify}(S)$ returns *accept*. Moreover, if $\mathsf{KCommit}(k) = f$ for some $k \in \mathcal{K}$, then $\mathsf{CSVerify}(k, 2, S)$ returns *accept*.

5.3.2 Non-transferability

We say that a concurrent signature scheme is *non-transferable* if there exists a polynomial time algorithm $\mathsf{FakeNTSign}$ that on input tuples $\langle X_i, X_j, x_j, M\rangle$, where $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$, $x_j \in \mathcal{SK}$ is the private key corresponding to X_j, and $M \in \mathcal{M}$, outputs NT signatures $\sigma' = \langle s', h'_i, h'_j\rangle$ such that $\langle\sigma, X_i, X_j, M\rangle$ is accepted by $\mathsf{NTVerify}$ and the distribution of σ' is polynomially indistinguishable from that of signatures σ produced by NTSign when run on inputs $\langle X_i, X_j, x_i, f, M\rangle$ where $f = \mathsf{KCommit}(k)$ for some random $k \in \mathcal{K}$.

5.3.3 Unforgeability

We give a formal definition of existential unforgeability of a concurrent signature scheme under a chosen message attack in the multi-party setting. To do this, we extend the definition of existential unforgeability against a chosen message attack of [63] to the multi-party setting. Our extension is strong enough to capture an adversary who can simulate and observe concurrent signature protocol runs between any pair of participants. It is defined using the following game between an adversary E and a challenger C.

Initialization: C runs Setup for a given security parameter l to obtain the public parameters and the descriptions of $\mathcal{PK}$, $\mathcal{SK}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{K}$, $\mathcal{F}$, and $\mathsf{KCommit} : \mathcal{K} \rightarrow \mathcal{F}$. C also generates the public and private keys X_i and x_i for each participant, where the number of participants is polynomial in l. E is given the public parameters and the set of public keys $\{X_i\}$. C retains the set of private keys $\{x_i\}$.

E can make the following types of query to the challenger C:

KCommit Queries: E can request that C select a keystone $k \in \mathcal{K}$ and return the keystone footprint $f = \mathsf{KCommit}(k)$. If E wishes to choose his own keystone, then he can compute his own keystone footprint using algorithm $\mathsf{KCommit}$ directly.

KReveal Queries: E can request that C reveal the keystone k that was used to produce a keystone footprint $f \in \mathcal{F}$ in a previous **KCommit** query. If f was not a previous **KCommit** output then C outputs *invalid*, otherwise C outputs k where $f = \mathsf{KCommit}(k)$.

NTSign Queries: E can request an NT signature for any input of the form $\langle X_i, X_j, h_j, m\rangle$ where $h_j \in \mathcal{F}$, $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$ and $m \in \mathcal{M}$. C responds with an NT signature $\sigma = \langle s, h_i, h_j\rangle = \mathsf{NTSign}(X_i, X_j, x_i, h_j, m)$. Note that using **NTSign** queries in conjunction with **KCommit** queries, E can obtain concurrent signatures $\langle k, \sigma\rangle$ for messages and pairs of users of his choice.

FakeNTSign Queries: E can request an NT signature for any input of the form $\langle X_i, X_j, m\rangle$ where $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$ and $m \in \mathcal{M}$. C responds with an NT signature $\sigma = \langle s, h_i, h_j\rangle = \mathsf{FakeNTSign}(X_i, X_j, x_j, m)$.

Corrupt Queries: E can request the private key corresponding to the public key of any participant with public key X_i. In response, C outputs the corresponding private key x_i.

Output: Finally E outputs a tuple $\sigma = \langle s, h_c, f \rangle$ where $s \in \mathcal{S}$, $h_c, f \in \mathcal{F}$, along with public keys $X_c, X_d \in \mathcal{PK}$, $X_d \neq X_c$ and a message $m \in \mathcal{M}$. The adversary wins the game if $\mathsf{NTVerify}(\langle s, h_c, f \rangle, X_c, X_d, m)$ returns *accept*, no **NTSign** query was made on $\langle X_c, X_d, f', m \rangle$ for any $f' \in \mathcal{F}$, no **FakeNTSign** query was made on $\langle X_c, X_d, m \rangle$, no **Corrupt** query was made on X_c, and if one of the following two cases hold:

1. No **Corrupt** query was made on X_d, or
2. Either f was a previous output from a **KCommit** query or E also outputs a keystone k such that $f = \mathsf{KCommit}(k)$.

Definition 5.3 We say that a concurrent signature scheme is *unforgeable* if the probability of success of any polynomially bounded adversary in the above game is negligible (as a function of the security parameter l).

Case 1 of the output conditions in the above game models forgery of an NT signature in the situation where the adversary does not have knowledge of either of the respective private keys. This condition is required so that the matching signer B is convinced that A's NT signature originated from A. Case 2 models forgery in the situation where the adversary knows one of the private keys and so applies to the situation where one of the two parties attempts to cheat the other. More specifically, it covers attacks where an initial signer forges a concurrent signature of a matching signer, or where a matching signer has access to an initial signer's NT signature and keystone footprint (but not the actual keystone) and forges a concurrent signature of the initial signer.

The challenger in the unforgeability game is able to answer **FakeNTSign** queries using algorithm FakeNTSign from Section 5.3.2. We consider it important to model such queries (as in the unforgeability game for DV signatures in Section 4.9.3) since an adversary may have access to such "faked" NT signatures that are produced by dishonest entities using FakeNTSign.

5.3.4 Fairness

We require the concurrent signature scheme and protocol to be fair for both an initial signer A, and a matching signer B. This concept is defined via the following game between an adversary E and a challenger C:

Initialization: This is as before in the unforgeability game of Section 5.3.3.

KCommit, KReveal, NTSign, FakeNTSign and Corrupt Queries: These queries are answered by C as in the unforgeability game of Section 5.3.3.

Output: Finally E outputs a keystone $k \in \mathcal{K}$, and $S = \langle \sigma, X_c, X_d, m \rangle$ where $\sigma = \langle s, h_c, f \rangle$, $s \in \mathcal{S}$, $h_c, f \in \mathcal{F}$, $X_c, X_d \in \mathcal{PK}$, $X_d \neq X_c$ and $m \in \mathcal{M}$, where $\langle k, 2, S \rangle$ is accepted by $\mathsf{CSVerify}$. The adversary wins the game if either of the following cases holds:

1. f was a previous output from a **KCommit** query and no **KReveal** query on input f was made, or
2. E also produces $S' = \langle \sigma', X_d, X_c, m' \rangle$, with $\sigma' = \langle s', h'_c, f \rangle$, $s' \in \mathcal{S}$, $h'_c, f \in \mathcal{F}$, and message $m' \in \mathcal{M}$, where $\mathsf{NTVerify}(S')$ returns *accept*, but $\langle k, 2, S' \rangle$ is not accepted by $\mathsf{CSVerify}$.

Definition 5.4 We say that a concurrent signature scheme is *fair* if any polynomially bounded adversary's probability of success in the above game is negligible.

Our definition of fairness formalizes our intuitive understanding of fairness for A in the protocol of Section 5.2.2 (in case 1 of the output conditions), since it guarantees that only the entity who generates a keystone can reveal it, thereby converting valid NT signatures into binding concurrent signatures. It also captures fairness for B (in case 2 of the output conditions), since it guarantees that all valid NT signatures produced using the same keystone footprint will all become binding. Thus B cannot be left in a position where a keystone binds his NT signature to him while A's initial NT signature is not also bound to A. However we note that our definition does not guarantee that B will ever receive the necessary keystone.

5.4 A Concrete Concurrent Signature Scheme

We present a concrete concurrent signature scheme which is based on the 2-party version of the ring signature scheme presented in Section 3.5. The scheme is defined via the following algorithms:

Setup: For some security parameter l, let p and q be large primes, where $q|(p-1)$. Let G be a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q and let g be a generator of G. Two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ are also selected and we define $\mathsf{KCommit}$ to be H_1. The public parameters are $params = \langle p, q, g, H_1, H_2 \rangle$ as well as descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{S}, \mathcal{F}, \mathcal{M}, \mathcal{K}$ which are defined as follows: $\mathcal{PK} = G$, $\mathcal{SK} = \mathbb{Z}_q^*$, $\mathcal{S} = \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} = \mathcal{K} = \{0,1\}^*$.

KeyGen: This algorithm takes as input the public parameters and selects a private key x_i at random from $\mathbb{Z}_q^*$, and the corresponding public key is computed as $X_i = g^{x_i} \bmod p$.

NTSign: This algorithm takes as input $\langle X_i, X_j, x_i, h_j, m\rangle$, where $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$, $x_i \in \mathbb{Z}_q$ is the private key corresponding to X_i, $h_j \in \mathcal{F}$ and $m \in \mathcal{M}$. The algorithm picks a random value $t \in \mathbb{Z}_q$ and then computes the values:

$$\begin{aligned} h &= H_2(X_i \| X_j \| m \| g^t {X_j}^{h_j} \bmod p), \\ h_i &= h - h_j \bmod q, \\ s &= t - x_i h_i \bmod q. \end{aligned}$$

The algorithm outputs $\sigma = \langle s, h_i, h_j\rangle$.

NTVerify: This algorithm takes as input $\langle \sigma, X_i, X_j, m\rangle$ where $\sigma = \langle s, h_i, h_j\rangle$, $s \in \mathcal{S}$, $h_i, h_j \in \mathcal{F}$, $X_i, X_j \in \mathcal{PK}$, and $m \in \mathcal{M}$. The algorithm checks that the equation

$$h_i + h_j \bmod q = H_2(X_i \| X_j \| m \| g^s X_i^{h_i} X_j^{h_j} \bmod p)$$

holds, and if so, outputs *accept*. Otherwise, it outputs *reject*.

CSVerify: This algorithm is defined in terms of KCommit and NTVerify, as described in Section 5.2.1.

The NTSign algorithm is almost identical to the RingSign algorithm of Section 3.5 on input $\langle m, R = \{X_i, X_j\}, i, x_i\rangle$ except that h_j is taken as an input parameter instead of being chosen randomly within the signature algorithm. The NTVerify algorithm is identical to the RingVerify algorithm of Section 3.5. We require that $X_j \neq X_i$ since otherwise the signature would be a standard Schnorr signature [99] and would not be non-transferable.

A concrete concurrent signature protocol can be derived directly from the algorithms defined above and the generic protocol described in Section 5.2.2.

5.5 Security of the Concrete Concurrent Signature Scheme

We now present some security results for the concrete scheme of Section 5.4.

Theorem 5.1 The concurrent signature scheme of Section 5.4 is *correct*.

Proof: This is trivial to verify, and we leave the details to the reader. □

Theorem 5.2 The concurrent signature scheme of Section 5.4 is *non-transferable* in the random oracle model.

Proof: We let the algorithm FakeNTSign be defined by running algorithm RingSign of Section 3.5 on input $\langle M, R = \{X_i, X_j\}, j, x_j \rangle$.

We recall that this algorithm selects random $t', h'_i \in \mathbb{Z}_q$ and computes

$$\begin{array}{rcl} z' & = & g^{t'} X_i^{h'_i} \bmod p \\ h' & = & H(X_i \| X_j \| M \| z') \\ h'_j & = & h' - h'_i \bmod q \\ s' & = & t' - x_j h'_j \bmod q \end{array}$$

The signature is $\sigma' = \langle s', h'_i, h'_j \rangle$. The correctness of the underlying ring signature scheme guarantees that FakeNTSign outputs an NT signature $\sigma' = \langle s', h'_i, h'_j \rangle$ such that $\langle \sigma', X_i, X_j, m \rangle$ is accepted by NTVerify.

The only difference between the NTSign and FakeNTSign algorithms is that NTSign takes the value f as input, whereas FakeNTSign selects the value h'_j randomly from $\mathbb{Z}_q$. The anonymity property of the underlying ring signature scheme therefore guarantee that the distribution of σ' is polynomially indistinguishable from that of signatures $\sigma = \langle s, h_i, f \rangle$ produced by $\mathsf{NTSign}(X_i, X_j, x_i, f, m)$ where $f = \mathsf{KCommit}(k)$ for some random $k \in \mathcal{K}$ as long as h'_j and f are indistinguishable. Since f is output by KCommit, which we model as a random oracle, f is independent of s and h_i, and is distributed uniformly at random in $\mathbb{Z}_q$. The algorithm RingSign chooses h'_j at random from $\mathbb{Z}_q$, so h'_j is also independent of s' and h'_i, and is distributed uniformly at random in $\mathbb{Z}_q$. Therefore and f and h'_j, and therefore also σ and σ', are indistinguishable as required. □

In order to analyze the unforgeability of our concurrent signature scheme, we first need to introduce a related non-generic (NG) signature scheme. We recall that generic signature schemes are simply digital signature schemes (as defined in Section 2.2.3) that take a certain form (which is described in Section 2.3.4). In addition, we recall from Section 2.3.5 that NG signature schemes are identical to generic signature schemes except that instead of generating signatures on a message M, NG signature schemes take an additional value T as input when generating or verifying a signature. Security for NG signature schemes is defined in the same way as security for digital signature schemes (in Section 2.3.1) except that the model is adapted to accommodate the additional value T.

Our concrete NG signature scheme, which we refer to as the NG CS signature scheme, is defined as follows.

- The Setup and KeyGen algorithms are identical to those of the concrete NIDV EDL

scheme except that the keystone space $\mathcal{K}$ and the function $\mathsf{KCommit}$ are not required.

- The Sign algorithm for a public key X takes as input a message $M = X_i, X_j, m$ and a private key $x_i \in \mathcal{SK}$, where $X_i, X_j \in \mathcal{PK}$, $X_i = X$ or $X_j = X$, and $m \in \mathcal{M}$, a value $T \in \mathcal{F}$.

  If $X_i = X$ then the algorithm runs in an identical way to $\mathsf{NTSign}(X_i, X_j, x_i, T, m)$ to produce a signature $\sigma = \langle s, h_i, T\rangle$. The algorithm sets $r_2 = \langle s, T\rangle$, $h = h_i + T$, $r_1 = g^s X_i^{h_i} X_j^T \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2\rangle$.

  If $X_j = X$ then the algorithm runs in an identical way to $\mathsf{FakeNTSign}(X_i, X_j, x_j, T, m)$ except that the value h_i' is set to be T instead of being chosen randomly from $\mathbb{Z}_q$. This produces a signature $\sigma = \langle s', T, h_j'\rangle$. The algorithm sets $r_2 = \langle s', h_j'\rangle$, $h = T + h_j$, $r_1 = g^s X_i^T X_j^{h_j} \bmod p$, and outputs $\sigma_{NG} = \langle r_1, h, r_2\rangle$.

- The Verify algorithm on input $M = X_i, X_j, m$, where $X_i = X$ or $X_j = X$, and a signature $\sigma_{NG} = \langle r_1, h, r_2\rangle$ where $r_2 = \langle s, h_j\rangle$, sets $\sigma = \langle s, h - h_j, h_j\rangle$ and runs in an identical way to $\mathsf{NTVerify}(\sigma, X_i, X_j, m)$.

Theorem 5.3 The concurrent signature scheme of Section 5.4 is *unforgeable* in the random oracle model, assuming the hardness of the discrete logarithm problem.

Proof:

We suppose that H_1 and H_2 are random oracles and there exists a polynomial time algorithm E that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_n **NTSignGen** and μ_f **FakeNTSign** queries, and wins the unforgeability game of Section 5.3.3 in time at most τ with non-negligible probability η' (in security parameter l) where the number of participants is bounded by ρ and $\eta' > 10\rho(\mu_s + 1)(\mu_s + \mu_2)/2^l$ where $\mu_s = \mu_n + \mu_f$.

In Step 1 of the proof, we show how E can be used to construct an algorithm B that makes at most μ_2 queries to the random oracle H_2, at most μ_s **Sign** queries to its challenger C, and wins the NG version of the unforgeability game of Section 2.3.1 for the NG CS signature scheme in time at most τ with non-negligible probability $\eta = \eta'/\rho$ where $\eta > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

In Step 2 of the proof, we then replace C with an algorithm C' that uses B to solve the discrete logarithm problem in G. Step 2 of the proof will make use of Lemma 2.6, the NG Forking Lemma.

Step 1 We will show that there exists an algorithm B that uses E to forge an NG CS signature with non-negligible probability when interacting with a challenger C in the NG version of the unforgeability game of Section 2.3.1.

The challenger C initializes the NG unforgeability game for B and gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{S}, \mathcal{F}, \mathcal{M}$, and access to the random oracle H_2.

B can make $\boldsymbol{H_2}$ as well as **Sign** queries on any message $M = m, \sigma, X_i, X_j$ (where $m \in \mathcal{M}$, $\sigma' \in \mathcal{S}$, $X_i, X_j \in \mathcal{PK}$, and $X_i = X$ or $X_j = X$), and a value $T \in \mathcal{F}$. B must eventually output a message M^*, a value T^*, and an NG CS signature $\sigma^*_{NG} = \langle r^*_1, h^*, r^*_2\rangle$. B wins the game if $\langle M^*, \sigma^*_{NG}\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*, T^*.

In order to win the above game, B in turn simulates a concurrent signature unforgeability game for E. *CS Unforgeability Simulation:*

B gives the parameters $\langle g, p, q\rangle$ and the descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{S}, \mathcal{F}, \mathcal{M}$ as well as a description of $\mathcal{K}$=$\{0,1\}^*$ to E. B generates a set of participants U, where $|U| = \rho(l)$ and ρ is a polynomial function of the security parameter l. B sets the public key of some randomly selected participant to be $X_\alpha = X$, and for each $i \neq \alpha$, B runs KeyGen to generate a private key x_i and public key X_i. We define the set of all participants' public keys to be $\mathcal{X}$. E is given $\mathcal{X}$.

B now simulates the challenger by simulating all the queries which E can make as follows:

$\boldsymbol{H_1}$-Queries: E can query the random oracle H_1 at any time. B simulates the random oracle by keeping a list L_{H_1} of tuples $\langle str_i, r_i\rangle$. When the oracle is queried with an input $str \in \{0,1\}^*$, B responds as follows:

1. If the string str is already in L_{H_1} in the tuple $\langle str = str_i, r_i\rangle$, then B outputs $g^{r_i} \bmod p$.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs $g^r \bmod p$ and adds $\langle str, r\rangle$ to L_{H_1}.

$\boldsymbol{H_2}$-Queries: E can query any string str on the H_2 oracle. B simulates the H_2 oracle by passing all H_2 queries to C and passing C's response back to E.

KCommit Queries: E can request that the challenger select a keystone $k \in \mathcal{K}$ and return a keystone footprint $f = H_1(k)$. B maintains a list L_K of tuples $\langle k, f\rangle$, and

answers queries by choosing a random keystone $k \in \mathcal{K}$ and computing $f = H_1(k)$. B outputs f and adds the tuple $\langle k, f \rangle$ to L_K. Note that L_K is a sublist of H_1-List, but is required in order to answer **KReveal** queries.

KReveal Queries: E can request the keystone of any keystone footprint $f \in \mathcal{F}$ produced by a previous **KCommit** Query. If there exists a tuple $\langle k, f \rangle$ on L_K, then B returns k, otherwise it outputs *invalid*.

NTSign Queries: E can request an NT signature for input $\langle X_i, X_j, h_j, m \rangle$ where $h_j \in \mathcal{F}$, $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$, and $m \in \mathcal{M}$.

If $X_i = X_\alpha$ then B sets $M = X_i, X_j, m$ and $T = h_j$ and makes a **Sign** query to C on M and T. C responds with an NG signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $r_2 = \langle s, T \rangle$. B sets $\sigma = \langle s, h - T, T \rangle$ and outputs σ to E.

If $X_i \neq X_\alpha$ then B runs $\mathsf{NTSign}(X_i, X_j, x_i, T, m)$ to produce an NT signature σ which it outputs to E.

FakeNTSign Queries: E can request a fake NT signature on input $\langle X_i, X_j, m \rangle$ where $X_i, X_j \in \mathcal{PK}$, $X_j \neq X_i$, and $m \in \mathcal{M}$.

If $X_j = X_\alpha$ then B selects a random value $h_j \in \mathcal{F}$, sets $M = X_i, X_j, m$ and $T = h_j$ and makes a **Sign** query to C on M and T. C responds with an NG signature $\sigma_{NG} = \langle r_1, h, r_2 \rangle$ where $r_2 = \langle s, T \rangle$. B sets $\sigma = \langle s, h - T, T \rangle$ and outputs σ to E.

If $X_j \neq X_\alpha$ then B runs $\mathsf{FakeNTSign}(X_i, X_j, x_j, m)$ to produce an NT signature σ which it outputs to E.

Corrupt Queries: E can request the private key for any public key X_i. If $X_i = X_\alpha$, then B terminates the simulation. Otherwise B returns the appropriate private key x_i.

Output: Finally, with non-negligible probability, E outputs a signature $\sigma = \langle s, h_c, f \rangle$ where $s \in \mathcal{S}$, $h_c, f \in \mathcal{F}$, along with public keys $X_c, X_d \in \mathcal{PK}$, $X_d \neq X_c$, and a message $m \in \mathcal{M}$, where $\mathsf{NTVerify}(\langle s, h_c, f \rangle, X_c, X_d, m)$ returns *accept*, σ was not output by any **NTSign** query on $\langle X_c, X_d, f, m \rangle$ or **FakeNTSign** query on $\langle X_c, X_d, m \rangle$, no **Corrupt** query was made on X_c, and one of the following two cases holds:

1. No **Corrupt** query was made on X_d, or
2. Either f was a previous output from a **KCommit** query or E also outputs a keystone k such that $f = \mathsf{KCommit}(k)$.

B takes E's output σ where $\sigma = \langle s, h_c, f\rangle$, and sets $M^* = X_c, X_d, m$, $T^* = f$, $h^* = h_c + f$ and $\sigma^*_{NG} = \langle r^*_1, h^*, r^*_2\rangle$ where $r^*_1 = g^s X_c^{h_c} X_d^f \bmod p$ and $r^*_2 = \langle s, f\rangle$. B outputs M^*, T^* and σ^*_{NG} to C.

Since E never made any **NTSign** queries on $\langle X_c, X_d, f, m\rangle$ or **FakeNTSign** queries on $\langle X_c, X_d, m\rangle$, B never made any **Sign** queries on $M^* = X_c, X_d, m$ and $T^* = f$.

The probability that B does not have to abort, E wins the game, and that either $X_V = X_J$, or $X_P = X_J$, is η'/ρ, which is non-negligible in security parameter l if η is. In this case, B would not have had to abort, and B wins the NG unforgeability game with probability $\eta = \eta'/\rho$ making at most μ_2 queries to the random oracle H_2, and at most $\mu_s = \mu_n + \mu_f$ **Sign** queries to C.

Step 2 We now define an algorithm C' that replaces B's challenger C and uses B to solve the discrete logarithm problem in G. C' will simulate the random oracle H_2 and the challenger in an NG unforgeability game with B. C''s goal is to solve the discrete logarithm problem on input $\langle g, X, p, q\rangle$, that is to find $x \in \mathbb{Z}_q$ such that $g^x = X \bmod p$, where g is of prime order q modulo prime p and generates group G.

NG Unforgeability Simulation:

The challenger C' initializes the NG unforgeability game for B as follows. C' gives B the public key X, the public parameters $\langle p, q, g\rangle$, descriptions of the spaces $\mathcal{PK}, \mathcal{SK}, \mathcal{S}, \mathcal{F}, \mathcal{M}$, and access to the random oracle H_2.

C' now simulates the challenger in an NG unforgeability game by simulating all the queries which B can make as follows:

***H_2*-Queries:** B can query the random oracle H_2 at any time. C' simulates the random oracle by keeping a list L_{H_2} of tuples $\langle str_i, r_i\rangle$. When the oracle is queried with an input $str \in \{0,1\}^*$, C' responds as follows:

1. If the string str is already in L_{H_2} in the tuple $\langle str = str_i, r_i\rangle$, then B outputs r_i.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs r and adds $\langle str, r\rangle$ to L_{H_2}.

Sign Queries: C' will also answer B's **Sign** queries on any messages $M = X_i, X_j, m$ where $X_i = X$ or $X_j = X$, and values $T \in \mathcal{F}$. C' picks random $s, h \in \mathbb{Z}_q$ and computes

$$r_1 = g^s X_i^{h-T} X_j^T \bmod p.$$

C' constructs the string $str = M, r_1$, and adds the tuple $\langle str, h\rangle$ to L_{H_2}. C' then sets $r_2 = \langle s, T\rangle$ and $\sigma_{NG} = \langle r_1, h, r_2\rangle$ which it outputs to B.

Output: Finally B should output a message M^* a value T^* and an NG CS signature $\sigma^*_{NG} = \langle r^*_1, h^*, r^*_2\rangle$. B wins the game if $\langle M^*, \sigma^*_{NG}\rangle$ is accepted by Verify and no **Sign** query was previously made on M^*, T^*.

Since H_2 is a random oracle, C' can simulate NG CS signatures that are indistinguishable from true NG CS signatures, so B can detect no inconsistency in the game.

From Step 1, we know that B that makes at most μ_2 queries to the random oracle H_2, at most $\mu_s = \mu_n + \mu_f$ **Sign** queries, and wins the above game in time at most τ with non-negligible probability $\eta = \eta'/\rho > 10(\mu_s + 1)(\mu_s + \mu_2)/2^l$.

By Lemma 2.6 of Section 2.3.5, (the NG Forking Lemma), C' can rewind B (and therefore also E) with the same random coins and repeat its simulation with a different random oracle H_2 so that B outputs another NG CS signature $\sigma_{NG} = \langle r^*_1, h, r_2\rangle$ on $M^* = X_c, X_d, m$, together with a value T, where $h \neq h^*$, $r_2 = \langle s', T\rangle$ and $r^*_1 = g^{s'} X_c^{h-T} X_d^T \bmod p$.

This means that B in fact obtained two valid NT signatures $\sigma = \langle s, h_c, f\rangle$ and $\sigma' = \langle s', h'_c, f'\rangle$ from E where the set of public keys and message for each signature are identical and where $h'_c + f' \neq h_c + f \bmod q$ and

$$g^{s'} {X_c}^{h'_c} {X_d}^{f'} = g^s {X_c}^{h_c} {X_d}^f \bmod p. \tag{5.1}$$

We now need to distinguish between two cases, depending on whether E wins the CS unforgeability game (simulated by B) by satisfying output condition 1 or 2. If E wins the game by satisfying condition 1, then we say that E is a Type-1 adversary, otherwise we say that E is a Type-2 adversary.

Case 1. We suppose that E is a Type-1 adversary. In this case, X_c and X_d were both uncorrupted in the NIDV soundness game, and we could have that $X_c = X$ or that $X_d = X$. If $h_c \neq h'_c$ and $X_c = X_\alpha$, then C' can solve equation 5.1 for the discrete logarithm of $X_c = X_\alpha$. The probability that $X_c = X_\alpha$ is 1/2 since $X_\alpha = X$ was chosen randomly from the set of participants and we know that $X_c = X_\alpha$ or $X_d = X_\alpha$. If $h_c = h'_c$ then $f \neq f'$. In this case, if $X_d = X_\alpha$ then C' can solve equation 5.1 for the discrete logarithm of $X_d = X_\alpha$. The probability that $X_d = X_\alpha$ is 1/2 since $X_\alpha = X$ was chosen randomly from the set of participants and we know that $X_c = X_\alpha$ or $X_d = X_\alpha$.

Case 2. We suppose that E is a Type-2 adversary. In this case, we know that X_c is uncorrupted (and we could have that $X_c = X$) and either f was a previous output from a **KCommit** query or E also outputs a keystone k such that $f = \mathsf{KCommit}(k)$.

If $h_c = h'_c$, then $f \neq f'$, so the values h_c and f (similarly h'_c and f') must have been computed after the H_2 query which resulted in h (or h^*), and satisfy the equations $f = h - h_c$ and $f' = h' - h'_c$. But we know that f is also an output of H_1 in the concurrent signature unforgeability game, either from a direct H_1 query, or via a **KCommit** query, and the probability that an output from an H_1 query matches (some function of) an output from an H_2 query is at most $\mu_2\mu_1/q$. This is negligible, so we assume that $f = f'$, and therefore that $h_c \neq h'_c$.

Since $h_c \neq h'_c$, if $X_c = X_\alpha = X$ then C' can now solve equation 5.1 for x, the discrete logarithm of $X_c = X_\alpha = X$. The probability that $X_c = X_\alpha$ is $1/2$ since $X_\alpha = X$ was chosen randomly from the set of participants and we know that $X_c = X_\alpha$ or $X_d = X_\alpha$.

From Cases 1 and 2 we find that the probability that C' can solve equation 5.1 for the discrete logarithm of X is at least

$$\gamma = \min\{\eta/2, \frac{\eta(q - \mu_1\mu_2)}{2q}\} = \frac{\eta(q - \mu_1\mu_2)}{2q}$$

which is non-negligible in security parameter l.

Since the NG Forking Lemma produces a second appropriate signature with expected time at most $\tau' = 120686\mu_s\tau$, we find that C' can solve the discrete logarithm problem in expected time $\tau'/\gamma = 120686\mu_s\tau/\gamma$.

This contradicts the hardness of the discrete logarithm problem. Therefore no such polynomially bounded adversary B can have non-negligible probability of winning the NG signature unforgeability game, and in turn no polynomially bounded adversary E can have non-negligible probability of winning the unforgeability game of Section 5.3.3.

□

Theorem 5.4 The concurrent signature scheme of Section 5.4 is *fair* in the random oracle model.

Proof: We suppose that H_1 and H_2 are random oracles, and suppose that there exists an algorithm E that with non-negligible probability wins the fairness game of Section 5.3.4. At the end of the game, we assume that E outputs outputs a keystone $k \in \mathcal{K}$, and $S = \langle \sigma, X_c, X_d, m \rangle$ where $\sigma = \langle s, h_c, f \rangle$, $s \in \mathcal{S}$, $h_c, f \in \mathcal{F}$, $X_c, X_d \in \mathcal{PK}$, $X_d \neq X_c$ and $m \in \mathcal{M}$, where $\langle k, 2, S \rangle$ is accepted by $\mathsf{CSVerify}$ and one of the following cases holds:

1. f was a previous output from a **KCommit** query and no **KReveal** query on input f was made, or

2. E also produces $S' = \langle \sigma', X_d, X_c, m' \rangle$, with $\sigma' = \langle s', h'_c, f \rangle$, $s' \in \mathcal{S}$, $h'_c, f \in \mathcal{F}$, and message $m' \in \mathcal{M}$, where $\mathsf{NTVerify}(S')$ returns *accept*, but $\langle k, 2, S' \rangle$ is not accepted by $\mathsf{CSVerify}$.

We show that such an E cannot exist.

Suppose case 1 of the output conditions occurs. Then E has found a keystone k and an output of a **KCommit** query f such that $f = H_1(k)$, but without making a **KReveal** query on input f. Since H_1 is a random oracle, E's probability of producing such a k is at most $\mu_1\mu_2/q$, where μ_1 is the number of H_1 queries made by E and μ_2 is the number of **KCommit** queries made by E. Since both μ_1 and μ_2 are polynomially bounded in the security parameter l and q is exponential in l, this probability is negligible. This contradicts our assumption that E wins the game with non-negligible probability, so Case 1 cannot occur.

Suppose case 2 of the output conditions occurs. Since $\langle k, 2, S \rangle$ is accepted by $\mathsf{CSVerify}$, we must have that $\mathsf{NTVerify}(S)$ returns *accept* and $\mathsf{KCommit}(k)=f$. But then, since S and S' share the value f and $\mathsf{NTVerify}(S')$ returns *accept*, we must also have that $\langle k, 2, S' \rangle$ is accepted by $\mathsf{CSVerify}$. This is a contradiction, so Case 2 cannot occur. □

5.6 Extensions and Open Problems

5.6.1 The Scheme Can Use a Variety of Keys

Our concurrent signature scheme can be based on any ring signature scheme, as long as it is of the correct form and compatible with the keystone idea. Thus it is feasible to build concrete concurrent signature schemes using a variety of key types. The security of such schemes could then be based on a variety of underlying hard problems. Furthermore, the key pairs in a single concurrent signature scheme may be of different types if the concurrent signature scheme is constructed using a separable ring signature scheme which is compatible with the keystone idea.

5.6.2 The Multi-party Case

It would be interesting to see if concurrent signatures could be extended to the multi-party case, that is, where many entities can fairly exchange signatures concurrently. The existing

two party scheme can trivially be extended to include multiple matching signers by using a ring signature scheme with the appropriate number of signers. However it appears to be difficult to construct an appropriate model of security for fairness in the multiparty case. We illustrate some of these issues by an example.

In the 2-party case, the matching signer is convinced of the initial signer's participation before responding. In the multiparty case, an intermediate signer may like assurance that the following signers will be committed before the keystone is revealed. However this release may be under the control of the initial signer, requiring the intermediate signers to trust the initial signer more than is necessary in the 2-party case. It would therefore be interesting to investigate methods whereby the revelation of keystones does not depend entirely on the initial signer, but on the other signing parties as well.

Susilo *et al.* propose a solution for the tripartite case in [107], however so far there have been no proposals extending the notion of concurrent signatures to the general multiparty case.

5.6.3 Extensions to Concurrent Signatures

A few authors have extended the idea of concurrent signature schemes (as was presented in [41]). In [108], Susilo *et al.* present the notion of perfect concurrent signatures. This paper improves the notion of ambiguity in [41] (or non-transferability as we refer to it in this chapter) by ensuring that even if both signers are known to be trustworthy, it is still infeasible to determine which ambiguous (NT) signature corresponds to which signer. By contrast, our ambiguous (NT) signatures use a common value, so if two signers are known to have followed the protocol, then the position of the common value indicates the position of the keystone which in turn indicates the true authorship of each signature. However concurrent signatures are designed to be used in environments where signers do not trust each other, so the value of the improvements made in [108] are questionable.

In parallel work to [108], Nguyen also presents a proposal to improve the ambiguity (or non-transferability) properties of concurrent signatures in [89]. The goals of the two papers appear to be very similar, although the solutions are slightly different.

The original work of [41] in fact briefly discusses the extension of concurrent signature schemes to the identity-based setting, and argues that the extension is trivial. Nevertheless, Chow and Susilo extend the ideas in [108] to the identity-based setting in [44].

5.7 Conclusion

We introduced the notion of concurrent signatures, presented a concurrent signature scheme and related its security to the hardness of the discrete logarithm problem in an appropriate security model. We have also discussed some applications for concurrent signatures, and the advantages they have over previous work. In particular, we have compared concurrent signatures to techniques for fair exchange of signatures, and presented some applications in which the full security of fair exchange may not be necessary and the more pragmatic solution of concurrent signatures suffices.

Part II

Key Agreement Protocols

Chapter 6

Introduction to Key Agreement

6.1 Basic Concepts

We start by defining some fundamental concepts that will be necessary for our discussions on key agreement protocols.

A protocol in which a shared secret intended for cryptographic use becomes available to two or more parties is called a *key establishment protocol*. Key establishment protocols result in shared secrets which are typically called *session keys*. A session key is usually intended to be an ephemeral secret, i.e., a secret value which will only be used for a short time period or session, after which it is securely erased.

The class of key establishment protocols may be subdivided into *key transport protocols* and *key agreement protocols* which are defined as follows.

Definition 6.1 [87] A *key transport protocol* is a key establishment protocol where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s).

Definition 6.2 [87] A *key agreement protocol* is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value.

In this thesis we are concerned with key agreement rather than key transport, therefore we focus on key agreement from now on.

When discussing communications on a channel, we require the following terminology:

Passes: The number of passes in a protocol is the total number of messages exchanged in the protocol.

Broadcast: A broadcast message is a message that is sent to every party in a protocol.

Rounds: A round consists of all the messages that can be sent and received in parallel in a protocol.

These notions can be affected by the type of network in which the protocol is operating. For example, not all networks allow messages to be broadcast, and in this case, a separate message must be sent to all parties.

We occasionally call a generic execution of a protocol between two or more participants a *run* of the protocol.

6.1.1 Adversarial Assumptions

As in Section 2.3, we define an *adversary* or *attacker* of a cryptographic protocol to be an entity which tries to defeat the intended security objective of the protocol. A *passive adversary* is one which only monitors communication channels. An *active adversary* is one which attempts to delete, add, or in some way modify the transmissions on a channel. We call such attacks made by an active adversary *active attacks*.

It is typically assumed that protocol messages are transmitted over channels or networks which are unprotected against attacks by an active adversary. When analyzing the security of a protocol, we therefore assume that the adversary has complete control over the network, with the ability to record, alter, delete, insert, redirect, reorder, replay past messages, and inject new messages. In addition, it is common to assume that an adversary is also capable of engaging unsuspecting authorized parties by initiating new protocol executions.

6.2 The Diffie-Hellman Protocol

The Diffie-Hellman key agreement protocol [50] revolutionized cryptography by introducing a fundamental technique for constructing key agreement protocols. Diffie-Hellman key agreement provided the first practical solution to the key distribution problem, allowing two parties who previously had no shared secret to establish such a shared secret by exchanging messages over an open channel.

Protocol 1 defines the original Diffie-Hellman protocol between two entities A and B.

Intuitively, the security of Protocol 1 appears to be related to the computational Diffie-Hellman (CDH) problem (defined in Section 2.1.3) since a passive adversary would have

Protocol 1: The Diffie-Hellman protocol (original version).

A and B begin by selecting an appropriate (large) prime p and a generator g of $\mathbb{Z}_p^*$. The following steps must be taken each time a session key is required:

1. A selects an ephemeral random integer $a, 1 \leq a \leq p-2$,
2. B selects an ephemeral random integer $b, 1 \leq b \leq p-2$.

A and B then exchange the following messages, in either order:

$$A \longrightarrow B : g^a \bmod p$$

$$B \longrightarrow A : g^b \bmod p$$

On receipt of the message $g^b \bmod p$, A computes $K_A = (g^b \bmod p)^a \bmod p$, and on receipt of the message $g^a \bmod p$, B computes $K_B = (g^a \bmod p)^b \bmod p$. We find that $K_A = K_B = K = g^{ab} \bmod p$ which can be used as a secret session key shared between A and B. The ephemeral values a and b should be erased on completion of the protocol.

to solve the CDH problem in order to determine the session key. In fact, when analyzed in an appropriate security model in which the adversary's task is to distinguish session keys from random strings, the security of this protocol is related to the decisional Diffie-Hellman problem (also defined in Section 2.1.3), and only provides secrecy of the resulting key against passive adversaries.

The original Diffie-Hellman protocol is in fact insecure against active adversaries since neither A nor B have any assurance of the source of the messages they receive or the identity of the party with whom they share the resulting key. This is demonstrated by a well-known attack on the original Diffie-Hellman protocol, known as a *man-in-the-middle attack*.

6.2.1 Man-in-the-Middle Attacks

The man-in-the-middle attack on Protocol 1 works as follows.

As for Protocol 1, we assume that A and B have selected an appropriate (large) prime p and a generator g of $\mathbb{Z}_p^*$, and the following steps have been performed:

1. A selects an ephemeral random integer $a, 1 \leq a \leq p-2$,
2. B selects an ephemeral random integer $b, 1 \leq b \leq p-2$.

If an adversary E wishes to launch a man-in-the-middle attack on this protocol run,

E selects ephemeral random integers $a', b', 1 \leq a', b' \leq p-2$. A and B then transmit the following messages, which E intercepts and replaces as follows:

$$\begin{array}{ccccc} A & & E & & B \\ g^a \bmod p & \longrightarrow & g^{a'} \bmod p & \longrightarrow & \\ & \longleftarrow & g^{b'} \bmod p & \longleftarrow & g^b \bmod p \end{array}$$

On receipt of the message $g^{b'} \bmod p$, A computes $K_A = (g^{b'} \bmod p)^a \bmod p$, and E computes $K_{EA} = (g^a \bmod p)^{b'} \bmod p$.

On receipt of the message $g^{a'} \bmod p$, B computes $K_B = (g^{a'} \bmod p)^b \bmod p$, and E computes $K_{EB} = (g^b \bmod p)^{a'} \bmod p$.

Although A and B believe that they share a key, they do not since $K_A \neq K_B$. Instead, A shares a key $K_A = K_{EA}$ with E, and B shares a key $K_B = K_{EB}$ with E.

Now when A sends a message to B encrypted with K_A, E can decipher it with K_{EA}, re-encrypt it with K_{EB}, and send it to B. Similarly, E can decrypt messages sent by B to A with K_{EB}, re-encrypt them with K_{EA}, and send them to A. In this way, A and B believe that they share a secure channel, while in fact E controls all the communication between them.

6.3 Authenticated Key Agreement

Protocol 1 is vulnerable to a man-in-the-middle attack since it not *authenticated*. By this we mean that entities who participate in the protocol have no way of verifying the identities of other entities with whom they may share the resulting key.

We now informally define some notions of authentication for key agreement protocols. Although the following notions apply to key establishment protocols in general, we are concerned only with key agreement protocols and therefore restrict our definitions to this case. The following definitions are formalized later in the thesis.

Definition 6.3 [87] *Key authentication* is the property whereby one party is assured that no other party aside from a specifically identified second party (and possibly additional identified trusted parties) may gain access to a particular secret key.

Definition 6.4 An *authenticated key agreement protocol* is a key agreement protocol which provides key authentication.

We suppose that entity A runs an authenticated key agreement protocol Π. The key authentication property of Π does not guarantee that the second party (B, say) actually

possesses the secret key, or that B was even involved in the protocol run. However it does guarantee that if any entity other than A can compute the secret key, then that entity must be B. For this reason, key authentication is sometimes referred to more precisely as (implicit) key authentication.

Definition 6.5 [87] *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

In practice, there are various ways to demonstrate possession of a key, including producing a one-way hash of the key itself or encrypting a known value using the key. The disadvantage of such methods is that some information about the value of the key is revealed, even if the information is not useful to a computationally bounded adversary. Alternatively, zero-knowledge techniques [34, 59, 62] may be employed to demonstrate possession of a key while providing no additional information regarding its value.

Definition 6.6 [87] *Explicit key authentication* is the property obtained when both (implicit) key authentication and key confirmation hold.

Definition 6.7 [87] *Entity authentication* is the process whereby one party is assured (through the acquisition of corroborative evidence) of the identity of a second party involved in a protocol and that the second party actually participated in the protocol.

The guarantee that the second party actually participated in the protocol ensures that the corroborative evidence that the first party receives is *fresh*, meaning that it is new evidence and has not simply been replayed (by some unauthorized entity) from some previous interaction with the second party.

Entity authentication is not a requirement in all protocols, although it can be used as a tool to construct authenticated key agreement protocols. However in this case, it is critical that in such a protocol, the party whose identity is corroborated is the same party with which the key is agreed.

There are many ways in which a key agreement may be authenticated. For example, entities that share a long-term secret may wish to generate an ephemeral secret session key. In this case, the entities may make use of the long-term shared secret in order to authenticate a key agreement protocol. An alternative approach is to use public key cryptography (where each entity has a long-term public and private key pair) to authenticate the key agreement protocol. In this case, certificates (or some PKI) would be required to authenticate the public keys. We examine this approach in more detail in the next section.

6.3.1 Security Attributes

There are a number of ways in which an attacker can attempt to break a key agreement protocol, and when constructing a key agreement protocol, the designer must consider what types of attack the protocol must resist.

Such analysis has let to the development of various desirable security attributes for key agreement protocols. We list the most common ones as described in [16, 76, 87]. These attributes can be vital in excluding certain realistic attacks.

Known session key security: A protocol has known session key security if knowledge of previous session keys does not allow an adversary to compromise other previous session keys or future session keys.

(Perfect) forward secrecy: A protocol has forward secrecy if the compromise of the long term private keys of one or more entities does not lead to the compromise of previously agreed session keys in the presence of a passive adversary. Perfect forward secrecy refers to the scenario when the long term private keys of all participating entities are compromised.

No key-compromise impersonation: When an adversary compromises an entity A's long-term private key, then an adversary can of course impersonate A. However a protocol is resistant to key compromise impersonation attacks if, after capturing A's long-term private key, an adversary cannot impersonate other entities to A in a key agreement protocol and obtain the resulting session key.

No unknown key-share: A key agreement protocol is resistant to unknown key-share attacks if an entity cannot be coerced into sharing a session key with a different party to the one intended without their knowledge. For example, A cannot be coerced into sharing a key with B when in fact A believes the key is shared with C. Unknown key-share attacks may lead to confusion when the key is applied. In such attacks, the adversary may even be one of the parties A, B or C.

No key control: A key agreement protocol has no key control if none of the participants (or an adversary) can force the session key to be a preselected value (or to lie within a small set of values), or predict the value of the session key. Mitchell *et al.* [88] discuss how the responder in a protocol almost always has an unfair advantage in controlling the value of the established session key. This can be avoided by

the use of commitments, although this seems to always require an extra round of communication.

Other attributes that are often desirable for key agreement protocols include:

Key freshness: A key is *fresh* if it can be guaranteed to be new (i.e. it is not an old key being reused). This is related to key control.

Efficiency: A protocol is efficient if its computational and communication complexity is minimized. Computational complexity is affected by the cost of the computations required by the protocol and the amount of precomputation that is possible. Communication complexity is affected by the number of passes and rounds that the protocol requires and the type of network being used.

Role symmetry: A protocol has role symmetry when the messages transmitted and the computations performed by all entities have the same structure.

Role symmetry is often a desirable attribute since it can simplify the implementation of a key agreement protocol. However, as we shall see in Chapter 7, role symmetry is in fact disallowed by many security models.

6.3.2 Authenticated Diffie-Hellman Protocols

We present some examples of simple protocols which provide implicit authentication for the original Diffie-Hellman protocol (Protocol 1) using public key techniques. We describe two of the authenticated key agreement protocols presented by Blake-Wilson *et al.* [16].

We start by describing [16, Protocol 3]. As in Protocol 1, we define the protocol between entities A and B who can communicate over an open channel and who wish to generate a shared secret session key. Since the protocol uses public key techniques for authentication, A and B require public and private key pairs $\langle X_A, x_A \rangle$ and $\langle X_B, x_B \rangle$ respectively.

We assume that p and q are large primes where $q|(p-1)$, and g is an element of $\mathbb{Z}_p^*$ of order q. We also assume that $H : \{0,1\}^* \rightarrow \{0,1\}^l$ is a cryptographic hash function for a fixed value l (usually the security parameter). We assume that x_A and x_B are chosen randomly from $\mathbb{Z}_q^*$ and that $X_A = g^{x_A} \bmod p$ and $X_B = g^{x_B} \bmod p$.

The next protocol we describe is [16, Protocol 4]. It is very similar to Protocol 2 above except that the session key is generated in a slightly different way. Although the

Protocol 2: Protocol 3 of [16].

The following steps must be taken each time a session key is required:

1. A selects an ephemeral random integer $a \in \mathbb{Z}_q$,
2. B selects an ephemeral random integer $b \in \mathbb{Z}_q$.

A and B then exchange the following messages, in either order:

$$A \longrightarrow B : T_A = g^a \bmod p$$

$$B \longrightarrow A : T_B = g^b \bmod p$$

On receipt of the message $g^b \bmod p$, A computes

$$K_A = H(T_B^a \bmod p, X_B^{x_A} \bmod p),$$

and on receipt of the message $g^a \bmod p$, B computes

$$K_B = H(T_A^b \bmod p, X_A^{x_B} \bmod p).$$

We find that $K_A = K_B = K = H(g^{ab} \bmod p, g^{x_A x_B} \bmod p)$ which can be used as a secret session key shared between A and B. The ephemeral values a and b are erased on completion of the protocol.

change seems to be a minor one, the resulting protocol, Protocol 3, has different security properties to Protocol 2 as will be discussed in Section 6.3.3.

We assume once again that p and q are large primes where $q|(p-1)$, and g is an element of $\mathbb{Z}_p^*$ of order q. We also assume that A and B have public and private key pairs $\langle X_A, x_A \rangle$ and $\langle X_B, x_B \rangle$ respectively which are generated as in Protocol 2.

We notice that in Protocol 3, the order of the message flows is important in computing the key. The initiator of the protocol (in this case the initiator is A since she sent the first message in the protocol) computes the session key in a different way to the responder (B in this case). If there is confusion about who initiated the protocol (e.g. A and B both believe that they initiated the protocol) then A and B will not generate the same session key.

6.3.3 Security Attributes of Protocols 2 and 3

We now informally examine whether Protocols 2 and 3 appear to have the security attributes of known-key security, (perfect) forward secrecy and resistance to key compromise impersonation attacks. Despite the apparent similarity between Protocols 2 and 3, the

Protocol 3: Protocol 4 of [16].

The following steps must be taken each time a session key is required:

1. A selects an ephemeral random integer $a \in \mathbb{Z}_q$,
2. B selects an ephemeral random integer $b \in \mathbb{Z}_q$.

A and B then exchange the following messages, in the following order:

$$A \longrightarrow B : T_A = g^a \bmod p$$

$$B \longrightarrow A : T_B = g^b \bmod p$$

On receipt of the message $g^b \bmod p$, A computes

$$K_A = H(T_B^{x_A} \bmod p, X_B^a \bmod p),$$

and on receipt of the message $g^a \bmod p$, B computes

$$K_B = H(X_A^b \bmod p, T_A^{x_B} \bmod p).$$

We find that $K_A = K_B = K = H(g^{x_A b} \bmod p, g^{x_B a} \bmod p)$ which can be used as a secret session key shared between A and B. The ephemeral values a and b are erased on completion of the protocol.

results of this analysis clearly illustrate some of the differences between the two protocols. We give informal arguments why each protocol does or does not not appear to have each of the security attributes considered.

Known session key security: Protocols 2 and 3 both appear to have known key security since it does not seem to be feasible for an adversary to gain knowledge of a new session key given knowledge of previous session keys. The main reason for this is that session keys are generated as outputs of a cryptographic hash function, and the inputs to this hash function change for each new session key established. If we assume that the hash function is one-way, then the adversary cannot determine the inputs to the hash function from the output. The adversary therefore does not learn any information from previous session keys that may be useful in determining the value of a new session key.

(Perfect) forward secrecy: Protocol 2 appears to have perfect forward secrecy. This is because even if an adversary knows the private keys x_A and x_B, the adversary cannot compute $g^{ab} \bmod p$ from the values T_A and T_B if we assume that the computational Diffie-Hellman problem is hard.

On the other hand, Protocol 3 does not have perfect forward secrecy since given the private keys x_A and x_B and the values T_A and T_B the adversary can compute the session key. However Protocol 3 does appear to have what might be called partial forward secrecy, since an adversary would need both private keys to compute the session key. With only one private key, the adversary would only be able to compute one of the inputs to the hash function, not both.

No key-compromise impersonation: Protocol 2 is not resistant to key compromise impersonation attacks. Given A's private key x_A, an adversary E could pretend to be B by choosing a value $b \in \mathbb{Z}_q$, computing $T_B = g^b \bmod p$, and sending this to A. E would receive A's value T_A in response. Now E can compute the value K_A as $H(X_B^{x_A} \bmod p, T_A^b \bmod p)$, and E now shares a key with A, while A believes her key is shared with B.

On the other hand, Protocol 3 appears to be resistant to key compromise impersonation attacks since it appears to be infeasible for an adversary to compute a session key from the values x_A, X_B, T_A and b.

It can be seen from the three security attributes analyzed above that Protocols 2 and 3 do indeed provide different security guarantees, despite their similarity. We do not consider whether Protocols 2 and 3 are resistant to unknown key-share attacks and key control since these properties require more complex analysis to establish.

The question of whether Protocols 2 and 3 should be considered secure depends on one's definition of security. Constructing a good definition of security for authenticated key agreement protocols is not a trivial task, and is the subject of Chapter 7.

Chapter 7

Models of Security for Key Agreement Protocols

7.1 Introduction

The design and analysis of key agreement protocols has proven to be a non-trivial task. Since the pioneering paper by Diffie and Hellman [50] which presented a solution for unauthenticated key agreement based on asymmetric techniques, many attempts have been made to construct key agreement protocols that provide implicit or explicit authentication.

Initially, protocol analysis was heuristic, and protocols were evaluated against known attack methods and recommended security attributes such as those listed in Section 6.3.1. However this was no guarantee that a given protocol would not fall prey to some new form of attack.

In 1993 Bellare and Rogaway [12] proposed the first formal treatment for the analysis of security of authenticated key agreement protocols. Their aim was to provide better security guarantees than are attainable from heuristic analysis. They proposed definitions for secure authenticated key agreement protocols as well as secure authenticated key agreement protocols with key confirmation using an appropriate security model. We refer to the model of Bellare and Rogaway [12] as the BR model.

Their model of security assumes that any two communicating parties have a shared long-term secret which can be used for authentication within the protocol. They also model multiple communicating parties who can participate in concurrent protocol runs. In their model the adversary is assumed to have complete control over the network.

Shortly after this work, Bellare and Rogaway adapted their original model of security to the 3-party case, in which a trusted authority participates in the key agreement

protocol [13] Other works [9, 16, 17] adapted the original BR model to the public key setting. Further extensions of the BR model include adaptations to the smart-card based setting [102], the identity-based setting [39], the tripartite key agreement setting [3], and adaptations to model dictionary attacks (in the password-based setting) [78, 10, 64].

Since the work in this thesis focusses on key agreement protocols in the public key setting, we present the security model of Blake-Wilson, Johnson and Menezes [16] (which we call the BJM model). This is possibly the most well-known adaptation of the BR model to the public key setting. In this model, each participant is assumed to possess a public and private key pair (where the public key is authenticated via some PKI), and these keys are used to provide authentication within the key agreement protocol.

7.2 The BJM Model

As in the original paper by Bellare and Rogaway [12], Blake-Wilson *et al.* [16] provide definitions of security for authenticated key agreement protocols (which they call AK protocols) and authenticated key agreement protocols with key confirmation (which they call AKC protocols). Although much of the notation we use follows that in [16], we change some of the notation to be consistent with subsequent models presented in this chapter. This will simplify comparisons between the models. We also present some aspects of the security model slightly differently to the way in which they are presented in [16], but this does not affect the functionality of the security model or the definition of security.

In the BJM model, all communication between protocol participants is controlled by the adversary. This is achieved by modeling protocol participants by oracles, who communicate only with the adversary. Oracles therefore never communicate directly with one another, only indirectly via the adversary. The adversary can read messages sent by oracles, provide its own messages, modify messages and delay or erase messages. The adversary may also initiate protocols, modeling the ability of parties to engage in many sessions of the protocol in parallel.

Although not usually made explicit in security models for key agreement, we assume that the key agreement protocol Π being run has some algorithm Setup. On input a security parameter l, Setup generates the public parameters *params* for the protocol. We also assume that there is a key generation algorithm KeyGen which on input *params* generates a public and private key pair $\langle PK, SK \rangle$ for a given protocol participant.

7.2.1 Protocol Participants

The model includes a set $\mathcal{U}$ of participant identifiers (IDs), and each instance of a participant is modeled by an oracle, e.g. $\Pi^i_{U,V}$ would model a participant $U \in \mathcal{U}$ carrying out a protocol session in the belief that it is communicating with another participant $V \in \mathcal{U}$, which we call U's *intended partner*, for the ith time (i.e. the ith run of the protocol between U and V). Each oracle $\Pi^i_{U,V}$ keeps a public transcript $T^i_{U,V}$ which records messages that it has sent or received as a result of queries it has answered. Participant oracles are modeled by probabilistic polynomial time Turing machines.

Each participant $U \in \mathcal{U}$ has a public and private key pair $\langle PK_U, SK_U \rangle$ which we assume is authenticated by some CA (or via some PKI), and each oracle instance of U has access to these keys.

When an oracle $\Pi^i_{U,V}$ receives some input M, it is recorded on $T^i_{U,V}$ and $\Pi^i_{U,V}$ proceeds according to the protocol. If the oracle produces some output, then this is also recorded on $T^i_{U,V}$.

At any stage, $\Pi^i_{U,V}$ may be in one of three states. The state of $\Pi^i_{U,V}$ is denoted $\delta^i_{U,V}$ and can be set to one of the following:

undecided: This is the initial state of the oracle and means that $\Pi^i_{U,V}$ has not yet terminated the protocol.

accepted: This is the state of the oracle if it has successfully terminated the protocol holding some session key $sk^i_{U,V}$.

rejected: This is the state of the oracle if it has terminated the protocol without holding a session key.

Upon termination, the oracle state (but not the session key if any) is recorded on the transcript $T^i_{U,V}$

The model also includes an adversary, E, who is not a participant. E is modeled by a probabilistic polynomial time Turing Machine and can interact with all the participants' oracles via queries. In addition E has access to the transcript of each oracle. Participant oracles only respond to queries by the adversary and do not communicate directly amongst themselves.

7.2.2 Oracle Queries

In attacking a key agreement protocol, we allow an adversary E to interact with a challenger C that simulates a set of participant oracles running the protocol. E can interact with the oracles by making various queries, and the responses are simulated by C.

For some security parameter l, C runs the Setup algorithm to generate the public parameters *params*. C also generates a set of participant IDs $\mathcal{U}$, where $|\mathcal{U}|=n_P$ and n_P is a polynomial function of l. For each participant $U \in \mathcal{U}$, C runs the KeyGen algorithm to generate a public and private key pair $\langle PK_U, SK_U \rangle$. In addition, we assume that each participant $U \in \mathcal{U}$ can engage in at most n_S sessions with any other participant $V \in \mathcal{U}$ where n_S is a polynomial function in l.

The set of oracles with which E can interact is

$$\{\Pi^i_{U,V} : U, V \in \mathcal{U}, i \in \{1, ..., n_S\}\}$$

where each oracle $\Pi^i_{U,V}$ has access to the public and private keys of participant U, the public parameters, and the public keys of all the other participants. E is given *params*, $\mathcal{U}$ and all the public keys. In addition, E has access to the oracle transcripts and can make the following queries:

Send$(\Pi^i_{U,V}, M)$**:** E can send message M to $\Pi^i_{U,V}$. M is recorded on $T^i_{U,V}$ and C (simulating $\Pi^i_{U,V}$) responds according to the protocol. Any output is recorded on $T^i_{U,V}$. If $M = \lambda$, then $\Pi^i_{U,V}$ initiates a protocol run (with intended partner V). An oracle $\Pi^i_{U,V}$ is called an *initiator oracle* if the first message it has received is λ. If $\Pi^i_{U,V}$ did not receive a message λ as its first message, then it is called a *responder oracle*.

Reveal$(\Pi^i_{U,V})$**:** E may request the session key held by $\Pi^i_{U,V}$. If $\Pi^i_{U,V}$ has accepted (i.e. $\delta^i_{U,V}$=accepted) and holds a session key $sk^i_{U,V}$, then this is output. An oracle $\Pi^i_{U,V}$ is called *revealed* if it has responded to a **Reveal** query.

Corrupt(U, PK'_U, SK'_U)**:** E may request the long-term private key of participant U, or may choose to replace U's key pair with the key pair $\langle PK'_U, SK'_U \rangle$. C outputs SK_U, and replaces U's key pair $\langle PK_U, SK_U \rangle$ with $\langle PK'_U, SK'_U \rangle$. All corresponding participant oracles are updated with the new key pair. A participant U is called *corrupted* if U has responded to a **Corrupt** query.

We say that E has *revealed* an oracle if it has issued a **Reveal** query to that oracle, and E has *corrupted* a participant if it has issued a **Corrupt** query to that participant.

7.2.3 Matching Conversations

The BJM model defines the notion of a *matching conversation*, which is used to reason about two oracles that have engaged in communication (via the adversary).

In order to determine whether two oracles have had a matching conversation, we examine the transcripts of the oracles in the presence of an adversary. For any oracle $\Pi^i_{U,V}$, its *conversation* is captured on its transcript $T^i_{U,V}$, which can be represented by a sequence:

$$T^i_{U,V} = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), ..., (\tau_m, \alpha_m, \beta_m)$$

This sequence records that at time τ_1, oracle $\Pi^i_{U,V}$ received α_1 and output β_1, then at time $\tau_2 > \tau_1$, the oracle received α_2 and output β_2, and so on, until at time τ_m, the oracle received α_m and output β_m.

In a particular execution of a protocol, the adversary's i-th query to an oracle is said to occur at time $\tau = \tau_i$. We do not specify an exact value for a given time τ_i, but we demand that $\tau_i < \tau_j$ when $i < j$. Notions of time that satisfy these requirements include "abstract time" where $\tau_i = i$, or "Turing machine time" where τ_i is the i-th step in the adversary's computation.

We notice that if $\alpha_1 = \lambda$, then $\Pi^i_{U,V}$ is an *initiator* oracle, otherwise $\Pi^i_{U,V}$ is a *responder* oracle. In what follows, we assume that the number of message passes R in the protocol is odd, so $R = 2\gamma + 1$ for some $\gamma \in \mathbb{N}$. The case for R even is analogous.

Definition 7.1 [16] Consider two oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ who run Π in the presence of an adversary E, where $\Pi^i_{U,V}$ is an initiator oracle with transcript $T^i_{U,V}$, and $\Pi^j_{V,U}$ is a responder oracle with transcript $T^j_{V,U}$. If there exist $\tau_0, \tau_1, .., \tau_R$ such that $T^i_{U,V}$ is prefixed by

$$(\tau_0, \alpha_0 = \lambda, \beta_0), (\tau_2, \beta_1, \beta_2), ..., (\tau_{R-1}, \beta_{R-2}, \beta_{R-1})$$

and $T^j_{V,U}$ is prefixed by

$$(\tau_1, \beta_0, \beta_1), (\tau_3, \beta_2, \beta_3), ..., (\tau_R, \beta_{R-1}, *)$$

then we say that $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ have engaged in a *matching conversation*.

We note that in $T^j_{V,U}$, the value $*$ means that any entry, possibly none, is allowed here.

The definition of matching conversations models the situation where a participant U believes he is communicating with a participant V (and vice versa), one of them is the initiator and the other is the responder, and the adversary acts like a wire, and

simply passes messages unaltered to and fro between the participants. We refer to such an adversary as a *benign* adversary.

If an oracle $\Pi^i_{U,V}$ has had a matching conversation with one of its intended partner's (V's) oracles $\Pi^j_{V,U}$, then we say that $\Pi^j_{V,U}$ is the *matching oracle* of $\Pi^i_{U,V}$ (and vice versa).

7.2.4 Freshness

The adversary can learn information about various session keys through its queries, since it can obtain session keys of any oracle that has accepted a session key via a **Reveal** query, and it can obtain the long-term private key of any participant via a **Corrupt** query.

However in order to model the security of a key agreement protocol, we need to identify oracles about whose session keys the adversary should not have learned any information. Such oracles are called *fresh*.

Definition 7.2 An oracle $\Pi^i_{U,V}$ is called *fresh* if it has accepted (and therefore holds a session key $sk^i_{U,V}$), it is not revealed, neither U nor V has been corrupted, and there is no revealed oracle $\Pi^j_{V,U}$ with which it has had a matching conversation.

It is important to note that an oracle $\Pi^i_{U,V}$ may be fresh but may not have had a matching conversation with any oracle at all (i.e. it may not have a matching oracle). This is possible since all oracles communicate only with the adversary and never directly between themselves. Indeed, the adversary may be able to format its messages to $\Pi^i_{U,V}$ in such a way that $\Pi^i_{U,V}$ accepts, but no other oracle is involved in the protocol run.

7.2.5 The BJM Game and Test Query

The security of a key agreement protocol is modeled via the following game between a challenger C and an adversary E.

Initialization(l): On input a security parameter l, C runs the Setup algorithm to generate the system parameters $params$. C also generates a set of participant IDs $\mathcal{U}$, where $|\mathcal{U}| = n_P$ and n_P is a polynomial function of l. For each participant $U \in \mathcal{U}$, C runs the KeyGen algorithm to generate a public and private key pair $\langle PK_U, SK_U \rangle$. Each oracle $\Pi^i_{U,V}$ for any $V \in \mathcal{U}$ and $i \in \{1, ..., n_S\}$ will have access to the private key SK_U. In addition, we assume that each participant $U \in \mathcal{U}$ can engage in at most n_S sessions with another participant $V \in \mathcal{U}$ where n_S is a polynomial function in l. E is given $params$, $\mathcal{U}$ and all the public keys.

Phase 1: C simulates a set of oracles

$$\{\Pi^i_{U,V} : U, V \in \mathcal{U}, i \in \{1, ..., n_S\}\}.$$

to which E can make **Send**, **Reveal** and **Corrupt** queries as were defined in Section 7.2.2.

Test($\Pi^i_{U^*,V^*}$): At some point, E may make a **Test** query to some *fresh* oracle $\Pi^i_{U^*,V^*}$. C randomly selects a bit b. If $b = 1$, then C outputs the session key $sk^i_{U^*,V^*}$, otherwise C outputs a randomly chosen element from the session key space.

Phase 2: E can continue making **Send**, **Reveal** and **Corrupt** queries to the oracles, except that E is forbidden from revealing $\Pi^i_{U^*,V^*}$ or its matching oracle (if any), and E cannot corrupt U^* or V^*.

Output: Finally E outputs a bit b'.

E *wins* the game if $b = b'$, and we define E's advantage in winning the game as

$$Advantage^E(l) = |\Pr[b' = b] - 1/2|.$$

We note that the original game in [16] was not adaptive, i.e. the **Test** query was the final query that the adversary could make, so there was no **Phase 2**. However the non-adaptive model was shown to be insufficiently powerful by Rackoff, and the model was fixed in [9]. We have adopted this fix here, and have presented the adaptive version of the model of [16].

7.2.6 AK security

We now present the definition of an authenticated key agreement (AK) protocol, as given in [16].

Definition 7.3 A protocol Π is a secure AK protocol if the following two conditions hold:

1. In the presence of the benign adversary, oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ always accept holding the same session key $sk = sk^i_{U,V} = sk^j_{V,U}$, and sk is distributed uniformly at random over the session key space.
2. For every adversary E:
   (a) If oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ have matching conversations and U and V are uncorrupted, then both oracles accept holding the same session key sk;
   (b) $Advantage^E(l)$ is negligible in l.

7.2.6.1 Notes on AK Security

Conditions 1 and 2(a) in the definition of AK security simply guarantee that the protocol is correct and achieves its goal when not under attack. In fact, conditions 1 and 2(a) are almost identical and indeed condition 1 may be discarded if we include the session key uniformity in 2(a).

The condition 2(b) guarantees that an adversary cannot distinguish a fresh oracle's session key from random, i.e. an adversary can learn no information about a fresh oracle's session key, even by making **Send**, **Reveal** and **Corrupt** queries to any oracles (except the fresh oracle and its intended partner).

A condition that is not immediately obvious, but which is implied by the model, is that if an adversary E can, with non-negligible probability, make any two oracles running protocol Π accept and hold the same session key sk, where the oracles are not matching oracles, then E can win the game with non-negligible probability. In this instance, Π is not a secure AK protocol.

This property is best illustrated by the following example. Suppose that with non-negligible probability η, E can make $\Pi^i_{U,V}$ and $\Pi^j_{K,L}$ accept, holding the same session key sk, but where $\Pi^i_{U,V}$ and $\Pi^j_{K,L}$ are not matching oracles (since they did not have a matching conversation). Then E can reveal the session key sk held by $\Pi^j_{K,L}$, and can select $\Pi^i_{U,V}$ for the **Test** query (this is allowed because $\Pi^i_{U,V}$ is still considered to be fresh). E can now win the game with probability η since it knows the session key sk of $\Pi^i_{U,V}$.

This condition implies that for any secure AK protocol, if oracles $\Pi^i_{U,V}$ and $\Pi^j_{K,L}$ accept holding the same session key, then with overwhelming probability they are matching oracles (i.e. they have had a matching conversation). This result, together with condition 2(a) of the security definition means that, with overwhelming probability, two oracles $\Pi^i_{U,V}$ and $\Pi^j_{K,L}$ running a secure AK protocol accept holding the same session key if and only if they are matching oracles.

7.2.7 AKC security

In order to define security for authenticated key agreement protocols with key confirmation, we require one further notion.

We consider an adversary E interacting with a challenger C simulating a set of oracles $\{\Pi^i_{U,V} : U, V \in \mathcal{U}, i \in \{1, ..., n_S\}\}$ running protocol Π. As in the game above, we allow the adversary to make **Send**, **Reveal** and **Corrupt** queries to the oracles. We then let

$\texttt{No-Matching}^E(l)$ denote the event that at some point during the interaction between the oracles and E, there exists an oracle $\Pi^i_{U,V}$ which has accepted (and which holds a session key $sk^i_{U,V}$), where U and V are uncorrupted, but where $\Pi^i_{U,V}$ did not have a matching conversation with any other oracle $\Pi^j_{V,U}$.

We now present the definition of an authenticated key agreement protocol with key confirmation (AKC protocol), as given in [16].

Definition 7.4 A protocol Π is a secure AKC protocol if the following two conditions hold:

1. In the presence of the benign adversary, oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ always accept holding the same session key $sk = sk^i_{U,V} = sk^j_{V,U}$, and sk is distributed uniformly at random over the session key space;

2. For every adversary E:

   (a) If oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ have matching conversations and U and V are uncorrupted, then both oracles accept holding the same session key sk;

   (b) The probability of $\texttt{No-Matching}^E(l)$ is negligible;

   (c) $Advantage^E(l)$ is negligible in l.

7.2.7.1 Notes on AKC Security

Conditions 1, 2(a) and 2(c) of the AKC definition are the same as those in the AK definition, so a secure AKC protocol is a secure AK protocol. However the definition of AKC security has the additional `No-Matching` condition 2(b).

The `No-Matching` condition means that in a secure AKC protocol, the only way for an adversary to get an uncorrupted entity to accept in a run of the protocol with any other uncorrupted entity is by allowing it to have a matching conversation with another oracle (i.e. relaying communications like a wire).

The `No-matching` condition together with condition 2(a) ensures that in a secure AKC protocol, an oracle $\Pi^i_{U,V}$ will accept if and only if it had a matching conversation with some other oracle $\Pi^j_{V,U}$. In addition, condition 2(a) ensures that if two oracles have had a matching conversation, then they both accept holding the same session key. Therefore no oracle $\Pi^i_{U,V}$ will accept unless it has indeed communicated with an oracle $\Pi^j_{V,U}$ and the two oracles share the same key. This provides the property of *key confirmation*.

7.2.8 Security Attributes of the BJM Model

We recall the security attributes presented in Section 6.3.1 which are commonly required of authenticated key agreement protocols. We consider which of these attributes are implied by the above definitions of secure AK or AKC protocols.

Known-key security: The property of known-key security is implied by the definitions of AK and AKC security. This can be seen by the following two properties of the model:

1. E is allowed to make Reveal queries to any oracles except for $\Pi^i_{U^*,V^*}$ and its matching oracle $\Pi^j_{V^*,U^*}$ to obtain any session keys except for the session key $sk^i_{U^*,V^*}$.
2. Even with the knowledge of many other session keys, E's ability to distinguish between $sk^i_{U^*,V^*}$ and a random number is still negligible if $\Pi^i_{U^*,V^*}$ is fresh.

Unknown key-share: We recall that a key agreement protocol is resistant to unknown key-share attacks if an entity cannot be coerced into sharing a session key with a different party to the one intended without that entity's knowledge.

Suppose that two oracles $\Pi^i_{U,V}$ and $\Pi^t_{K,L}$ share a session key. Then with overwhelming probability, we have that $\Pi^t_{K,L} = \Pi^j_{V,U}$ for some $j \in \{1, ..., n_S\}$. This is because, as was pointed out in Section 7.2.6.1, the definitions of AK and AKC security imply that if any two oracles share a session key, then with overwhelming probability, they must have had matching conversations.

So U can only share a key with his intended partner (in this case V), and no unknown key-share attack can succeed. Hence the definitions of AK and AKC security imply resilience to unknown key-share attacks.

We note that the definitions of AK and AKC security do not imply that a secure protocol has forward secrecy or is resistant to key compromise impersonation attacks. In order to model these attacks, we would have to allow E to corrupt the participant corresponding to the oracle on which it makes the **Test** query (or its intended partner), and the definition of freshness in the BJM model does not allow this.

Moreover, the definitions of AK and AKC security do not imply resistance to key control attacks that are launched by one of the protocol participants (although key control attacks launched by an outside adversary are captured by the BJM model). In the BJM

model, all participants are assumed to be honest participants unless they are corrupted (in which case E can impersonate a participant). But the BJM model does not allow E to corrupt the participant corresponding to the oracle on which it will make a **Test** query (or that oracle's intended partner). However we do know that if the participants are honest and the protocol is not attacked, then the session key established is distributed uniformly at random in the session key space.

7.3 A Modified BJM Model

The BJM approach provides a reasonable model of security for authenticated key agreement protocols. However it does not capture certain security attributes (such as forward secrecy and resilience to key compromise impersonation attacks), and in some ways it appears to be unnecessarily complex. For example, the definition of matching conversations is very complicated, and the notation $\Pi^j_{U,V}$ for oracles can be rather confusing, especially since $\Pi^j_{U,V}$ may not be communicating with participant V at all.

We therefore present a modification of the BJM model (which we call the mBJM model) which captures more attacks than the BJM model and which we believe simplifies some of the concepts in the BJM model. Our modifications of the BJM model are inspired largely by the model of Bellare *et al.* [10], which, although developed in the password-based setting, addresses many of the issues of the BJM model. We will use our mBJM model in the next chapter to examine the security of some concrete protocols.

The mBJM model presented in this chapter is based on the mBR model presented in [73]. We have changed the name since the model is more closely related to the BJM model than to the BR model.

Our model includes a set of participant IDs $\mathcal{U}$, where each participant $U \in \mathcal{U}$ has a long-term public key PK_U and a long-term private key SK_U. We use Π^i_U to denote an oracle modeling the ith instance of participant U.

Oracles follow the rules of the protocol, responding to input messages. Each oracle maintains a public transcript T^i_U which records all messages they have sent or received as a result of queries they have answered.

At any stage Π^i_U may be in one of three states. The state of Π^i_U is denoted δ^i_U and can be set to one of the following:

undecided: This is the initial state of the oracle and means that Π^i_U has not yet terminated the protocol.

accepted: This is the state of the oracle if it has successfully terminated the protocol (accepted) holding some session key sk_U^i.

rejected: This is the state of the oracle if it has terminated the protocol without holding a session key.

An oracle Π_U^i may accept at any time, and once accepted it should hold a role $role_U^i \in \{initiator, responder\}$, a partner ID $pid_U^i \in \mathcal{U}$ (this corresponds to the intended partner in the BJM model and is the ID of the oracle with which it assumes it is communicating), a session ID sid_U^i and a session key sk_U^i. We note that the value i in Π_U^i is not the same as the sid_U^i but rather an internal session counter for each oracle. This may act as an internal identifier for the session until sid_U^i is established, and thereafter the session is (uniquely) identified by sid_U^i. By the end of the protocol, the role, partner ID, session ID and oracle state (but not the session key if any) are recorded on T_U^i.

As in the BJM model, the mBJM model includes an adversary E that is modeled by a probabilistic polynomial time Turing Machine. E can interact with all the participants' oracles via queries and has access to the transcript of each oracle. Participant oracles only respond to queries by the adversary and do not communicate directly amongst themselves.

7.3.0.1 Partners

Definition 7.5 When running the protocol, if oracles Π_U^i holding $(sk_U^i, sid_U^i, pid_U^i)$ and $\Pi_{U'}^j$ holding $(sk_{U'}^j, sid_{U'}^j, pid_{U'}^j)$ have both accepted and the following conditions hold:

1. $sid_U^i = sid_{U'}^j$, $sk_U^i = sk_{U'}^j$, $pid_U^i = U'$ and $pid_{U'}^j = U$,
2. $role_U^i = initiator$ and $role_{U'}^j = responder$ or vice versa,
3. No oracle in E's game besides Π_U^i or $\Pi_{U'}^j$ accepts with session ID equal to sid_U^i,

then Π_U^i and $\Pi_{U'}^j$ are said to be *partners*.

This definition roughly corresponds to the definition of matching oracles in the BJM model, and will be used to define freshness of an oracle in the mBJM game.

7.3.1 Oracle Queries

As in the BJM model, the security of a key agreement protocol is modelled via the following game between a challenger C and an adversary E.

For some security parameter l, C runs the Setup algorithm to generate the system parameters $params$. C also generates a set of participant IDs $\mathcal{U}$, where $|\mathcal{U}|{=}n_P$ and n_P is a polynomial function of l. For each participant $U \in \mathcal{U}$, C runs the KeyGen algorithm to generate a public and private key pair $\langle PK_U, SK_U \rangle$. In addition, we assume that each participant $U \in \mathcal{U}$ can engage in at most n_S sessions with any other participant $V \in \mathcal{U}$ where n_S is a polynomial function in l.

The set of oracles with which E can interact is

$$\{\Pi_U^i : U \in \mathcal{U}, i \in \{1, ..., n_S\}\}$$

where each oracle Π_U^i has access to the public and private keys of participant U, the public parameters, and the public keys of all the other participants. E is given $params$, $\mathcal{U}$ and all the public keys. In addition, E has access to the oracle transcripts and can make the following queries:

Send(Π_U^i, M)**:** E can send the oracle Π_U^i a message M. M is recorded on T_U^i and Π_U^i responds according to the protocol. Any output is recorded on T_U^i. If $M = \lambda$ and M is the first message received by Π_U^i, then Π_U^i initiates a protocol run (with some intended partner pid_U^i), sets $role_U^i = initiator$. In this case Π_U^i is called an *initiator oracle*. If Π_U^i did not receive a message λ as its first message, then it sets $role_U^i = responder$ and is called a *responder oracle*.

Reveal(Π_U^i)**:** E may request the session key held by Π_U^i. If Π_U^i has accepted and holds a session key sk_U^i, then this is output. An oracle Π_U^i is called *revealed* if it has responded to a **Reveal** query.

Corrupt(U)**:** E may request the long-term private key of participant U, and may choose to replace U's key pair with the key pair $\langle PK'_U, SK'_U \rangle$. C outputs SK_U, and replaces U's key pair $\langle PK_U, SK_U \rangle$ with $\langle PK'_U, SK'_U \rangle$. All corresponding participant oracles are updated with the new key pair. A participant U is called *corrupted* if U has responded to a **Corrupt** query.

We say that E has *revealed* an oracle if it has issued a **Reveal** query to that oracle, and E has *corrupted* a participant if it has issued a **Corrupt** query to that participant.

7.3.2 Freshness

The definition of freshness serves the same purpose as in the BJM model, i.e. to identify oracles about whose session keys the adversary should have learned no information.

Definition 7.6 An oracle Π_U^i is called *fresh* if it is not revealed, it does not have a revealed partner, and if the participant pid_U^i is uncorrupted.

We note that, unlike in the BJM model, an oracle Π_U^i may still be considered fresh if U is corrupted. This will be important for modeling key compromise impersonation attacks.

7.3.3 The mBJM Game and the Test Query

As in the BJM model, the security of a key agreement protocol in our mBJM model is modeled via a game between a challenger C and an adversary E. This game is as follows:

Initialization(l): On input a security parameter l, C runs the Setup algorithm to generate the system parameters *params*. C also generates a set of participant IDs $\mathcal{U}$, where $|\mathcal{U}| = n_P$ and n_P is a polynomial function of l. For each participant $U \in \mathcal{U}$, C runs the KeyGen algorithm to generate a public and private key pair $\langle PK_U, SK_U \rangle$. Each oracle Π_U^i for any $i \in \{1, ..., n_S\}$ will have access to the public and private key pair $\langle PK_U, SK_U \rangle$, as well as the public parameters and the public keys of all other participants. In addition, we assume that each participant $U \in \mathcal{U}$ can engage in at most n_S sessions with any other participant $U' \in \mathcal{U}$, where n_S is a polynomial function in l. E is given *params*, $\mathcal{U}$ and all the public keys.

Phase 1: C simulates a set of oracles

$$\{\Pi_U^i : U \in \mathcal{U}, i \in \{1, ..., n_S\}\}.$$

to which E can make **Send**, **Reveal** and **Corrupt** queries as were defined in Section 7.3.1.

Test($\Pi_{U^*}^i$): At some point, E may make a **Test** query to some *fresh* oracle $\Pi_{U^*}^i$. C randomly selects a bit b. If $b = 1$, then C outputs the session key $sk_{U^*}^i$, otherwise C outputs a randomly chosen element from the session key space.

Phase 2: E can continue making **Send**, **Reveal** and **Corrupt** queries to the oracles, except that E is forbidden from revealing $\Pi_{U^*}^i$ or its partner oracle (if any), and E cannot corrupt participant $pid_{U^*}^i$.

Output: Finally E outputs a bit b'.

E *wins* the mBJM game if $b = b'$, and we define E's advantage in winning the game as

$$Advantage^E(l) = |\Pr[b' = b] - 1/2|.$$

7.3.4 Definition of security

A *benign adversary* is defined as in the BJM model and is one who simply relays messages between parties without modification. We then define the notion of a secure authenticated key agreement protocol as follows:

Definition 7.7 A protocol is an mBJM-AK secure protocol if the following two conditions hold:

1. In the presence of a benign adversary, two oracles running the protocol both accept holding the same session key and session ID, and the session key is distributed uniformly at random on the session key space.
2. For any adversary E, $Advantage^{E}(l)$ is negligible.

We say that protocol Π is *mBJM-AK insecure* if it is not mBJM-AK secure. That is, there exists an adversary E which, with non-negligible probability (in l), wins the mBJM game against challenger C. We say that such an adversary E can successfully *mBJM attack* protocol Π.

7.3.4.1 Notes on mBJM-AK Security

Our definition of mBJM-AK security is similar to the definition of AK security in the BJM model. We recall that in the BJM model, the definition of AK security guaranteed that if two oracles shared the same session key, then with overwhelming probability, they were matching oracles. The analogous result for mBJM-AK security in the mBJM model is that if two oracles running a secure mBJM-AK protocol accept holding the same session key, then with overwhelming probability, they are partners. We recall that the converse must be true since, by Definition 7.5, two oracles must share the same session key to be partners.

Our model is very similar to the BJM model except that we work with partners instead of matching oracles, we use the notion of partner IDs instead of the notation $\Pi^{i}_{U,V}$, and we allow a corrupted participant's oracle to still be considered fresh. We allow this because corruption in our mBJM model is simply a query to a participant which reveals the long-term secret key of the participant. The adversary does not learn any other internal state of the participant's oracles and does not gain control of these oracles. Therefore a corrupted participant's oracle may still be considered to be fresh and can therefore still be chosen as a **Test** oracle. This allows us to model key compromise impersonation attacks.

Most of these deviations from the BJM model are inspired by the model of Bellare *et al.* [10]. The main difference between our mBJM model and that of [10] is that our model is in the public key setting. In addition we do not explicitly distinguish between acceptance and termination as is done in [10], and we do not model perfect forward secrecy. The perfect forward secrecy property can be added as is done in [10] by permitting the adversary to corrupt both participants of the **Test** session on the condition that the adversary does not alter the massages transmitted in the **Test** session in any way.

7.3.5 Security Attributes of the mBJM Model

We recall that the definition of AK security ensured that a key agreement protocol has known-key security and resistance to unknown key-share attacks. Using similar arguments to Section 7.2.8, the same can be shown for key agreement protocols that are mBJM-AK secure.

In addition, the definition of mBJM-AK security also ensures resistance to key compromise impersonation attacks. We recall from Section 6.3.1 that a key compromise impersonation attack on an entity A has occurred if an adversary is able to impersonate some other entity (say B) to A in a key agreement protocol and obtain the resulting session key by compromising A's long-term private key (but not B's long-term private key). Such attacks are captured by our model since the adversary is permitted to corrupt a participant, continue to interact with the corrupted participant's oracles, and then select one of these oracles for a **Test** query.

Our definition of mBJM-AK security does not model perfect forward secrecy. However as mentioned before, our model can be extended (as is done in [10]) to model perfect forward secrecy as well.

We direct the reader to [8, 10, 12, 16, 32, 101] for details of alternative models illustrating different approaches to dealing with partnering, corruptions and freshness.

7.3.6 mBJM-AKC Security

The definition of mBJM-AK security can be extended to a definition of security in the mBJM model for authenticated key agreement protocols with key confirmation (AKC) protocols as is done in [16].

As for the definition of AKC security in Section 7.2.7, we require an additional condition, which we call `No-Partnering`. This is closely related to the `No-Matching` condition in Definition 7.4. We let $\texttt{No-Partnering}^E(l)$ denote the event that at some point during

the interaction between the oracles and E, there exists an oracle Π_U^i which has $pid_U^i = V$ where V is uncorrupted, which has accepted (and which holds a session key sk_U^i), but where Π_U^i has no partner.

Definition 7.8 A protocol is an mBJM-AKC secure protocol if the following three conditions hold:

1. In the presence of a benign adversary, two oracles running the protocol both accept holding the same session key and session ID, and the session key is distributed uniformly at random on the session key space.
2. The probability of $\texttt{No-Partnering}^E(l)$ is negligible;
3. For any adversary E, $Advantage^E(l)$ is negligible.

7.4 Identity-based Models

The BJM and mBJM models are suitable for protocols in the standard public key setting. However the models do need to be modified slightly in order to model protocols which use identity-based long-term public and private keys. This is because identity-based public keys can be derived from any identifying string in a dynamic way by the adversary. A security model in an identity-based environment should adequately model such features of the identity-based environment.

In the usual public key setting, the challenger determines the set of public keys at the beginning of the game. By contrast, in the identity-based setting we allow the adversary to generate identity-based public keys dynamically during the attack game.

Since public keys can always be derived from identifiers, when modeling protocols in the identity-based setting, we therefore only need to work with the identifiers (IDs) of protocol participants, and not their actual public keys.

We now consider how the BJM model can be adapted to the identity-based (ID-based) setting. We call the identity-based version of the BJM model the ID-BJM model, and this model will be required in the next chapter to examine the security of specific identity-based key agreement protocols.

We note that similar changes would be required to adapt the mBJM model to the identity-based setting, but this will not be required in this thesis, so we leave the details to the reader.

7.4.1 The ID-BJM model

The ID-BJM model is very similar to the BJM model, with each participant having a unique identifier U, and instances of U are modeled as oracles $\Pi^i_{U,V}$, where this represents participant U communicating with intended partner V for the i-th time. Each oracle maintains a transcript $T^i_{U,V}$ and state $\delta^i_{U,V}$ as in the BJM model, and matching conversations are defined as before.

However in the ID-based setting, a participant's public key is generated directly from his identifier U. This means that the adversary E can generate new public keys from any identifier of its choice. In order to model this, we therefore modify the oracle queries that E can make.

As in the BJM model, E can make various queries to a challenger C that simulates a set of oracles (or participants) running the protocol. However in the ID-based setting, C does not generate a set of participant IDs beforehand. Rather, C sets $\mathcal{U}$ to be the ID space (usually $\mathcal{U} = \{0,1\}^*$). The specific participant IDs will be chosen by E in the game itself, and C will initialize new participants and their oracle instances as required. However the number of new participant IDs that E can generate is bounded by some value n_P, and we also still assume that each participant U can engage in at most n_S sessions with any other participant $V \in \mathcal{U}$, where n_P and n_S are polynomial functions in l.

C also simulates the trusted authority (TA) in this environment, and therefore generates the public parameters of the TA and gives these to E. C also generates a master secret s from which it can generate a private key SK_U from any given ID U. C keeps s private. E can make the following queries:

Send(U, V, i, M)**:** Message M is sent to $\Pi^i_{U,V}$. If participants U and V do not yet exist, then C firstly initializes these participants (and their oracle instances) and generates their public and private keys using their IDs U and V and the master secret s. M is recorded on $T^i_{U,V}$ and $\Pi^i_{U,V}$ responds according to the protocol. Any output is recorded on $T^i_{U,V}$. If $M = \lambda$ and M is the first message received by $\Pi^i_{U,V}$, then $\Pi^i_{U,V}$ initiates a protocol run (with intended partner V). In this case oracle $\Pi^i_{U,V}$ is called an *initiator oracle*. If $\Pi^i_{U,V}$ did not receive a message λ as its first message, then it is called a *responder oracle*.

Reveal$(\Pi^i_{U,V})$**:** E may only request the session key held by an existing oracle $\Pi^i_{U,V}$. If $\Pi^i_{U,V}$ has accepted and holds a session key $sk^i_{U,V}$, then this is output. An oracle $\Pi^i_{U,V}$ is called *revealed* if it has responded to a **Reveal** query.

Private key extract(U): E may request the private key corresponding to ID U. If participant U does not yet exist, then C firstly initializes this participant (and its oracle instances) and generates its public and private keys using its ID U and the master secret s. C outputs the private key SK_U. The private key of U is said to have been *extracted* if U has responded to a **Private key extract** query.

The definition of freshness is then modified appropriately to be:

Definition 7.9 An oracle $\Pi^i_{U,V}$ is called *fresh* if it has accepted (and therefore holds a session key $sk^i_{U,V}$), it is not revealed, the private keys of neither U nor V has been extracted, and there is no revealed oracle $\Pi^j_{V,U}$ with which it has had a matching conversation.

The ID-BJM game and definitions of AK and AKC security are identical to the BJM game and definitions of AK and AKC security, except that **Corrupt** queries are replaced by **Private key extract** queries, and the **Send**, **Reveal** and **Private key extract** queries are answered as described above.

7.5 A Modular Approach to the Construction of KA Protocols

The above definitions of security for key agreement protocols provide a strong security guarantee for a key agreement protocol, provided a proof of security can be constructed. However in general it appears to be rather difficult to prove efficient key agreement protocols secure in these models, and therefore relatively few protocols have full proofs of security in these models.

To address this problem, a more "modular" approach to constructing key agreement protocols was advocated by Bellare, Canetti and Krawczyk [8]. Instead of designing a key agreement protocol and then attempting to prove its security in a suitable model, Bellare *et al.* proposed an approach by which secure protocols can be constructed.

The approach entails constructing a basic protocol which is secure given communication over ideally authenticated channels. These are called *authenticated links*. In other words, the basic protocol should be secure against a passive adversary.

This basic protocol then needs to be transformed into a protocol which is secure given *unauthenticated links* (or insecure channels). This is achieved by applying what are called *authenticators* to each message flow in the protocol (and we then say that the resulting protocol is authenticated).

Informally, if an authenticator is secure, then the origin of each authenticated message is verifiable and no adversary can forge the origin of a message or alter the message contents without detection. Therefore if the underlying unauthenticated protocol is secure against passive attacks, then the authenticated protocol is secure against all (active and passive) adversaries.

The advantage of such an approach is that the basic protocol and the authenticator may be constructed and proven secure independently, and the resulting protocol is guaranteed to be secure. In fact, libraries of basic protocols and authenticators may be built up (as is done in [8, 23, 32, 67, 110, 111]), from which many different secure key agreement protocols may be constructed. Such a modular approach greatly simplifies the protocol design process, thereby reducing the risk of errors.

The disadvantage of using this modular approach is that it is only useful for constructing new protocols, and results in protocols of a specific form. However this modular approach sheds no light on how to evaluate the security of the many existing protocols in the literature that have not been constructed in this modular way.

In addition, cryptographic primitives such as encryption, signatures or MACs are usually required to build secure authenticators, and the application of these authenticators often increases the computational and communication complexity of a protocol. Therefore the modular method of construction advocated by Bellare *et al.* [8] generally does not result in the most efficient key agreement protocols. Of course protocols constructed using this modular approach may be modified to be more efficient using various techniques, but then the security proof may no longer be valid.

7.6 Universal Composability

A proof that a particular protocol Π is secure in some security model (such as the BJM model) ensures that if Π is exposed to attacks of the type captured by the security model, then it remains secure. However the proof of security does not provide us with any guarantees of what will happen if our protocol is exposed to attacks or influences outside of the security model (i.e. events that are not captured by the security model).

The security models that we have considered so far generally model a set of participants which only run the protocol Π. Therefore, a proof of security in such a model guarantees that the protocol is secure against adversaries with capabilities specified by the model and in an environment in which participants run only the protocol Π. However in today's world, protocols are very rarely run in isolation. A single machine may be running many

different protocols concurrently with many other entities or machines, and these other interactions may be related to the key agreement protocol Π. We would like to know that even in such an environment, where our protocol Π is composed with many other protocols or processes, it will remain secure.

Recent developments in provable security for key agreement protocols [29, 30, 32, 33, 31] have therefore been concerned with whether such protocols remain secure when composed with an unbounded number of unknown (and perhaps even malicious) protocols, or more generally, when the protocol is used as a component of an arbitrary distributed system. This has resulted in the development of what is called the *universal composition* framework (UC framework). For completeness, we give a brief overview of this framework.

The UC framework [29, 30] attempts to model protocols running in an arbitrary environment. It is claimed by Canetti and Krawczyk that a proof of security for a protocol Π in the UC framework guarantees that Π may be composed in an arbitrary distributed system and will still achieve its goals.

A definition of security of a protocol in the UC framework is formulated in a slightly different way to the way in which it is formulated in models such as the BJM model. In the UC framework, we ascertain the goals or functionality of a protocol in order to construct an *ideal process* which achieves the goals of the protocol in a secure way. In the ideal process, all parties usually interact with a trusted third party or *ideal functionality* which performs the function of the protocol. The ideal process can therefore be thought of as a formal specification of the goals and security requirements of the protocol.

As in previous models (such as the BJM model), a protocol instance in the UC framework is modeled by a probabilistic polynomial time Turing machine, and interacts with an adversary which is also modeled by a probabilistic polynomial time Turing machine. However unlike in previous models, an additional machine called the *environment* is added to the model of computation. The environment can interact with the protocol being run as well as the adversary. This environment machine represents everything that is external to the current protocol execution.

Informally, a protocol is then considered to be secure if it *securely realizes* its goals, or *emulates* the ideal process from the point of view of the environment. The environment therefore serves as an interactive distinguisher between the protocol and the ideal process. This means that an adversary interacting with the protocol has the same affect as (and can cause no more damage than) an adversary interacting with the ideal process.

We remark that the UC framework has been used to model not only key agreement

protocols, but also many other cryptographic primitives and protocols (such as public key encryption, digital signatures and protocols involving more than two parties). However since we are only concerned with key agreement protocols at this stage, we have only presented an informal description of the UC framework in the context of key agreement protocols.

The UC framework provides a very strong definition of security. But it appears to be even more difficult to construct protocols that realize UC security than it is to construct protocols that realize other definitions of security (e.g. AK security in the BJM model). Various relaxations of the UC model have been proposed, although these result in a loss of security, particularly with respect to composability. Canetti and Krawczyk [33] present certain key agreement protocols that achieve UC security as well as protocols that achieve a relaxed version of UC security. However all the protocols that have been shown to achieve this level of security are constructed using the modular approach of Bellare *et al.* [8]. It is not known whether simpler, more efficient protocols that are not constructed using the modular approach can be shown to be UC secure.

Chapter 8

Modular Security Proofs for Key Agreement Protocols

8.1 Introduction

It is becoming increasingly common for designers of key agreement protocols (and indeed designers of all cryptographic primitives) to have to provide proofs of security for their protocols in appropriate security models before their protocols will be considered for practical use. However at the same time, protocol designers are also under pressure to provide protocols that are optimized for efficiency.

The modular approach of [8] for constructing key agreement protocols (which was described in Section 7.5) can be employed to construct secure protocols, and if the protocol designer has access to libraries of secure basic protocols and authenticators, the protocol designer may have a large number of secure protocols from which to choose. However, as was pointed out in Section 7.5, protocols generated in such a manner are often not the most efficient.

In many environments, the benefits of being able to easily design secure protocols outweigh the possible disadvantages. However there exist environments in which efficiency is of utmost importance, and most key agreement protocols optimized for efficiency are not constructed in a modular way. Indeed we can find several efficient key agreement protocols in the literature which do not have formal proofs of security (such as protocols in [16, 51, 76, 79, 103]) or have only proofs of security in weakened security models (such as protocols in [3, 39, 83]. Since the structure of these protocols is not compatible with the modular approach in [8], complete proofs of security for such protocols still appear to be difficult to construct.

In this chapter, we consider protocols which are not designed in a modular way but which we nevertheless wish to prove secure in an appropriate security model. Since this type of protocol is not designed in a modular way, typical proofs of security are often complicated and error-prone. We therefore develop a technique by which the proof process for a large class of key agreement protocols can be simplified.

Informally, our technique for proving the security of a protocol Π works as follows. The first step is to prove that protocol Π has a property that we call *strong partnering* (which is defined in Section 8.3.1). The second step is to prove that a related protocol π is secure in a highly reduced security model. Finally, as the main result of the chapter, we show how the proof of security of π in the reduced model can be translated into a proof of security for Π in the full security model using a Gap assumption.

Each step above is far simpler than a single proof of security in the full security model. The result is a modular technique for constructing proofs of security for a large class of key agreement protocols which are not constructed using the modular approach of [8].

We then use this technique to consider certain key agreement protocols in the literature that were previously without proofs or that only had incomplete proofs of security. Using our techniques, full proofs of security can be generated for protocols in [16, 39, 103] (possibly after slight modifications to the protocols if necessary). We focus in detail on Protocol 3 in Section 6.3.2 (originally presented in [16, Protocol 4]), and the identity-based key agreement (AK) protocols of [39] and [103].

We also hope that our methods will aid future designers of lightweight key agreement protocols in the formal analysis of their protocols in simplifying their task by breaking it up into components.

8.1.1 Published Work

An earlier version of this work appears in [73] and forms the basis for this chapter. The nomenclature in this chapter differs slightly from the published work. In particular, what we refer to here as the mBJM model is referred to as the mBR model in [73].

8.2 Gap Assumptions

Our technique makes use of Gap assumptions, as defined by Okamoto and Pointcheval [93]. Informally, a Gap problem is usually the problem of solving some computational problem (e.g. computational Diffie-Hellman) with the help of a corresponding decisional

oracle (in this case a decisional Diffie-Hellman oracle). The decisional problem may be easy or hard; irrespective of this a Gap problem may still be defined.

Gap assumptions have recently found several applications in cryptography. In particular, Gap assumptions have been used in [1, 69, 110] to prove the security of certain key agreement protocols. However we show how Gap assumptions can be systematically applied to a class of protocols in order to obtain proofs of security, whereas previous works applied Gap assumptions in an ad-hoc manner.

Following the notation of Okamoto and Pointcheval [93], we informally define a family of Gap problems.

Let $f : X \times Y \rightarrow \{0, 1\}$ be any relation on sets X and Y. The *computational* problem (or *inverting* problem in the language of [93]) of f is, given $x \in X$, to compute any $y \in Y$ such that $f(x, y) = 1$ if such a y exists, or to return `Fail` otherwise.

The *decisional* problem of f is, given $(x, y) \in X \times Y$, to decide whether $f(x, y) = 1$ or not.

Definition 8.1 The Gap problem of f is to solve the computational problem of f using an oracle which solves the decisional problem of f.

We always refer to Gap problems in terms of computational and decisional problems. Although the definition of Gap problems in [93] is not restricted to this case, we only present this scenario since we will only use Gap problems in this context. As an example, we define the computational, decisional and Gap Diffie-Hellman problems.

Let p and q be primes where $q|p - 1$. Let G be a multiplicative subgroup of $\mathbb{Z}_p^*$, of order q, and let $g \in G$ generate G. We denote by $DL(g, h) \in \mathbb{Z}_q$ the discrete logarithm of $h \in G$ with respect to base g. So $g^{DL(g,h)} = h \bmod p$.

Given $a, b, c \in \mathbb{Z}_q$, we define the Diffie-Hellman relation f_{DH} as follows:

$$f_{DH} : (G \times G) \times G \rightarrow \{0, 1\}, \text{ where } f_{DH}(g^a, g^b, g^c) = \begin{cases} 1 & \text{if } g^{ab} = g^c \bmod p \\ 0 & \text{otherwise} \end{cases}$$

We can now define the computational, decisional and Gap problems of f_{DH}, better known as the computational, decisional and Gap Diffie-Hellman problems, respectively.

Computational Diffie-Hellman (CDH) Problem: Given $g^a, g^b \in G$, where $a, b \in_R \mathbb{Z}_q$, compute $g^c \in G$, such that $f_{DH}(g^a, g^b, g^c) = 1$. That is, compute $g^c = g^{ab} \bmod p$.

Decisional Diffie-Hellman (DDH) Problem: Given $g^a, g^b, g^c \in G$, where $a, b \in_R \mathbb{Z}_q$, determine $f_{DH}(g^a, g^b, g^c)$. That is, determine whether $c = ab \bmod q$ or not.

Gap Diffie-Hellman (GDH) Problem: Given $g^a, g^b \in G$ where $a, b \in_R \mathbb{Z}_q$, as well as an oracle that solves the DDH problem in G, compute $g^{ab} \bmod p$.

The corresponding assumptions are that the above problems are *hard*, that is, they are infeasible to solve in polynomial time in a security parameter used to define the problem instances.

8.3 Modular Security Proofs in the mBJM Model

The model of security in which we work in this chapter is the mBJM model presented in Section 7.2. However analogous versions of our results also hold in the models of [12, 16, 32] and the ID-BJM model of Section 7.4.1. We refer the reader back to Chapter 7 for details of the mBJM security model.

From now on, we assume that we are only dealing with key agreement protocols that produce a *hashed session key* on completion of the protocol. By this we mean that the key agreement protocol Π specifies that the session key be computed as the hash H of some string.

Definition 8.2 Suppose Π is a protocol in the mBJM model that produces a hashed session key using the cryptographic hash function H. Then the *session string* for a particular oracle Π_U^i is denoted $ss_{\Pi_U^i}$, and is defined to be the string which is hashed to produce the session key sk_U^i. So we have that $sk_U^i = H(ss_U^i)$.

This reliance on hashing to produce a session key does not seem to be too strong a restriction since it is fairly common to use a key derivation function to obtain a session key from a secret value established during a key agreement protocol, and this key derivation function is usually implemented via a hash function.

We make this restriction since our modular proof technique will only work on key agreement protocols that produce hashed session keys. It will also require that we model the hash function by a random oracle in the proof of security.

8.3.1 Protocol Partnering

When trying to establish that a protocol Π is secure in the mBJM model, we need to ensure that an adversary cannot trivially win the game defined in Section 7.3.3 by an attack on the partnering properties of Π.

Definition 8.3 Suppose Π is a key agreement protocol. If there exists an adversary E, which when attacking Π in an mBJM game defined in Section 7.3.3 and with non-negligible probability in the security parameter l, can make some two oracles Π_U^i and $\Pi_{U'}^j$ accept holding the same session key when they are not partners, then we say that Π has *weak partnering*. If Π does not have weak partnering, then we say that Π has *strong partnering*.

If a protocol Π had weak partnering, then there would exist an adversary E that could make oracles Π_U^i and $\Pi_{U'}^j$ accept holding the same session key but without being partners. The rules of the mBJM game would then allow the adversary to reveal the session key held by Π_U^i, and then choose $\Pi_{U'}^j$ for the **Test** query, allowing E to trivially win the security game.

Therefore, for Π to be a secure key agreement protocol as defined in Definition 7.7, Π must have strong partnering. This can be ensured by including appropriate "partnering information" in the session string $ss_{\Pi_U^i}$ (and therefore in the computation of the session key sk_U^i), where partnering information is information from which it can be established whether the two session participants are partners or not. If appropriate partnering information is included in the session string, then weak partnering (i.e. where two oracles generate equal session keys with different partnering information) implies a collision in the hash function H. Since H is modeled as a random oracle, this occurs with negligible probability, so the protocol has strong partnering. In fact, we only really require H to be collision resistant to ensure strong partnering.

In the mBJM model, partnership is defined via session keys, session IDs and partner IDs. For oracles Π_U^i and $\Pi_{U'}^j$ to accept holding the same (unique) session key but without being partners, they must have different sids and/or pids. To guarantee that protocol Π has strong partnering in the mBJM model, we must therefore ensure that the session IDs are unique to each session and that (with overwhelming probability) $sk_U^i = sk_{U'}^j$ only if $role_U^i \neq role_{U'}^j$, $sid_U^i = sid_{U'}^j$, $pid_U^i = U'$ and $pid_{U'}^j = U$. In this case we can use the values sid_U^i, U and pid_U^i (if U is the initiator and pid_U^i is the responder) or sid_U^i, pid_U^i and U (if U is the responder and pid_U^i is the initiator) as the partnering information, and the inclusion of this information in the session string will ensure strong partnering.

The idea of including appropriate partnering information in the session string to ensure strong partnering applies equally to our mBJM model as it does to the BR, BJM and ID-BJM models, even though the concept of partnering is slightly different in each of these models. For example, in the models of [12, 13, 16], partnering is defined via matching conversations, or matching transcripts, and where each participant believes they are

communicating with the other. Therefore a key agreement protocol secure in these models can never allow two oracles to share the same key without having matching transcripts and corresponding intended partners. Strong partnering in these models can therefore be ensured by including the identities of the two participants (starting with the initiator) as well as the protocol message flows (obtained from the protocol transcripts) in the session string of each oracle.

8.3.2 Reduced Games

We now consider two reduced mBJM games. The first game is identical to the mBJM game defined in Section 7.3.3 except that the adversary E is not allowed to make any **Reveal** queries. We call this reduced game a No-Reveals mBJM (NR-mBJM) game. The second game is identical to the NR-mBJM game, except that the adversary no longer makes the normal **Test** query. Instead, to win the game, the adversary must select an accepted and fresh oracle on which to make a (modified) **Test** query at the end of its computation and output the session key held by this oracle. Since the adversary in this game must actually compute the session key of an oracle (instead of having to decide between a session key and a random value from the key space), we call this game a computational NR-mBJM (cNR-mBJM) game. We define E's advantage, denoted $Advantage^E(l)$, in the cNR-mBJM game to be the probability that E outputs a session key sk such that $sk = sk_{\Pi^i_U}$ where Π^i_U is the oracle selected by the adversary for the (modified) **Test** query.

We define NR-mBJM (and cNR-mBJM) security as follows:

Definition 8.4 A protocol Π is a (c)NR-mBJM-secure key agreement protocol if:

1. In the presence of the benign adversary, two oracles running the protocol both accept holding the same session key and session ID, and the session key is distributed uniformly at random on $\{0,1\}^l$; and
2. For any adversary E, $Advantage^E(l)$ in the (c)NR-mBJM game is negligible.

We say that protocol Π is *(c)NR-mBJM-insecure* if it is not (c)NR-mBJM-secure. That is, there exists an adversary E which, with non-negligible probability (in l), wins the (c)NR-mBJM game against challenger C. We say that such an E can successfully *(c)NR-mBJM-attack* protocol Π.

As part of our modular proof technique for a given protocol Π which produces hashed session keys on completion of the protocol, we will consider a related protocol π. When it

is not clear from the context, we will use use the notation Π_U^i for an oracle running Π and $sk_{\Pi_U^i}, sid_{\Pi_U^i}$ and $pid_{\Pi_U^i}$ for the session key, session ID and partner ID of Π_U^i. Similarly, we will use the notation π_U^i for an oracle running π and $sk_{\pi_U^i}, sid_{\pi_U^i}$ and $pid_{\pi_U^i}$ for the session key, session ID and partner ID of π_U^i. When it is clear which protocol we are referring to, we will revert back to the simpler notation introduced in Section 7.3.

Protocol π is defined in the same way as Π except that the session key generated by π is defined to be the session string of Π rather than the hash of this string. That is, $sk_{\pi_U^i} = ss_{\Pi_U^i}$. As part of our proof technique it will be necessary to prove that protocol π is cNR-mBJM secure. Since the cNR-mBJM game is a highly reduced game, it is usually fairly easy to establish a protocol's security in this model. Although it may not be obvious how a proof of security in this reduced model may be helpful, in Section 8.3.3 we present a theorem which shows how a proof of cNR-mBJM security for π can be transformed into a proof of mBJM security for Π using a Gap assumption, provided that Π has strong partnering.

The reason that we defined NR-mBJM security when cNR-mBJM security is our main focus is that, although it is a more complex game than the cNR-mBJM game, a number of recent papers presenting new key agreement protocols prove that the protocols meet such a weakened definition of security [3, 16, 39, 83]. That is, they take an appropriate security model, and prove the security of their protocols in the No-Reveals (NR) variant of the security model.

It is trivial to see that if protocol Π is NR-mBJM secure, then it is also cNR-mBJM secure. We also have the following result relating the NR-mBJM security of Π and the cNR-mBJM security of the related protocol π.

Theorem 8.1 If protocol Π produces a hashed session key via hash function H and is NR-mBJM secure, then the related protocol π is cNR-mBJM secure.

Proof: We provide a sketch of the proof of this theorem. The details are left to the reader. We show that if there exists an adversary E that can cNR-mBJM-attack π, then we can build an adversary A that can NR-mBJM-attack Π.

Suppose that an adversary E wins the cNR-mBJM game for protocol π with non-negligible probability η. Suppose also that A runs an NR-mBJM game with challenger C. A in turn acts as a challenger for E in a cNR-mBJM game. A passes all E's queries to C and returns all C's outputs to E. Finally E will output the session key $sk_{\pi_U^i}$ of some fresh oracle π_U^i. Recall however that $sk_{\pi_U^i} = ss_{\Pi_U^i}$.

A then chooses Π_U^i for the **Test** query and receives a key sk. If $sk = H(sk_{\pi_U^i})$ then A outputs 1, otherwise A outputs 0. It is easy to see that A wins the NR-mBJM game against Π with probability η. □

We note that in the proof of the above theorem, no assumption is required concerning the properties of H.

8.3.3 Handling Reveal Queries using Gap Assumptions

We now consider a protocol Π which has strong partnering and for which the related protocol π is known to be cNR-mBJM secure. Our aim is to translate these results into a proof of mBJM security for Π. In order to do this, we will need to be able to construct a challenger C in an mBJM game for Π which can answer an adversary E's **Reveal** queries.

At first glance, it seems that C needs to be able to compute the session key sk_U for any oracle Π_U^i that E may wish to reveal during the mBJM game. However this is not the case if Π produces a hashed session key (via hash function H) and if H is modelled as a random oracle. We will see below in Theorem 8.2 that in this case, C only needs to be able to solve the following decisional problem:

> Given the public parameters, the transcript T_U^i of oracle Π_U^i in an mBJM game, as well as P_U and $P_{U'}$ (the public keys of U and U' where $pid_U^i = U'$) and s, where s is a string, decide whether $s = ss_{\Pi_U^i}$, where $ss_{\Pi_U^i}$ is the session string of oracle Π_U^i.

We call this decisional problem the *session string decisional problem for protocol* Π. We note that the corresponding computational problem is to compute the session string of Π, which is the same as computing the session key of π.

We now present the main result of this chapter.

Theorem 8.2 Suppose that key agreement protocol Π produces a hashed session key on completion of the protocol (via hash function H) and that Π has strong partnering. If the cNR-mBJM security of the related protocol π is probabilistic polynomial time reducible to the hardness of the computational problem of some relation f, and the session string decisional problem for Π is polynomial time reducible to the decisional problem of f, then the mBJM security of Π is probabilistic polynomial time reducible to the hardness of the Gap problem of f, assuming that H is a random oracle.

Proof:

Since the cNR-mBJM security of π is probabilistic polynomial time reducible (in security parameter l) to the hardness of the computational problem of some relation f, there exists an algorithm A that, on input a problem instance of the computational problem of f and acting as a challenger for an adversary E which has non-negligible probability η of winning the cNR-mBJM game for π in time τ, is able to solve the computational problem of f with some non-negligible probability $g(\eta)$ and in time $h(\tau)$, where g and h are polynomial functions.

We now define an algorithm B which, given an adversary D which has non-negligible probability η' of winning the mBJM game for Π in time τ', is able to solve the Gap problem of f with some non-negligible probability $g'(\eta')$ and in time $h'(\tau')$ where g' and h' are polynomial functions. B will act as a challenger for D. B will also run algorithm A and will simulate an adversary for A. Since B attempts to solve the Gap problem of f, B will also have access to a decisional oracle for f.

Since Π has strong partnering, we know that if two oracles in D's attack share the same session key, then they must be partners (with overwhelming probability). We therefore know that D will never reveal a session key sk where sk is equal to the **Test** query oracle Π_T^i's session key $sk_{\Pi_T^i}$. This is because D is not permitted to reveal the session key of the oracle on which a **Test** query was made or its partner (if it exists).

We also assumed that the session string decisional problem for Π is polynomial time reducible to the decisional problem of f. That is, there exists some algorithm C which, given a decisional oracle for f, is able to solve the session string decisional problem for Π in polynomial time τ'' (and with probability 1).

Since B's goal is to solve the Gap problem of f, B is initialized with an instance of the computational problem of f. B then runs A on the instance of the computational problem of f and simulates an adversary E for A. A sets up a cNR-mBJM game for B and gives all the public parameters to B. B in turn passes these public parameters to adversary D. B now answers all of D's queries as follows.

B passes all D's queries besides **Reveal** and H queries to A. Since, in any session, protocol π is identical to protocol Π until the session is completed and the session key is computed, these queries will all be answerable by A. B passes A's responses back to D.

In order for B to answer D's **Reveal** queries, B maintains a Guess session key list (G-List). Each element on the G-List is a tuple of the form $(T_V^j, P_V, P_{V'}, R_V^j)$ where T_V^j is the transcript of oracle Π_V^j, P_V is the public key of V, $P_{V'}$ is the public key of V' where $pid_{\Pi_V^j} = V'$, and R_V^j is a random guess for the session key sk_V^j of oracle Π_V^j. Initially the

G-List is empty.

In order for B to answer E's H queries, B maintains an (initially empty) H-List containing tuples of the form (s_i, sk_i, str). For each H query on string s that D makes, B checks whether s is on the H-List as the first component in some tuple (s_i, sk_i, str). If it is, then B outputs sk_i. If s is not on the H-List then B uses the algorithm C to determine whether s is a valid session string for any oracle Π_V^j on the G-List. If $s = ss_{\Pi_V^j}$ is the session string for some oracle Π_V^j on the G-List, then B outputs R_V^j and adds the tuple (s, R_V^j, str) where str="V,j" to the H-List. Otherwise B selects a random sk from the session key space, adds the tuple (s, sk, str) (where str is the empty string λ) to the H-List, and outputs sk.

When D makes a **Reveal** query on any oracle Π_U^i which has accepted, B proceeds as follows. If Π_U^i has an entry on the G-List of the form $(T_U^i, P_U, P_{U'}, R_U^i)$, B outputs the value R_U^i. Otherwise B checks whether any entry on the H-List of the form (s_i, sk_i, str) where $str = \lambda$ has $s_i = ss_{\Pi_U^i}$ using algorithm C. If such an entry (s_i, sk_i, str) exists, then str is set to "U,i" on the H-List and the entry $(T_U^i, P_U, P_{U'}, R_U^i)$ is added to the G-List, where $R_U^i = s_i$, T_U^i is the transcript of Π_U^i, P_U is the public key of U and $P_{U'}$ is the public key of U' where $pid_{\Pi_U^i} = U'$. Otherwise a random session key R_U^i is selected and the entry $(T_U^i, P_U, P_{U'}, R_U^i)$ is added to the G-List. To answer the **Reveal** query, B outputs the value R_U^i in every case.

In this way, B can consistently answer D's **Reveal** and H queries. At some point D selects an oracle Π_T^i for the **Test** query. B selects a random element sk from the session key space and gives this to D.

Since Π has strong partnering, $sk_{\Pi_T^i}$ will not be equal to any other revealed session key, so the only way that D could distinguish (with non-negligible probability) whether $sk = sk_{\Pi_T^i}$, would be to query the session string $ss_{\Pi_T^i}$ on H.

If D does not query H on the **Test** query oracle's session string $ss_{\Pi_T^i}$, then D can only win with probability $1/S_H$ where S_H is the size of the output space of H, which we assume is negligible in security parameter l. If D wins the game, then with overwhelming probability $1 - 1/S_H$, D queries H on $ss_{\Pi_T^i}$. B can detect this value by checking which of the tuples (s_i, sk_i, str) on the H-List with $str = \lambda$ has $s_i = ss_{\Pi_T^i}$ using algorithm C. B gives this s_i to A as A's adversary E's output of the **Test** query.

Since $ss_{\Pi_T^i} = sk_{\pi_T^i}$, B has simulated a valid adversary E for A which has non-negligible success probability $\eta = \eta' \cdot (1 - 1/S_H)$ and which runs in polynomial time $\tau = \tau' + \tau'' \cdot N_H \cdot (N_R + 1)$. Here N_H and N_R are the number of H and **Reveal** queries that D makes,

respectively. So A outputs the solution to the instance of the computational problem of f with non-negligible probability $g(\eta)$ and in time $h(\tau)$.

Therefore B solves the Gap problem of f with non-negligible probability $g(\eta)$ and in time $h(\tau)$.

□

8.4 Applying the Technique to Different Security Models

Analogous results to the ones in Section 8.3 can be obtained for the security models of [10, 12, 13, 16, 32] and the ID-BJM model of Section 7.4.1.

For each of these models, an equivalent definition of strong partnering can be made. In the models of [10, 32], partnering is defined in a similar way to the way it is defined in our mBJM model, but in the models of [12, 13, 16] partnering is defined via the concept of matching conversations, so strong partnering would be defined in this context as well. NR and cNR versions of the security model can also be defined in the same way as for the mBJM model, and the definition of the related protocol π is independent of the model used. It is then possible to prove analogous versions of Theorem 8.2 (and Theorem 8.1) for these models. These in turn illustrate how proofs in these models can be constructed in a modular way.

As an example, we present the analogous version of Theorem 8.2 for the BJM model of Section 7.2.

We consider a protocol Π in the BJM model. We can define the NR-BJM and cNR-BJM games in exactly the same way as the NR-mBJM and cNR-mBJM games were defined in Section 8.3.2. The session string decisional problem and the related protocol π can also be defined exactly as before. It remains to show how *strong partnering* is defined in the BJM model.

Definition 8.5 Suppose Π is a key agreement protocol. If there exists an adversary E, which when attacking Π in a BJM game defined in Section 7.2.5 and with non-negligible probability in the security parameter l, can make any two oracles $\Pi^i_{U,V}$ and $\Pi^t_{K,L}$ accept holding the same session key when they are not matching oracles, then we say that Π has *weak partnering*. If Π does not have weak partnering, then we say that Π has *strong partnering*.

To ensure that the protocol Π has strong partnering, we must ensure that (with over-

whelming probability) $sk^i_{U,V} = sk^t_{K,L}$ if and only if $K = V$ and $L = U$, and $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$ are matching oracles (or partners). If a protocol Π does not have strong partnering, then this can be ensured by modifying the protocol slightly to include the partnering information in the computation of the session key. In the BJM model, the partnering information is the information required to determine whether two oracles had matching conversations or not.

We recall from Definition 7.1 that in order to have matching conversations, each message that one oracle sends must be received by the other without modification, and in the correct order. Moreover, there can also be no confusion as to which oracle is the initiator and which is the responder.

Suppose that in a protocol run between oracles $\Pi^i_{U,V}$ and $\Pi^j_{V,U}$, oracle $\Pi^i_{U,V}$ sends and receives a sequence of messages $\beta_0, .., \beta_n$ where this does not include the initializing message λ or the termination output $*$. If the IDs of the initiator and responder (in that order) as well as the sequence $\beta_0, .., \beta_n$ are included in the session string $ss_{\Pi^i_{U,V}}$ (and therefore in the computation of the session key $sk^i_{U,V}$), then strong partnering can be guaranteed.

We now state versions of Theorems 8.1 and 8.2 adapted to the BJM model.

Theorem 8.3 If a protocol Π produces a hashed session key via hash function H and is NR-BJM secure, then the related protocol π is cNR-BJM secure.

Proof: The proof of this Theorem is identical to the proof of Theorem 8.1 except that references to NR-mBJM and cNR-mBJM games (and the resulting notation) are replaced by NR-BJM and cNR-BJM games (and the resulting notation). □

Theorem 8.4 Suppose that key agreement protocol Π produces a hashed session key on completion of the protocol (via hash function H) and that Π has strong partnering. If the cNR-BJM security of the related protocol π is probabilistic polynomial time reducible to the hardness of the computational problem of some relation f, and the session string decisional problem for Π is polynomial time reducible to the decisional problem of f, then the BJM security of Π is probabilistic polynomial time reducible to the hardness of the Gap problem of f, assuming that H is a random oracle.

Proof:

The proof of this theorem is almost identical to the proof of Theorem 8.2 except that references to mBJM, NR-mBJM and cNR-mBJM games (and the resulting notation) are

replaced by BJM, NR-BJM and cNR-BJM games (and the resulting notation), and the term partners is replaced by matching oracles. The changes to the security model otherwise have no effect on the proof of the theorem. □

The ID-BJM model is identical to the BJM model except for the way that participants in the game are initialized and private keys are extracted. Therefore NR-ID-BJM and cNR-ID-BJM games can be defined in exactly the same way as the NR-BJM and cNR-BJM games are defined, and the session string decisional problem, the related protocol π and strong partnering can all be defined exactly as in the BJM model.

It is therefore easy to see that Theorems 8.3 and 8.4 apply equally well to the ID-BJM model as they do to the BJM model.

8.5 Applying the Technique to Existing Protocols

We now consider Protocol 3 from Section 6.3.2 (originally presented in [16, Protocol 4]). This protocol was conjectured to be secure in [16] but this has never been proven. In fact there is not even a partial proof of security for this protocol.

However, when modified to ensure strong partnering, the protocol can be proven secure in the mBJM model. We now present the modified version of Protocol 3 which we refer to as Protocol 4.

As in Section 6.3.2, the modified key agreement protocol requires a Setup algorithm which generates large primes p and q where $q|p-1$. The group G is chosen to be a multiplicative subgroup of $\mathbb{Z}_p^*$, where G has order q, and element $g \in G$ generates G. A cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^l$ for a fixed value l (usually the security parameter) is also selected.

Suppose that A and B are participants with public and private key pairs $\langle X_A, x_A \rangle$ and $\langle X_B, x_B \rangle$ respectively, where x_A and x_B are chosen randomly from $\mathbb{Z}_q^*$, $X_A = g^{x_A} \bmod p$ and $X_B = g^{x_B} \bmod p$. A and B run Protocol 4 to generate a shared session key.

It is easy to see that in Protocol 4 we have

$$K_A = K_B = K = H(g^{x_A b} \bmod p, g^{x_B a} \bmod p, A, B, T_A, T_B).$$

So K can be used as a secret session key shared between A and B. The ephemeral values a and b are erased on completion of the protocol.

The modified version of Protocol 4 in which the session key is equal to the session string of Protocol 4 is denoted by Protocol 4′.

We now establish the security of Protocol 4 in the mBJM model using Theorem 8.2. To do this we need to establish a pair of lemmas.

Protocol 4: Modified from Protocol 3 to ensure strong partnering.

The following steps must be taken each time a session key is required:

1. A initiates session Π^i_A, setting $pid^i_A = B$ and $role^i_A = initiator$. A selects an ephemeral random value $a \in \mathbb{Z}_q$,
2. B initiates session Π^j_B setting $pid^j_B = A$ and $role^j_B = responder$. B selects an ephemeral random value $b \in \mathbb{Z}_q$.

A and B then exchange the following messages, in the following order:

$$A \longrightarrow B : T_A = g^a \bmod p$$

$$B \longrightarrow A : T_B = g^b \bmod p$$

On receipt of the message $g^b \bmod p$, A checks that $X_B, T_B \in G$ and computes

$$sid^i_A = A, B, T_A, T_B \quad \text{and} \quad K_A = H(T_B^{x_A} \bmod p, X_B^a \bmod p, sid^i_A)$$

and accepts with session key $sk^i_A = K_A$. On receipt of $g^a \bmod p$, B checks that $X_A, T_A \in G$ and computes

$$sid^j_B = A, B, T_A, T_B \quad \text{and} \quad K_B = H(X_A^b \bmod p, T_A^{x_B} \bmod p, sid^j_B)$$

and accepts with session key $sk^j_B = K_B$.

Lemma 8.5 The cNR-mBJM security of Protocol $4'$ is probabilistic polynomial time reducible to the hardness of the CDH problem in G.

Proof: We assume that for security parameter l there exists an adversary E for Protocol $4'$ that can win the cNR-mBJM game with non-negligible advantage η and in polynomial time τ. Suppose that the number of participants in E's game is n_P and that the maximum number of sessions each participant may be involved in is n_S, where n_P and n_S are polynomial functions of l.

We now construct from E an algorithm F which solves the CDH problem in G with non-negligible probability. That is, given as input elements $g, g^x, g^y \in G$, F's task is to compute and output the value $g^{xy} \bmod p$.

F simulates a challenger in a cNR-mBJM game with E. F sets up the game with the group G and generator $g \in G$. F generates a set of participants of size n_P. For each

participant I, F sets I's private key to be a randomly chosen $x_I \in \mathbb{Z}_q$ and sets I's public key to be $X_I = g^{x_I} \bmod p$. However for some randomly selected participant P, F sets P's public key to be $X_P = g^x$. F also picks a random participant $Q \neq P$, and a random session number $t \in \{1, .., n_S\}$. F starts E and answers E's queries as follows.

Send: E may make a special **Send** query Π_I^s which sets $pid_I = X_{I'}$ and instructs I to initiate a protocol run with its partner I'. E can also send any oracle Π_I^s a message M, and the oracle responds according to the protocol. However if E initializes or sends a message to oracle Π_Q^t, then Π_Q^t outputs g^y.

Corrupt(U)**:** If E corrupts participant P, then F aborts. Otherwise F gives E the long-term private key of the participant.

The probability that E chooses oracle Π_Q^t for the **Test** query and that $pid_Q = X_P$ is at least $\frac{1}{n_P^2.n_S}$. In this case, we note that E could not have corrupted participant P, and so F would not have aborted.

E finally outputs a session key of the form (a, b, c) where $a, b \in G$ and $c \in G^4$. If Π_I^s was an initiator, then F outputs b as its guess for the value $g^{xy} \bmod p$, otherwise F outputs a as its guess. It is now easy to see that F solves the CDH problem on input g, g^x, g^y with probability at least $\eta' = \eta.(\frac{1}{n_P^2.n_S})$ (which is non-negligible in l), and in time τ. □

It is interesting to note how short the proof of this theorem is; this is due to the simplicity of the cNR-mBJM model.

We note that a common error when proving that a protocol Π is mBJM-secure (or even NR-mBJM or cNR-mBJM secure) is to make the assumption that the **Test** query oracle Π_U^i has a partner, and that the input to Π_U^i comes from this partner. In fact the challenger has no control over the input to Π_U^i since the adversary controls all communications between oracles. This error can be seen in papers such as [40, 84] where proofs of security were attempted in the full security model. Their corrected versions [39, 83] only provide proofs in the NR versions of the original models.

Lemma 8.6 Protocol 4 has strong partnering in the random oracle model.

Proof: With overwhelming probability, the session ID is unique to each session since it includes the values T_A and T_B which are randomly chosen by each oracle. It is also trivial to verify that appropriate partnering information is included in the session string, and therefore strong partnering is guaranteed. We leave the details to the reader. □

Corollary 8.7 Protocol 4 is secure in the random oracle model assuming the hardness of the Gap Diffie-Hellman problem in G.

Proof: This result comes immediately from Theorem 8.2, and Lemmas 8.5 and 8.6, and the simple observation that the session string decisional problem for Protocol 4 is reducible to the decisional Diffie-Hellman problem (in linear time).

□

8.5.1 Notes on Protocol 4

We note that Protocol 4 can easily be extended to have perfect forward security by including the value $g^{ab} \bmod p$ in the session string. This extended Protocol 4 can then be proven secure in an extended mBJM model which models perfect forward secrecy.

In Protocol 4, each participant sets their *pid* at the start of the protocol. However without some prior communication, there may be confusion as to the identity of the protocol participants. In this case, the participants may have incorrect *pid* values, even in a protocol run between two honest participants, and the participants will fail to generate the same session key.

This does not affect the security of the protocol in the mBJM model, but it is not an acceptable situation in practice. It is therefore advisable to include the participant identities in the message flows and for the responder's *pid* to be set after receipt of the message from the initiator.

In the mBJM security model, it is assumed that public keys are properly authenticated, and a participant's identity is securely bound to their public key via some PKI. However in reality, we may not be able to make this assumption, and this can lead to attacks such as unknown key-share attacks [72, 85]. It may therefore also be advisable to include the identities of the protocol participants as well as their public keys in the computation of the session key.

8.6 Applying the Technique to Protocols with Partial Proofs

Our technique can also be applied to key agreement protocols in the literature with only partial proofs. We find numerous protocols [16, 39, 83] which use a hash function to derive a session key and which have proofs of security reducing to the hardness of some

computational problem f, but only in the NR version of the security model used[1]. If the necessary conditions are met for such protocols, full proofs of security in the relevant model can be obtained as follows.

1. It must be shown that the protocol Π has strong partnering. If Π does not have strong partnering, this can be achieved by modifying the protocol to include the appropriate partnering information (for the security model used) in the session string. It should be checked that such modifications do not affect the existing proof of security.

2. The appropriate version of Theorem 8.1 can now be applied to Π to guarantee that the related protocol π is secure in the cNR version of the security model used.

3. It must be shown that a decisional oracle for f can be used to solve the session string decisional problem of Π.

4. The appropriate version of Theorem 8.2 may now be used to obtain a complete security proof for Π in the full version of the security model used. Security will depend on the hardness of the Gap problem of f.

The proof of security for [16, Protocol 3] can be completed in the manner described above, although the protocol does require some modifications to achieve strong partnering. A suitably modified version of this protocol is in fact presented in [71] together with a proof of security. Interestingly, [16, Protocol 3] and the modified version in [71] are vulnerable to a key compromise impersonation attack. However this does not affect the proof of security since the model of [16] does not capture security against these attacks.

Unfortunately, the partial proofs for the protocols in [39, 83] cannot be completed using our modular technique. This is because the partial proof of security in [83] is in fact incorrect, and the session string decisional problem is not reducible to the decisional problem of f in [39]. In the next section we show how to adapt our modular technique to the protocol of [39].

8.7 When the Modular Technique Cannot be Used

When applying the modular technique to prove the security of a given protocol, various conditions must hold in order to obtain a full proof of security. However the protocol under

[1] A proof for the protocol of [39] appearing in [43] allows the adversary to make some but not all Reveal queries

consideration may not satisfy these conditions, and in such circumstances the modular technique cannot be applied. However it may still be possible to obtain a full proof of security for the protocol using similar ideas.

As examples, we consider the AK protocol of Chen and Kudla (Protocol 2 in [39, 40]) which is proven secure in the NR-ID-BJM model, and the AK protocol of Smart [103], which until now has not been proven secure.

In order to define these two protocols, we first need to introduce some mathematical background. We introduce the notion of bilinear maps (or pairings) on elliptic curves. Pairings have been used extensively in the construction of identity-based cryptographic primitives, and they are required for both the protocols of Chen and Kudla and Smart.

8.7.1 Pairings and Related Problems

Let G_1 and G_2 denote two groups of prime order q, where G_1, with an additive notation, denotes a subgroup of the group of points on an elliptic curve; and G_2, with a multiplicative notation, denotes a subgroup of the multiplicative group of a finite field. A pairing is a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ between these two groups. The map must satisfy the following properties:

Bilinear: Given $P, Q, W \in G_1$ then

$$\hat{e}(P+Q, W) = \hat{e}(P, W) \cdot \hat{e}(Q, W), \text{ and } \hat{e}(P, Q+W) = \hat{e}(P, Q) \cdot \hat{e}(P, W),$$

and consequently for any $a \in \mathbb{Z}_q$, $\hat{e}(aP, Q) = \hat{e}(P, aQ) = \hat{e}(P, Q)^a$.

Non-degenerate: There exists a $P \in G_1$ such that $\hat{e}(P, P) \neq 1$.

Computable: If $P, Q \in G_1$, then $\hat{e}(P, Q) \in G_2$ is efficiently computable.

A bilinear map satisfying the above properties is called an *admissible* bilinear map. An admissible bilinear map can obtained by modifying either the Weil pairing [86] or the Tate pairing [54].

The main computational problem associated with pairings is the *computational bilinear Diffie-Hellman problem* (CBDH problem), and is defined as follows.

Definition 8.6 Let G_1 and G_2 denote two groups of prime order q, let P be a generator of G_1, and let $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be an admissible bilinear map. The CBDH problem in $\langle G_1, G_2, \hat{e} \rangle$ is as follows: Given $P, xP, yP, zP \in G_1$ for some $x, y, z \in \mathbb{Z}_q^*$, compute $W = \hat{e}(P, P)^{xyz} \in G_2$.

We say that algorithm A has advantage ϵ in solving the CBDH Problem in $\langle G_1, G_2, \hat{e}\rangle$ if

$$Pr[A(P, xP, yP, zP) = \hat{e}(P, P)^{xyz}] \geq \epsilon$$

This probability is measured over the random choices of $P, xP, yP, zP \in G$ and the random inputs of A, if any.

The decisional problem arising from the CBDH problem is the *decisional bilinear Diffie-Hellman problem* (DBDH problem), and is defined as follows.

Definition 8.7 Let G_1 and G_2 denote two groups of prime order q, let P be a generator of G_1, and let $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be an admissible bilinear map. Given $P, xP, yP, zP \in G_1$ for some $x, y, z \in \mathbb{Z}_q^*$, as well as $W \in G_2$, the DBDH problem in $\langle G_1, G_2, \hat{e}\rangle$ is to determine if $\hat{e}(P, P)^{wxy} = W$.

The CBDH and DBDH problems can be used to define a related Gap problem.

Definition 8.8 Let G_1 and G_2 denote two groups of prime order q, let P be a generator of G_1, and let $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be an admissible bilinear map. The Gap bilinear Diffie-Hellman (GBDH) problem in $\langle G_1, G_2, \hat{e}\rangle$ is as follows: Given $P, xP, yP, zP \in G_1$ for some $x, y, z \in \mathbb{Z}_q^*$, as well as an oracle that solves the DBDH problem in $\langle G_1, G_2, \hat{e}\rangle$, compute $W = \hat{e}(P, P)^{xyz} \in G_2$.

Informally, the computational, decisional and Gap bilinear Diffie-Hellman assumptions are that no polynomially bounded adversary has non-negligible advantage in solving the computational, decisional and Gap bilinear Diffie-Hellman problems, respectively.

8.7.2 The Chen-Kudla Protocol

We start by considering the AK protocol of Chen and Kudla (AK-CK protocol) [39, Protocol 2]. Although this protocol has a partial proof of security, the session string decisional problem is not reducible to the appropriate decisional problem. This obstacle prevents us from applying the identity-based version of Theorem 8.4 to obtain a full proof of security. However, after slight modifications to ensure strong partnering, it is still possible to obtain a full proof of security for this protocol.

The identity-based AK-CK protocol [39, 40] requires a trusted authority (TA) from which each protocol participant can acquire their private identity-based key.

To provide a private key generation service, the TA selects groups G_1 and G_2 of prime order q, a generator P of G_1, and an admissible map $\hat{e} : G_1 \times G_1 \rightarrow G_2$. The TA selects a

master private key s chosen randomly from $\mathbb{Z}_q^*$. The TA also selects a cryptographic hash function $H_1 : \{0,1\}^* \to G_1$. The public parameters of the TA are descriptions of G_1, G_2 and $\langle P, sP, \hat{e}, H_1 \rangle$. When a user A requests the private key for their identity A, the TA issues a private key $S_A = sH_1(A)$.

If two participants with identities A and B (we refer to these participants simply as A and B) wish to share a session key, they obtain the public parameters of the TA and select an additional cryptographic hash function $H_2 : \{0,1\}^* \to \{0,1\}^l$ for a fixed value l (usually the security parameter). We assume that A and B have corresponding private keys S_A and S_B that are issued by the TA. The AK-CK protocol (modified for strong partnering) is shown in Protocol 5.

Protocol 5: A Modification of the AK-CK Protocol to ensure strong partnering.

The following steps must be taken each time a session key is required:

1. A selects an ephemeral random value $a \in \mathbb{Z}_q^*$,
2. B selects an ephemeral random value $b \in \mathbb{Z}_q^*$.

A and B then exchange the following messages, in the following order:

$$A \longrightarrow B : T_A = aH_1(A)$$

$$B \longrightarrow A : T_B = bH_1(B)$$

On receipt of T_B, A checks that $T_B \in G_1$ and computes

$$K_A = H_2(\hat{e}(S_A, T_B + aH_1(B)), A, B, T_A, T_B)$$

and accepts with session key $sk^i_{A,B} = K_A$. On receipt of T_A, B checks that $T_A \in G_1$ and computes

$$K_B = H_2(\hat{e}(T_A + bH_1(A), S_B), A, B, T_A, T_B)$$

and accepts with session key $sk^j_{B,A} = K_B$.

It is easy to see that in Protocol 5 we have

$$K_A = K_B = K = H_2(\hat{e}(H_1(A), H_1(B))^{s(a+b)}, A, B, T_A, T_B))$$

which can be used as a secret session key shared between A and B. The ephemeral values a and b are erased on completion of the protocol.

Protocol 5 differs from the AK-CK protocol in that the values A, B, T_A and T_B are included in the session string to ensure that the protocol has strong partnering.

Lemma 8.8 Protocol 5 has strong partnering in the random oracle model.

Proof: It is trivial to verify that this condition holds because the ordered protocol transcript and participant IDs (partnering information) are included in the session string. We leave the details to the reader. □

Although we cannot use our modular technique to prove the security of Protocol 5, we can still obtain a proof of security for Protocol 5 in the ID-BJM model. To do this, we use some of the techniques used in the proof of Theorem 8.2, and the security of the protocol relies on the hardness of the GBDH assumption.

Theorem 8.9 Protocol 5 is ID-BJM AK secure, assuming the hardness of the Gap bilinear Diffie-Hellman problem in $\langle G_1, G_2, \hat{e}\rangle$, and assuming that H_1 and H_2 are random oracles.

Proof:

Conditions 1 and 2(a) of Definition 7.3 are trivial to verify. It remains to prove that condition 2(b) holds.

We assume that there exists an adversary E who can win the ID-BJM game with non-negligible advantage $\eta(l)$ (where l is the security parameter), making at most μ_1 queries to the H_1 random oracle, μ_2 queries to the H_2 random oracle, and where E initiates at most μ_S sessions (i.e. for any oracle $\Pi^n_{I,J}$, $n \in \{1, ..., \mu_S\}$).

We now construct from E an algorithm F which solves the CBDH problem with non-negligible probability. F takes as input descriptions of the two groups G_1, G_2, the bilinear map $\hat{e}$, a generator P of G_1, and a triple of elements $xP, yP, zP \in G_1$ with $x, y, z \in \mathbb{Z}^*_q$ where q is the prime order of G_1 and G_2. In addition, F has access to a DBDH oracle, which on input any $\langle aP, bP, cP, W\rangle$ outputs 1 if $W = \hat{e}(P, P)^{abc}$ and 0 otherwise. F's task is to compute and output the value $g^{xyz} \in G_2$ where $g = \hat{e}(P, P)$.

Using similar techniques to the proof of Theorem 8.2, F will use the DBDH oracle to ensure consistency between the way that the H_2 oracle is simulated and the way that **Reveal** queries are answered. Due to the way that F simulates the game, F will not able to compute the session keys of all oracles, and E may be able to detect this by making H_2 queries on the appropriate session strings. However F will use the DBDH oracle to detect such session strings, and will therefore be able to answer **Reveal** and H_2 queries in a consistent manner.

F chooses distinct random integers u and v from $\{1, ..., \mu_1\}$ and a value $p \in \{1, ..., \mu_S\}$. F gives E the public parameters consisting of the descriptions of the groups G_1, G_2, the parameters $\langle P, xP, \hat{e}\rangle$, and access to the random oracles H_1 and H_2. F will simulate all

oracles required during the game and answers all E's queries as follows.

H_1 **queries:** F simulates the random oracle H_1 by keeping a list of tuples $\langle I, r_I Q_I \rangle$ which is called the H_1-List. When the H_1 oracle is queried with an input $I \in \{0,1\}^*$, F responds as follows. If I is already on the H_1-List in the tuple $\langle I, r_I, Q_I \rangle$, then F outputs Q_I. Otherwise:

1. If I is the v-th distinct H_1 query, then the oracle outputs $Q_I = yP$ and adds the tuple $\langle I, \perp, Q_I \rangle$ to the H_1 list. Otherwise F selects a random $r_I \in \mathbb{Z}_q^*$ and outputs $Q_I = r_I P$, and then adds the tuple $\langle I, r_I, Q_I \rangle$ to the H_1 list.
2. In addition, F sets up a new participant I with public key Q_I and private key $S_I = r_I x P$.

We assume that U was the u-th distinct identity to be queried on H_1, and that V was the v-th distinct identity to be queried on H_1. We note that for participant V, where $Q_V = yP$, F is unable to compute the private key S_V.

As in the proof of Theorem 8.2, F maintains a guess session key list, called the G-List. For each oracle of the form $\Pi^n_{V,J}$ (for any participant J) there is an entry on the G-List of the form $\langle n, J, sk^n_{V,J} \rangle$ where $sk^n_{V,J}$ is the session key of oracle $\Pi^n_{V,J}$. Initially the session keys on the list are set to $\perp$.

Send queries: E can send a message m to any oracle $\Pi^n_{I,J}$. If $\Pi^n_{I,J} = \Pi^p_{U,V}$, then F retrieves the value r_U on the H_1 List such that $H_1(U) = r_U P$ and outputs $zH_1(U) = r_U zP$. Otherwise F chooses a random $\alpha \in \mathbb{Z}_q^*$ and outputs $\alpha H_1(I)$.

H_2 **queries:** E can query any string s on the H_2 random oracle. In order to answer these queries, F maintains an H_2-List of tuples $\langle s, t \rangle$ where $t \in \{0,1\}^l$.

If s is already on the H_1-List in the tuple $\langle s, t \rangle$, then F outputs t. Otherwise if s is not of the form g, V, J, T_V, T_J or g, J, V, T_J, T_V for some participant J and some $g \in G_2$, then F selects a random $t \in \{0,1\}^l$, adds the tuple $\langle s, t \rangle$ to the H_2-List and outputs t.

Suppose that s is of the form g, V, J, T_V, T_J or g, J, V, T_J, T_V for some participant J and some $g \in G_2$. Then for each oracle $\Pi^n_{V,J}$ for any $n \in \{1, ..., \mu_S\}$ on the G-List with $sk^n_{V,J} \neq \perp$, F proceeds as follows. If $\Pi^n_{V,J}$ received a message T_J and output $T_V = \alpha Q_V$, then F submits the tuple $\langle yP, \alpha H_1(J) + T_J, xP, g \rangle$ to the DBDH oracle and receives a response b.

If, for any of these queries to the DBDH oracle, $b = 1$, then F outputs $sk^n_{V,J}$. Otherwise F outputs a random $t \in \{0,1\}^l$, adds the tuple $\langle s, t\rangle$ to the H_2-List and outputs t.

Reveal queries: E can request the session key of any oracle $\Pi^n_{I,J}$ that has accepted.

If $\Pi^n_{I,J} = \Pi^p_{U,V}$ then F terminates E and aborts. If $\Pi^n_{I,J} \neq \Pi^n_{V,J}$ (for all J), then F outputs $sk^n_{I,J}$. Otherwise, we have that $\Pi^n_{I,J} = \Pi^n_{V,J}$ for some participant J, and we assume that $\Pi^n_{V,J}$ received a message T_J and output $T_V = \alpha Q_V$. F proceeds as follows.

F considers each tuple $\langle s, t\rangle$ on the H_2-List where s is of the form g, V, J, T_V, T_J or g, J, V, T_J, T_V for some $g \in G_2$. For each such tuple on the H_2-List, F submits the tuple $\langle yP, \alpha H_1(J) + T_J, xP, g\rangle$ to the DBDH oracle and receives a response b.

If, for any of these queries to the DBDH oracle, $b = 1$, then F sets $sk = t$ where $t = H_2(s)$. Otherwise F selects a random $sk \in \{0,1\}^l$. F then sets the value $sk^n_{V,J} = sk$ on the G-List and outputs sk.

Private Key Extract queries: On input an identity I, F queries I on the H_1 oracle. If $I \neq V$ then F retrieves the value r_I from the tuple $\langle I, r_I, Q_I\rangle$ on the H_1 list, and outputs $S_I = r_I xP$. Otherwise F terminates E and aborts.

Test query: At some point in the simulation, E will ask a Test query of some oracle. If E does not choose the oracle $\Pi^p_{U,V}$ for the Test query, then F aborts. F simply outputs a random session key $sk^* \in \{0,1\}^l$.

Output: When E has finished querying oracles, E outputs a bit b'.

The probability that F does not abort at some point in the simulation is $1/\mu_1^2\mu_S$ since u and v were chosen randomly from $\{1, ..., \mu_1\}$. In this case, E cannot detect any inconsistency in F's simulation, and has probability η of winning the game where oracle $\Pi^p_{U,V}$ was chosen for the Test query.

If E wins the game where oracle $\Pi^p_{U,V}$ was chosen for the Test query, then with overwhelming probability $1 - 1/2^l$, E queried the session string $ss^p_{U,V}$ on H_2. If this is not the case, then E cannot distinguish between a random key and the true session key for the Test session. F therefore picks a random tuple $\langle s, t\rangle$ on the H_2 List where $s = g, U, V, T_U, T_V$ (if $\Pi^p_{U,V}$ is an initiator oracle) or $s = g, V, U, T_V, T_U$ (if $\Pi^p_{U,V}$ is a responder oracle) and guesses that $s = ss^p_{U,V}$ (this occurs with probability at least $1/\mu_2$).

F retrieves the value r_U on the H_1 List such that $H_1(U) = r_U P$ and computes $S_U = r_U x P$. We recall that $H_1(V) = yP$ and $T_U = r_U z P$, so if $s = ss^p_{U,V}$ then $g = \hat{e}(S_U, T_V + zyP) = \hat{e}(r_U x P, T_V) \cdot \hat{e}(r_U x P, zyP)$. F therefore outputs $(g/\delta)^{1/\gamma}$ as its guess for $\hat{e}(P,P)^{xyz}$, where $\delta = \hat{e}(r_U x P, T_V)$ and $\gamma = r_U$.

F solves the CBDH problem with probability $\eta/\mu_1^2\mu_2\mu_S(1-1/2^l)$ which is non-negligible in l, contradicting the hardness of the CBDH problem.

□

8.7.3 Smart's Protocol

The setup procedure for Smart's AK protocol is identical to the setup for the AK-CK protocol. The public parameters of the TA, as before, are descriptions of G_1, G_2 and $\langle P, sP, \hat{e}, H_1 \rangle$. When a user A requests the private key for their identity A, the TA issues a private key $S_A = sH_1(A)$.

As before, we assume that A and B have corresponding private keys S_A and S_B that are issued by the TA. Smart's AK protocol (modified for strong partnering) is shown in Protocol 6.

Protocol 6: A Modification of Smart's AK Protocol to ensure strong partnering.

The following steps must be taken each time a session key is required:

1. A selects an ephemeral random value $a \in \mathbb{Z}_q^*$,
2. B selects an ephemeral random value $b \in \mathbb{Z}_q^*$.

A and B then exchange the following messages, in the following order:

$$A \longrightarrow B : T_A = aP$$

$$B \longrightarrow A : T_B = bP$$

On receipt of T_B, A checks that $T_B \in G_1$ and computes

$$K_A = H_2(\hat{e}(aH_1(B), sP) \cdot \hat{e}(S_A, T_B), A, B, T_A, T_B)$$

and accepts with session key $sk^i_{A,B} = K_A$. On receipt of T_A, B checks that $T_A \in G_1$ and computes

$$K_B = H_2(\hat{e}(bH_1(A), sP) \cdot \hat{e}(S_B, T_A), A, B, T_A, T_B)$$

and accepts with session key $sk^j_{B,A} = K_B$.

It is easy to see that in Protocol 6 we have

$$K_A = K_B = K = H_2(\hat{e}(bH_1(A) + aH_1(B), sP), A, B, T_A, T_B))$$

which can be used as a secret session key shared between A and B. The ephemeral values a and b are erased on completion of the protocol.

We can show that Protocol 6 has strong partnering, and it is possible to obtain a proof of security of Smart's protocol in the cNR-ID-BJM model assuming the hardness of the CBDH problem. Unfortunately, as with Protocol 5, we cannot apply the modular technique because the session string decisional problem is not reducible to the DBDH problem.

Despite the fact that we cannot make us of the modular technique for Protocol 6, as with Protocol 5, we can still obtain a full proof of security for Protocol 6 in the ID-BJM model using the proof techniques of Theorem 8.2. This proof is very similar to the proof of Theorem 8.9.

However if we make a small modification to Protocol 6, we find that the session string decisional problem is reducible to the DBDH problem, and we can obtain a full proof of security for the resulting protocol using our modular technique. The modified version of Protocol 6 requires exactly the same setup as Protocol 6 and is shown in Protocol 7.

We notice that the only difference between Protocols 6 and 7 is in the generation of the key. In Protocol 6, the outputs of the pairing computations are multiplied together, whereas in Protocol 7, the outputs of the pairing computations are concatenated.

The modified version of Protocol 7 in which the session key is equal to the session string of Protocol 7 is denoted by Protocol 7′.

We now establish the security of Protocol 7 in the ID-BJM model using the identity-based version of Theorem 8.4. To do this we need to establish a pair of lemmas.

Lemma 8.10 The cNR-ID-BJM security of Protocol 7′ is probabilistic polynomial time reducible to the hardness of the CBDH problem in $\langle G_1, G_2, \hat{e}\rangle$.

Proof: We assume that for security parameter l there exists an adversary E for Protocol 7′ that can win the cNR-ID-BJM game with non-negligible advantage $\eta(l)$ (where l is the security parameter), making at most μ_1 queries to H_1, μ_C **Private Key Extract** queries, and where E initiates at most μ_S sessions (i.e. for any oracle $\Pi^n_{I,J}$, $n \in \{1, ..., \mu_S\}$).

We now construct from E an algorithm F which solves the CBDH problem with non-negligible probability. F takes as input descriptions of the two groups G_1, G_2, the bilinear

Protocol 7: A Modification of Protocol 6 to allow modular proofs.

The following steps must be taken each time a session key is required:

1. A selects an ephemeral random value $a \in \mathbb{Z}_q^*$,
2. B selects an ephemeral random value $b \in \mathbb{Z}_q^*$.

A and B then exchange the following messages, in the following order:

$$A \longrightarrow B : T_A = aP$$

$$B \longrightarrow A : T_B = bP$$

On receipt of T_B, A checks that $T_B \in G_1$ and computes

$$K_A = H_2(\hat{e}(aH_1(B), sP), \hat{e}(S_A, T_B), A, B, T_A, T_B)$$

and accepts with session key $sk^i_{A,B} = K_A$. On receipt of T_A, B checks that $T_A \in G_1$ and computes

$$K_B = H_2(\hat{e}(bH_1(A), sP), \hat{e}(S_B, T_A), A, B, T_A, T_B)$$

and accepts with session key $sk^j_{B,A} = K_B$.

map $\hat{e}$, a generator P of G_1, and a triple of elements $xP, yP, zP \in G_1$ with $x, y, z \in \mathbb{Z}_q^*$ where q is the prime order of G_1 and G_2. F's task is to compute and output the value $g^{xyz} \in G_2$ where $g = \hat{e}(P, P)$.

F simulates a challenger in a cNR-ID-BJM game with E. F chooses distinct random integers u and v from $\{1, ..., \mu_1\}$ and a value $p \in \{1, ..., \mu_S\}$. F gives E the public parameters consisting of the descriptions of the groups G_1, G_2, the parameters $\langle P, xP, \hat{e}\rangle$, and access to the random oracle H_1. F will simulate all oracles required during the game and answers all E's queries as follows.

H_1 **queries:** F simulates the random oracle H_1 by keeping a list of tuples $\langle I, r_I Q_I\rangle$ which is called the H_1-List. When the H_1 oracle is queried with an input $I \in \{0,1\}^*$, F responds as follows. If I is already on the H_1-List in the tuple $\langle I, r_I, Q_I\rangle$, then F outputs Q_I. Otherwise:

1. If I is the v-th distinct H_1 query, then the oracle outputs $Q_I = yP$ and adds the tuple $\langle I, \perp, Q_I\rangle$ to the H_1 list. Otherwise F selects a random $r_I \in \mathbb{Z}_q^*$ and outputs $Q_I = r_I P$, and then adds the tuple $\langle I, r_I, Q_I\rangle$ to the H_1 list.
2. In addition, F sets up a new participant I with public key Q_I and private key

$S_I = r_I xP$.

We assume that U was the u-th distinct identity to be queried on H_1, and that V was the v-th distinct identity to be queried on H_1. We note that for participant V, where $Q_V = yP$, F is unable to compute the private key S_V.

Send queries: E can send a message m to any oracle $\Pi^n_{I,J}$. If $\Pi^n_{I,J} = \Pi^p_{U,V}$, then F outputs zP. Otherwise F chooses a random $\alpha \in \mathbb{Z}^*_q$ and outputs αP.

Private Key Extract queries: On input an identity I, F queries I on the H_1 oracle. If $I \neq V$ then F retrieves the value r_I from the tuple $\langle I, r_I, Q_I \rangle$ on the H_1 list, and outputs $S_I = r_I xP$. Otherwise F terminates E and aborts.

The probability that E chooses oracle $\Pi^p_{U,V}$ for the **Test** query is at least $\frac{1}{\mu_1^2 \cdot \mu_S}$. In this case, we note that E could not have made a **Private key extract** query on V, so F would not have aborted.

E finally outputs a session key of the form (a, b, I, J, c, d) where $a, b \in G_2$, $I, J \in \{0,1\}^*$ and $c, d \in G_1$. If $\Pi^p_{U,V}$ was an initiator, then F outputs a as its guess for the value $g^{xyz} \in G_2$ where $g = \hat{e}(P, P)$, otherwise F outputs b as its guess. It is now easy to see that F solves the CBDH problem on input $xP, yP, zP \in G_1$ with probability at least $\eta' = \frac{\eta}{\mu_1^2 \cdot \mu_S}$ (which is non-negligible in l). □

Lemma 8.11 Protocol 7 has strong partnering in the random oracle model.

Proof: It is trivial to verify that this condition holds because the ordered protocol transcript and participant IDs (partnering information) are included in the session string. We leave the details to the reader. □

Corollary 8.12 Protocol 7 is secure in the ID-BJM model assuming that H_1 and H_2 are random oracles, and assuming the hardness of the Gap Bilinear Diffie-Hellman problem in $\langle G_1, G_2, \hat{e} \rangle$.

Proof: This result comes immediately from the identity-based version of Theorem 8.4, and Lemmas 8.10 and 8.11, and the simple observation that the session string decisional problem for Protocol 7 is reducible to the DBDH problem (in constant time). □

8.8 Special Gap Groups

The Gap assumptions may not be acceptable to all, since in developing security proofs, one must assume the use of an oracle which is not known to exist: a decisional oracle. For instance, for Protocol 1, the proof of security ultimately requires an oracle which solves DDH in the group G. This is thought to be a hard problem, so there is no known method of constructing such an efficient oracle. Nevertheless, the security proof still gives a concrete basis for assessing the security of the protocol.

However there do exist groups in which the computational problem is thought to be hard but where the decisional problem is known to be easy, for instance, groups of points on an elliptic curve on which an efficient bilinear map (or pairing operation) is defined. In such groups, the pairing operation can be used to construct an efficient DDH oracle, and the Gap problem is in fact equivalent to the computational problem. Therefore if Protocol 1 had been defined over such a group, then its security would in fact reduce to the CDH problem in that group.

Of course, one may object to using a group in which the decisional problem is known to be easy since this may indicate other, yet to be discovered weaknesses of the group.

8.9 Conclusions and Open Problems

We have presented a modular technique that makes use of Gap assumptions for simplifying proofs of security for key agreement protocols which are not built using the modular approach of [8]. Protocols of this type have traditionally been notoriously hard to prove secure, and we have indicated how the proofs of security of many such protocols in the literature may be constructed or completed using our technique. Our technique works not only with the model presented in this chapter, but also with the models of [12, 13, 16, 32].

We considered in detail a long-standing protocol presented in [16] which previously lacked a proof of security. We then provided a full proof of security for a slightly modified version of this protocol using the techniques introduced in this chapter. We also considered protocols in [16, 39, 103] which up till now had only partial proofs of security or no security proofs at all. For such protocols we indicated how full proofs of security may be constructed (either using our modular technique, or the proof techniques used in this chapter), and we provide full proofs of security for the (slightly modified) AK protocols of [39] and [103].

In future work, it would be interesting to extend the idea of modular proofs to multi-party protocols as well as other types of key agreement protocols such as key agreement

protocols with key confirmation. It would also be interesting to investigate the use of Gap assumptions in other cryptographic primitives, such as encryption or signature schemes, and extend the idea of modular proof techniques to these primitives.

Bibliography

[1] M. Abdalla, O. Chevassut, and D. Pointcheval. One-time verifier-based encrypted key exchange. In S. Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 47–64. Springer-Verlag, 2005.

[2] M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer-Verlag, 2002.

[3] S. Al-Riyami and K. Paterson. Authenticated three party key agreement protocols from pairings. In K. Paterson, editor, *Proceedings of 9th IMA International Conference on Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 332–359. Springer-Verlag, 2003.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606. Springer-Verlag, 1998.

[5] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[6] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In U. Montanari et al., editor, *ICALP '00: Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 524–535. Springer-Verlag, 2000.

[7] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer-Verlag, 2004.

[8] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing STOC*, pages 419–428. ACM, 1998.

[9] M. Bellare, E. Petrank, C. Rackoff, and P. Rogaway. Authenticated key exchange in the public key model. Manuscript, 1995-96.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

[11] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[12] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology - CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1994.

[13] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing STOC*, pages 57–66. ACM, 1995.

[14] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. 3rd Theory of Cryptography Conference - TCC 2006 (to appear), 2006. Available at `http://eprint.iacr.org/2005/304`.

[15] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and reduced SHA-1. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer-Verlag, 2005.

[16] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In M. Darnell, editor, *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer-Verlag, 1997.

[17] S. Blake-Wilson and A. Menezes. Entity authentication and key transport protocols employing asymmetric techniques. In *Security Protocols Workshop*, 1997.

[18] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer-Verlag, 2001.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer-Verlag, 2003.

[20] D. Boneh and R. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In N. Koblitz, editor, *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer-Verlag, 1996.

[21] D. Boneh and M. Naor. Timed commitments. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[22] J. Boyar, D. Chaum, I. Damgård, and T. P. Pedersen. Convertible undeniable signatures. In A. Menezes and S. Vanstone, editors, *Advances in Cryptology – CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 189–205. Springer-Verlag, 1991.

[23] C. Boyd, W. Mao, and K. Paterson. Key agreement using statically keyed authenticators. In M. Jakobsson et al., editor, *Applied Cryptography and Network Security: Second International Conference, ACNS 2004*, volume 3089 of *Lecture Notes in Computer Science*, pages 388–401. Springer-Verlag, 2004.

[24] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[25] E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures for ad-hoc groups. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 465–480. Springer-Verlag, 2002.

[26] J. Camenisch. Efficient and generalized group signatures. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer-Verlag, 1997.

[27] J. Camenisch and M. Michels. Confirmer signature schemes secure against adaptive adversaries. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, 2000.

[28] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer-Verlag, 2003.

[29] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, last updated January 2005, 2000. Available at `http://eprint.iacr.org/`.

[30] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science - FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[31] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer-Verlag, 2005.

[32] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, 2001.

[33] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 2002.

[34] D. Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO 1986*, volume 263 of *LNCS*, pages 195–199. Springer-Verlag, 1986.

[35] D. Chaum. Zero-knowledge undeniable signatures. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 458–464. Springer-Verlag, 1990.

[36] D. Chaum. Designated confirmer signatures. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 86–91. Springer-Verlag, 1994.

[37] D. Chaum and H. van Antwerpen. Undeniable signatures. In G. Brassard, editor, *Advances in Cryptology – CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer-Verlag, 1990.

[38] D. Chaum and E. van Heyst. Group signatures. In D. Davies, editor, *Advances in Cryptology – EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[39] L. Chen and C. Kudla. Identity based authenticated key agreement from pairings. Cryptology ePrint Archive, Report 2002/184, 2002. Available at `http://eprint.iacr.org/`.

[40] L. Chen and C. Kudla. Identity based authenticated key agreement from pairings. In *IEEE Computer Security Foundations Workshop – CSFW-16 2003*, pages 219–233. IEEE Computer Society Press, 2003.

[41] L. Chen, C. Kudla, and K. Paterson. Concurrent signatures. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 287–305. Springer-Verlag, 2004.

[42] M. Cherepnev. On the connection between the discrete logarithms and the Diffie-Hellman problem. Discrete Math. Appl., 1996.

[43] K.-K. Choo, C. Boyd, and Y. Hitchcock. On session key construction in provably-secure key establishment protocols. In S. Vaudenay, editor, *Proceedings of International Conference on Cryptology in Malaysia - Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, page 116 131. Springer-Verlag, 2005. Available at `http://eprint.iacr.org/2005/206`.

[44] S. Chow and W. Susilo. Generic construction of (identity-based) perfect concurrent signatures. In S. Qing et al., editor, *Proceedings of the 7th International Conference on Information and Communications Security - ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 194 – 206. Springer-Verlag, 2005.

[45] D. Coppersmith and I. Shparlinski. On polynomial approximation and the parallel complexity of the discrete logarithm problem and breaking the Diffie-Hellman cryptosystem. Preprint, Nov. 1996.

[46] R. Cramer, I. Damgard, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO 1994*, volume 893 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1995.

[47] I. Damgård and T. Pedersen. New convertible undeniable signature schemes. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, 1996.

[48] B. den Boer. Diffie-Hellman is as strong as discrete log for certain primes. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 530–539. Springer-Verlag, 1989.

[49] Y. Desmedt and M. Yung. Weakness of undeniable signature schemes. In D. Davies, editor, *Advances in Cryptology – EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, 1991.

[50] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[51] W. Diffie, P. C. van Oorschot, and M. J. Weiner. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[52] Y. Dodis and L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In M. Yung, editor, *DRM '03: Proceedings of the 2003 ACM workshop on Digital rights management*, pages 47–54. ACM Press, 2003.

[53] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[54] G. Frey, M. Müller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.

[55] S. D. Galbraith and W. Mao. Invisibility and anonymity of undeniable and confirmer signatures. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, page 8097. Springer-Verlag, 2003.

[56] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In M. Wiener, editor, *Advances in Cryptology – Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.

[57] J. Garay and C. Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In R. Wright, editor, *Financial Cryptography 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 190–207. Springer-Verlag, 2003.

[58] O. Goldreich. A simple protocol for signing contracts. In D. Chaum, editor, *Advances in Cryptology – CRYPTO 1983*, pages 133–136. Plenum Press, 1983.

[59] O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO 1986*, volume 263 of *LNCS*, pages 171–185. Springer-Verlag, 1986.

[60] S. Goldwasser and Y. Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003*, pages 102–113. IEEE Computer Society, 2003.

[61] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[62] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[63] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[64] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *ACM Conference on Computer and Communications Security*, pages 122–131, 1998.

[65] J. Herranz and G. Sáez. Forking lemmas for ring signature schemes. In T. Johansson and S. Maitra, editors, *Proceedings of 5th International Conference on Cryptology in India - INDOCRYPT 2003*, volume 2904 of *Lecture Notes in Computer Science*, pages 266–279. Springer-Verlag, 2003.

[66] J. Herranz and G. Sáez. New ID-based ring signature schemes. In J. Lopez et al., editor, *Proceedings of the 6th International Conference on Information and Communications Security - ICICS'04*, volume 3269 of *Lecture Notes in Computer Science*, pages 27–39. Springer-Verlag, 2004.

[67] Y. Hitchcock, Y. Tin, J. G. Nieto, C. Boyd, and P. Montague. A password-based authenticator: Security proof and applications. In T. Johansson and S. Maitra, editors, *Proceedings of 4th International Conference on Cryptology in India INDOCRYPT 2003*, volume 2904 of *Lecture Notes in Computer Science*, pages 388–401. Springer-Verlag, 2003.

[68] M. Jakobsson. Blackmailing using undeniable signatures. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 425–427. Springer-Verlag, 1994.

[69] M. Jakobsson and D. Pointcheval. Mutual authentication and key exchange protocol for low power devices. In P. Syverson, editor, *Financial Cryptography, 5th International Conference, FC 2001*, volume 2339 of *Lecture Notes in Computer Science*, page 178195. Springer-Verlag, 2002.

[70] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In U. Maurer, editor, *Advances in Cryptology - EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, 1996.

[71] I. Jeong, J. Katz, and D. Lee. One-round protocols for two-party authenticated key exchange. In M. Jakobsson et al., editor, *Applied Cryptography and Network Security: the Second International Conference, ACNS 2004*, volume 3089 of *Lecture Notes in Computer Science*, pages 220 – 232. Springer-Verlag, 2004.

[72] B. Kaliski, Jr. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and Systems Security*, 4(3):275–288, 2001.

[73] C. Kudla and K. Paterson. Modular security proofs for key agreement protocols. In B. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565. Springer-Verlag, 2005.

[74] C. Kudla and K. Paterson. Non-interactive designated verifier proofs and undeniable signatures. In N. Smart, editor, *10th International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 136–154. Springer-Verlag, 2005.

[75] F. Laguillaumie and D. Vergnaud. Designated verifier signatures: Anonymity and efficient construction from *any* bilinear map. In C. Blundo and S. Cimato, edi-

tors, *SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 105–119. Springer-Verlag, 2005.

[76] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[77] H. Lipmaa, G. Wang, and F. Bao. Designated verifier signature schemes: Attacks, new security notions and a new construction. In L. Caires et al., editor, *Automata, Languages and Programming, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 459–471. Springer-Verlag, 2005.

[78] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Security Protocols Workshop*, 1997.

[79] T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key-distribution systems. *Electronics Letters*, E69(2):99–106, 1986.

[80] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In Y. Desmedt, editor, *Advances in Cryptology — CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer-Verlag, 1994.

[81] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *Theory of Cryptography, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer-Verlag, 2004.

[82] U. Maurer and S. Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[83] N. McCullagh and P. Barreto. A new two-party identity-based authenticated key agreement. Cryptology ePrint Archive, Report 2004/122, 2005. Available at `http://eprint.iacr.org/`.

[84] N. McCullagh and P. Barreto. A new two-party identity-based authenticated key agreement. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 262–274. Springer-Verlag, 2005.

[85] A. Menezes. Another look at HMQV. Cryptology ePrint Archive, Report 2005/205, 2005. Available from `http://eprint.iacr.org/2005/205`.

[86] A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[87] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[88] C. Mitchell, M. Ward, and P. Wilson. Key control in key agreement protocols. *Electronics Letters*, 34:980–981, 1998.

[89] K. Nguyen. Asymmetric concurrent signatures. In S. Qing et al., editor, *Proceedings of the 7th International Conference on Information and Communications Security - ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 181 – 193. Springer-Verlag, 2005.

[90] J. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer-Verlag, 2002.

[91] W. Ogata, K. Kurosawa, and S. Heng. The security of the FDH variant of Chaum's undeniable signature scheme. Cryptology ePrint Archive, Report 2004/290, 2004. Available from `http://eprint.iacr.org/2004/290`.

[92] W. Ogata, K. Kurosawa, and S. Heng. The security of the FDH variant of Chaum's undeniable signature scheme. In S. Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 328–345. Springer-Verlag, 2005.

[93] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In K. Kim, editor, *Public Key Cryptography – PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer-Verlag, 2001.

[94] J. Park, E. Chong, and H. Siegel. Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures. In *Proceedings of the*

22nd annual ACM symposium on Principles of Distributed Computing - PODC '03, pages 172–181. ACM Press, 2003.

[95] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 1996.

[96] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, vol. 13, pp. 361–396, 2000.

[97] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer-Verlag, 2001.

[98] S. Saeednia, S. Kremer, and O. Markowitch. An efficient strong designated verifier signature scheme. In J. Lim and D. Lee, editors, *Information Security and Cryptology - ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 40–54. Springer-Verlag, 2003.

[99] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[100] A. Shamir. Identity-based cryptosystems and signature schemes. In G. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1984.

[101] V. Shoup. On formal models for secure key exchange. IBM Technical Report RZ 3120, 1999. Available at `http://shoup.net/papers`.

[102] V. Shoup and A. Rubin. Session key distribution using smart cards. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 321–331. Springer-Verlag, 1996.

[103] N. Smart. An identity based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, 38(13):630–632, 2002.

[104] R. Steinfeld, L. Bull, H. Wang, and J. Pieprzyk. Universal designated-verifier signatures. In C. Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 523–542. Springer-Verlag, 2003.

[105] R. Steinfeld, H. Wang, and J. Pieprzyk. Efficient extension of standard Schnorr/RSA signatures into universal designated-verifier signatures. In F. Bao et al., editor, *PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, 2004.

[106] D. Stinson. *Cryptography: Theory and Practice, Second Edition.* Chapman & Hall/CRC, 2002.

[107] W. Susilo and Y. Mu. Tripartite concurrent signatures. In *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, pages 425 – 441. Kluwer, 2005.

[108] W. Susilo, Y. Mu, and F. Zhang. Perfect concurrent signature schemes. In J. Lopez et al., editor, *Proceedings of the 6th International Conference on Information and Communications Security - ICICS'04*, volume 3269 of *Lecture Notes in Computer Science*, pages 14–26. Springer-Verlag, 2004.

[109] W. Susilo, F. Zhang, and Y. Mu. Identity-based strong designated verifier signature schemes. In H. Wang et al., editor, *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 313–324. Springer-Verlag, 2004.

[110] Y. Tin, C. Boyd, and J. G. Nieto. Provably secure mobile key exchange: Applying the Canetti-Krawczyk approach. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 166–179. Springer-Verlag, 2003.

[111] Y. Tin, H. Vasanta, C. Boyd, and J. G. Nieto. Protocols with security proofs for mobile applications. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 358–369. Springer-Verlag, 2004.

[112] G. Wang. An attack on not-interactive designated verifier proofs for undeniable signatures. Cryptology ePrint Archive, Report 2003/243, 2003. Available from `http://eprint.iacr.org/`.

[113] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2005.

[114] X. Wang, Y. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005.

[115] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.

[116] X. Wang, H. Yu, and Y. Yin. Efficient collision search attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005.

[117] F. Zhang and K. Kim. ID-based blind signature and ring signature from pairings. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 533–547. Springer-Verlag, 2002.