Université Catholique de Louvain
Faculté des Sciences Appliquées
Laboratoire de Microélectronique

Provable Security and Fairness in Cryptographic Identification and Signature Schemes

Julien Cathalo

Thèse soutenue en vue de l'obtention du grade de
Docteur en Sciences Appliquées

Composition du jury:

Pr. Jean-Jacques Quisquater (UCL - DICE) – Promoteur
Dr. Marc Girault (France Telecom)
Dr. François Koeune (UCL - DICE, K2Crypt)
Pr. David Naccache (Université Paris II, École normale supérieure)

Pr. Luc Vandendorpe (UCL - TELE) – Président

Louvain-la-Neuve, Belgique
Septembre 2007

Abstract

Identification schemes are public-key cryptographic primitives that allow an entity (called the prover) to prove his or her identity to another entity. An identification scheme is secure if no attacker can impersonate the prover.

Digital signature schemes allow an entity to produce a signature on a message; given the message and the signature, another entity can check the identity of the signer and verify that the message was not modified. A secure digital signature scheme is such that an attacker can not produce a forgery, i.e. a false signature.

There has been a lot of work aiming at establishing, in a provable manner, the security of such schemes. Given a cryptographic scheme and a security property, one should be able to demonstrate whether the scheme satisfies it or not. The approach is usually the following: when the security property is not satisfied, show it with an efficient attack; when it is satisfied, use a security proof. But even when they are provably secure, cryptographic schemes can be attacked: it can be at the implementation level, or because of a flaw in the proof.

The goal of this thesis is to apply these approaches to study the security of several public-key cryptographic schemes. We study the GPS identification scheme and show how some implementations can be broken by an efficient attack. We show how to securely sign long messages with RSA. We break a fair exchange signature scheme based on GPS and RSA. We consider a new problem called fair identification and propose a fair identification scheme.

Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse, Jean-Jacques Quisquater, de m'avoir accueilli au sein du groupe Crypto, d'avoir encadré mon travail de doctorat et de m'avoir permis de mener ma recherche avec une grande liberté.

Ma reconnaissance va ensuite aux membres de mon jury, Marc Girault, François Koeune et David Naccache, et à Luc Vandendorpe qui a accepté de présider ce jury.

Merci à toutes les personnes qui ont collaboré aux travaux de recherche présentés dans cette thèse : Jean-Sébastien Coron, François Koeune, Benoît Libert, David Naccache, Omkant Pandey et Jean-Jacques Quisquater.

Je tiens aussi à remercier les membres du groupe Crypto, passés et présents, pour l'excellente ambiance de travail et de bonne humeur qui règne au groupe.

Je remercie François Koeune d'avoir pris le temps de relire en détail cette thèse et de m'avoir aidé à l'améliorer par ses conseils pertinents.

Merci à tous mes proches qui m'ont encouragé pendant ces années de doctorat et en particulier à Anne-Hélène pour son soutien permanent.

Contents

Notations and abbreviations

Notations

$\mathbb{Z}$	Set of integers
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}_q$	Set of integers $\{0, 1, \ldots, q-1\}$
$\mathbb{Z}_q^*$	Set of elements having a modular multiplicative inverse in $\mathbb{Z}_q$
$\phi(q)$	Euler function, cardinal of the set $\mathbb{Z}_q^*$
$\in$	Membership of a set
$\in_R$	Membership of a randomly and uniformly distributed sample from a set
$:=$	Assignment of a value to a variable
$\xleftarrow{R}$	Assignment of a random and uniformly chosen value from a set
a div b	Quotient of the euclidian division of a by b
a mod b	Remainder of the euclidian division of a by b
$\equiv$	Congruence modulo an integer
$\oplus$	Bitwise exclusive OR
$\{0,1\}^t$	Set of strings of t bits
$\{0,1\}^*$	Set of strings of arbitrary but finite length
$a\|\|b$	Concatenation of $a \in \{0,1\}^{n_0}$ and $b \in \{0,1\}^{n_1}$; $a\|\|b \in \{0,1\}^{n_0+n_1}$
$W(x)$	Hamming weight of integer x
$d(x, x')$	Hamming distance between x and x' i.e. $W(x \oplus x')$
$\|b\|$	Size in bits of integer b
b_j	j-th bit of integer b, starting from the least significant bit

Introduction

1. Motivation and Scope

Identification schemes and digital signatures are two related kinds of public-key cryptographic primitives. In an identification scheme, a prover convinces a verifier of his identity, while in a signature scheme a signer produces a digital signature that binds him to a document. In both cases, a verifier can check the identity of the other entity, either interactively (in the case of the identification scheme) or non-interactively (in the case of a signature scheme). These kind of tools find many applications today like smart card payment systems for example.

The RSA [57] primitive was initially designed as a public-key encryption and digital signature scheme; it is the most widely used public-key primitive today. The GPS [38, 55] identification scheme is computationally very efficient. Interestingly, these two schemes have a lot in common. Both schemes have their security based on problems related to the integer factorization (the RSA problem in the case of RSA and the discrete logarithm with short exponents problem modulo a composite number in the case of GPS). The similarity in their construction was used several times: in the original GPS paper, the idea was to use RSA with GPS in order to provide self-certified public keys (i.e. public keys that do not need a separate certificate); later, it was used by Markowitch and Saeednia [53] to design a fair exchange scheme based on GPS and RSA.

But there are still many open questions about the security of these two schemes. When trying to evaluate this security, two approaches are possible. In order to prove that a security property is not satisfied, one can describe an attack on the scheme, showing how an attacker can efficiently break the given security property. On the other hand, in order to prove that a given security property is satisfied, one can provide a security proof, demonstrating that no attacker can break the given security property.

Sometimes, schemes have rigorous security proofs but can be broken by side-channel attacks, therefore one needs to know which attacks are possible in order to counter them. Moreover, some new cryptographic problems arise and need their security models and proofs.

In this thesis we apply these different approaches to study the security of several identification and secure schemes related to GPS and RSA.

Moreover, we want to investigate the property of *fairness*. In electronic commerce, it often happens that two (or more) users want to obtain something from each other and agree to make an exchange. Informally, an exchange protocol is *fair* when it ensures that either all the parties get what they want, or none of them gets nothing. This problem has been studied in the case of signatures but the situation where two persons want to fairly exchange proofs identities is a new and distinct problem that is interesting to investigate.

2. Organization of the Work

This thesis is organized as follows. The first part is an introduction to several concepts in public-key cryptography. Four new results are then presented in the second and third part.

The first part introduces all the concepts that are necessary to understand the new results. It provides an introduction to identification and signature schemes, fair exchange schemes, and also discusses practical aspects such as side-channel attacks and efficient implementation.

The second part of the thesis describes two new results about the security of identification and signature schemes. The first result is a new side-channel strategy called *Hamming weight cryptanalysis* and its application to the GPS identification scheme. The second result is about encoding for RSA signature schemes and presents a new technique to encode long messages.

The third part of the thesis presents two new results related to the topic of fairness in cryptography. The first result is an attack that breaks a fair exchange scheme based on GPS and RSA. The second result is a new problem called *fair identification* that is introduced and studied.

Part 1

Preliminaries

CHAPTER 1

Public-Key Cryptography

1. Public-Key Identification and Signature

1.1. Secret-Key vs. Public-Key Cryptography. Cryptography provides tools for the security of communication systems in a wide variety of situations. The main kind of cryptographic primitives are *encryption*, which ensures the confidentiality of a communication, *digital signatures* that are meant to be the electronic equivalent of handwritten signatures and *identification schemes* that allow to verify the identity of a user or a device.

One can make a distinction between two kind of cryptographic schemes: secret-key schemes and public-key schemes.

Secret-key cryptography (also called *symmetric* cryptography) requires that users share a secret. For example, a symmetric encryption scheme allow two entities Alice and Bob to communicate in a confidential way, provided they previously agreed on a secret key. The secret key is used by Alice to encrypt a message then used by Bob to decrypt the message. This secret key is only known to Alice and Bob. The requirement that both entities must know a common secret value is strong, as users must find a way to choose such a value in a secure way.

Public-key cryptography (also called *asymmetric cryptography*), invented in 1976 by Diffie and Hellman [33], aims at removing this need for a common secret. It allows users to perform cryptographic operations without sharing a secret. The principle is that each user has two distinct but related keys: a private key and a public key. The private key is known to the user only and must be kept secret, while the public key is meant to be available to anyone.

1.2. Types of Public-Key Schemes.

1.2.1. *Public-Key Encryption.* An public-key encryption scheme allows users to encrypt messages under the recipient's public key, in such a way that only the intended recipient can decrypt the message using his private key.

It works as follows. Assume that Alice wants to encrypt a message m, called the *plaintext*, that is intended to Bob. Bob has a private key sk and has already published his corresponding public key pk. Alice encrypts the message using an *encryption algorithm* E and Bob's public key. She obtains an encrypted message $c = E_{\mathsf{pk}}(m)$ called the *ciphertext*. When Bob receives the ciphertext, he uses a *decryption algorithm* D and his private key in order

to obtain the plaintext $m = D_{\mathsf{sk}}(c)$. Notice that Alice only used public information in order to encrypt the message: anyone can encrypt a message intended to Bob, but only Bob can decrypt it thanks to his private key. The most famous public-key encryption scheme is the RSA [57] encryption scheme.

1.2.2. *Identification.* An identification scheme allows an entity called the *prover* to prove his or her identity to another entity called the *verifier*. In a public-key identification scheme, the verifier knows the public key of the prover. The prover interacts with the verifier, using his private key without revealing it. The verifier becomes convinced that the prover knows the private key, which confirms the identity of the prover.

1.2.3. *Digital Signature.* In a signature scheme, when the signer wants to sign a message m, he computes a *digital signature* of m with his private key. This signature can be verified by anyone using the signer's public key; the verification algorithm will confirm the integrity of the message and the identity of the signer.

1.3. Hash Functions. A *cryptographic hash function* or *hash function* for short is a function that takes as input a bit string of arbitrary but finite length and outputs a bit string of fixed length. Hash functions are designed to be fast and to yield few collisions (i.e. two different inputs that have the same output); they are used in a wide variety of cryptographic schemes.

2. Number-Theoretic Reference Problems

The security of a public-key scheme usually relies on the intractability of a computational problem. Some of the most frequently used problems are presented in this section. While their exact computational complexity is not known, they are believed hard. More precisely, one usually makes the assumption that no polynomial-time algorithm can solve them.

In practice this means that, when appropriate parameters sizes are used, an adversary cannot solve the problem, even given an important computing power.

Given two related problems, it useful to know if one is "harder" than the other; this happens if there is a reduction between the two problems.

Definition 2.1. *Let A and B be two computational problems. A is said to* polytime reduce *to B if there exists an algorithm that solves A using as a subroutine a hypothetical algorithm that solves B, which runs in polynomial time if the algorithm for B does.*

If A polytime reduces to B, we write $A \leq_P B$. This means that A is not harder than B as the existence of a polynomial algorithm that solves B would imply the existence of a polynomial algorithm that solves A.

2.1. Factorization and RSA Problem. The security of many cryptographic schemes relies on the hardness of factoring a large integer or solving equations modulo such an integer.

DEFINITION 2.2. *Let N be a positive integer. The* integer factorization problem *is to find the prime factorization of N, i.e. to write $N = \prod_{i=1}^{k} {p_i}^{e_i}$ where p_i are distinct prime numbers and each $e_i \geq 1$.*

In practice, N is often chosen as the product of two primes.

DEFINITION 2.3. *The* RSA problem *is the following: given $N = pq$ where p and q are two distinct, odd primes, given e such that e is coprime with $(p-1)(q-1)$, and an integer X, find x such that $x^e \equiv X \pmod{N}$.*

In other words, the RSA problem consists in computing e-th roots modulo N. The RSA problem polytime reduces to the factorization problem: given p and q, one can compute $\phi(N) = (p-1)(q-1)$ then $d = e^{-1} \bmod \phi(N)$. Then, given $X = x^e \bmod N$, x can be computed as $x^d \bmod N$.

A variant of the RSA problem is when the attacker is given access to an RSA inversion oracle and is challenged with values for which it has to find e-th roots; the one more RSA problem is to find a number n of e-th roots using strictly less than n queries.

We denote by $K_{rsa}(k)$ the RSA key generation algorithm where k is a security parameter.

DEFINITION 2.4. *An rsa-omi adversary is a randomized, polynomial-time algorithm I that gets input N,e and has access to two oracles. The first is an RSA-inversion oracle $(.)^d \bmod N$ that given $Y \in \mathbb{Z}_N^*$ returns $Y^d \bmod N$. The second is a challenge oracle that, each time it is invoked (it takes no inputs), returns a random challenge point $W \in \mathbb{Z}_N^*$. The game considered is to run the RSA key generation algorithm $K_{rsa}(k)$ and then run $I(N, e)$ with its oracles. Let $W_1, \ldots, W_n$ denote the challenges returned by I's challenge oracle. We say that I wins if its output is a sequence of points $w_1, \ldots, w_n \in \mathbb{Z}_N$ satisfying $w_i \equiv W_i^d \pmod{N}$ meaning I inverts all the challenge points and also the number of queries made by I to its RSA-inversion oracle is strictly less than n. The rsa-omi advantage of I, denoted $Adv_{K_{rsa},I}^{rsa-omi}(k)$, is the probability that I wins, taken over the coins of K_{rsa}, the coins of I, and the coins used by the challenge oracle across its invocations. We say that K_{rsa} is OMI secure if the function $Adv_{K_{rsa},I}^{rsa-omi}(k)$ is negligible for any rsa-omi adversary I of time complexity polynomial in k.*

2.2. Discrete Logarithm Problem. Another frequently used hard problem is the discrete logarithm problem. It has several variants, depending on the group and the size of the exponent.

DEFINITION 2.5. *The* discrete logarithm problem *is the following problem: given a prime number p, a generator g of $\mathbb{Z}_p^*$ and an element X of $\mathbb{Z}_p^*$, find $x \in [0, p-1[$ such that $g^x \equiv X \pmod{p}$.*

It can be generalized to any finite cyclic group (written multiplicatively):

DEFINITION 2.6. *The* generalized discrete logarithm problem *is the following problem: given a finite cyclic group G of order n, a generator g of G and an element X of G, find $x \in [0, n[$ such that $g^x = X$.*

In Schnorr's identification and signature scheme, for example, G is the subgroup of $\mathbb{Z}_p^*$ generated by an element of order q where q is a prime.

One can also put a restriction on the size of the exponent.

DEFINITION 2.7. *The* generalized discrete logarithm problem with short exponents *is the following problem: given a finite cyclic group G of order n, a generator g of G, an integer $S < n$ and an element X of G such that $g^x = X$ where $x \in [0, S[$, find x.*

The generalized discrete logarithm problem with short exponents polytime reduces to the generalized discrete logarithm problem if n/S is polynomial.

In the previous problem, we can choose G as the group generated by an element g modulo a composite number N:

DEFINITION 2.8. *The* discrete logarithm problem with short exponents modulo a composite integer *is the following problem: given an integer N such that N is the product of two primes p and q, an integer $g \in [0, N[$, an integer $S < ord_N(g)$ and an integer X of G such that $g^x \equiv X \pmod N$ where $x \in [0, S[$, find x.*

2.3. Diffie-Hellman Groups. We now present two variants of the Diffie-Hellman problem: the computational Diffie-Hellman problem (CDH for short) and the decisional Diffie-Hellman problem (DDH).

In the following definitions, we are given a multiplicative group G of prime order p.

DEFINITION 2.9. *The* computational Diffie-Hellman problem *is: given $g \in G$ and g^a, g^b, compute g^{ab}.*

In other words, the computational Diffie-Hellman problem consists in finding $u^{\log_g h}$ given three elements $g, h, u \in G$.

DEFINITION 2.10. *The* decisional Diffie-Hellman problem *is : given $g \in G$, g^a, g^b and $h \in G$, decide whether $h = g^{ab}$.*

In other words, the decisional Diffie-Hellman problem consists in determining if $\log_g h = \log_u v$ given four elements g, h, u, v in G. In case they do, the tuple (g, h, u, v) is called a *DDH-tuple*.

The DDH problem reduces to the CDH problem: given an instance g, g^a, g^b, h of the DDH problem, one can use the algorithm that solves the DDH problem with g, g^a, g^b in order to compute g^{ab}, then check whether $h = g^{ab}$.

There are some groups where the DDH problem is easy while the CDH problem is hard; these groups are called *gap Diffie-Hellman groups* [46, 45]. We can give a more formal definition:

DEFINITION 2.11. *A prime order group G is called a GDH group if there exists an efficient polynomial time algorithm which solves the DDH problem, but no polynomial time algorithm can solve the CDH problem with non-negligible probability, when the inputs g, a, b are chosen at random.*

3. Provable Security

3.1. Standard Model. A cryptographic scheme has no practical use if it is insecure; when studying a scheme, one must try to find guarantees that the scheme is indeed secure. For example, given a signature scheme, one needs to prove that it is unforgeable, i.e. that no attacker can produce valid signatures without knowing the corresponding private key.

In most cases, one cannot ensure unconditional security; one needs to prove that a scheme is secure under some computational assumption.

The approach is to choose a computational problem (e.g. the factorization problem) that is supposedly hard, then to prove than the scheme is secure if the problem is hard.

More precisely, one needs to define:

- The attacker's goal
- The attacker's means
- A computational assumption

A *security proof* is a mathematical demonstration that if the computational assumption is hard, then no attacker can succeed.

The method used to build a security proof is usually the following. One assumes that an attacker (that succeeds with non-negligible probability) exists, then one designs an algorithm that solves the computational assumption. The attacker is used as a subroutine of this algorithm. This demonstrates that, if the computational problem cannot be solved, then the attack is not feasible.

3.2. Random Oracle Model. There are some situations in which a standard security proof is hard to find. This is often due to the fact that when using the attacker as a subroutine, he makes some requests and the algorithm has to answer those requests. For example, when proving the security of a signature scheme against chosen message attacks, the attacker is going to ask for signatures of messages of his choice thus the algorithm needs to be able to compute these signatures.

In order to overcome this technical difficulty, Bellare and Rogaway introduced the random oracle model in 1993 [16]. In this security model, hash functions are replaced by *random oracles*. A random oracle is a function that is indistinguishable from a perfectly random function from the point of view of the attacker.

In the security proof, when the attacker needs the hash of a value, instead of computing the outputs of the hash function himself, he will send the value to an oracle that will send back a value as the output, such that the values sent by the oracle are uniformly distributed over the output space of the function.

This means that the reduction algorithm that uses the attacker as a subroutine will now answer the hash queries; it can output a value chosen in a smart way, as long as it "looks like a random value" to the attacker.

As an example of security proof in the random oracle model, we give the main idea of the security proof of the Full Domain Hash [28] signature scheme. In this scheme, RSA parameters are generated. (N, d) is the signer's private key and (N, e) is the corresponding public key; a hash function H such that the outputs of H are uniformly distributed over $\mathbb{Z}_N^*$ is chosen. The signature of a message m is computed as $H(m)^d \bmod N$.

During an adaptive chosen-message attack, the attacker has access to a signing oracle; he sends signature queries for m_i of his choice and receives the corresponding signatures. In the proof, we assume that the attacker also sends hash queries to a random oracle, and that when he sends a query m_i to the signature oracle, he previously sent a hash query for the same message.

The reduction algorithm answers both kind of queries. When it receives a hash query for message m_i, it computes $H(m_i)$ as $y_i^e \bmod N$ for a random $y_i \in \mathbb{Z}_N^*$. When it receives a signature query for m_i, it sends the value y_i.

For the attacker, $H(m_i)$ is a random value in $\mathbb{Z}_N^*$, but for the reduction algorithm it means that if the attacker later asks for the signature of m_i, it will be able to answer correctly since $H(m_i)^d \bmod N = y_i$.

This shows how, by forcing the attacker to send hash queries, the random oracle model gives more freedom to design an efficient reduction algorithm. Some schemes have security proofs both in the standard model and in the random oracle model, but usually proofs employing random oracles have better reduction rates or rely on less strong assumptions.

Security in the random oracle model does not imply security in the real world: some schemes are secure in the random oracle model but can be broken [26, 11]. A proof in this model should not be seen as a formal proof that the scheme is secure; it is nevertheless a heuristic way to validate the design of the cryptographic scheme.

CHAPTER 2

Public-Key Identification Schemes

1. Introduction

Identification schemes allow one entity (called the verifier) to verify the identity of another entity (called the prover). In real life, there are many applications where an electronic device must prove its identity to access a resource or a place; this occurs for example when someone uses a smart card in order to enter a restricted area in a building (the reader at the door must ensure that the smart card is indeed a card that belongs to someone allowed in the area). Another example is cash withdrawal at an automatic cash dispenser: the machine has to check the identity of the withdrawer before giving the money.

In an identification scheme, the prover knows a secret and wants to convince the verifier of this knowledge. The verifier will accept the identity of the prover if he can confirm that the prover knows the secret. An identification scheme is a protocol that describes a way to do that in a secure manner, allowing that any entity trying to impersonate another entity will be detected by the verifier.

Secret key techniques can be used to ensure identification. In this case, the verifier must know the prover's secret key. The simplest technique is simply to reveal the secret key to verifier. This requires a secure communication channel between the prover and the verifier; otherwise an eavesdropper can obtain the prover's secret key. More advanced techniques also exist; one-time passwords for example are secure against eavesdroppers as a password will never be used twice by the prover. One can also use challenge-response protocols based on an encryption scheme: the verifier sends an encrypted random message and the prover must answer with the decryption of the message, which proves that he knows the secret key.

Secret-key identification techniques are computationally efficient, but have the same drawback than other secret-key primitives: the need for the prover and the verifier to have a common secret. If Alice uses the same secret-key to be identified by Bob and Charlie, then Bob can run an identification protocol with Charlie and impersonate Alice. This means that such secret-key techniques should only be used in situations where the prover completely trusts the verifier; more precisely, if Alice wants to prove her identity to Bob, she must trust Bob and check that she is indeed talking to Bob. In many practical applications, this assumption is not realistic.

The solution to avoid this is to use public-key techniques where the prover will never have to reveal her private key. In the remaining of this thesis, we focus on public-key techniques.

In this area too, many ways can be employed to design schemes. One can use public-key encryption: the verifier sends to the prover a random message m, encrypted under the prover's public key. The prover decrypts the message with her private key and sends the decrypted message to the verifier. The verifier checks that the decrypted message is m and becomes convinced of the identity of the prover.

A similar idea is to use digital signatures: the verifier sends a random message m to the prover. The prover signs the message with her private key and sends the signature to the verifier. The verifier checks the signature and accepts the identity of the prover.

Another approach is to invent specific identification schemes. In some cases, this allows to propose more efficient schemes; sometimes the motivation is also to obtain primitives whose security is easier to prove, as it was the case with the invention of zero-knowledge schemes.

The remaining of this chapter gives an overview of the provable security of identification schemes then presents a few of the most well-known schemes.

2. Provable Security of Identification Schemes

In order to discuss the security of identification schemes, we need to define three entities and their respective goals.

- The *prover* is an entity that wants to convince the verifier of its identity.
- The *impersonator* is an entity, distinct from the prover, who wants to make the verifier believe that it is the prover.
- The *verifier* is an entity that wants to be able to distinguish between the prover and the impersonator.

An identification scheme is a protocol between an entity and the verifier; at the end of the protocol, the verifier should be able to know if the entity is the prover or the impersonator. The verifier can be seen as an algorithm that outputs a bit at the end of the interaction with an entity, such that this bit equals 1 if the entity is the prover and 0 otherwise.

2.1. Kinds of Attacks. In an identification scheme, the goal of the attacker is to impersonate the prover. More precisely, the attack is successful if the attacker obtains a way to be accepted by the verifier as the honest prover with a non-negligible probability.

The means of the attacker may be:

- *Key-only attack*: The attacker has only access to the public key of the prover.
- *Passive attack*: The attacker has access to transcripts of identifications between the prover and the verifier.

- *Active attack*: The attacker impersonates the verifier and interacts with the prover.
- *Concurrent attack*: The attacker impersonates the verifier and interacts with several clones of the prover concurrently.

2.2. Kinds of Security Proofs. Two approaches exist when it comes to proving the security of an identification scheme. The first one is the zero-knowledge paradigm invented by Goldwasser, Micali and Rackoff [41] in 1989. It is a heuristic method where one checks that the identification protocol satisfies some properties.

While the zero-knowledge approach and some variants like the witness-indistinguishable approach [35] have been used for years, they are usually not considered as formal proofs of security as they do not directly demonstrate that attacks are impossible. Another approach was proposed by Bellare and Palacio in 2002 [15]; they use a *reductionist* proof, i.e. a proof under a computational assumption that no attacker can break the scheme. The drawback of this approach is the need to use stronger computational assumptions. We now detail these two approaches.

2.2.1. *The Zero-Knowledge Paradigm.* Zero-knowledge protocols allow the prover to demonstrate the knowledge of a secret without revealing *any* information on the secret. They are an example of interactive proofs. Zero-knowledge protocols can be used as identification schemes in the following way. The prover generates a private key s_k and the corresponding public key p_k. Prior to an identification session, the verifier is given p_k and wants to check that the prover is indeed in possession of the corresponding private key s_k. The prover convinces the verifier that he knows s_k without revealing s_k. More precisely, during the protocol, the prover does not reveal any information about the secret (apart from the fact that he knows the secret).

A zero-knowledge identification scheme should satisfy three properties: completeness, soundness and zero-knowledge.

Definition 2.1. *A protocol is* complete *if, when executed between an honest prover and an honest verifier, it succeeds with overwhelming probability.*

Definition 2.2. *A protocol is* sound *if there exists an expected polynomial time algorithm M such that if a dishonest prover can with non-negligible probability successfully execute the protocol with the verifier, then M can be used to extract from this prover the private key of the honest prover (or a knowledge that would allow successful subsequent protocol executions).*

A protocol that satisfies these first two properties is called a *proof of knowledge*. Informally, a proof a knowledge is a protocol in which a verifier can determine whether the prover knows the secret or not. The most simple example of proof of knowledge is a protocol where the prover reveals the secret to the verifier.

A proof of knowledge is *zero-knowledge* if an eavesdropper cannot learn anything about the secret from his observations of the executions of the protocol between the honest prover and the verifier. The idea is that the eavesdropper can *simulate* these executions of the protocol, without knowing the secret, and that the outputs of this simulation look exactly like real executions of the protocol.

It can be formalized as follows:

DEFINITION 2.3. *A proof of knowledge is* zero-knowledge *if there exists an expected polynomial-time algorithm (called the* simulator*) which can produce, without interacting with the prover, transcripts indistinguishable from those resulting from interactions with the prover.*

Sometimes, the distribution of simulated transcripts and the distribution of those resulting from real interactions are very close but not equal. If, in practice, it is impossible to distinguish between the two distributions, this means that the eavesdropper will not be able to extract the secret from the transcripts of the real interactions. This notion is captured by the concept of statistical zero-knowledge.

DEFINITION 2.4. *Let X_1, X_2 be two random variables defined over S_1 and S_2. The* statistical distance *between X_1 and X_2 is*

$$SD[X_1, X_2] = \frac{1}{2} \sum_a |Pr[X_1 = a] - Pr[X_2 = a]|$$

DEFINITION 2.5. *A proof of knowledge is* statistically zero-knowledge *if there exists an expected polynomial-time algorithm (called the* simulator*) which can produce, without interacting with the prover, transcripts such that the statistical distance between the distribution of these transcripts and the distribution of observable transcripts is negligible.*

2.2.2. *Reductionist Security Proofs.* In 2002, Bellare and Palacio [15] proved the security of the GQ and Schnorr schemes under concurrent attacks. Rather than using the zero-knowledge paradigm, they propose a reductionist security proof. They means that they demonstrate that if the underlying computational problem is hard, then no attacker can succeed in breaking the given security property.

Their proofs are in the standard model. The security of the GQ identification scheme is proven under the one more RSA assumption and the security of the Schnorr identification scheme is proven under the one more discrete logarithm assumption.

3. Fiat-Shamir Identification Scheme

The Fiat-Shamir scheme was proposed in 1986 [36]. It is a proof of knowledge of a square root modulo N; its security is based on the factorization problem. We present a simplified version of this protocol in order to highlight its similarity with Guillou-Quisquater.

3.1. Description. An authority chooses two strong primes p and q then computes their product $N = pq$.

The prover's private key is a random element $x \in [0, N[$. Her public key is $X = x^{-2} \bmod N$. An identification proceeds as follows:

(1) *Commitment:* The prover picks a random integer $y \in [0, N[$. She computes $Y = y^2 \bmod N$ and sends Y to the verifier.
(2) *Challenge:* The verifier sends a random bit $c \in \{0, 1\}$ to the prover.
(3) *Response:* The prover computes $z = yx^c \bmod N$. She sends z to the prover.
(4) *Verification:* The verifier checks that $z^2 X^c \equiv Y \pmod N$.

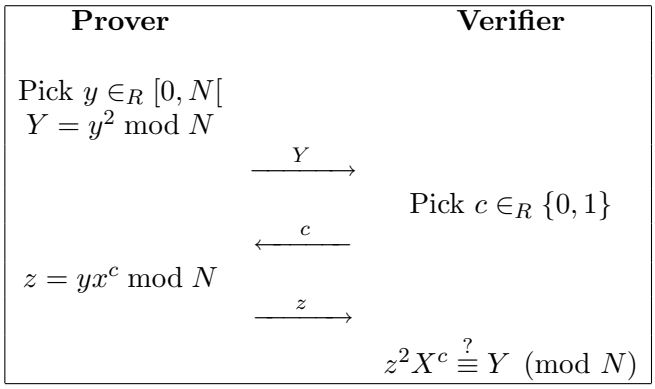

FIGURE 1. A Round of Fiat-Shamir

3.2. Security Discussion. Completeness follows from

$$z^2 X^c \equiv (yx^c)^2 (x^{-2})^c \equiv y^2 \equiv Y \pmod N$$

In order to prove the soundness, one shows that an attacker that can perform a successful identification with probability higher than $1/2$ can find the private key x. We first note that a dishonest prover can succeed in an identification if it can predict the value of the challenge c: he simply can choose any $z \in [0, N[$ and compute $Y = z^2 X^c \bmod N$. He then sends Y to the verifier and waits for c; if the guess on c was correct, he answers with z such that $Y = z^2 X^c \bmod N$ so he will fool the verifier. Using this technique, the attacker can succeed with a probability of $1/2$ as only two values are possible for c.

But if an attacker can succeed with a probability higher than $1/2$, then he can find a value Y such that he can answer two challenges $c_0 = 0$ and $c_1 = 1$. This means that the attacker knows Y, z_0 and z_1 such that

$$Y \equiv {z_0}^2 X^{c_0} \equiv {z_1}^2 X^{c_1} \pmod N$$

Thus ${z_0}^2 \equiv {z_1}^2 X \pmod N$. The attacker can then compute the private key x as $x = {z_0}^{-1} z_1 \bmod N$. This shows the soundness of the scheme.

The scheme is zero-knowledge: the simulator produces (Y, c, z) triples such that $Y = z^2 X^c \bmod N$ by first choosing at random z and c. Such triples are indistinguishable from the ones observed during real executions of the protocol.

In practice, the probability of success of the attacker should be negligible. Therefore the protocol should be used n times and the verifier should confirm the identity of the prover if all n runs are successful; the probability of success of an attacker becomes $(1/2)^n$.

4. Guillou-Quisquater Identification Scheme

Guillou and Quisquater proposed their identification scheme in 1986 [43]. The scheme is often called GQ. Its security is based on the RSA problem and can be seen as an adaptation of Fiat-Shamir where square roots are replaced by e-th roots.

4.1. Description. An authority generates RSA parameters: it chooses two strong primes p and q, computes their product $N = pq$. It also chooses an integer e such that e is coprime with $(p-1)(q-1)$.

The prover's private key is a random element $x \in [0, N[$. Her public key is $X = x^{-e} \bmod N$. An identification proceeds as follows:

(1) *Commitment:* The prover picks a random integer $y \in [0, N[$. She computes $Y = y^e \bmod N$ and sends Y to the verifier.
(2) *Challenge:* The verifier sends a random integer $c \in [0, e[$ to the prover.
(3) *Response:* The prover computes $z = yx^c \bmod N$. She sends z to the prover.
(4) *Verification:* The verifier checks that $z^e X^c \equiv Y \pmod p$.

Prover		**Verifier**
Pick $y \in_R [0, N[$ $Y = y^e \bmod N$		
	$\xrightarrow{Y}$	
		Pick $c \in_R [0, e[$
	$\xleftarrow{c}$	
$z = yx^c \bmod N$		
	$\xrightarrow{z}$	
		$z^e X^c \stackrel{?}{\equiv} Y \pmod p$

FIGURE 2. A Round of GQ

4.2. Security Discussion. Completeness follows from

$$z^e X^c \equiv (yx^c)^e (x^{-e})^c \equiv y^e \equiv Y \pmod{N}$$

Soundness can be proven in a similar manner than Fiat-Shamir. An attacker can succeed with probability $1/e$ by guessing the challenge; but if he can succeed with a higher probability, he can find Y, c_0, c_1, z_0, z_1 (where c_0 and c_1 are different challenges) such that

$$Y \equiv {z_0}^e X^{c_0} \equiv {z_1}^e X^{c_1} \pmod{N}$$

From these values, the attacker can compute the e-th root of X, i.e. the private key x.

There is an issue when one tries to prove that the scheme is zero-knowledge. If we assume that the prover is honest, then the simulator can generate (Y, c, z) triples such that $Y \equiv z^e X^c \pmod{N)}$ by simply choosing z and c at random. These triples are indistinguishable from observed triples. Therefore GQ is *honest-verifier zero-knowledge*: no information on the secret leaks during an execution of the protocol between the prover and an *honest* verifier.

But when one tries to simulate executions of the protocol between the prover and a dishonest verifier, one must take into account the case where the verifier chooses his challenges according to a strategy that may depend on Y and on the result of previous rounds. This means that when the simulator generates a triple (Y, c, z), it must check that c corresponds to the challenge that the attacker would have sent; if not, the triple is not valid and another has to be computed until the c value fits. This means that the complexity of the simulation will depend on the value of e. Formally, GQ is zero-knowledge if e is polynomial; in practice, this means that the guarantee that no information on x leaks decreases as the size of e increases.

Since the probability of success of an attacker not knowing the secret is $1/e$, there is a trade-off to find for the value of e; usually, values of 16, 32 or 64 bits are chosen.

5. Schnorr Identification Scheme

Schnorr's identification scheme was proposed by Schnorr in 1991. Its security is based on the discrete logarithm problem modulo a prime number.

5.1. Description. An authority generates two strong primes p and q such that $q|p-1$. It also chooses an integer $g \in [0, p[$ and an integer B.

The prover's private key is a random element $x \in [0, q[$. Her public key is $X = g^{-x} \bmod p$. An identification with Schnorr proceeds as follows:

(1) *Commitment:* The prover picks a random integer $y \in [0, q[$. She computes $Y = g^y \bmod p$ and sends Y to the verifier.
(2) *Challenge:* The verifier sends a random integer $c \in [0, B[$ to the prover.

(3) *Response:* The prover computes $z = y + cx \bmod q$. She sends z to the prover.
(4) *Verification:* The verifier checks that $g^z X^c \equiv Y \pmod p$.

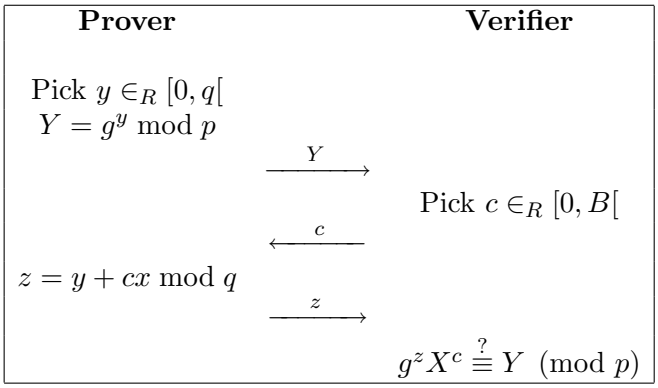

FIGURE 3. A Round of Schnorr

5.2. Security Discussion. Completeness follows from

$$g^z X^c \equiv g^{y+cx}(g^{-x})^c \equiv g^y \equiv Y \pmod p$$

The soundness of Schnorr can be proven in a similar way than the soundness of GQ. The protocol is also honest-verifier zero-knowledge; there is the same issue with the zero-knowledge property than with GQ, meaning that in practice B should be 16, 32 or 64 bits long.

6. Girault-Poupard-Stern Identification Scheme

GPS is an identification scheme initially proposed by Girault [38]. This protocol was designed for smart cards, allowing fast identification even on low-cost processors. GPS is based on a modification of Schnorr's scheme; its main difference with Schnorr is that the computation of the response does not involve modular computations. The scheme, whose name means Girault-Poupard-Stern, was selected as a standard for identification in the European project Nessie ("New European Schemes for Signatures, Integrity, and Encryption") [27].

6.1. Description. An authority generates two strong primes p and q and computes $N = pq$. It also chooses an integer g and three security parameters A, B and S. We set $E = A + (B-1)(S-1)$.

The prover's private key is a random element $x \in [0, S[$. Her public key is $X = g^{-x} \bmod N$. An identification with GPS proceeds as follows:

(1) *Commitment:* The prover picks a random integer $y \in [0, A[$. She computes $Y = g^y \bmod N$ and sends Y to the verifier.
(2) *Challenge:* The verifier sends a random integer $c \in [0, B[$ to the prover.

(3) *Response:* The prover checks that $c \in [0, B[$ and computes $z = y + cx$. She sends z to the prover.
(4) *Verification:* The verifier checks that $z \in [0, E[$ and $g^z X^c \equiv Y \pmod N$.

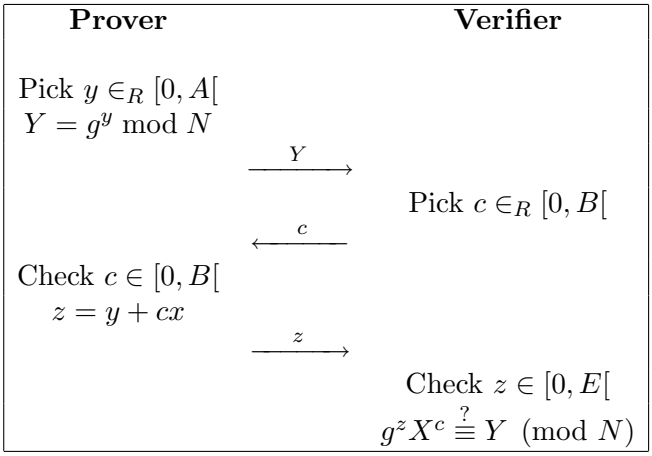

FIGURE 4. A Round of GPS

6.2. Security Discussion. Completeness follows from

$$g^z X^c \equiv g^{y+cx}(g^{-x})^c \equiv g^y \equiv Y \pmod p$$

The soundness of GPS was proven by Poupard and Stern in [55]. They also show that the scheme is statistically zero-knowledge if B is polynomial and if $A >> BS$.

We must remark here that the fact that the prover must check that $c \in [0, B[$ is crucial for the security of the scheme. Otherwise, a dishonest verifier could send a larger value for x. Since $x = z/c - y/c$, the attacker tries to guess y/c (he already knows z/c). If $c > A$, for example, the attacker knows that $y/c \in [0, 1[$ thus can compute x as $\lfloor z/c \rfloor$. This means that the attacker can find the private key in only one round of GPS. This attack is captured by the security proof of [55] since the (statistical) zero-knowledge property requires that $A >> BS$.

6.3. Parameters Sizes. We recall the minimum parameter sizes as recommended in [40], taking into account the constraints that B should be chosen with the usual trade-off and that $A >> BS$ to ensure the statistical zero-knowledge property.

The S value should satisfy $|S| \geq 160$. One should choose B such that $|B| = 16, 32$ or 64 and A such that $|A| = |B|+|S|+64$ or $|A| = |B|+|S|+80$.

When choosing the size of the modulus N, two cases may occur:

- Each user has his own modulus. In this case, $|N|$ should be at least 1024, and the factors of N, p and q, can be revealed to the prover, allowing her to use the Chinese Remainder Technique to speed-up the commitment phase.
- Several users share the same modulus N. In that case, $|N|$ should be at least 2048.

GPS authors [40] recommend the value $g = 2$, as it fits the security requirements while allowing good performance.

CHAPTER 3

Digital Signature Schemes

1. Introduction

A digital signature is a public-key cryptographic primitive that simulates the security properties of a handwritten signature. A handwritten signature binds a user to a document: when the signer puts her signature on the document, she links her identity to the content of the document. Anyone should be able to verify the signature, i.e. to check the identity of the signer, and no one should be able to produce a false signature.

Digital signatures work in a similar manner. The digital signature of a message m is a string that depends on m and on some secret known only to the signer, in such a way that anyone can check the validity of the signature. Signatures should be *unforgeable*: it should be impossible to compute a forgery, i.e. a false signature. A digital signature allows to authenticate the message, i.e. to check the identity of the signer and to ensure that the message was not modified by anyone else.

Definition 1.1 (Signature scheme). *A signature scheme is defined by three algorithms:*

- *The key generation algorithm* Setup *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and secret keys,* $(\mathsf{pk}, \mathsf{sk})$.
- *The signing algorithm* Sign *takes the message* m *to be signed and the secret key* sk *and returns a signature* $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$*. The signing algorithm may be probabilistic.*
- *The verification algorithm* Verify *takes a message* m*, a candidate signature* $\overline{\sigma}$ *and the public key* pk*. It returns a bit* $\mathsf{Verify}_{\mathsf{pk}}(m, \overline{\sigma})$*, equal to one if the signature is accepted, and zero otherwise. We require that if* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$*, then* $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1$.

The key generation algorithm outputs the public and private keys of the signer. The signing algorithm is executed by the prover: using the private key and the message, it outputs the signature. The verification algorithm is run by the verifier: using the message, the public key and the candidate signature, it returns 1 if the signature is valid and 0 otherwise.

In this chapter, we explain how to study the security of signature schemes and present a few signature schemes that will be studied in the other parts of this thesis.

2. Security of Signature Schemes

A signature scheme is secure if no attacker can produce a forgery. In order to formalize this notion, we need to precisely describe the *goal* of the attacker and his *means*.

The possible ways to break a signature scheme are:

- *Total break*: The attacker obtains the private key of the signer.
- *Selective forgery*: The attacker obtains a valid signature on a message of his choice.
- *Existential forgery*: The attacker obtains a valid signature on a message that he did not choose.

They are ordered by difficulty, the hardest problem being the first in the list. In the case of a total break, the attacker obtains the private key; that will allow him to sign *any* message of his choice.

The means of the attacker depends on what he has access to. One can assume that the attacker will always have access to public data like the signer's public key. Obtaining signed messages is also feasible in many situations, for example by watching publicly available signed documents. In some cases, the attacker may even have some limited access to a way to sign messages and use this to obtain some knowledge that would allow him later to sign other messages. These means are classified in the following way:

- *Key-only attack*: The attacker has only access to the public key of the signer and other public parameters.
- *Known-message attack*: The attacker has access to signatures of known messages.
- *Chosen-message attack*: The attacker can obtain signatures on messages of his choice before he starts attempting to break the scheme. He obtains these signatures only once, i.e. messages are chosen before any signature is seen.
- *Adaptive chosen-message attack*: The attacker has access to a *signing oracle* that provides signatures on messages of the attacker's choice. The attacker can choose the messages depending on previously received signatures.

In order to obtain a high level of security, one usually considers an attacker with powerful means and a modest goal; if the signature scheme is secure against this kind of attack, it will be secure against all the other kind of attacks.

The strongest security requirement for a signature scheme is thus resistance against existential forgeries under adaptive chosen-message attacks.

2.1. Security Model. The security of signature schemes was formalized in an asymptotic setting by Goldwasser, Micali and Rivest [42]. Here we use the definitions of [19] which provide a framework for the concrete security analysis of digital signatures. We consider resistance against adaptive chosen-message attacks: a forger $\mathcal{F}$ can dynamically obtain signatures of

messages of its choice and attempt to output a valid forgery. A *valid forgery* is a message/signature pair (m, σ) such that $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1$ whilst the signature of m was never requested by $\mathcal{F}$.

DEFINITION 2.1. *A forger $\mathcal{F}$ is said to $(t, q_{sig}, \varepsilon)$-break the signature scheme* (Setup, Sign, Verify) *if after at most $q_{sig}(k)$ signature queries and $t(k)$ processing time, it outputs a valid forgery with probability at least $\varepsilon(k)$ for any $k > 0$.*

A signature scheme is secure is no attacker can break the scheme with non-negligible probability and in a reasonable time:

DEFINITION 2.2. *A signature scheme* (Setup, Sign, Verify) *is (t, q_{sig}, ε)-secure if there is no forger who $(t, q_{sig}, \varepsilon)$-breaks the scheme.*

3. RSA Signature Scheme

RSA [57] is the most famous and widely used public-key cryptosystem since its publication in 1977. It can be used to provide both encryption schemes and digital signatures; its security is based on the RSA problem. We now describe how it can be used to generate signatures.

The idea is to compute a modulus N as a product of primes p and q and an exponent e coprime to $\phi(N)$. A signature of a message is a e-th root modulo N of the message. The basic RSA signature scheme can be described as follows:

- Setup : The RSA generator RSA, on input 1^k, randomly selects two distinct $k/2$-bit primes p and q and computes the modulus $N = pq$. It randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$ and computes the corresponding decryption exponent d such that $e \cdot d = 1 \bmod \phi(N)$. The output of RSA is (N, e, d); the public key is (N, e) and the private key is (N, d).
- Sign : To compute a signature on a message m, the signer computes $s = m^d \bmod N$. The signature is s.
- Verify : To verify a signature s on m, the verifier checks that $m = s^e \bmod N$.

This basic scheme should not be used in practice as it is not resistant to chosen-message attacks. An attacker wanting to obtain the signature on a given message m could find two messages m_1 and m_2 such that $m = m_1 \cdot m_2 \bmod N$, ask for the signatures of m_1 and m_2 then compute the signature of m as the product modulo N of the signatures of m_1 and m_2. This is because

$$\mathsf{Sign}(m) = m^d \bmod N = (m_1^d \cdot m_2^d) \bmod N = \mathsf{Sign}(m_1) \cdot \mathsf{Sign}(m_2) \bmod N$$

This means that more elaborate constructions should be used in practice. The usual approach is to first apply a kind of hash or encoding function μ to the message before raising it to the exponent d modulo N.

Let μ be an encoding function taking as input a message of size ℓ bits and returning a k-bit integer. The RSA scheme with an encoding function works as follows:

Setup : The RSA generator RSA, on input 1^k, randomly selects two distinct $k/2$-bit primes p and q and computes the modulus $N = pq$. It randomly picks an encryption exponent $e \in \mathbb{Z}^*_{\phi(N)}$ and computes the corresponding decryption exponent d such that $e \cdot d = 1 \bmod \phi(N)$. The output of RSA is (N, e, d); the public key is (N, e) and the private key is (N, d).

Sign : To compute a signature on a message m, the signer computes $s = \mu(m)^d \bmod N$. The signature is s.

Verify : To verify a signature s on m, the verifier checks that $\mu(m) = s^e \bmod N$.

4. GPS Signature Scheme

A method proposed by Fiat and Shamir [36] allows to turn an interactive three-passes identification scheme into a non-interactive signature scheme. The idea is that the signer is doing an identification without interacting with anyone, by replacing the random challenge sent by the verifier in the identification scheme by a hash of the commitment and the message. The size of the challenge must be larger than in the identification scheme to prevent off-line attacks on the hash function.

The GPS signature scheme is an example of the use of this transform; it is derived from the GPS identification scheme. It works as follows.

Setup : The TTP generates an RSA modulus $N = pq$ where p and q are primes. It chooses three integers A, B and S such that $A >> BS$, and a hash function h such that the outputs of h are uniformly distributed over $[0, B[$. The TTP chooses an integer g of large order modulo N. It publishes N, g, h.

To generate her key pair, the signer chooses a random integer $x \in [0, S[$ as her secret key and computes $X = g^{-x} \bmod N$ as her public key.

Sign : To compute a signature on a message m, the signer takes a random integer $y \in [0, A[$ and computes $Y = g^y \bmod N$, and $z = y + h(Y, m)x$. The signature is (Y, z).

Verify : To verify a signature (Y, z) on m, the verifier checks that $h(Y, m) \in [0, B[$, that $z \in [0, A + (B - 1)(S - 1)[$ and that $Y \equiv g^z X^{h(Y,m)} \pmod N$.

The security of the scheme against existential forgeries under adaptive chosen message attacks was proven in the random oracle model in [55] under the assumption that computing discrete logarithms with short exponents modulo N is hard.

5. Gap Diffie-Hellman Signature Scheme

We now describe the gap Diffie-Hellman (GDH) signature scheme [22]. This scheme is a good example of the use of the gap Diffie-Hellman problem to design a scheme.

Setup : The TTP takes a GDH group G of order p and random $g \in G$, $x \in \mathbb{Z}_p$. It computes $h = g^x$ and sets the public key to be (g, h). The secret key is x. It also chooses a hash function H whose outputs are elements of G (in the security proof, H will be considered as a random oracle).

Sign : To compute a signature on a message m, the signer computes $\sigma = H(m)^x$.

Verify : To verify σ on m, the verifier checks that $(g, h, H(m), \sigma)$ form a DDH-tuple.

The security of this scheme was proven in [22] by Boneh, Shacham and Lynn:

THEOREM 5.1. *If G is a GDH group, then the GDH signature scheme is existentially unforgeable under adaptive chosen message attack in the random oracle model.*

Informally, in order to check the signature, the verifier must verify that $(g, h, H(m), \sigma)$ is a DDH-tuple, which is possible since G is a GDH group. On the other hand, an attacker trying to compute an forgery on a given message m' must be able to find σ' such that $(g, h, H(m'), \sigma')$ is a DDH-tuple, where he cannot choose g, h or $H(m')$; this means that the attacker must solve an instance of the CDH problem, which is impossible under the GDH assumption.

CHAPTER 4

Optimistic Fair Exchange

1. Introduction

The issue of fairness in cryptography arises when two (or more) entities want to achieve a transaction but do not trust each other completely.

For example, assume that Alice and Bob meet over the internet and that Alice wants to buy an object from Bob and Bob is willing to sell it; since they do not really know each other, Alice does not want to sign the payment unless she is sure that she will get the object, and Bob does not want to send the object unless he is sure that Alice is going to pay him.

This is an example of an important issue in electronic commerce and digital rights management: the problem of exchanging items and information in a fair way. Let us consider a situation where Alice is ready to sign a document if Bob fulfills a given obligation; Bob agrees to fulfill this obligation if Alice signs the document. This problem is called *fair exchange*.

A fair exchange protocol involves Alice, Bob and a third party (that we will call Charlie) trusted by both Alice and Bob. At the beginning of the protocol, Alice and Bob agree on a message m; Alice agrees to give her signature on m to Bob and Bob agrees to fulfill his obligation. At the end of the protocol, either both parties are satisfied (i.e Alice signed m and Bob fulfilled his obligation) or none of them. This ensures that no one is cheated and is called the *fairness property* of the protocol.

We now describe a simple fair exchange protocol. Alice sends her signature to Charlie and Charlie checks this signature; if the signature is valid, Charlie asks Bob to fulfill his obligation. Once Charlie is convinced that Bob fulfilled his obligation, he sends Alice's signature to Bob. If Alice cheats (or if for some reason Charlie does not receive her signature), Charlie will ask nothing to Bob; if Bob cheats (or cannot fulfill his obligation), Charlie will not send him Alice's signature. This demonstrates that this protocol is fair.

However, this basic fair exchange protocol involves Charlie in all the steps of the exchange; this implies a heavy communication load and a heavy computational load for Charlie, especially in a setting where a lot of transactions occur at the same time. It is thus natural to try and minimize the role of Charlie, by designing protocols where Charlie only intervenes in case of problem, i.e. if Alice or Bob tries to cheat or if a technical problem prevents them from completing the protocol. Such protocols are called *optimistic* fair exchange protocols.

The first to formally study optimistic fair exchange protocols were Asokan, Shoup and Waidner [4, 5]. Several fair exchange of signatures protocols have been proposed since then [6, 8, 21, 23, 54]. They all rely on some cryptographic primitive allowing the commitment to a signature. This was noticed by Dodis and Reyzin in [34], who proposed a model for non-interactive fair exchange protocols. The model relies on a kind of cryptographic primitives called *verifiably committed signatures* that generalize verifiably encrypted signatures (where a verifier can check that a given ciphertext is the encryption of a signature on a given message m) and two-signatures (i.e. sequential two-party multisignature schemes).

2. Verifiably Committed Signatures

The principle of verifiably committed signatures can be summarized as follows. The participants in the protocol are Alice the signer, Bob the verifier, and Charlie the semi-trusted arbitrator. The signer Alice wants to give Bob a signature on a message m, but only if Bob fulfills some obligation I in exchange. Bob does not want to fulfill I until he is sure that he will eventually get Alice's signature on m. In order to do this, Alice first computes a partial signature σ' on m. Bob can check that this partial signature corresponds to Alice's valid signature on m, but cannot extract the valid signature from the partial one. Then Bob fulfills I. Alice then sends the final signature σ on m back to Bob. In case Alice aborts the protocol, Bob can call the arbitrator Charlie, and if Bob can prove that he fulfilled I, Charlie extracts the full signature from the partial one. Thus Charlie is only needed in case of problem.

Definition 2.1 (Verifiably Committed Signature scheme). *A verifiably committed signature scheme is defined by six algorithms:*

- *The key generation algorithm* Setup *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and secret keys* $(\mathsf{pk}, \mathsf{sk})$ *for Alice. It also generates a secret arbitration key* ask *for Charlie.*
- *The partial signing algorithm* PSign *takes the message* m *to be signed and the secret key* sk *and returns a partial signature* $\sigma' = \mathsf{PSign}_{\mathsf{sk}}(m)$.
- *The partial verification algorithm* $\mathsf{PVerify}$ *takes a message* m*, a candidate partial signature* $\overline{\sigma'}$ *and the public key* pk*. It returns a bit* $\mathsf{PVerify}_{\mathsf{pk}}(m, \overline{\sigma'})$*, equal to one if the partial signature is accepted, and zero otherwise. We require that if* $\sigma' \leftarrow \mathsf{PSign}_{\mathsf{sk}}(m)$*, then* $\mathsf{PVerify}_{\mathsf{pk}}(m, \sigma') = 1$.
- *The signing algorithm* Sign *takes the message* m *to be signed and the secret key* sk *and returns a signature* $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$.
- *The verification algorithm* Verify *takes a message* m*, a candidate signature* $\overline{\sigma}$ *and the public key* pk*. It returns a bit* $\mathsf{Verify}_{\mathsf{pk}}(m, \overline{\sigma})$*,*

equal to one if the signature is accepted, and zero otherwise. We require that if $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$*, then* $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1$.

- *The dispute resolution algorithm* Res *takes a message* m*, a valid partial signature* σ'*, the public key* pk *and the secrete arbitration key* ask*. It outputs a signature* σ *such that* $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1$.

Dodis and Reyzin state that a verifiably committed signature scheme should satisfy the following (informal) security requirements [34]:

- *Security against Alice:* Alice should not be able to produce a valid partial signature σ' which Charlie cannot convert into a valid full signature σ.
- *Security against Bob:* Bob should not be able to produce a valid partial signature σ' which he did not get from Alice, and Bob should not be able to produce a valid full signature σ which he did not get from Alice or Charlie.
- *Security against Charlie:* Charlie should not be able to produce a valid full signature σ without seeing a valid partial signature σ' computed by Alice.

In case Charlie is dishonest, he can agree with Bob to cheat Alice by opening her signature to Bob; however, he will not be able to forge a signature of Alice on a message of his choice. For this reason, the required level of trust in Charlie that is not as high as the one of a certification authority for example. Charlie is often called a semi-trusted arbitrator.

3. Optimistic Fair Exchange of Signatures

We now give an example of application of fair exchange that allows Alice and Bob to fairly exchange two signatures. Once we are given a verifiably committed signature scheme, it is relatively straightforward to use it to design an optimistic fair exchange of signatures scheme where Alice and Bob exchange their signatures on messages m_A and m_B respectively.

In the verifiably committed signature scheme, Bob must fulfill an obligation; in the corresponding fair exchange of signatures scheme, this obligation is to send his signature on a message m_B to Alice. Notice that Bob can use any secure signature scheme (Sign',Verify'); in particular, Bob does not have to use the verifiably committed signature of Alice.

The protocol will take three steps :

(1) Alice sends her partial signature on message m_A to Bob
(2) Bob checks the partial signature and sends his signature on m_B to Alice
(3) Alice checks Bob's signature then sends her full signature on m_A to Bob

In case Bob does not receive Alice's full signature or receives an invalid signature, he calls Charlie and sends Alice's partial signature along with a proof that he sent his signature to Alice. Charlie checks Alice's partial

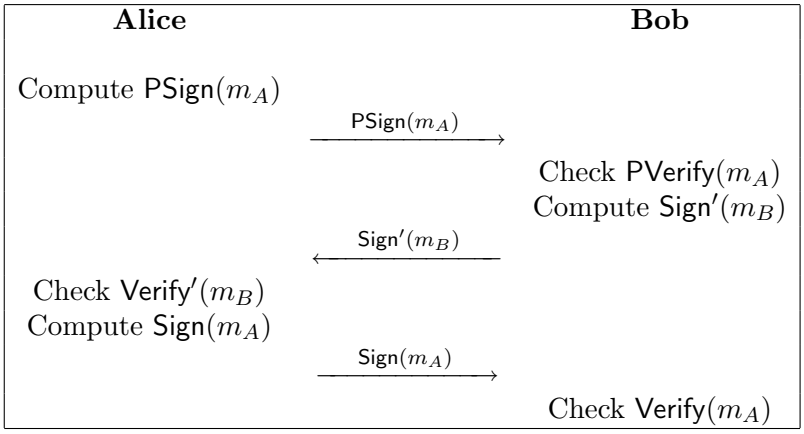

FIGURE 1. Fair Exchange based on Verifiably Committed Signatures

signature and Bob's proof then opens Alice's partial signature, i.e. computes Alice's full signature (or a signature that is indistinguishable from Alice's full signature) on m_A and sends it to Bob.

The protocol may fail at any step, because of a technical difficulty (if the communication channel is cut, for example) or because one of the participants is trying to cheat. Let us give a look at every possible case :

(1) The first step fails, i.e. Bob never receives a valid partial signature from Alice. The protocol stops and is fair since no party obtained what he or she wanted.
(2) The first step succeeds, but the second step fails, i.e. Alice sent her partial signature but Bob failed to send his signature or sent an invalid one. Bob cannot open Alice's signature without the help of Charlie since he cannot prove to Charlie that he sent his signature so Alice. Again, the protocol stops and is fair since no party obtained what he or she wanted.
(3) The first two step succeed, but the third step fails, i.e. Alice has got Bob's signature but Bob only obtained Alice's partial signature. Bob calls Charlie and obtains Alice's full signature. The protocol is fair since both parties obtained what they wanted.
(4) All the steps succeed. Again, the protocol is fair since both parties obtained what they wanted.

This shows that in all the cases, the scheme is fair. No one can obtain the other's signature without revealing his or her signature.

4. Dodis-Reyzin Fair Exchange Scheme

We now give an example of fair exchange scheme. We choose to describe Dodis and Reyzin's scheme [34] because of its simplicity and because it is a provably secure scheme in the model that we described above.

The idea is to transform the GDH signature scheme into a verifiably committed signature scheme based on GDH groups.

Setup : Alice picks a GDH group G of order p, chooses random $g \in G$ and $x, x_1 \in \mathbb{Z}_p$ then computes $x_2 = x - x_1 \bmod p$, $h = g^x$, $h_1 = g^{x_1}$ and sets $PK = (g, h)$, $SK = (x, x_1)$, $APK = h_1$, $ASK = x_2$. She sends PK, APK, ASK to Charlie, who checks that $h = h_1 g^{x_2}$.

PSign : Alice computes a partial signature on m as a GDH signature with public key h_1 : her partial signature is $\sigma' = H(m)^{x_1}$.

PVerify : Bob verifies that $(g, h_1, H(m), \sigma')$ is a DDH-tuple.

Sign : Alice computes $\sigma' = H(m)^x$.

Verify : Bob verifies that $(g, h, H(m), \sigma)$ is a DDH-tuple.

Res : In case of problem, given σ' and m, Charlie first checks that $(g, h_1, H(m), \sigma')$ is a DDH-tuple (i.e. Charlie performs the PVer algorithm). If it is the case, Charlie outputs $\sigma = \sigma' H(m)^{x_2}$.

The authors show that this scheme is as secure as the regular GDH signature. In particular, it is secure in GDH groups in the random oracle model.

CHAPTER 5

Secure and Efficient Implementation

Consider a smart card implementation of a signature scheme in a payment system where a signature is computed each time a user pays something. Even if the scheme itself is provably secure, what if the computation of a signature takes several minutes ? The card will be useless as people want to be able to pay with their card in a few seconds at most. What if the power consumption of the card reveals some information on the private key ? A dishonest person could build a system that allows to copy the card, then use a copy to steal money from the legitimate owner of the card. These examples show that cryptographers must not only design provably secure schemes; they also have to consider the following questions:

- *The scheme's efficiency*: Can it be implemented such that the time required to perform a signature will be reasonable ? Several other efficiency aspects might be important to consider, such as the size of the keys, the size of the signatures, the time needed for verification.
- *The security of the implementation*: For example, does the implementation time leak some information on the private key ? More generally, this is the problem of side-channel attacks.

The remaining of this chapter presents a few techniques used to implement efficiently and securely a scheme.

1. Exponentiation Techniques

Computing an exponentiation is computing g^x in some group G given an element $g \in G$ and an integer x. This computation occurs in many public-key cryptographic schemes. Often, one has to compute a *modular* exponentiation, where the problem is to compute $g^x \bmod M$ given three integers g, x, and M. For example, RSA signature and verification involve modular exponentiations modulo a composite integer N.

Exponentiations are often the heaviest kind of operations that occur in number-theoretic based schemes; therefore, a fast algorithm for modular exponentiation is a crucial component in the implementation of a scheme.

The square and multiply algorithm is an exponentiation algorithm whose complexity is linear in the size of the exponent. The square-and-multiply algorithm and its variants are often used to implement exponentiations in cryptography. The idea is to use the binary representation of $x = (x_t x_{t-1} \dots x_1 x_0)_2$ and to write g^x as a product of values of the form $g^{2^i x_i}$.

The binary representation of x can be read from right to left or for left to right:

Algorithm 1 Right to Left Square-and-Multiply

Input: $g \in G$ and a positive integer $x = (x_t x_{t-1} \dots x_1 x_0)_2$
Output: g^x
  $A := 1$
  $S := g$
  for i from 0 to t **do**
    if $x_i = 1$ **then**
      $A := A.S$
    end if
    $S := S^2$
  end for
  return A

Algorithm 2 Left to Right Square-and-Multiply

Input: $g \in G$ and a positive integer $x = (x_t x_{t-1} \dots x_1 x_0)_2$
Output: g^x
  $A := 1$
  for i from t down to 0 **do**
    $A := A^2$
    if $x_i = 1$ **then**
      $A := A.g$
    end if
  end for
  return A

2. Efficiency Optimization of Identification Schemes

Several tricks can be used in order to make efficient implementations of three-pass identification schemes.

2.1. Precomputed Commitments. The commitment step of a three-pass identification scheme can be computed before knowing the challenge sent by the verifier. This means that one or several commitments can be computed in advance by the device that does the identification, or even by another trusted, more powerful, device. Care must be taken as such commitments must be used only once then discarded; otherwise, the security of the scheme is compromised.

In the case of schemes like Schnorr or GPS, this means that no online modular exponentiation is needed. With Schnorr, the only online computation is $z = y + cx \pmod q$ and with GPS it is $z = y + cx$, meaning that these schemes have a very low computational cost for the prover when using precomputed commitments.

2.2. Hashed Commitments. Another common trick can be used to save communication load. The idea is that the prover applies a hash function to the commitment before sending it to the verifier. This requires an example. In the Schnorr scheme, the prover will compute $Y = g^y \bmod p$ then $Y' = h(Y)$ where h is a hash function. At the verification step, instead of checking if $Y \equiv g^z X^c \pmod p$, the verifier will check if $Y' = h(g^z X^c \bmod p)$. In practice, p may be a 1024 bits integer while the outputs of h may be 256 bits long, this means that the amount of data that the prover needs to send at the commitment step was divided by 4.

2.3. Precomputed Parts of the Answer. When the device that performs the identification has good storage capacities but a small computing power, a technique can be used to further simplify the computation of the answer.

This technique is based on two remarks. The first one is that, in most well-known identification schemes, the most expensive part of the computation of the answer only involves the private key and the challenge. In the GQ scheme, the answer is $z = yx^c \bmod N$ thus the expensive part involves x and c. In GPS, the answer is $z = y + cx$ and the expensive part is cx. The second remark is that, in practice, the challenge sent by the verifier can be relatively small, 16 or 32 bits for example.

Thus the idea is to precompute the expensive part of the answer for all possible values. In the case of GPS, this means computing cx for all possible values of c. In this case, for the price of storing B precomputed values, an identification only costs one addition.

A similar idea was used to efficiently implement GPS in [39].

3. Side-Channel Attacks

3.1. Principle. Side-channel attacks concern the implementation of a scheme on a specific device (e.g a laptop, a smart card). Such a device usually performs operations on secret data like a message or a key. The idea is to observe the physical behavior of the device; when there is a correlation between this behavior and the secret data stored in the device, then an attacker may analyze his observations to derive information on the data (and sometimes the whole data).

Side-channel attacks on public-key cryptographic primitives usually target the exponentiation algorithm and aim at finding the exponent. In an RSA signature, this exponent is the private key d; in a Schnorr signature, this exponent is an ephemeral secret y whose knowledge allows to find the private key as $x = c^{-1}(z - y) \bmod q$. This means that an attacker able to find this exponent completely breaks the scheme.

The basis of several attacks (including the first example of side-channel attacks proposed by Kocher [48] in 1996) is to make the assumption that the implementation uses a square-and-multiply algorithm. In this algorithm, a squaring occurs for each bit on the exponent and a multiplication occurs

only if this bit equals 1. If an attacker can observe the operations that the device performs and distinguish squarings from multiplications, he can recover the exponent. Often, the attacker will only obtain a list of data that is correlated to the behavior of the device, meaning that he will have to perform some kind of statistical analysis of this data.

3.2. Kinds of Attack. Commonly studied side-channel attacks are :

- *Power analysis* based on measuring the power consumption of the device. They were introduced by Kocher, Jaffe and Jun [49].
- *Electromagnetic radiation analysis* where the attacker monitor the electromagnetic field surrounding the device. The first scheme that was attacked was the DES.
- *Timing attacks* based on measuring the time between two observable events, for example the time between a question to the device and its answer
- *Fault attacks* where the attacker ensures that the device performs a computing error such that the device outputs some useful information on the secret

Such attacks often require a precise knowledge of the specific implementation that is analyzed.

3.3. Protection Against Side-Channel Attacks. The protection against side-channel attacks can be done at several levels. For example, in the case of electromagnetic attacks, one can use tricks that reduce the field surrounding the device.

The cryptographer can also design algorithms whose behavior reveals less information on the secret. An exponentiation algorithm whose execution time is constant will prevent timing attacks (but not necessarily other attacks).

Part 2

Security of Identification and Signature Schemes

CHAPTER 6

Hamming Weight Cryptanalysis of GPS

1. Introduction

1.1. Motivation. The GPS identification scheme is very efficient, it has a security proof and is a part of Nessie's portfolio of cryptographic primitives [27]. All these features make it suitable for implementation in several kinds of applications. However, an important open question remains: its resistance to side-channel attacks.

Many cryptographic schemes that are provably secure can be broken by a side-channel attack. Thus a security proof is not enough to ensure the security of a scheme in the real world: one needs to try to protect the scheme from physical attacks. The motivation of this chapter is to study the feasibility of a side-channel attack on the GPS scheme.

We show that a simple physical attack can efficiently break a non-protected implementation of GPS. By "simple", we mean that we simply need to be able to observe a modular exponentiation algorithm and find the Hamming weight of the exponent. This can be done in several ways, even with a basic timing attack. We then show how to protect the scheme from our attack.

1.2. Common Belief on Timing Attacks. Timing attacks against exponentiation schemes have been known for several years, and various countermeasures have been proposed against them. It is widely believed that these countermeasures are efficient and that timing attacks are not a critical security threat.

Since some of these countermeasures are precisely based on randomizing the exponent, one may feel tempted to believe that an algorithm where the exponent is random by nature, and where a different exponent is used for each execution may not be subject to a timing attack. We show that this is not true.

1.3. Using Partial Information on Ephemeral Secrets. The idea to attack a Schnorr-like scheme by recovering some information on ephemeral secrets then to use it to find the private key is not new; it is similar to a method proposed by Nguyen and Shparlinski to attack Schnorr's scheme.

In [56], they assume that they are given some known bits of the ephemeral keys (that were found via a side-channel attack for example).

They exploit the relationship between the ephemeral keys and the private key (namely, that $z = y + cx \mod q$ where y is the ephemeral key, x is the private key and z and c are known to the attacker) with the Lenstra-Lenstra-Lovász lattice basis reduction algorithm (LLL) [50] in order to find the private key.

1.4. Summary of the Results. We propose a side-channel attack strategy on GPS in two steps:

(1) A side-channel attack that finds the Hamming weight of the ephemeral secrets
(2) An algorithm that computes the private key from those Hamming weights

In the commitment step of GPS, a modular exponentiation $g^y \mod N$ occurs where the ephemeral secret y is the exponent. Side-channel attacks often aim at finding the exponent itself; in our work we only want to find its Hamming weight. This is not a strong requirement and, depending on the implementation, several side-channel attacks may be used to find this Hamming weight. The general idea is that in some implementations, the number of modular multiplications is equal to the number of 1's in the binary representation of the exponent. Thus, any side-channel attack that can provide some information about the number of multiplications that occurred will give some information about the Hamming weight.

A timing attack seems to be the easiest and simplest way to find those Hamming weights, therefore in the remaining of this chapter, we will consider a timing attack scenario.

In our scenario, the attacker impersonates the verifier, and is able to measure precisely the computation time for the commitment step. Apart from this, the attacker has no knowledge of the implementation, such as multiplication algorithm, time needed for an individual multiplication, ...

To our knowledge, none of the previously known timing attacks needed only the leakage of the exponent's Hamming weight ([31] and [58], for example, were based on modular reductions occurring in Montgomery multiplications). As a consequence, several of the countermeasures proposed against timing attacks turn out to be useless in our context.

Lastly, previous timing attacks required big amounts of data to work. Our attack only needs 600 timings to obtain the key with a probability of success of 72% after a few hours of treatment on a single PC, while 1000 timings allow 89% success after a few seconds of treatment. Collecting such a number of timings is easy (we recall that [40] states that the total time of the commitment followed by the answer takes less than 100 milliseconds with a crypto-processor card).

The second step of the strategy, that finds the private key given the Hamming weights of the ephemeral secrets, bears some similarities with the Nguyen-Spharlinski attack on Schnorr. But our method is different in that

it requires Hamming weights instead of bits and that our algorithm does not involve LLL.

2. Recovering Hamming Weights

2.1. The GPS Commitment Step. When implementing GPS, two methods are possible to manage the commitment issue. The first one consists in pre-computing commitments outside the device and then storing them in the device, using several tricks to save memory space (see [9] for details). This solution is interesting since it can be implemented on a device without crypto-processor, but it becomes a problem for applications that require a lot of identifications to be performed (such as pay-TV, for example): the device has to be reloaded with fresh commitments in a secure way periodically, or can only perform a limited number of identifications (an optimized version with 6000 pre-computed commitments needs about 36 kb of memory space).

The second method consists in making the device compute the commitments itself. This solution allows it to perform enough identifications for any application. Two variants of this method are possible: the computation can be done online, i.e. during the identification, or offline. The online variant requires a crypto-processor. In that case, the commitment can be computed in less that 100 milliseconds [40]. In the offline variant, the device computes the commitment before the execution of the identification itself. Speed is not an issue anymore, thus no crypto-processor is needed; however, the device needs to be supplied with current – and therefore, for classical smart cards, to be connected to a reader – during this computation.

2.2. Timing Attack. A timing attack on GPS is feasible under some conditions that we describe here.

We assume that the attacker is able to accurately measure the computation time of the commitment. We furthermore assume that this computation time is roughly linear in the Hamming weight of the exponent. This section briefly discusses the realism of these assumptions.

First of all, this will of course only be possible if the commitments are computed by the device, either online or offline. In the offline case, running times may be difficult to obtain by direct measurements, but indirect methods (e.g. based on power consumption) are still possible.

We also assume that the attacker is able to impersonate a honest verifier, which is a weak assumption: as resistance to dishonest verifier attacks is a natural requirement for an identification scheme, there is usually no need to bother about the verifier's identity. In its core version, GPS does not check the verifier's identity before entering the commitment phase.

As far as accuracy of the measurement is concerned, we believe our assumption to be realistic, for example in the context of a smart card (a typical target for side-channel attacks) since smart cards are usually not equipped with an internal clock, but have their clock signal provided by the reader they are put in; in this context, accurate running time is easy

to obtain with a rogue reader. To resist side-channel attacks, some modern smart cards get equipped with an internal clock. However, our attack seems to be robust against small imprecision in running time, and could therefore be carried out simply by measuring the elapsed time between startup and commitment.

Finally, let us explain why, in the case of a classical square and multiply algorithm, the running time will be a linear function of the Hamming weight of the exponent. In such algorithms, a multiplication occurs each time a 1 is read in the binary representation of the exponent, and a squaring occurs independently of the value of the bit; therefore the number of multiplications that occurred is *equal* to the number of 1's in the binary representation of the exponent, i.e. its Hamming weight. Moreover, the total running time of the exponentiation algorithm is equal to the time needed for squarings (which is constant) plus the time needed for multiplications. Thus this total running time is a linear function of the Hamming weight of the exponent.

Other methods, such as sliding windows or Walter's division chains [63], are discussed in section 5.

Remark: For the sake of simplicity, we will assume in this description that A is a power of 2 (in fact, it is likely that many implementations choose this value, because it makes generation of random elements in $[0, A[$ easier). Taking this value allows us to use a very simple formula linking the Hamming weight of the ephemeral key y and the probability for one of its bits to be 0. However, a different A makes computations more fastidious, but has no effect on the attack's efficiency.

2.3. Test Platform. To validate our attack, we performed timing measurements using a smart card development kit [51, 1]. Although not strictly practical, this scenario should be very close to the reality. Since the emulator is designed to allow implementors to optimize their code before "burning" actual smart cards, its predictions almost perfectly match the smart card's behavior. We therefore believe physical attack of an actual smart card should not induce much more measurement errors than the ones we encountered here.

2.4. Time Measurements. The attacker impersonates the honest verifier k times, always sending the same challenge[1] $c = 1$, and collects a list of couples $(t^{(i)}, z^{(i)}), i = 1 \dots k$, where $t^{(i)}$ is the computation time for the commitment of the i-th interaction, and $z^{(i)}$ is the corresponding answer $(z^{(i)} = y^{(i)} + x)$.

[1]Always sending $c = 1$ improves both the precision and the simplicity of the attack, but this is not a necessary condition. For example, the principle of the attack also works when the verifier always sends powers of two as challenges: the pair $(w(y), z = y + 2^v x)$ will leak information on bits $x_{|S|-1} \dots x_v$ of x.

Assuming the exponentiation's running time is roughly a linear function of the Hamming weight, i.e.

$$t^{(i)} = \alpha \times w(y^{(i)}) + \beta + \epsilon^{(i)},$$

with unknown parameters α and β ($\epsilon^{(i)}$ represents the error), the attacker estimates the Hamming weights of the set of ephemeral keys.

Figure 1 gives a clear idea of how parameters α and β can be estimated. The left graph shows the sorted running times of 80 exponentiations performed on our test platform (with the following parameters: $g = 2$, $|N|$=1024, $A = 2^{240}$ and without CRT) and the right graph shows the corresponding Hamming weights. Linear regression techniques on the left graph immediately provide good estimates for α and β.

These figures might deserve a bit more attention. As we can see, running times are grouped by "steps", and the attacker can safely assume that these "steps" correspond to several exponents having the same Hamming weight, whereas the average height between two "steps" is the time for a modular multiplication (i.e. α). The right graph shows that this estimate is pretty accurate: actual Hamming weight is rarely further than 2 from its estimate.

Remarks:

- We purposely took a small number of samples in order not to overload the figures. In practice, using a much larger set (say, 1000 samples) will of course reduce the error.
- We emphasize that this estimation is possible *whether the Chinese Remainder Technique is used or not.* Without CRT, the exponentiation time leaks information on $w(y)$. But with CRT, the device first computes $g^y \bmod p$ then $g^y \bmod q$ (since $y \ll p, q$, we have $y = y \bmod (p-1) = y \bmod (q-1)$), and the timing leaks information on $2 \times w(y)$.
- Smart card implementations of GPS probably require a constant time between the insertion of the card in the reader and the beginning of the computation of the commitment, meaning that the attacker gets a list $(t^{(i)} + R)$, where R is constant and unknown to the attacker, instead of a list $(t^{(i)})$. Thanks to the linear regression, this has no effect on the attack's efficiency.

3. From Hamming Weights to the Private GPS Key

In this section we describe an algorithm that recovers the private key given a list of Hamming weights of y values.

3.1. Overview. The attack proceeds in two phases: computation of a candidate for the key and key recovering.

The core principle of the attack goes as follows: suppose the attacker has obtained $w(y)$, the Hamming weight of some y generated by the prover.

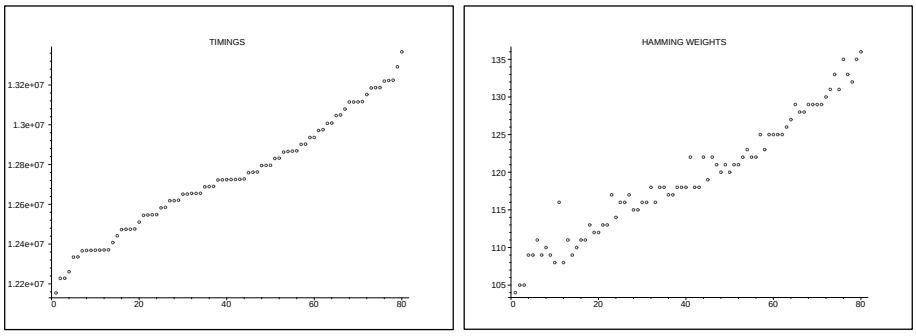

FIGURE 1. Sorted Timings for 80 Samples and Corresponding Hamming Weights

Impersonating the verifier and sending the challenge $c = 1$, he has also obtained the prover's response, $z = y + x$.

He starts by attacking the least significant bit of the secret, x_0. If the Hamming weight of y is smaller than half of y's bit length, he may assume that the least significant bit of y is more likely to be equal to 0 than to 1. More precisely, he assumes that the probability $P_0 = P(y_0 = 0)$ is equal to $1 - \frac{w(y)}{|A|}$.

Therefrom, and knowing the value of z, he can easily deduce a probability for the least significant bit of x to be zero.

Of course, this single guess has a non-negligible chance to be wrong. However, if the attacker is able to repeat this experiment over several identifications and compute the average of these probabilities, then this error risk shrinks.

Once he has obtained x_0, the attacker can deduce the actual value of the least significant bit $y_0^{(i)}$ of every message of his sample, which will allow him to attack the second bit x_1. Note that it is not necessary to collect new sets of samples.

3.2. Phase I: Computing a Candidate for the Key. In this phase, the attacker uses the collected data $(w^{(i)}, z^{(i)})$ to recover a candidate $\overline{x}$ for the private key.

3.2.1. *Estimation of* $P(y_j^{(i)} = 0)$. At step j of the procedure, we must estimate $P(y_j^{(i)} = 0)$ for each $i = 1, \ldots, k$. A basic estimation would be $P(y_j^{(i)} = 0) = 1 - (w^{(i)}/|A|)$, but we use two tricks to refine it:

- since $y^{(i)}$ is at least 64 bits longer than x, the first bits of $y^{(i)}$ are very likely to be the same as the first bits of $z^{(i)}$. We have $z_{|A|-1}^{(i)} \cdots z_{|S|+10}^{(i)} = y_{|A|-1}^{(i)} \cdots y_{|S|+10}^{(i)}$ with probability $1 - \frac{1}{2^{10}}$. For simplicity, we will assume in the following that this relationship always holds.

- if we already know $x_{j-1} \dots x_0$, then we can use $z^{(i)}$ to compute $y_{j-1}^{(i)} \dots y_0^{(i)}$.

This leads to the following estimation of $P(y_j^{(i)} = 0)$:

$$P(y_j^{(i)} = 0) = 1 - \frac{w(y_{|S|+9}^{(i)} \dots y_j^{(i)})}{|S| + 10 - j}$$

with

$$w(y_{|S|+9}^{(i)} \dots y_j^{(i)}) = w^{(i)} - w(z_{|A|-1}^{(i)} \dots z_{|S|+10}^{(i)}) - w(y_{j-1}^{(i)} \dots y_0^{(i)})$$

Our procedure will only store k values $w^{(i)}$ corresponding to the weight of the part of $y^{(i)}$ that has not been guessed yet.

3.2.2. *Guessing x_0.* For each $i = 1, \dots, k$, we have $z^{(i)} = y^{(i)} + x$. Thus $z_0^{(i)} = y_0^{(i)} \oplus x_0$. Therefore we have $P(x_0 = 0) = P(z_0^{(i)} = y_0^{(i)})$. For each i, we estimate $P(y_0^{(i)} = 0)$, then $P(z_0^{(i)} = y_0^{(i)})$:

$$P(z_0^{(i)} = y_0^{(i)}) = \begin{cases} P(y_0^{(i)} = 0) \text{ if } z_0^{(i)} = 0 \\ \\ 1 - P(y^{(i)} = 0) \text{ if } z_0^{(i)} = 1 \end{cases}$$

Using all the couples $(w^{(i)}, z^{(i)})$, we can use the following estimation:

$$P(x_0 = 0) = \frac{1}{k} \sum_{i=1}^{k} P(z_0^{(i)} = y_0^{(i)})$$

If we get $P(x_0 = 0) > 1/2$, we set $\overline{x}_0 = 0$; else we set $\overline{x}_0 = 1$.

3.2.3. *Guessing x_j for $j > 0$.* To try to guess x_j for $j > 0$, the situation is a little different, because we now have to deal with a carry: $z_j^{(i)} = y_j^{(i)} \oplus x_j \oplus carry_j^{(i)}$. We use our previous guesses $(\overline{x}_{j-1}, \dots, \overline{x}_0)$ to guess the carry. As we already explained, we also use $(\overline{x}_{j-1}, \dots, \overline{x}_0)$ to refine our estimation of $P(y_j^{(i)} = 0)$.

Each couple $(w^{(i)}, z^{(i)})$ leads to an estimation of $P(x_j = 0)$:

$$P(x_j = 0) \simeq P(y_j^{(i)} = z_j^{(i)} \oplus carry_j^{(i)})$$

We take the mean of these estimations for $i = 1, \dots, k$.

$$P(x_j = 0) = \frac{1}{k} \sum_{i=1}^{k} P(y_j^{(i)} = z_j^{(i)} \oplus carry_j^{(i)})$$

Again, if we get $P(x_j = 0) > 1/2$, we set $\overline{x}_j = 0$; else we set $\overline{x}_j = 1$.

Our procedure $FindCandidate$ stores the list $carry^{(i)}, i = 1, \ldots, k$, where $carry^{(i)}$ equals $carry_{j-1}^{(i)}$ at the beginning of procedure $PartialEstimate(i, j)$ and $carry_j^{(i)}$ at the end of this procedure.

Algorithm 3 $PartialEstimate(i, j)$: computes the i-th estimation of $P(x_j = 0)$ and the carry $carry_j^{(i)}$

$\overline{y}_{j-1}^{(i)} := z_{j-1}^{(i)} \oplus carry_{j-1}^{(i)} \oplus \overline{x}_{j-1}^{(i)}$
$w^{(i)} := w^{(i)} - \overline{y}_{j-1}^{(i)}$
if $\overline{x}_{j-1} + \overline{y}_{j-1}^{(i)} + carry_{j-1}^{(i)} \geq 2$ **then**
  $carry_j^{(i)} := 1$
else
  $carry_j^{(i)} := 0$
end if
Clear $carry_{j-1}^{(i)}$
Store $carry_j^{(i)}$
$P_j^{(i)} := 1 - (w^{(i)}/(|S| + 10 - j))$
if $z_j^{(i)} \oplus carry_j^{(i)} = 0$ **then**
  return $P_j^{(i)}$
else
  return $1 - P_j^{(i)}$
end if

Remark: If the estimation is wrong at step j, meaning that $\overline{x}_j \neq x_j$, the main consequence is an error on the next carry [2]. It is easy to see that this error will essentially propagate like a carry error in a classical addition, meaning that only a small block of the output bits will be erroneous. Our experiments confirmed this fact.

3.3. Phase II: Using the Candidate to Find the Key. At the end of Phase I, the attacker finds a candidate $\overline{x}$. If $g^{-\overline{x}} \equiv X \pmod{N}$, the attack is a success. Even if $\overline{x} \neq x$, the attacker can make an exhaustive search on values x' such that $d(x', \overline{x})$, the Hamming distance between x' and $\overline{x}$, is small, testing whether $g^{-x'} \equiv X \pmod{N}$. The maximum distance such that the attack can succeed depends on the capacity of the attacker; practical values are given in the next section.

4. Practical Results

We chose typical GPS values for the parameters in our tests: $g = 2$, $|S| = 160$, $|N| = 1024$, $A = 2^{240}$, then used the test platform described in

[2]Actually, a wrong estimation on x_j also induces an error on the updated estimation $w^{(i)}$. This error is small and does not affect the success of our attack.

Algorithm 4 *FindCandidate*: computes $\overline{x}$

for i=1 to k **do**
  $w^{(i)} := w^{(i)} - w(z^{(i)}_{|A|-1} \dots z^{(i)}_{|S|+10})$
end for
for $j = 0$ to $|S| - 1$ **do**
  $P := 0$
  for $i = 1$ to k **do**
    $P := P + PartialEstimate(i, j)$
  end for
  $P := P/k$
  if $P > 0.5$ **then**
    $\overline{x}_j := 0$
  else
    $\overline{x}_j := 1$
  end if
  Store $\overline{x}_j$
end for
return $\overline{x}_{|S|-1} \dots \overline{x}_0$

section 2.3 to obtain running times. Our exponentiation algorithm does not use the Chinese Remainder Technique.

For Phase II, we assumed that the attacker can perform 400 tests per second (which roughly corresponds to computations on a personal computer with a 2 Ghz processor with the GMP library). This leads to the following classification for the candidates:

(1) If $\overline{x} = x$, $\overline{x}$ is already the correct key
(2) $d(\overline{x}, x) \leq 2$, the attacker will need less than 16 seconds on average to find the correct key; we say that $\overline{x}$ is a "seconds" candidate
(3) $d(\overline{x}, x) \leq 4$, the attacker will need less than 9 hours on average to find the correct key; we say that $\overline{x}$ is a "hours" candidate
(4) $d(\overline{x}, x) \leq 5$, the attacker will need less than 12 days on average to find the correct key; we say that $\overline{x}$ is a "days" candidate

Our attack's results are summarized in Table 1.

TABLE 1. Simulation Results

k (number of samples)	200	400	600	800	1000
Immediate keys	0%	2%	21%	52%	72%
"seconds" candidates	0%	3%	54%	80%	89%
"hours" candidates	0%	6%	72%	94%	97%
"days" candidates	0%	10%	77%	96%	98%
avg. distance $d(\overline{x}, x)$	45.97	16.11	3.88	1.43	0.7

Phase I itself completes in a few seconds. With as little as 200 samples, the attack does not produce directly exploitable results; we get a value $\overline{x}$ that has 114 bits in common with the private key x while a random 160 bits value would have on average 80 bits in common with x. This means that we already obtain substantial information on the key.

5. Countermeasures Against the Timing Attack

Several countermeasures against timing attacks are known today. However, some care must be taken, as they will not all be efficient against our attack. In order to be efficient, a countermeasure must ensure that there is no correlation between the number of modular multiplications and the Hamming weight of the exponent.

Most timing attacks known so far exploited the modular reduction occurring in a Montgomery (resp. Barrett, ...) multiplication. Therefore, several countermeasures [32, 61, 62] typically consisted in removing the time variation in this multiplication. Since our attack is not based on the same property, these countermeasures are pointless.

Another frequently suggested countermeasure consists in randomizing the exponent by adding a random multiple of $\phi(N)$ to it, a modification that does not affect the final result. However, GPS, that was designed for efficiency, uses a short (typically, between 240 and 304 bits) exponent. Clearly, adding a multiple of $\phi(N)$ (typically, 1024 or 2048 bits) will imply a very serious performance drawback.

Simple sliding window techniques are probably too basic to make the attack impossible (in short, there is still a correlation between running time and Hamming weight).

Nevertheless, some other countermeasures are possible.

5.1. Pre-computed Commitments. A natural countermeasure is to use pre-computed commitments. This obviously makes the attack impossible. However, this solution may not be possible in some applications, where the device that performs the identifications cannot store too many pre-computed commitments.

5.2. Square and Multiply Always. Using dummy multiplications during the exponentiation allows to hide the hamming weight of the ephemeral keys. This countermeasure increases the computation time by about 30%.

5.3. MIST algorithm. A much more efficient countermeasure could be the use of Walter's MIST exponentiation algorithm [63]. MIST is not strictly constant-time (the greater the exponent, the longer the division chain). However, the information it could leak is probably limited to the strong bits of the exponent [64], which are already known to any eavesdropper (as they correspond to the strong bits of the answer). Thus MIST, which was designed to resist power analysis for RSA-like systems, i.e. when a same

exponent is used many times, seems to fit our needs with GPS, where a new exponent is used for each commitment.

6. How to Extend the Attack to Schnorr

A natural approach is to try to apply this attack to other schemes than GPS. First, let us notice that the nature of the attack requires an discrete-log based scheme, since finding the Hamming weight of the ephemeral secret seems to be only possible when it is used as an exponent. It turns out that the attack can be applied to Schnorr's scheme. Let us observe the main difference between Schnorr and GPS in the case of the attack. With GPS, the attacker gets a list $(w^{(i)}, z^{(i)})$ where $w^{(i)} = W(y^{(i)})$ and $z^{(i)} = y^{(i)} + x$. With Schnorr, a modular reduction occurs so the attacker gets a list $(w^{(i)}, z^{(i)})$ where $w^{(i)} = W(y^{(i)})$ and $z^{(i)} = y^{(i)} + x \bmod q$.

The idea is to get rid of this modular reduction. We first notice that when $z = y + x \bmod q$ where y and x are in $[0, q[$, then either $z = y + x$ if $z \geq x$ or $z = y + x - q$ if $z < x$.

If we sort and reindex the z values, we obtain a list $z^{(1)} < \ldots < z^{(k)}$. Let $t \in [0, k]$ be the index such that $z^{(t)} < x$ and $z^{(t+1)} \geq x$.

We have $z^{(i)} = y^{(i)} + x - q$ for $i = 1, \ldots, t$ and $z^{(i)} = y^{(i)} + x$ for $i = t + 1, \ldots, k$. Therefore, from this list, if we know the index t, we can compute a list of k integers $z'^{(i)}$ such that $z'^{(i)} = y^{(i)} + x$ for $i = 1, \ldots, k$, simply by setting $z'^{(i)} = z^{(i)} + q$ if $i = 1, \ldots, t$ and $z'^{(i)} = z^{(i)}$ otherwise.

From this list $z'^{(i)}$ and the list of corresponding Hamming weights of $y^{(i)}$, we can apply the same Hamming weight cryptanalysis that in GPS.

One difficulty remains: how to find the index t ? A first possibility is an exhaustive search. For each of the $k + 1$ possible values for t, one computes the corresponding list $z'^{(i)}$ then applies the Hamming weight cryptanalysis. This means that the complexity of the attack is multiplied by a factor $k + 1$ which may be considered as acceptable.

Another way to find t exploits the list of Hamming weight values $w^{(i)}$. Small values of $y^{(i)}$ will probably have several 0's at the beginning of their binary representation, thus a sequence of values with low Hamming weight is likely to start at index $t + 1$.

7. Conclusion

This works shows how straightforward GPS implementations can easily be broken. Our attack is efficient in that it does not need a lot of samples and only requires limited computing power. It demonstrates that it is necessary to use countermeasures that hide the Hamming weight of the exponent when implementing GPS, Schnorr or other discrete-log based identification and signature schemes.

Because of the history of timing attacks on RSA and the fact that countermeasures against these attacks are well known, people seem to believe that the problem of timing attacks is solved in general; we show with the

example of GPS that this is not true. Some schemes, even when implemented with basic countermeasures, can succumb to a timing attack.

CHAPTER 7

Bimodular RSA Padding

1. Introduction

Finding provably secure ways to sign with RSA is an important topic of research. Given RSA parameters (N, e, d), the problem is to design a provably secure signature scheme. A common approach is to apply some kind of encoding function to the message then raising the result to the power d modulo N. Namely, the signing algorithm is $m \to \mu(m)^d \bmod N$ where μ is the encoding function. This is the basis of algorithms like RSA-PSS [19].

Definition 1.1. *We say that a function μ is a* secure encoding function *if the signature scheme whose signing generation algorithm is $m \to \mu(m)^d \bmod N$ is existentially unforgeable under adaptive chosen message attacks.*

In the random oracle model, if μ behaves as a random oracle then it is a secure encoding function; this is the idea of the Full Domain Hash (FDH for short) scheme. However, security proofs in the random oracle model can be only considered as heuristic, since in the real world random oracles are necessarily replaced by functions which can be computed by all parties. A famous result by Canneti, Goldreich and Halevi [26] shows that a security proof in the random oracle model does not necessarily imply security in the standard model. In the standard model, finding a secure encoding function is still an open problem.

A question that arises on this topic is the importance of the length of the messages that need to be signed. One could think that designing an encoding function for messages of arbitrary length is more difficult that designing an encoding function for messages of fixed, relatively short length.

This question was studied by Coron, Koeune and Naccache at Asiacrypt 2000 [29]. They showed that if a secure encoding function for short messages exists, then it can be turned into a secure encoding function for arbitrarily long messages. In other words, there is no need to worry about message length when trying to design a secure encoding function.

However, their construction requires that the input size ℓ of $\mu(m)$ be larger than the size of N (hereafter denoted k). Several standards (for example the ISO/IEC 9796-1 standard [2]) fail to comply with this property. The authors left as an open problem the case $\ell \leq k$.

Our contribution is to solve this open problem and provide a construction for any input size ℓ. A variant of this problem was already solved by

Arboit and Robert in [3], who proposed a construction similar to [29] that works for any ℓ, but at the cost of a new security assumption, namely the division intractability of the encoding function $\mu(m)$. The advantage of our construction is that we do not make any additional assumptions, namely if RSA signature with $\mu(m)$ is secure against existential forgery under a chosen-message attack, then so is RSA with $\mu'(m)$ for messages of arbitrary size. As is the case for the constructions in [29] and [3], a practical advantage of our construction is that it allows to perform some pre-computations on partially received messages, *e.g.* on IP packets which are typically received in random order.

2. The Classical RSA Paradigm

We recall in figure 1 the classical RSA paradigm where an encoding function μ is applied.

System parameters
- two integers $k > 0$ and $\ell > 0$
- a function $\mu : \{0,1\}^\ell \rightarrow \{0,1\}^k$

Key generation: Setup
- $(N, e, d) \leftarrow RSA(1^k)$
- public key : (N, e)
- private key : (N, d)

Signature generation: Sign
- let $y \leftarrow \mu(m)$
- return $y^d \bmod N$

Signature verification: Verify
- let $y \leftarrow x^e \bmod N$
- let $y' \leftarrow \mu(m)$
- if $y = y'$ then return one else return zero.

FIGURE 1. The Classical RSA Paradigm: Using μ for Signing Fixed-Length Messages.

3. The Coron-Koeune-Naccache Construction

We recall in figure 2 the construction proposed by Coron, Koeune and Naccache in [29]. It assumes that the encoding function μ can handle inputs of size $k+1$ where k is the size of the modulus and allows to sign $2^a \cdot (k-a)$ bit messages where $0 \leq a \leq k-1$. The construction can be recursively iterated to sign messages of arbitrary length.

It is shown in [29] that the scheme described in figure 2 is secure against existential forgery under a chosen message attack:

System parameters
- two integers $k > 0$ and $a \in [0, k-1]$
- a function $\mu : \{0,1\}^{k+1} \to \{0,1\}^k$

Key generation: Setup*
- $(N, e, d) \leftarrow RSA(1^k)$
- public key : (N, e)
- private key : (N, d)

Signature generation: Sign*
- Split the message m into $(k-a)$-bit blocks such that $m = m[1]||\ldots||m[r]$.
- let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$ where i in $0||i||m[i]$ is an a-bit string representing i.
- let $y \leftarrow \mu(1||\alpha)$
- return $y^d \bmod N$

Verification: Verify*
- let $y \leftarrow x^e \bmod N$
- let $\alpha = \prod_{i=1}^{r} \mu(0||i||m[i]) \bmod N$
- let $y' \leftarrow \mu(1||\alpha)$
- if $y = y'$ then return one else return zero.

FIGURE 2. Coron-Koeune-Naccache Encoding of Arbitrary Length Messages.

THEOREM 3.1. *If the signature scheme* (Setup, Sign, Verify) *is* $(t, q_{sig}, \varepsilon)$ *secure, then the signature scheme* (Setup*, Sign*, Verify*) *which signs* $2^a \cdot (k-a)$ *bit messages is* $(t^*, q^*_{sig}, \varepsilon^*)$ *secure, where:*

$$t^*(k) = t(k) - 2^a \cdot q_{sig}(k) \cdot \mathcal{O}(k^2)\,, \tag{1}$$

$$q^*_{sig}(k) = q_{sig}(k) - 2^{a+1}\,, \tag{2}$$

$$\varepsilon^*(k) = \varepsilon(k)\ . \tag{3}$$

4. Bimodular Encoding

The drawback of the previous construction is that the the input size ℓ of $\mu(m)$ needs to be larger than the size of the modulus N. In this section, we describe a construction wherein the input size ℓ of the encoding function μ does not need to be larger than k. We denote by $\ell(k)$ the input size of the encoding function μ as a function of the security parameter k. In the following, we assume that $\ell(k)$ is an increasing function of k. For example, for the ISO/IEC 9796-1 standard [2], we have $\ell(k) \simeq k/2$.

The new signature scheme (Setup', Sign', Verify') is described in figure 3. The new signature scheme is parameterized by two security parameters

System parameters
- two positive integers k_1, k_2 such that $k_2 > k_1$
- an integer $a \in [0, k_1 - 1]$
- two functions $\mu_i : \{0,1\}^{\ell_i} \to \{0,1\}^{k_i}$ for $i = 1, 2$ such that $\ell_2 \geq k_1$.

Key generation: Setup′
- $(N_1, e_1, d_1) \leftarrow RSA(1^{k_1})$
- $(N_2, e_2, d_2) \leftarrow RSA(1^{k_2})$
- public key : (N_1, N_2, e_2)
- private key : (N_1, N_2, d_2)

Signature generation: Sign′
- Split the message m into $(\ell_1 - a)$-bit blocks such that $m = m[1]||\ldots||m[r]$.
- let $\alpha = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1$ where i in $i||m[i]$ is an a-bit string representing i.
- let $y \leftarrow \mu_2(\alpha)$
- return $y^{d_2} \bmod N_2$

Verification: Verify′
- $y \leftarrow x^{e_2} \bmod N_2$
- let $\alpha = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1$
- let $y' \leftarrow \mu_2(\alpha)$
- if $y = y'$ then return one else return zero.

FIGURE 3. Bimodular Encoding of Arbitrary Length Messages.

k_1, k_2 such that $k_1 < k_2$. As the previous construction, it is a deterministic signature scheme. The construction uses the same encoding function μ with two distinct moduli N_1 and N_2 of sizes k_1 and k_2 bits, respectively. For the sake of clarity and since encoding functions take the modulus as a parameter, we will write μ_i when μ is used with modulus N_i. We denote by $\ell_1 = \ell(k_1), \ell_2 = \ell(k_2)$ the input sizes of μ_1, μ_2 respectively. Our construction requires that $\ell_2 \geq k_1$. Since by assumption $\ell(k)$ is an increasing function of k, this means that a sufficiently large security parameter k_2 must be selected in order to have $\ell_2 = \ell(k_2) \geq k_1$

Our construction enables to sign $2^a \cdot (\ell_1 - a)$ bit messages where $0 \leq a \leq \ell_1 - 1$. The maximum length that can be handled by the new construction is therefore $2^{\ell_1 - 1}$ bits for $a = \ell_1 - 1$ or $a = \ell_1 - 2$ and, as in [29], the construction can be recursively iterated so as to sign arbitrarily long messages.

A possible realization example is the following: assume that we are given an encoding function μ that takes as input $k/2$-bit messages and outputs k-bit strings, for signing with a k-bit RSA modulus. If we take for example

$k_1 = 1024$, $k_2 = 2048$ and $a = 24$, then messages of size up to $2^{24} \cdot 488 \simeq 8.2 \cdot 10^9$ bits can be signed. First, one applies the encoding function $\mu_1 : \{0,1\}^{512} \to \{0,1\}^{1024}$ to the 2^{24} blocks of 488 bits; then one multiplies together the resulting 1024-bit integers modulo N_1 and obtains a 1024-bit integer which is finally signed using the encoding function $\mu_2 : \{0,1\}^{1024} \to \{0,1\}^{2048}$ modulo N_2. Notice that d_1 is not used for signing and e_1 is not needed for the verification either; thus (e_1, d_1) is to be deleted after the generation of N_1.

The following theorem states that this construction preserves the resistance against chosen message attacks of the original signature scheme:

THEOREM 4.1. *If the signature scheme* (Setup, Sign, Verify) *is* $(t, q_{sig}, \varepsilon)$ *secure, then the signature scheme* (Setup', Sign', Verify') *which signs* $2^a \cdot (\ell_1 - a)$ *bit messages is* $(t', q'_{sig}, \varepsilon')$ *secure, where:*

$$t'(k_1, k_2) = t(k_1) - q'_{sig} \cdot 2^a \cdot \left(T_\mu(k_2) + \mathcal{O}({k_2}^3)\right) , \tag{4}$$

$$q'_{sig}(k_1, k_2) = q_{sig}(k_1) - 2^{a+1} , \tag{5}$$

$$\varepsilon'(k_1, k_2) = 4 \cdot \varepsilon(k_1) \ . \tag{6}$$

and $T_\mu(k_2)$ *is the time required to compute* $\mu(m)$ *for security parameter* k_2.

PROOF. Without loss of generality, we can assume that $t(k)$, $q_{sig}(k)$ and $T_\mu(k)$ are increasing functions of k, and that $\varepsilon(k)$ is a decreasing function of k.

Let $\mathcal{F}'$ be a forger that breaks the signature scheme (Setup', Sign', Verify') for the parameters (k_1, k_2). We construct a forger $\mathcal{F}_1$ for the signature scheme (Setup, Sign, Verify) for the parameter $k = k_1$ and a forger $\mathcal{F}_2$ for same signature scheme with parameter $k = k_2$. When the same property holds for both $\mathcal{F}_1$ and $\mathcal{F}_2$, we write this property for a generic forger $\mathcal{F}$. The forger $\mathcal{F}$ will run $\mathcal{F}'$ in order to produce a forgery; it will answer the signature queries of $\mathcal{F}'$ by itself. $\mathcal{F}$ has access to a signing oracle $\mathcal{S}$ for (Setup, Sign, Verify) .

First, we pick a random bit b. If $b = 1$, we construct a forger $\mathcal{F}_1$ for the parameter $k = k_1$. If $b = 0$, we construct a forger $\mathcal{F}_2$ for the parameter $k = k_2$.

$\mathcal{F}$ is first given as input (N, e) where N, e were obtained by running Generate for the parameter k defined previously. The forger $\mathcal{F}$ then starts running $\mathcal{F}'$ with the public key (N_1, N_2, e_2), where N_1, N_2, e_2 are defined as follows:

If $b = 1$, the forger $\mathcal{F}_1$ sets $N_1 \leftarrow N$, $e_1 \leftarrow e$ and runs $RSA(1^{k_2})$ to obtain (N_2, e_2, d_2). Otherwise (if $b = 0$) the forger $\mathcal{F}_2$ sets $N_2 \leftarrow N$, $e_2 \leftarrow e$ and runs $RSA(1^{k_1})$ to obtain (N_1, e_1, d_1).

We observe that the view of the forger $\mathcal{F}'$ in independent of the bit b, since in both cases the moduli N_1 and N_2 are generated using $RSA(1^{k_1})$ and $RSA(1^{k_2})$, either by $\mathcal{F}$ itself or through (N, e) given as input to $\mathcal{F}$.

When $\mathcal{F}'$ asks the signature of the j-th message m_j with $m_j = m_j[1]||\dots||m_j[r_j]$, $\mathcal{F}$ computes:

$$\alpha_j = \prod_{i=1}^{r_j} \mu_1(i||m_j[i]) \bmod N_1$$

If $b = 0$ then $\mathcal{F}_2$ requests the signature s_j of α_j from $\mathcal{S}$. If $b = 1$ then $\mathcal{F}_1$ can compute $s_j = \mu_2(\alpha_j)^{d_2} \bmod N_2$ directly since it knows d_2. Let q'_{sig} be the total number of signatures requested by $\mathcal{F}'$.

Eventually $\mathcal{F}'$ outputs a forgery (m', s') for the signature scheme (Setup', Sign', Verify') with $m' = m'[1]||\dots||m'[r']$, from which $\mathcal{F}$ computes:

$$\alpha' = \prod_{i=1}^{r'} \mu_1(i||m'[i]) \bmod N_1 \tag{7}$$

We denote by β the probability that $\alpha' \notin \{\alpha_1, \dots, \alpha_q\}$. Note that since the view of $\mathcal{F}'$ is independent of b, this event is independent of b as well. We distinguish three cases:

First case: $\alpha' \notin \{\alpha_1, \dots, \alpha_q\}$ and $b = 0$. From the remark above, this happens with probability $\beta/2$. In which case $\mathcal{F}_2$ outputs the forgery (α', s') and halts. This is a valid forgery for the signature scheme (Setup, Sign, Verify) since $s' = \mu_2(\alpha')^{d_2} \bmod N_2$ and the signature of α' was never asked to the signing oracle $\mathcal{S}$.

Second case: $\alpha' \in \{\alpha_1, \dots, \alpha_q\}$ and $b = 1$. This happens with probability $(1 - \beta)/2$. Let c be such that $\alpha = \alpha_c$. We write $m = m_c$, $\alpha = \alpha_c$ and $r = r_c$, which gives using (7):

$$\prod_{i=1}^{r'} \mu_1(i||m'[i]) \bmod N_1 = \prod_{i=1}^{r} \mu_1(i||m[i]) \bmod N_1 \tag{8}$$

We show that the previous equation leads to a multiplicative forgery for the modulus $N_1 = N$, which enables $\mathcal{F}_1$ to compute a forgery.

First, the message m' must be distinct from m because the signature of m has been requested by $\mathcal{F}'$ whereas the signature of m' was never requested by $\mathcal{F}$, since m' is the message for which a forgery was obtained. Consequently there exists an integer j such that either:

$$j||m'[j] \notin \{1||m[1], \dots, r||m[r]\} \tag{9}$$

or:

$$j||m[j] \notin \{1||m'[1], \dots, r'||m'[r']\} \tag{10}$$

We assume that condition (9) is satisfied (condition (10) leads to the same result). Therefore from (8) we can write:

$$\mu(j||m'[j]) = \Big(\prod_i \mu(i||m[i])\Big)\Big(\prod_{i \neq j} \mu(i||m'[i])\Big)^{-1} \bmod N_1 \tag{11}$$

Consequently, $\mathcal{F}_1$ asks the signing oracle $\mathcal{S}$ for the signatures x_i of the messages $i||m[i]$, $1 \leq i \leq r$, and for the signatures x'_i of the messages $i||m'[i]$, $1 \leq i \leq r'$, $i \neq j$. Using (11), $\mathcal{F}_1$ can compute the signature of $j||m'[j]$ from the other signatures:

$$x'_j = \mu(j||m'[j])^{d_1} = \Big(\prod_i x_i\Big)\Big(\prod_{i \neq j} x'_i\Big)^{-1} \bmod N_1$$

and $\mathcal{F}_1$ finally outputs the forgery $(j||m'[j], x'_j)$. This is a valid forgery for the signature scheme (Setup, Sign, Verify) since the signature of $j||m'[j]$ was never asked to the signing oracle.

Third case: $\alpha' \notin \{\alpha_1, \ldots, \alpha_q\}$ and $b = 1$, or $\alpha' \in \{\alpha_1, \ldots, \alpha_q\}$ and $b = 0$. In this case, $\mathcal{F}$ fails. This happens with probability $1/2$.

To summarize, from a forger $\mathcal{F}'$ that breaks the signature scheme (Setup', Sign', Verify') with probability $\varepsilon'(k_1, k_2)$ for the parameters (k_1, k_2), we construct a forger $\mathcal{F}$ that breaks the signature scheme (Setup, Sign, Verify) with probability $\varepsilon' \cdot \beta/2$ for the parameter k_2, and with probability $\varepsilon' \cdot (1-\beta)/2$ for the parameter k_1, for some (unknown) β.

Therefore, if we assume that the signature scheme (Setup, Sign, Verify) cannot be broken in time $t(k)$ with probability greater than $\varepsilon(k)$ for all k, we must have:

$$\varepsilon'(k_1, k_2) \cdot \beta/2 \leq \varepsilon(k_2)$$

and

$$\varepsilon'(k_1, k_2) \cdot (1-\beta)/2 \leq \varepsilon(k_1)$$

which implies using $\varepsilon(k_2) \leq \varepsilon(k_1)$ that:

$$\varepsilon'(k_1, k_2) \leq 4 \cdot \varepsilon(k_1)$$

which gives (6).

If $b = 0$, then for each of the q'_{sig} queries of $\mathcal{F}'$, the forger $\mathcal{F}_2$ makes at most 2^a multiplications modulo N_1 and one query to $\mathcal{S}$. Thus $\mathcal{F}_2$ runs in time

$$t(k_2) = t'(k_1, k_2) + q'_{sig} \cdot 2^a \cdot \big(T_\mu(k_1) + \mathcal{O}({k_1}^2)\big) \tag{12}$$

If $b = 1$ then for each query of $\mathcal{F}'$, the forger $\mathcal{F}_1$ makes at most 2^a multiplications modulo N_1 and one exponentiation modulo N_2. After it has received the forgery, it makes at most 2^{a+1} multiplications modulo N_1 to compute its own forgery. Thus $\mathcal{F}_1$ runs in time:

$$t(k_1) = t'(k_1, k_2) + q'_{sig} \cdot \big(2^a \cdot \big(T_\mu(k_1) + \mathcal{O}({k_1}^2)\big) + T_\mu(k_2) + \mathcal{O}({k_2}^3)\big) \tag{13}$$

From inequalities (12) and (13), and using $t(k_1) \leq t(k_2)$ and $T_\mu(k_1) \leq T_\mu(k_2)$ we obtain (4).

Finally, the forger $\mathcal{F}_2$ makes at most q'_{sig} queries to the signing oracle, and the forger $\mathcal{F}_1$ makes at most 2^{a+1} queries to the signing oracle. This

gives $q_{sig}(k_2) \leq q'_{sig}(k_1, k_2)$ and $q_{sig}(k_1) \leq 2^{a+1}$. Using $q_{sig}(k_1) \leq q_{sig}(k_2)$, we obtain

$$q_{sig}(k_1) \leq 2^{a+1} + q'_{sig}(k_1, k_2),$$

which gives (5). □

5. Conclusion

This work shows how to construct a secure RSA encoding scheme for signing arbitrarily long messages, given any secure encoding scheme for signing fixed-size messages. This solves a problem left open by Coron *et al.* in [29]. We believe that our work focuses the question of finding a secure encoding for RSA signatures, by showing that the difficulty in building secure encoding schemes for RSA is not in handling messages of arbitrary length, but rather in finding a secure redundancy function for short messages, which remains an open problem in the standard model.

Part 3

Fair Exchange of Signatures and Identities

CHAPTER 8

Breaking a Fair Exchange Scheme Based on GPS and RSA

1. Introduction

In [53], Markowitch and Saeednia proposed an optimistic fair exchange scheme. The scheme is based on a verifiably committed signature scheme that the authors introduce. This verifiably committed signature scheme is constructed from the basic RSA encryption scheme and the GPS signature scheme and was also used later in the context of a non-repudiation protocol [52].

One advantage of this scheme is its rather low communication and computational cost thanks to the use of GPS. An interesting feature of the scheme is that it uses the RSA primitive to allow partial signatures to be opened by Charlie.

In this chapter, we show that the security against Bob is not satisfied in the verifiably committed signature from [53]. We describe an attack allowing to extract the full signature σ from a partial signature σ'. Our attack is extremely efficient (it only requires that the attacker previously obtained a very small number of full signatures, and has a computational cost of a few exponentiations). As a consequence, the fairness property of two protocols [53, 52] is broken.

Of independent interest, we show how the combination of GPS and RSA can be very sensitive to attacks. Informally, the problem of extracting a RSA root of a GPS commitment has some poor security properties: there is a (non-negligible) subset of the instances of this problem for which the knowledge of a solution allows the computation of the solution for *any* instance.

2. Description of the Committed Signature Scheme

This section describes the verifiably committed signature scheme of [53]. In this scheme, the idea is to use GPS signatures combined with the RSA encryption primitive, a valid signature being a GPS signature along with the RSA decryption of the corresponding commitment. As a consequence, the hardness of extracting a full signature from a partial signature implicitly relies on the problem of computing the RSA decryption of a GPS commitment given the corresponding challenge and answer, but without knowing neither the GPS private key nor the RSA private key.

We now describe the committed signature scheme.

Setup : Charlie generates an RSA modulus $N = pq$ where $p = 2p' + 1, q = 2q' + 1$ and p, p', q, q' are primes. He chooses a hash function h. Charlie chooses an integer g of order $p'q'$. He also chooses an integer e coprime to $p'q'$, and computes $d = e^{-1} \bmod p'q'$. Charlie publishes N, g, h and e. Charlie's secret arbitration key is d.

To generate her key pair, Alice chooses a random integer x as her private key and computes $X = g^{-x} \bmod N$ as her public key.

PSign : To compute a partial signature on a message m, Alice takes a random integer y and computes $Y = g^{ey} \bmod N$, and $z = ey + h(Y, m)x$. The partial signature is (Y, z).

PVerify : To verify a partial signature (Y, z) on m, Bob - or any verifier - checks if $Y \equiv g^z X^{h(Y,m)} \pmod N$.

Sign : To compute a full signature on m corresponding to her partial signature (Y, z), Alice computes $Y' = g^y \bmod N$, with the value of y used in the partial signature process. The full signature is (Y', z) with the same z than in the partial signature.

Verify : To verify a full signature, (Y', z) on m, Bob - or any verifier - checks if $Y'^e \equiv g^z X^{h(Y'^e \bmod N, m)} \pmod N$.

Res : In case Alice refuses to open her signature to Bob, Charlie uses the partial signature (Y, z) and his secret arbitration key d to compute the corresponding final signature (Y', z) where $Y' = Y^d \bmod N$.

The signature protocol is as follows: Alice computes a partial signature (Y, z), where $Y = g^{ey} \bmod N$, sends this partial signature to Bob, and stores the value $Y' = g^y \bmod N$. Bob checks that $Y \equiv g^z X^{h(Y,m)} \pmod N$, then fulfills his obligation I. Alice sends Y' to Bob, and Bob checks that $Y'^e \equiv Y \pmod N$.

Remarks:

(1) Note that this verifiably committed signature scheme does not exactly fit the model of Dodis and Reyzin, since both the Sign and Verify algorithm depend on the public arbitration key e.
(2) Since $z \equiv h(Y, m)x \pmod e$, where $z, h(Y, m)$ and e are public, it is clear that some information on $x \bmod e$ leaks. For this reason, the authors of [53] recommend to take $e = 3$ to minimize the amount of information leaked on the private key x.

3. Extracting Full Signatures from Partial Signatures

In this section we describe an attack that allows an attacker to extract the full signature (Y', z) on m from the partial signature (Y, z). The main idea of the attack is that computing Y', the e-th root of $Y = g^z X^a \bmod N$ for a given integer a, is simple when one knows the e-th root of the value

$g^{ax \bmod e} X^{a \bmod e} \bmod N$ thanks to the euclidian division. We now explain this more in the detail.

3.1. Preliminary: the F function. Let F be the application from $[0, e[$ to $[0, N[$ defined by:

$$F(c) = \left(g^{cx \bmod e} X^c\right)^d \bmod N$$

Note that $F(0) = 1$. We are going to show how the properties of this function allow attacking the scheme. The two first properties already show the insecurity of the scheme for small values of e; the third property may be viewed as an enhancement of the attack.

PROPERTY 1. *Knowing a full signature (Y', z) allows computing $F(a \bmod e)$, where $a = h(Y'^e \bmod N, m)$.*

Proof: Since $Y'^e \equiv g^z X^a \pmod N$, we apply an euclidian division on z and a and we have:

$$Y'^e \equiv g^{(z \text{ div } e)e + z \bmod e} X^{(a \text{ div } e)e + a \bmod e} \pmod N$$

Therefore

$$\begin{aligned}
(Y' g^{-(z \text{ div } e)} X^{-(a \text{ div } e)})^e &\equiv g^{z \bmod e} X^{a \bmod e} \pmod N \\
&\equiv g^{ax \bmod e} X^{a \bmod e} \pmod N \\
Y' g^{-(z \text{ div } e)} X^{-(a \text{ div } e)} &\equiv (g^{ax \bmod e} X^{a \bmod e})^d \pmod N \\
&\equiv F(a \bmod e) \pmod N \qquad (14)
\end{aligned}$$

Thus $F(a \bmod e)$ can be computed as $Y' g^{-(z \text{ div } e)} X^{-(a \text{ div } e)} \bmod N$. □

PROPERTY 2. *Given a partial signature (Y, z), the knowledge of $F(a \bmod e)$ where $a = h(Y \bmod N, m)$ allows computing the corresponding full signature (Y', z).*

Proof: From equation (14), we have

$$Y' = g^{z \text{ div } e} X^{a \text{ div } e} F(a \bmod e) \bmod N$$

□

PROPERTY 3. *Knowing $F(a_0 \bmod e)$ such that $a_0 = h(Y_0'^e \bmod N, m)$ is coprime to e for some full signature (Y_0', z_0) allows computing $F(a)$ for any $a \in [0, e[$.*

Proof: Since a_0 is coprime with e, we can compute $k = {a_0}^{-1} a \bmod e$.

By euclidian division,

$$\begin{aligned}
k(a_0 \bmod e) &= [k(a_0 \bmod e)] \text{ div } e \times e + [k(a_0 \bmod e)] \bmod e \\
&= [k(a_0 \bmod e)] \text{ div } e \times e + k a_0 \bmod e \\
&= [k(a_0 \bmod e)] \text{ div } e \times e + a
\end{aligned}$$

$$\begin{aligned} k(a_0 x \bmod e) &= [k(a_0 x \bmod e)] \text{ div } e \times e + [k(a_0 x \bmod e)] \bmod e \\ &= [k(a_0 x \bmod e)] \text{ div } e \times e + k a_0 x \bmod e \\ &= [k(a_0 x \bmod e)] \text{ div } e \times e + a x \bmod e \end{aligned}$$

therefore

$$\begin{aligned} F(a_0 \bmod e)^{ek} &\equiv \left(g^{a_0 x \bmod e} X^{a_0 \bmod e}\right)^k \pmod N \\ &\equiv g^{k(a_0 x \bmod e)} X^{k(a_0 \bmod e)} \pmod N \\ &\equiv \left(g^{k(a_0 x \bmod e) \text{ div } e} X^{k(a_0 \bmod e) \text{ div } e}\right)^e g^{ax \bmod e} X^a \pmod N \\ &\equiv \left(g^{k(z_0 \bmod e) \text{ div } e} X^{k(a_0 \bmod e) \text{ div } e}\right)^e (F(a))^e \pmod N \end{aligned}$$

We raise to the power d modulo N and we obtain

$$F(a) = F(a_0 \bmod e)^k g^{-(k(z_0 \bmod e) \text{ div } e)} X^{-(k(a_0 \bmod e) \text{ div } e)} \bmod N \tag{15}$$

□

3.2. Attack Strategy. The attack proceeds in two steps. In the first step, the attacker obtains a small number of full signatures from the signer. In the second step, he asks the signer any number of committed signatures and extracts the corresponding full signature.

(1) In this step, the attacker obtains a number of full signatures until he finds a signature (Y_0', z_0) (on a message m_0) such that $a_0 \bmod e$ is coprime with e, where $a_0 = h(Y_0'^e \bmod N, m_0)$. Once he has obtained such a signature, he computes $F(a_0 \bmod e)$ and stores the triple $(a_0 \bmod e, z_0 \bmod e, F(a_0 \bmod e))$.

(2) In this step, the attacker plays the role of the verifier. Once he gets a committed signature (Y, z), he computes $F(a \bmod e)$ where $a = h(Y \bmod N, m)$, using the stored triple $(a_0 \bmod e, z_0 \bmod e, F(a_0 \bmod e))$ and equation (15). He then uses this value of $F(a \bmod e)$ to extract the full signature from the committed one, according to property 2. The attacker thus obtains the full signature without fulfilling his obligation. This step obviously succeeds with probability 1, and can be done with any number of committed signatures.

It is important to notice that the first step can be *active* or *passive*. It can be performed by Bob, who behaves honestly with Alice during a few executions of the protocol, or by an outsider who eavesdrop the signatures.

3.3. An Example with Artificially Small Parameters. Let us show an example of the attack with artificially small parameters. We take $e = 3$. Alice, whose private key is $x = 110$, takes a random $y = 213$, and computes $Y = g^{3 \times 213} \bmod N$. She computes $h(Y, m) = 16$, and $z = ey + h(Y, m)x = 3 \times 213 + 16 \times 110 = 2399$. The full signature contains Y' such that $Y'^3 \equiv g^{2339} X^{16} \pmod N$.

The attacker sees this full signature. Since $2339 = 799 \times 3 + 2$ and $16 = 5 \times 3 + 1$, the attacker computes $\tau = Y' g^{-799} X^{-5} \bmod N$. This τ is such that $\tau^3 \equiv g^2 X \pmod N$.

Now, the attacker starts the protocol with Alice and receives a partial signature from which is he is going to extract the full signature. Assume that the partial signature is $(\widetilde{Y}, \widetilde{z})$ where $\widetilde{z} = 1933$ and $h(\widetilde{Y}, \widetilde{m}) = 14$. He has $\widetilde{Y} = g^{1933} X^{14} \bmod N = g^{3\times 644+1} X^{3\times 4+2} \bmod N$, thus he needs to find a e-th root of $gX^2 \bmod N$. This root is $\tau^2 g^{-1}$ thanks to equation (15).

Thus the attacker can compute $\widetilde{Y}' = g^{644} X^4 \tau^2 g^{-1} \bmod N$, where $\widetilde{Y}'^e \bmod N = \widetilde{Y}$, meaning that $(\widetilde{Y}', \widetilde{z})$ is the full signature corresponding to the committed one: the attack is a success.

3.4. Efficiency Discussion. Since the second step always succeeds once the first step ended successfully, the efficiency of the attack only depends on the number of full signatures the attacker has to obtain in the first step.

If we make the reasonable assumption that the outputs of the hash function taken modulo e are uniformly distributed over $[0, e[$, then, for each signature (Y', z), the probability that $a \bmod e$ is coprime with e is $\phi(e)/e$. Thus the first step succeeds after at most l tries with probability

$$P_l = 1 - \left(1 - \frac{\phi(e)}{e}\right)^l$$

This formula allows us to evaluate the efficiency of the attack depending on the value of the e parameter.

3.4.1. *The $e = 3$ case.* It is natural to study this single case since it is the value suggested by the authors. We have $\phi(3) = 2$, thus the attack succeeds after at most l tries with probability

$$P_l = 1 - \frac{1}{3^l}$$

Obtaining one signature allows a success probability of 66%, whereas 5 signatures allow a probability of success of more than 99%.

3.4.2. *The $e \leq 2^{10}$ case.* In [53], it is specified that e must be a very small integer. Thus we can assume that $e < 2^{10}$. To compute a lower bound of $\phi(e)/e$ for values of e between 2 and 2^{10}, we simply compute all the possible values of $\phi(e)/e$. The lower bound is reached for $e = 210$. Thus we have that

$$\forall e \in [1, 2^{10}], \quad \frac{\phi(e)}{e} \geq \frac{\phi(210)}{210} = \frac{8}{35} \approx 0.22857$$

While with a single signature, the lower bound on the probability of success is already 22%, we can see that, with 10 signatures, we have a probability of success of at least 92%.

4. Repairing the Scheme

4.1. First Attempt: Increasing the Value of e. A natural idea to repair the scheme, since the efficiency of our attack depends on $\phi(e)/e$, is to minimize this value by taking a larger and specific value for e. Of course, since $x \bmod e$ leaks during any execution of the protocol, we have to take a larger value for x. More precisely, we have to ensure that $x \operatorname{div} e$ is large enough (at least 160 bits) because $Xg^{-x \bmod e} = (g^e)^{x \operatorname{div} e} \pmod N$ thus finding x is equivalent to solving the discrete logarithm of $Xg^{-x \bmod e} \bmod N$ in basis $g^e \bmod N$. This implies a performance decrease since the size of the commitments must be such that $y >> 2^{|h|}e$.

But we have the following well-known bound: when e is any number such that $e \geq 5$,

$$\phi(e) \geq \frac{e}{6 \ln \ln e} \tag{16}$$

Thus even if e is 4000 bit long, then 2.1% of the full signatures are such that a is coprime with e. Thus with 33 full signatures the attacker will succeed with probability 1/2.

In short, taking a larger value for e does not prevent the attack, but just makes it a little less efficient. We conclude that the scheme is insecure for *any* value of e.

4.2. How to Repair the Scheme with GQ. We propose a modified scheme that uses non-interactive proofs-of-knowledge. The idea is that the attack is possible because Y', the e-th root of Y, is revealed by Alice. Thus we keep the same protocol except for the full signature process. Alice proves that she knows Y', but this time she does not reveal it. This can be done with a non-interactive zero-knowledge proof-of-knowledge of e-th root of Y, such as a non-interactive version of the Guillou-Quisquater protocol, for example.

We now describe the modified scheme. The Setup, PSign and PVerify algorithms are the same as in the original scheme. The other algorithms work as follows:

- Sign : To compute a full signature on m corresponding to her partial signature (Y, z), Alice computes $Y' = g^y \bmod N$, with the value of y used in the partial signature process. Then she picks $u \in \mathbb{Z}_N^*$, computes $U = u^e \bmod N$ and $V = u(Y')^{h(U)} \bmod N$. The full signature is (Y, z, U, V) with the same z than in the partial signature.
- Verify : To verify a full signature (Y, z, U, V), Bob checks if $V^e \equiv UY^{h(U)} \pmod N$ and if $Y \equiv g^z X^{h(Y,m)} \pmod N$.
- Res : In case Alice refuses to open her signature to Bob, Charlie uses the partial signature (Y, z) and his secret arbitration key d to compute $Y' = Y^d \bmod N$. Then he picks $u \in \mathbb{Z}_N^*$, computes

$U = u^e \bmod N$ and $V = u(Y')^{h(U)} \bmod N$. The full signature is (Y, z, U, V).

This modified scheme is less efficient than the original one but seems to resist the attack.

5. GPS and RSA can be Securely Combined

In order to show that GPS and RSA can be securely combined, we recall a way to use them together invented by Girault. The GPS scheme was initially proposed as a system with *self-certified* public keys. The motivation is that any public key needs a way to be certified. Usually, one uses a certificate, i.e. a document produced by an authority that links the public key to the identity of its owner. In order to remove the need for such certificates, Girault proposed in [38] the concept of self-certified public keys, where the user chooses a private key and the authority computes the corresponding public key without learning the private key.

The TTP generates an RSA modulus $N = pq$ where p and q are primes, a public exponent $e \in [0, N[$, computes $d = e^{-1} \bmod \phi(N)$, and chooses an integer g of large order modulo N. It publishes N, g and e. The prover, whose identity is I, picks a random integer $x \in [0, S[$ as her secret key and sends $g^{-x} \bmod N$ to the authority. The authority computes the prover's public key as $X = (g^{-x} - I)^d \bmod N$. Therefore the prover knows x such that $g^{-x} \equiv X^e + I \pmod N$.

In the GPS identification protocol with self-certified public keys, the prover proves that she knows that x. It is the usual GPS scheme where the public key X is replaced by $X^e + I \pmod N$.

Prover		**Verifier**
Pick $y \in_R [0, A[$ $Y = g^y \bmod N$		
	$\xrightarrow{Y}$	
		Pick $c \in_R [0, B[$
	$\xleftarrow{c}$	
Check $c \in [0, B[$ $z = y + cx$		
	$\xrightarrow{z}$	
		Check $z \in [0, E[$ $g^z (X^e + I)^c \stackrel{?}{\equiv} Y \pmod N$

FIGURE 1. A Round of GPS using Self-Certified Public Keys

6. Conclusion

We showed that the security against the verifier was not verified in the committed signature scheme from [53] (also used in [52]) by describing an efficient method to extract a full signature from a partial signature.

This attack is a new example of a well-known fact: constructing a new cryptographic primitive from two secure primitives can lead to a totally insecure result. But, interestingly, we notice that GPS and RSA can be securely combined as it was the case with the initial design of GPS with self-certified public keys [38].

We conclude that the security of a cryptographic protocol should not rely on the hardness of computing the RSA decryption of a GPS commitment.

CHAPTER 9

A New Problem: Fair Identification

1. Motivation

We present a new cryptographic problem called fair identification, where two members of a group agree to exchange proofs of identities (i.e. to simultaneously *reveal* and *prove* their identities).

We consider a group of anonymous users. By anonymous, we mean that each user has an identity (it can be a name, an address, a pseudonym) but two users can meet and discuss without knowing each other's identity. One can imagine an organization where participants of a meeting wear masks, or a chat session on the internet. In such a setting, users have a motivation to hide their identity: it can be for privacy reasons or for their personal security for example. Thus it may be against someone's interests, or even dangerous, to reveal his or her identity.

However, situations can occur where two people that met want to learn each other's identity. Each of them must be ready to reveal his or her identity in order to learn the identity of the other person. In this case, it is natural for each person to require a *proof* of the other's identity. Otherwise, it would be very simple to cheat as a dishonest user could simply give a false identity; the other, honest, user would reveal his or her identity and the exchange would be unfair.

Without a dedicated protocol, each user will hesitate in revealing his or her identity first. They need a *fair* protocol that ensures that no one is cheated, i.e. no one revealed his or her identity without learning the other's identity. This new problem is similar to the problem of fair exchange of signatures (even if they are some fundamental differences that will be discussed later) and we call it *fair exchange of identities* or *fair identification* for short.

We call the two persons that want to exchange their identities A and B; a fair identification protocol should satisfy the following security properties:

- *Completeness:* If A and B are honest, both of them learn each other's identity.
- *Fairness:* If anyone is cheating, either no one learns anything about the identity of the other or both of them learn each other's identity.
- *Secrecy:* Identities of A and B remain secret to an outsider against active attacks, unless the outsider is ready to reveal his identity.

If we want to minimize the intervention of the trusted third party, we should look for *optimistic* protocols where the third party is needed only in case of disputes. This fourth property is inspired by optimistic fair exchange of signatures.

Harder variants of this problem are also possible: for example, one could consider concurrent attacks instead of active attacks, but in this work we will focus on active attacks only.

2. Similar Problems

Several problems similar to fair identification have been studied in the literature. It is thus a natural approach to examine a few kinds of cryptographic primitives used to solve such problems to see if they can trivially achieve fair identification.

Mutual Authentication: A mutual authentication scheme [18, 12, 44] allows two parties to authenticate each other. Once A proved her identity, B can abort the scheme without revealing his identity, therefore such schemes do not achieve fairness.

Fair Exchange of Signatures: In a fair exchange of signatures protocol, each party obtains the other's signature in a fair manner. If A and B run a fair exchange of signatures, A will commit to her signature. Once B has verified this committed signature, he will learn A's identity without revealing his identity. Therefore fair exchange of signature protocols do not provide any direct solution to the fair identification problem.

Identity Escrow: Identity escrow schemes [47] are inspired by group signatures and allow an entity A to send some information to B that commits to A's identity, meaning that this information would allow an authorized third party to recover A's identity. A and B could run a fair identification protocol as follows:

(1) A runs the identity escrow protocol with B.
(2) B confirms his true identity to A.
(3) A confirms her true identity to B.

If A is cheating, B can go to the escrow agent with transcripts of identity escrow protocol in step 1 to obtain A's identity.

This protocol is fair but eavesdroppers can learn the identities of A and B. Encrypting the communication prevents passive attacks but not active attacks. Thus this protocol does not satisfy the requirements for fair identification. However, we will use this idea to commit to one's identity in our fair identification protocol.

Since there seems to be no trivial way to achieve fair identification, we propose a new scheme. Rather than build a scheme from scratch, we use existing cryptographic primitives and combine them to design a fair identification scheme in order to emphasize on the security mechanisms that we need.

3. Building Blocks

In this section, we introduce the cryptographic primitives that we use as building blocks for our fair identification protocol. These building blocks are a signature scheme with a special feature, a public key encryption scheme, and a group signature scheme.

3.1. Signatures with Key-Independent Coupons. Some signature schemes have the interesting feature that a part of the signature can be computed prior to the knowledge of the message to sign. They are sometimes called on-line/off-line signatures [59] or coupon-based signatures, and the pre-computed part is called the coupon.

In this work, we additionally require that the coupon can be computed without knowing the signer's private key and that the coupon does not reveal anything about the identity of the signer. We call such a scheme a *signature scheme with key-independent coupons*.

Many known signature schemes satisfy this requirement; in fact, many signature schemes obtained by applying the Fiat-Shamir heuristic do (the coupon is the commitment of the corresponding identification scheme), like the Schnorr, GQ and GPS signature schemes for example, as long as all users have the same system-wide parameters. In the GQ scheme, for example, users must all have the same modulus N and exponent e. In that setting, a coupon will be a value $y^e \bmod N$; the knowledge of the coupon does not reveal the identity of the signer.

More formally, a signature scheme with key-independent coupons is a tuple of algorithms $\mathcal{SS} = (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$ satisfying the usual properties of a signature scheme where: $\mathcal{K}_S$, $\mathcal{S}$, and $\mathcal{V}$ are the key-generation, signing and verification algorithms respectively. Additionally, $\mathcal{S}$ internally works as follows ($s =$ signing key and $\hat{st} =$ some state information, $m =$ message to be signed): $\mathsf{Algorithm}\ \mathcal{S}(m, s) : \{(x, \hat{st}) \leftarrow \mathcal{S}_1(); y \leftarrow \mathcal{S}_2(x, \hat{st}, m, s); \mathrm{Return}(x, y); \}$.

Here, $\mathcal{S}_1$ is the algorithm that generates the coupon and $\mathcal{S}_2$ is the algorithm that generates the remaining part of the signature.

Let *cert* denote the certificate for the public key of a signature scheme with key-independent coupons. This certificate contains the identity I_U of user U.

3.2. Public Key Encryption in Multi-User Setting. By $\mathcal{PE}$ we denote an IND-CCA secure public-key encryption scheme secure in a multi-user setting according to the definitions of Bellare et al [10].

A public-key encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms. The *key generation* algorithm $\mathcal{K}$ is a randomized algorithm that takes nothing as input and returns a pair (pk, sk) of matching public and secret keys; we write $(pk, sk) \stackrel{R}{\leftarrow} \mathcal{K}()$. The *encryption* algorithm $\mathcal{E}$ is a randomized algorithm that takes the public key pk and a *plaintext* M (from the message space $\mathrm{MsgSp}(pk)$) to return a *ciphertext* C'; we write $C' \stackrel{R}{\leftarrow} \mathcal{E}_{pk}(M)$.

The *decryption* algorithm $\mathcal{D}$ is a deterministic algorithm that takes the secret key sk and a ciphertext C' to return the corresponding plaintext M (or a special symbol $\perp$ if C' is invalid); we write $M \leftarrow \mathcal{D}_{sk}(C')$.

Two ideal examples of such encryption schemes are Cramer-Shoup [30, 10] and RSA-OAEP [17, 13, 60, 37].

3.3. Group Signatures. Group signatures allow a group member to sign anonymously on behalf of the group. If needed, the signature can be opened by a trusted third party, called group manager, to reveal the identity of the signer. For an in-depth discussion on group signatures see [25, 24, 7, 14, 20].

Informally, a group signature scheme $\mathcal{G} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ is a 5-tuple of algorithms, where:

–Setup is an algorithm which takes no input. It initializes the system and outputs the group public key GPK, secret data for the group manager, and any other parameters needed.

–Join is an interactive protocol executed between the group manager and a user (say A). As a result, A learns his secret data g_A for generating group signatures and the group manager might also learn some data to aid him later in opening the signatures if required.

–Sign is the signing algorithm. It takes as input the message m to be signed and the secret data g_A of any user A to produce a group signature σ_A.

–Verify is the algorithm to verify the correctness of a group signature on a given message. It takes as input the message m, the signature σ and the group public key GPK; it outputs 1 if σ is a valid group signature on m, and 0 otherwise.

–Open is the algorithm that only the group manager can use to identify the signer of a particular group signature. It takes as input the signature σ, manager's secret data for opening group signatures and perhaps some other information; its output is a proof identifying the signer of the signature.

For notation, $\sigma_A(m)$ will denote the group signature of A on message m. When only σ_A is written, it means that it is a group signature of A on whatever message and that opening it would identify A as its signer.

The following properties are desirable for group signatures:

- *Correctness:* Group signatures produced using Sign are always accepted by Verify.
- *Unforgeability:* Only group members can sign efficiently.
- *Anonymity:* Given a group signature, it is computationally hard to identify its signer for everyone but the group manager.
- *Unlinkability:* Deciding whether two valid group signatures were computed by the same group member, is computationally hard.
- *Exculpability:* Neither a group member nor the group manager can produce a group signature on behalf of any other member.

- *Openability:* The group manager is always able to open and identify the actual signer of a valid group signature.
- *Coalition Resistance:* A colluding subset of group members (even if comprised of the entire group) cannot generate a group signature that the group manager cannot open.

Let us explain why and how we use group signatures in our scheme. Our approach to design a fair identification scheme is inspired by the way fair exchange of signature schemes are built. In order to fairly exchange signatures, users needs a way to commit to their signatures such that a third party can reveal the signature if needed: this is the concept of verifiably committed signatures. Similarly, in order to fairly exchange identities, users need a way to commit to their identities. Group signatures solve this problem and can be seen as "verifiable commitments to identity".

4. A Fair Identification Protocol

4.1. Sketch. In this section, we describe a fair identification protocol. We start by giving a sketch of our protocol. First, A generates and sends the coupon of his signature. Then B generates a group signature on A's coupon and sends it to A. A now has the commitment of B to his identity and hence he now sends his identity and the remaining part of the signature. B verifies the signature and then sends his identity and signature to A. To remain anonymous to outsiders, they encrypt the communication using temporary keys (using $\mathcal{PE}$). To avoid active attacks, these keys are signed under the group signature of B. Let T denote the trusted third party which will also take the role of group manager in the group signature scheme.

The protocol works as follows.

(1) A computes a signature coupon x_A and a pair of keys (pk_A, sk_A) for the encryption scheme, then sends the coupon and the public key to B.
(2) B generates a pair of keys (pk_B, sk_B) for the encryption scheme and computes a group signature on $x_A \| pk_A \| pk_B$. B sends pk_B and the group signature σ_B to A.
(3) A verifies the group signature; if it is valid, A computes the signature (x_A, y_A) (where x_A is the coupon) on $x_A \| \sigma_B$ then encrypts this signature and his certificate $cert_A$ under B's public encryption key pk_B. A sends this encrypted message to B.
(4) B decrypts the message, obtains the certificate of A, learns A's identity and checks the signature using the certificate. If it is valid, B signs σ_B and sends an encryption of σ_B and $cert_B$ under A's public key pk_A.
(5) A decrypts the message, obtains the certificate of B, learns B's identity and checks the signature using the certificate.

If A does not receive the final message, A can go to the third party and convince it that B agreed to reveal his identity to A by showing the

transcripts of the protocol. The third party will open the group signature, find B's identity and reveal it to A.

4.2. Protocol Description.

4.2.1. *Setup.* System specific parameters:

(1) T declares: (a) A secure signature scheme with key-independent coupons $\mathcal{SS} = (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$, and (b) An IND-CCA secure public-key encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ to be used by each player, whenever needed.
(2) Pick a group signature scheme $\mathcal{G} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ with T being the group manager. T creates an instance of $\mathcal{G}$ by running the procedure Setup. T learns the secrets corresponding to the group manager and let GPK be the group public key.

User specific parameters:

(1) Each user U first decides his public and private keys, I_U and s_U respectively, for the signature scheme $\mathcal{SS}$ by running $\mathcal{K}_S$. U proves to T that I_U is his public key.
(2) Now U runs the Join protocol of $\mathcal{G}$, with T, to learn his group specific secret g_U needed to produce the group signatures.
(3) T generates a certificate $cert_U$ mentioning that person with public key I_U is registered with T.

4.2.2. *Exchange.* The exchange protocol is described in figure 1.

4.2.3. *Resolve.* Party A claiming to be cheated, presents σ_B to T and the corresponding message components x_A, pk_A, pk_B, full signature S_A on σ_B and identifies herself as A to T. T sends a proof identifying the signer (B) of σ_B to A and sends the full signature S_A to B.

5. Security Discussion

We now discuss the security of our fair identification protocol.

5.1. Completeness. If all the steps of the protocol succeed, then A and B provably learn each other's identity: A receives B's signature on message (σ_B, s_B) after step 4. Since σ_B itself depends on some values sent by A (namely, x_A and pk_A), this means that A's was indeed talking to the owner of the private key s_B; the certificate $cert_B$ proves that the owner of this key has identity I_B. Thus A obtains I_B and is convinced that this is B's identity.

Similarly, B receives A's signature on a message that contains σ_B after step 3. When he receives the certificate, he obtains I_A and is convinced that this is A's identity. This shows that the scheme is complete.

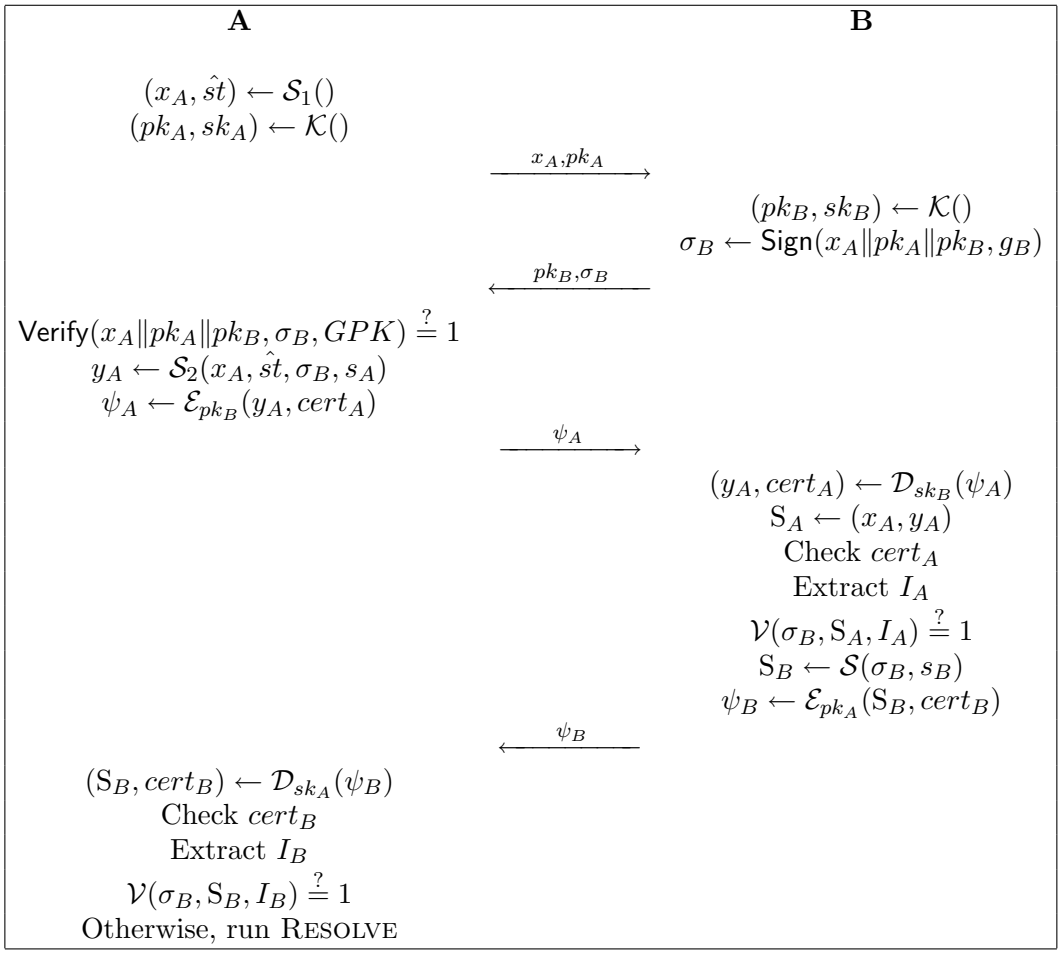

FIGURE 1. Proposed Fair Identification Protocol

5.2. Fairness. We show that in all circumstances the protocol ends in a fair situation (i.e. a situation where A and B learned nothing about each other's identity, or a situation where they both learned each other's identity).

In step 1, A must send x_A and pk_A to B.

- If B does not receive these values, the protocol stops and the situation is fair.
- Otherwise, it goes on.

In step 2, B must send pk_B and σ_B to A.

- If A does not receive these values, or if σ_B is invalid, the protocol stops. B learned x_A and pk_A. By assumption, the coupon x_A does not reveal anything on A's private key or any information on A's identity. Moreover, pk_A is a random output of the setup procedure

of the encryption scheme. Therefore, B has no information on A's identity: the situation is fair.
- Otherwise, the protocol goes on.

In step 3, A must send ψ_A to B.

- If B does not receive a valid ψ_A, the protocol stops. A learned pk_B and σ_B. The security of the group signature ensures that σ_B does not reveal B's identity and pk_B is a random output of the setup procedure of the encryption scheme. This shows that A cannot learn B's identity: the situation is fair.
- Otherwise, the protocol goes on.

In step 4, B must send ψ_B to A.

- If A does not receive a valid ψ_B, the protocol stops. B obtains y_A and $cert_A$, therefore B learns A's identity. A goes to the third party and learns B's identity. The situation is fair.
- Otherwise, the valid ψ_B contains a proof of B's identity. The situation is fair.

5.3. Secrecy. We now consider an attacker trying to learn A or B's identity.

5.3.1. *Passive Attacks.* In the case of a passive attack, the attacker only has access to the values that are exchanged by A and B: x_A, pk_A, pk_B, σ_B, ψ_A and ψ_B.

In order to find A's identity, the attacker must decrypt ψ_A; this is impossible if the encryption scheme is secure.

In order to find B's identity, the attacker must decrypt ψ_B or open the group signature σ_B. This is impossible if the encryption scheme and the group signature scheme are secure.

5.3.2. *Active Attacks.* In the case of A's identity, we assume that the attacker plays the role of B and selects his messages in order to learn A's identity. The attacker will obtain A's identity in step 3 but only if he can send A a valid group signature on $x_A \| pk_A \| pk_B$. Therefore the attacker has two solutions: he can break the group signature scheme or compute a group signature scheme. Under the assumption that the group signature scheme is secure, the attacker will therefore have to commit to his identity and A will be able to learn this identity.

In the case of B's identity, we assume that the attacker plays the role of A and selects his messages in order to learn B's identity. The attacker can try to open B's group signature, or try to obtain the certificate $cert_B$. Under the assumption that the group signature scheme is secure, B cannot open σ_B. Thus he has to obtain ψ_B, thus send a valid ψ_A to B, therefore revealing his identity.

6. Extensions and Future Work

In this section, we discuss a few variants to the problem of fair identification. While not all of these variations might find a practical application, they constitute an interesting challenge.

6.1. Transferable Proofs of Identity. One could imagine a scenario where users want to fairly exchange *transferable* proofs of identity. In this case, the precise statement of the problem will be: how should two parties identify each other so that either each party learns a transferable proof of the other's identity or none of them learns nothing about the identity of the other.

In our fair identification protocol, the proofs of identity that users get if the protocol ends normally are indeed transferable (because those proofs are signatures), but it does not necessarily mean that our protocol is a fair exchange of identity proofs. This property can be ensured if group signatures can be opened in a transferable manner, i.e. when the third party opens the group signature, it outputs a transferable proof that identifies its signer. Though it might not be possible for every group signature scheme, it is indeed the case in modern schemes. For example, in ACJT scheme [7], this proof is actually an interactive zero-knowledge proof of equality of two discrete logarithms, between the group manager and the party interested in identifying the signer. This can be made transferable by applying the Fiat-Shamir heuristic. Thus, if cheating occurs in our protocol, the cheated party can also get the transferable proof which will be nothing but the group signature together with a transferable proof identifying its signer (obtained from T).

Now let us consider a stronger requirement: the proofs obtained with the help of the third party should be at least computationally indistinguishable from the proofs obtained directly from the concerned player. This requirement has an interesting link with the fair exchange of signatures where signatures opened by the third party should be indistinguishable from signatures computed by the signer.

Such indistinguishability cannot be achieved using our protocol. It would be interesting to see whether existing protocols for fair exchange of signatures can be modified to provide fair identification too.

6.2. Nontransferable Proofs of Identity. If the fair identification protocol between A and B ended successfully, A revealed her identity to B, so B can claim to another user that he performed an exchange with A. But A might want to be sure that B cannot *prove* that they indeed met. A protocol that would ensure this for A and B would be a fair exchange of nontransferable proofs of identities.

More precisely, if A and B execute the fair identification protocol and identify each other, then after the protocol completes, transcripts of the run

prove nothing to anyone – transcripts could have very well been simulated by A (or B) himself.

Our protocol cannot be used to guarantee nontransferable proofs of identity. One might think of using interactive proof protocols instead of signatures to achieve this goal. Although this could work, neither this construction nor its security proof are trivial.

6.3. Conditional Fair Identification. Assume a situation where the parties know in advance whom they want to identify fairly. More precisely, A is willing to achieve fair identification only with B, and B is willing to achieve fair identification only with A. With very little tweaking, our protocol should work for this kind of problem. However a more general situation is the following one: A is willing to achieve fair identification with B if and only if B satisfies *some condition*, say $\mathcal{C}_B$. Similarly, B is willing to do fair identification with A if and only if A satisfies some condition, say $\mathcal{C}_A$. We term it as conditional fair identification. Our protocol may work for this, depending upon what these conditions are.

6.4. Perfect Fairness. In this work, the coalition of adversaries was considered to be one single adversary. Following an approach used in traitor tracing schemes, we aimed at identifying at least one user in the coalition. However, a stronger notion of fairness is possible where one could aim at identifying *each* player in the coalition. Precisely, perfect fairness for A would mean that if A is identified by any player B in a particular run of the protocol, A will also identify *exactly* B. With this perfect fairness notion, it should not matter whether B is in a coalition or not. In order to achieve perfect fairness, it is necessary to use non-transferable proofs; otherwise, just one person could interact with A and obtain a transferable proof of identifying A and then show it to everyone in the coalition. Perfect fairness seems to be the most challenging property to ensure.

Conclusion

This thesis studied several examples of approaches to establish in a provable manner the security (and, sometimes, the insecurity) of several public-key identification and signature schemes.

We first showed how a side-channel attack could break a provably secure identification and signature scheme with the example of GPS. This result illustrates the fact that even when using basic countermeasures, cryptographic schemes can be the target of new side-channel attacks. We believe that future research should continue to try to find new ways to resist physical attacks.

We then designed a construction allowing to sign arbitrarily long messages with RSA, given a secure encoding for short messages. While finding a secure encoding for RSA signatures in the standard model remains an open problem, our result shows that the difficulty is not in handling messages of arbitrary length, but rather in finding a secure encoding for short messages.

The Markowitch-Saeednia verifiably committed signature scheme highlighted the interest of using GPS and RSA to design efficient primitives; however, by showing how to break the fairness property of this scheme, we demonstrated the difficulty of designing such schemes in a *secure* manner. There are several open questions related to this result: what schemes can be designed that take advantage of the features of GPS and RSA ? How to design efficient verifiably committed signature schemes ?

We studied a new problem called fair identification and proposed a scheme that solves this problem. Our scheme can be seen as a proof that fair identification can be achieved, but designing efficient and provably secure fair identification schemes is a new and challenging open problem.

List of Figures

Bibliography

[1] Cascade (Chip Architecture for Smart CArds and portable intelligent DEvices). Project funded by the European Community, see `http://www.dice.ucl.ac.be/crypto/cascade`.

[2] ISO/IEC 9796-1. Information technology - Security techniques - Digital signature scheme giving message recovery, Part 1: Mechanisms using redundancy, 1991.

[3] G. Arboit and Robert J.-M. From Fixed-Length Messages to Arbitrary-Length Messages Practical RSA Signature Padding Schemes. In D. Naccache, editor, *Topics in Cryptology - CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 44–51. Springer-Verlag, 2001.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt 98*, 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 Communication*, 18(4):593–610, 2000.

[6] G. Ateniese. Efficient Verifiable Encryption (and Fair Exchange) of Digital Signatures. In G. Tsudik, editor, *Sixth ACM Conference on Computer and Communication Security and Privacy*, pages 138–146. ACM, November 1999.

[7] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[8] F. Bao, R. Deng, and W. Mao. Efficient and Practical Fair Exchange Protocols with Off-line TTP. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[9] O. Baudron, F. Boudot, P. Bourel, E. Bresson, J. Corbel, L. Frisch, H. Gilbert, M. Girault, L. Goubin, J.-F. Misarsky, P. Nguyen, J. Patarin, D. Pointcheval, J. Stern, and J. Traoré. GPS, an asymetric identification scheme for *on the fly* authentification of low cost smart cards. Submitted to the European NESSIE Project, 2000. Available at `https://www.cryptonessie.org/`.

[10] M. Bellare, A. Boldyreva, and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt'00*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000.

[11] M. Bellare, A. Boldyreva, and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Springer, editor, *Eurocrypt'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188, 2004.

[12] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. In *Procceedings of the 30th annual Symposium on the Theory of Computing – STOC*, pages 419–428. ACM Press, 1998.

[13] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - Crypto'98*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[14] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simpliefied requirements, and a construction based on general assumptions. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[15] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In M. Yung, editor, *Advances in Cryptology - Crypto'02*, Lecture Notes in Computer Science. Springer, 2002.

[16] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computers and Communications Security*. ACM, 1993.

[17] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Advances in Cryptology - Eurocrypt'94*, Lecture Notes in Computer Science. Springer, 1995.

[18] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Procceedings of the 27th annual Symposium on the Theory of Computing – STOC*, pages 57–66. ACM Press, 1995.

[19] M. Bellare and P. Rogaway. The exact security of digital signatures - How to sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology - Eurocrypt 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 1996.

[20] D. Boneh, X. Boyen, and H. Shacham. Short Group Signatures. In *Advances in Cryptology - CRYPTO 2004*, Lecture Notes in Computer Science. Springer, 2004.

[21] 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.

[22] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the weil pairing. In C. Boyd, editor, *Advances in Cryptology - Asiacrypt'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[23] J. Camenisch and I.B. Damgård. Verifiable Encryption, Group Encryption, and their Application to Group Signatures and Signature Sharing Schemes. In T. Okamoto, editor, *Advances in Cryptology - Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 331–345. Springer-Verlag, 2000.

[24] J. Camenisch and M. Michels. A Group Signature Scheme with Improved Efficiency. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 1998.

[25] J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. In B.S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[26] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In ACM, editor, *STOC '98*, 1998.

[27] Nessie consortium. Portfolio of recommended cryptographic primitives. Available at `https://www.cryptonessie.org/deliverables/decision-final.pdf`, 2003.

[28] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[29] J.-S. Coron, F. Koeune, and D. Naccache. From fixed-length to arbitrary-length RSA padding schemes. In T. Okamoto, editor, *Advances in Cryptology - Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 90–96. Springer-Verlag, 2000.

[30] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - Crypto'98*, Lecture Notes in Computer Science. Springer, 1998.

[31] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In J.-J. Quisquater and B. Schneier, editors, *Proc. CARDIS 1998, Smart Card Research and Advanced Applications*, LNCS. Springer, 1998.

[32] J.F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards*. PhD thesis, Université catholique de Louvain - UCL Crypto Group - Laboratoire de microélectronique (DICE), May 1998.

[33] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Trasanctions on Information Theory*, 22(6):644–654, 1976.

[34] Y. Dodis and L. Reyzin. Breaking and Repairing Optimistic Fair Exchange from PODC 2003. In M. Yung, editor, *ACM Workshop on Digital Rights Management (DRM)*, pages 47–74, 2003.

[35] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, New York, NY, USA, 1990. ACM Press.

[36] A. Fiat and A. Shamir. How to prove yourself : practical solutions of identification and signature problems. In G. Brassard, editor, *Advances in Cryptology - Proceedings of CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[37] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure under the RSA Assumption. Available on eprint archive – `http://eprint.iacr.org/2000/061`.

[38] M. Girault. Self-Certified Public Keys. In D.W. Davies, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 1991*, volume 0547 of *Lecture Notes in Computer Science*, pages 490–497. Springer, 1991.

[39] Marc Girault and David Lefranc. Public key authentication with one (online) single addition. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2004.

[40] Marc Girault, Guillaume Poupard, and Jacques Stern. Some modes of use of the GPS identification scheme. In *Proceedings of the third Nessie workshop*, November 6-7 2002.

[41] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[42] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. In *SIAM Journal of computing*, volume 17, pages 281–308, april 1988.

[43] L. Guillou and J.-J. Quisquater. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In C.G. Günther, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 1988*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988.

[44] M. Jakobsson and D. Pointcheval. Mutual authentication and key-exchange protocols for low power devices. In *Financial Cryptography*, pages 178–195. Springer, 2001.

[45] A. Joux. A One-Round Protocol for Tripartite Diffie-Hellman. In Springer-Verlag, editor, *ANTS-IV Conference*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394, 2000.

[46] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups, 2001. Cryptology ePrint Archive, Report 2001/003. Available at `http://eprint.iacr.org/2001/003/`.

[47] J. Kilian and E. Petrank. Identity Escrow. In *Advances in Cryptology - CRYPTO '98*, volume 1642 of *Lecture Notes in Computer Science*, pages 169–185. Springer, 1998.

[48] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology - Proceedings of CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[49] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[50] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. In *Mathematische Annalen*, volume 261, pages 515–534. Springer, 1982.

[51] Advanced RISC Machines Ltd. *ARM Software Developpment Toolkit version 2.11: User guide*. Advanced RISC Machines Ltd, 1997. Document number: ARM DUI 0040C.

[52] O. Markowitch and S. Kremer. An Optimistic Non-Repudiation Protocol with Transparent Trusted Third Party. In G.I. Davida and Y. Frankel, editors, *Proceedings of the 4th International Conference on Information Security (ISC 2001)*, volume 2200 of *Lecture Notes in Computer Science*, pages 363–378. Springer, 2001.

[53] O. Markowitch and S. Saeednia. Optimistic Fair Exchange with Transparent Signature Recovery. In P.F. Syverson, editor, *Proceedings of Financial Cryptography 2001*, volume 2339 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 2002.

[54] J.M. Park, E. Chong, H. Siegel, and I. Ray. Constructing Fair Exchange Protocols for E-Commerce via Distributed Computation of RSA Signatures. In *Proceedings of the 22th Annual ACM Symposium on Principles of Distributed Computing (PODC 2003)*, pages 172–181, July 2003.

[55] G. Poupard and J. Stern. Security Analysis of a Practical *on the fly* Authentification and Signature Generation. In K. Nyberg, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436. Springer, 1998.

[56] P. Q. Nguyen and I.E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *Journal of Cryptology*, 15(3):151–176, June 2002.

[57] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. LCS TM82, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1977.

[58] W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In Ç. Koç and C. Paar, editors, *Proc. of Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *LNCS*, pages 109–124. Springer, 2000.

[59] A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.

[60] V. Shoup. OAEP Reconsidered. Available on eprint archive – `http://eprint.iacr.org/2000/060`.

[61] Colin D. Walter. Montgomery Exponentiation Needs no Final Subtractions. *Electronics Letters*, 35(21):1831–1832, October 1999.

[62] Colin D. Walter. Montgomery's Multiplication Technique: How to Make It Smaller and Faster. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93. Springer-Verlag, August 1999.

[63] Colin D. Walter. MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In *Topics in Cryptology - CT-RSA 2002*, Lecture Notes in Computer Science. Springer, April 2002.

[64] Colin D. Walter. Seeing through MIST given a small fraction of an RSA private key. In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003*, volume 2612. Springer, 2003.

APPENDIX A

Publications list

Publications in refereed international conferences:

- Julien Cathalo, François Koeune, Jean-Jacques Quisquater, *A New Type of Timing Attack: Application to GPS*, In C. Walter, C. K. Koc, C. Paar, editors, Fifth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Volume 2779 of Lecture Notes in Computer Science, pages 291-303, Springer, September 2003.
- Julien Cathalo, Benoît Libert, Jean-Jacques Quisquater, *Cryptanalysis of a Verifiably Committed Signature Scheme based on GPS and RSA*, In Kan Zhang, Yuliang Zheng, editors, 7th Information Security Conference, ISC 2004, Volume 3225 of Lecture Notes in Computer Science, pages 52-60, Springer, September 2004.
- Julien Cathalo, Jean-Sébastien Coron, David Naccache, *From Fixed-Length to Arbitrary-Length RSA Encoding Schemes Revisited*, In Serge Vaudenay, editor, 8th International Workshop on Practice and Theory in Public-Key Cryptography (PKC 2005), Volume 3386 of Lecture Notes in Computer Science, pages 234-243, Springer, January 2005.
- Julien Cathalo, Benoît Libert, Jean-Jacques Quisquater, *Efficient and Non-Interactive Timed-Release Encryption*, In S. Qing, W. Mao, J. Lopez and G. Wang, editors, Seventh International Conference on Information and Communications Security (ICICS 2005), Volume 3783 of Lecture Notes in Computer Science, pages 291-303, Springer, December 2005.
- Omkant Pandey, Julien Cathalo, Jean-Jacques Quisquater, *Fair Identification*, In D. Pointcheval, editor, CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, Volume 3860 of Lecture Notes in Computer Science, pages 52-63, Springer, January 2006.