Special signature schemes

Special Signature Schemes

Eugene J.L.J. van Heijst

Special Signature Schemes

door

Eugène Josephus Leonardus Johannes van Heijst

Special Signature Schemes

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J. H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op maandag 6 juli 1992 om 16.00 uur

door

Eugène Josephus Leonardus Johannes van Heijst

geboren te Eindhoven.

Dit proefschrift is goedgekeurd door
de promotoren
prof. dr. ir. H. C. A. van Tilborg
en
prof. dr. J. H. van Lint
en de copromotor
dr. D. Chaum

Contents

1

Introduction

1.1. Introduction

In this section we will give a brief overview of all the (well known) cryptographic tools and methods used in this thesis. It includes public key cryptography, message confidentiality and authentication (digital signature), the cryptosystems based on discrete logarithm and RSA, blobs and zero-knowledge. Also all the assumptions which will be used in this thesis are stated, but if an assumption is used somewhere in this thesis, it is indicated explicitly.

1.2. Public key cryptography

In contemporary computer-controlled communication systems, the conventional cryptosystems have turned out to possess two major disadvantages: the problem of key management and distribution in the case of many users, and the problem of authentication.

In [DH76] a new cryptosystem, called a *public key system*, is introduced (see Figure 1.1) that solves these two problems. Each user U of this cryptosystem creates his own encryption algorithm E_U and decryption algorithm D_U. Each user U makes his encryption algorithm E_U public by putting it in a public key book, called a Trusted Public Directory (TPD) of public keys (It must be "trusted", otherwise some attacks are possible, see [RS84]). The decryption algorithm D_U however is kept secret by U. The pairs E_U, D_U are required to satisfy the properties:

P1. $D_U(E_U(m)) = E_U(D_U(m)) = m$, for all messages m and all users U.
P2. E_U and D_U are algorithms that are easy to perform.

P3. It is *computationally infeasible* for eavesdroppers to find an algorithm D_U^* from E_U that satisfies $D_U^*(E_U(m)) = E_U(D_U^*(m)) = m$ for a non negligible fraction of all m.

The notion "*computational feasibility*" can be defined formally by using Turing machines (see Section 2.2), but here we will use the following intuitive definition. If $a_1,\ldots,a_t$ are binary strings chosen according to some prescribed probability distribution and b is a binary string with $b=f(a_1,\ldots,a_t)$ for some function f, then we say that it is feasible to compute b from $a_1,\ldots,a_t$ if there is an efficient algorithm that outputs b with non-negligible probability when it is given $a_1,\ldots,a_t$ as input. In this thesis we shall freely use the notion of computational feasibility in statements of propositions, corollaries, and so forth.

Property P3 makes it possible to publish the encryption algorithm without endangering the privacy of the transmitted messages. This property is often based on some (unproven) complexity theoretic assumption (see Sections 1.4 and 1.5).

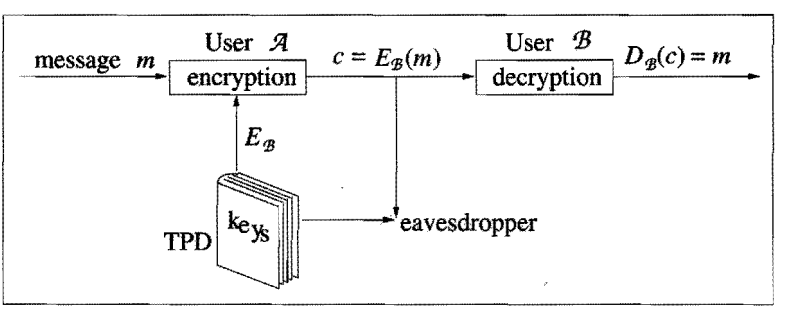


Fig. 1.1. Description of a public key system.

We define the following four kinds of functions, whose description must be publicly known and the function must be easy to perform:

- *one-way function*: given a random image, it is computationally infeasible to find a preimage. (In the literature another definition is sometimes used: for given x and $f(x)$ it is computationally infeasible to find an $x' \neq x$ such that $f(x) = f(x')$.)
- *trapdoor one-way function*: a one-way function, for which it is feasible to compute preimages, given certain additional information.
- *collision-free function*: it is computationally infeasible to find two distinct numbers that map to the same image. (In the literature also called *collision-resistant*. A *collision* is a pair $\{x, x'\}$ satisfying $x \neq x'$ and $f(x) = f(x')$).
- *hash function*: a collision-free function that maps arbitrary-length input to a fixed-length output.

Note that with these definitions, collision-free and one-way functions are not related, according to Table 1.1 (in which p is a large prime and g a generator of $\mathbb{Z}_p^*$; see Section 1.3.1).

function	collision-free	one-way
$f(x,y)=x+y$	no	no
$f(x)=x$	yes	no
$f(x,y)\equiv g^{x+y} \pmod p$	no	yes
$f(x)\equiv g^x \pmod p$	yes, for $x\in\{0,1,\ldots,p-2\}$	yes

Table 1.1. Four functions to illustrate that collision-free and one-way functions are not related.

Message confidentiality

If user $\mathcal{A}$ wants to send a confidential message m to user $\mathcal{B}$, he looks up in the TPD the public encryption algorithm $E_{\mathcal{B}}$ of $\mathcal{B}$, and sends the encrypted message

$$c = E_{\mathcal{B}}(m).$$

User $\mathcal{B}$ can recover the message m from c by applying his secret decryption algorithm $D_{\mathcal{B}}$ on c, because $D_{\mathcal{B}}(E_{\mathcal{B}}(c)) = c$ by property P1 (see Figure 1.1). By property P3, no other user than $\mathcal{B}$ can recover the message: it is computationally infeasible for an eavesdropper knowing $E_{\mathcal{B}}$ and $c = E_{\mathcal{B}}(m)$ to compute m. Thus everyone can encrypt the message and only one person can decrypt the message.

Message authentication (digital signature)

If user $\mathcal{A}$ (called the *signer*) wants to send message m to user $\mathcal{B}$ (called the *recipient*) provided with his digital signature, he sends the message m and his signature

$$s = D_{\mathcal{A}}(m).$$

User $\mathcal{B}$ can verify the signature s by applying the public encryption algorithm $E_{\mathcal{A}}$ to s, because (by property P1) $E_{\mathcal{A}}(s) = E_{\mathcal{A}}(D_{\mathcal{A}}(m)) = m$. By property P3 no other user than $\mathcal{A}$ can have created this digital signature on m. Thus only one person can compute the signature and everybody can verify the signature. This is an example how to construct digital signatures; see the next section for the general case.

Protocols

A *protocol* can be taken to be a set of rules according to which messages are transmitted between parties. Generally the parties apply cryptographic operations (such as computation of digital signatures and encryption) to the messages sent and received, in order to protect their interests. Hence the descriptions above of message confidentiality and message authentication are examples of protocols, which will be often depicted in a figure. See Figure 1.2 for the protocol of a digital signature, in which $\xrightarrow{x}$ denotes the transmission of the value x from one party to another party.

A *move* is a message sent from one party to the other (so the protocol in Figure 1.2

consists of one move, while the coin-flipping protocol of Figure 1.6 consists of 3 moves). An *interactive protocol* is one in which the two parties involved both influence (in some way) the numbers sent during the protocol (see for instance Figures 1.6, 1.7 and 1.8).

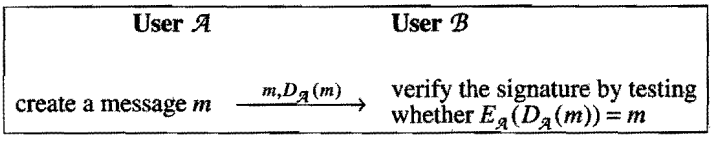


Fig. 1.2. Protocol of a digital signature.

1.3. Signatures

In [DH76] the concept of (*conventional*) *digital signatures* was introduced, and a formal definition of what was thought to be the optimal security for digital signatures was given in [GMR85] and [GMR88]. For these signatures, there is a publicly known test predicate, so that the recipient of the signature can verify efficiently the correctness of the signature. For instance, in the previous section user $\mathcal{B}$ verifies the signature $s = D_{\mathcal{A}}(m)$ of user $\mathcal{A}$ by applying the public encryption algorithm $E_{\mathcal{A}}$ to s. All known conventional signature schemes are based on a generally trusted, but unproven complexity-theoretic assumption (such as property P3 of the previous section). Hence signatures can be forged if this assumption turns out to be false. Hence the signers have only computational security. The recipients of the signatures have unconditional security: if a received signature passes the test predicate, then this signature will always be valid, no matter how much computing power the signer has.

Recently it has been discovered that the definition of [GMR88] was not as general as possible: there are schemes possible with more (or different) security features, such as the following schemes.

Undeniable signatures

In [CvA89] undeniable signatures are introduced: for the verification of these signatures no test function is available, but the recipient has to perform a *confirmation protocol* with the signer. This means that a recipient of such a signature cannot show the correctness of the signature to other persons without the help of the signer. Moreover, a presumed "signer" can disavow a forged signature by using a *disavowal protocol*. The signer has computational and the recipient unconditional security.

Convertible signatures

In [BCDP90] convertible signatures are introduced: these are undeniable signatures that have the feature that they can be changed into (conventional) digital signatures (by releasing some numbers by the signer) (see also Chapter 5).

Signatures unconditionally secure for the signer

A scheme is called *unconditionally secure for the signer* if an attacker with unlimited computing power cannot fake signatures of a signer $\mathcal{S}$, i.e., he cannot create the correct signature of $\mathcal{S}$ on an arbitrary message. An exponential small error probability is tolerated, because the attacker can always guess the secret key of $\mathcal{S}$ correctly. So the signer has unconditional security and the recipient computational security (for constructions see Chapter 5).

Fail-stop signatures

In [WP89] fail-stop signatures are introduced: these are signatures unconditionally secure for the signer having the following properties (unforgeability also relies on a complexity-theoretic assumption):

- If a signature is forged, the presumed signer can prove that the signature is a forgery
- The signer cannot make signatures which he can later prove to be forgeries.

In Remark 5.1 of Subsection 5.4.1 we show that undeniable signatures unconditionally secure for the signer are not fail-stop signatures.

1.4. Discrete Logarithm

1.4.1. Discrete Logarithm modulo a prime number

In [DH76] the following one-way function is proposed. Let p be a large prime such that $p-1$ contains a large prime divisor, and let g be a *generator* of $\mathbb{Z}_p^*$. Thus each element c of $\mathbb{Z}_p^*$ can be written as

$$c \equiv g^s \pmod{p},$$

where s is unique modulo $(p-1)$. Given s, one can compute c in polynomial (in $\log p$, where this logarithm is base-2)-time. But the opposite problem of finding s when c, p, g are given is assumed to be infeasible. More precise, this scheme is based on the following complexity-theoretic assumption:

Assumption 1.1. *It is infeasible to compute the discrete logarithm modulo a large prime number p, provided $p-1$ contains a large prime divisor.*

The last condition in this assumption is needed, otherwise the discrete logarithm can easily be computed by the algorithm of Pohlig and Hellman [PolHel78]. Assumption 1.1 can be strengthened to the Certified Discrete Log Assumption:

Assumption 1.2. *It is infeasible to compute the discrete logarithm modulo a large prime number p, provided p–1 contains a large prime divisor, even if the factorization of p–1 is known.*

Another concept based on the discrete logarithm is the one in which g does not generate $\mathbb{Z}_p^*$, but a subgroup of $\mathbb{Z}_p^*$ of prime order (see Section 5.2).

Diffie and Hellman [DH76] proposed the following key-exchange protocol (called the *Diffie-Hellman key-exchange*). Each user U randomly chooses a *secret key* $s_U \in \{1,\ldots,p-2\}$; the *public key* will be $c_U \equiv g^{s_U} \pmod p$. Suppose users $\mathcal{A}$ and $\mathcal{B}$ want to exchange a secret key over a public channel. They can both compute the secret key $k \equiv g^{s_\mathcal{A} s_\mathcal{B}} \pmod p$ because $(c_\mathcal{A})^{s_\mathcal{B}} \equiv g^{s_\mathcal{A} s_\mathcal{B}} \equiv (c_\mathcal{B})^{s_\mathcal{A}} \pmod p$. With this key exchange protocol, each user must be sure that he uses the correct public key of the other, otherwise an active eavesdropper can decrypt all the messages sent, without being noticed (see [RS84]). Therefore a *Trusted* Public Directory must be used.

There are also signature schemes based on discrete logarithm; see for instance [ElG85].

What information about the bits of the exponent s is revealed by $c \equiv g^s$? Write $p = 2^d q+1$, where q is odd. Using the Pohlig-Hellman algorithm [PolHel78], the d least significant bits of s are easy to compute. Peralta [Per85] proved that for some constant c the bits $s_{d+1},\ldots,s_{d+c\log\log p}$ are *simultaneously hard* (this means that these $c\log\log p$ bits are polynomially indistinguishable from a random $c\log\log p$ - bit string) (see Figure 1.3).

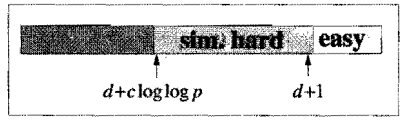


Fig. 1.3. Complexity map of the discrete logarithm modulo a prime (not to scale).

1.4.2. Discrete Logarithm modulo a composite number

The idea of using a composite modulus for the discrete logarithm can be found in [Shm85], [McC88] and [SS90]. Shmuely [Shm85] proved (roughly stated) that any algorithm that will break the *composite Diffie-Hellman key-exchange* (with composite modulus) for a nonnegligible fraction of bases g, can be used to factor the modulus. McCurley [McC88] notes that this result is not as strong as we might like for the security of the scheme, because it is unlikely that the cryptanalyst may specify random base g during the attack. He proved that any algorithm that breaks the scheme using g=16 can be used to factor the modulus.

[SS90] considered the same problem as [Per85] (see Figure 1.4). They proved that if the modulus $N = pq$ is a large *Blum integer* (this means that both primes p and q are congruent to 3 mod 4), g is a *quadratic residue* (so $g \equiv w^2 \pmod N$ for some $w \in \mathbb{Z}_n^*$) and p, q are of equal size (they have the same number of bits), and that if it is infeasible to factor Blum integers, then:

- for arbitrarily small constant ε the bits s_i $(1 \le i \le (1-\varepsilon)\log N)$ are *hard* (so each s_i is polynomially indistinguishable from a random bit);
- the bits $s_1, \ldots, s_{\frac{1}{2}\log N}$ are simultaneously hard.

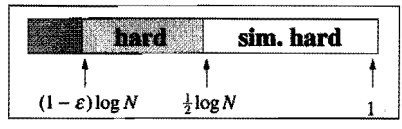


Fig. 1.4. Complexity map of the discrete logarithm modulo a Blum integer (not to scale).

1.5. RSA

Define $\mathbb{Z}_N^*$ to be the set $\{a \mid a \in \mathbb{N}, 1 \le a \le N, \gcd(a,N) = 1\}$, where N is a positive integer. The *Euler's Totient function* $\varphi(N)$ is defined as $\varphi(N) = |\mathbb{Z}_N^*|$, and it has the property that for all numbers $m \in \mathbb{Z}_N^*$ holds that

$$m^{\varphi(N)} \equiv 1 \pmod N.$$

In [RSA78] the following trapdoor one-way function is proposed. Initially, each user $\mathcal{U}$ chooses two large "random" primes $P_{\mathcal{U}}$ and $Q_{\mathcal{U}}$, and computes their product $N_{\mathcal{U}}$, which will be used as modulus. $\mathcal{U}$ further chooses an integer $d_{\mathcal{U}} > 1$ coprime to $\varphi(N_{\mathcal{U}}) = (P_{\mathcal{U}}-1)(Q_{\mathcal{U}}-1)$ and computes an integer $\bar{d}_{\mathcal{U}}$ with $d_{\mathcal{U}}\bar{d}_{\mathcal{U}} \equiv 1 \pmod{\varphi(N_{\mathcal{U}})}$[†]. He makes $N_{\mathcal{U}}$, $d_{\mathcal{U}}$ public in a TPD, but keeps $P_{\mathcal{U}}$, $Q_{\mathcal{U}}$, $\varphi(N_{\mathcal{U}})$ and $\bar{d}_{\mathcal{U}}$ secret (these numbers are the trapdoor information for $\mathcal{U}$). Hence the encryption algorithm of $\mathcal{U}$ is $E(m) \equiv m^{d_{\mathcal{U}}} \pmod{N_{\mathcal{U}}}$ and his decryption algorithm is $D(c) \equiv c^{\bar{d}_{\mathcal{U}}} \pmod{N_{\mathcal{U}}}$.

In stead of writing $c^{\bar{d}_{\mathcal{U}}} \pmod{N_{\mathcal{U}}}$ we write $c^{1/d_{\mathcal{U}}} \pmod{N_{\mathcal{U}}}$, and we will call it the $d_{\mathcal{U}}^{\text{th}}$ *RSA-root* of $c \in \mathbb{Z}_N^*$ (because it is the unique solution $y \in \mathbb{Z}_N^*$ of the congruence $y^{d_{\mathcal{U}}} \equiv c \pmod{N_{\mathcal{U}}}$). Thus if user $\mathcal{A}$ wants to issue his signature on m to user $\mathcal{B}$, his signature will be

$$S \equiv m^{1/d_{\mathcal{A}}} \pmod{N_{\mathcal{A}}},$$

which can be easily verified by $\mathcal{B}$ (see Figure 1.5). In this thesis we usually assume one *signature authority* $\mathcal{Z}$ which issues signatures to *individuals*, and we will also

[†] For efficiency, Carmichael's function $\lambda(N)$ can be used in stead of $\varphi(N)$. This function is defined to be the *smallest* positive number $\lambda(N)$ such that for all m coprime to N, we have that $m^{\lambda(N)} \equiv 1 \pmod N$ (i.e., the highest order of the elements from $\mathbb{Z}_N^*$). Hence $\lambda(N)$ divides $\varphi(N)$.

consider more complicated RSA-signatures, such as $S \equiv m_1^{2/3} \cdot m_2^{1/9} \pmod{N}$.

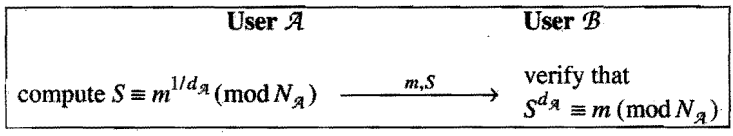


Fig. 1.5. An RSA-based signature-issuing protocol in which user $\mathcal{A}$ issues a signature on message m to user $\mathcal{B}$.

The security of the RSA-scheme is based on the following assumption.

Assumption 1.3. *Without the trapdoor information it is infeasible to compute RSA-roots.*

According to this assumption it is infeasible to split numbers that are the product of several large primes; and it is infeasible to compute discrete logarithms modulo a composite number that is the product of some large primes. The proof of the last claim can be found in [CEvdG87]: suppose that we have an efficient algorithm AL that for each pair $\{g,c\}$ with $g \in \{1,\ldots,N-1\}$, $\gcd(g,N)=1$ and $c \in \langle g \rangle$, computes an integer s with $g^s \equiv c \pmod{N}$. Choose r at random from $\{1,\ldots,N-1\}$, coprime to N, and choose a "probable prime" p between N and $2N$. Then with high probability, p is coprime to $\varphi(N)$. Compute $g := \equiv r^{2p} \pmod{N}$ and $c := \equiv r^2 \pmod{N}$, and because $\gcd(p,\varphi(N)) = 1$ we have that $c \in \langle g \rangle$. If we apply AL on $\{g,c\}$, we obtain an integer s with $g^s \equiv c$. Then $r^{2ps} \equiv r^2 \pmod{N}$, so $r^{2(ps-1)} \equiv 1$, and thus is r^{ps-1} a square root of 1. The number 1 has four square roots modulo N, so with probability 1/2, this root is not equal to 1 or -1, and thus it yields the factorization of N. But according to Assumption 1.3 it is infeasible to factor numbers that are the product of several large primes, so this algorithm AL does not exist.

1.6. Blobs

A *bit commitment scheme* is a scheme that allows $\mathcal{P}$ (prover) to commit himself to some bits (this commitment is called a *blob*) in such a way that it prevents $\mathcal{V}$ (verifier) from learning these bits without the help of $\mathcal{P}$ (so that a blob is a one-way function for $\mathcal{V}$ and a collision-free function for $\mathcal{P}$). Later on, $\mathcal{P}$ can open this blob to reveal the committed bits, and $\mathcal{V}$ is convinced that $\mathcal{P}$ has not changed these bits in the meantime. These blobs can be created by physical means (e.g., envelopes), by quantum mechanics (see [Brass88]), or by cryptography (see [BCC88] for an overview). In the last case, there is a public function $\mathcal{B}$ that is efficiently computable. If $\mathcal{P}$ wants to commit himself

to value b (from a certain set), he chooses randomly a number r (from some other fixed set), computes the blob $B = \mathcal{B}(b,r)$ and issues B to $\mathcal{V}$. If later on, this blob has to be opened, then $\mathcal{P}$ reveals b and r to $\mathcal{V}$, who will verify that this blob B indeed satisfies $B = \mathcal{B}(b,r)$.

According to [BCC88], a blob has the following four properties:

(i) $\mathcal{P}$ can create blobs to commit to any value (from some set).

(ii) $\mathcal{P}$ can open any blob he has committed to in only one way.

(iii) The verifier $\mathcal{V}$ cannot learn anything about the value that $\mathcal{P}$ has himself committed to in an unopened blob.

(iv) Blobs are uncorrelated to any secret that $\mathcal{P}$ wants to keep from $\mathcal{V}$.

An example of the use of blobs is *coin-flipping over the phone* [Blu82]: if two persons $\mathcal{P}$ and $\mathcal{V}$ want to flip a coin over the phone, they have to use blobs, otherwise neither party will be convinced that the other is honest. They can perform the following protocol (see Figure 1.6), that uses two coin-flips in stead of one. First $\mathcal{P}$ flips a coin and let b_1 be the outcome of this coin-flipping (for instance, let b_1 be "0" for head and "1" for tail). $\mathcal{P}$ sends to $\mathcal{V}$ a blob containing this number b_1. Thus $\mathcal{P}$ cannot change the outcome of his coin-flip anymore, and $\mathcal{V}$ cannot learn this number b_1. $\mathcal{V}$ also flips a coin and sends the outcome b_2 to $\mathcal{P}$. So also $\mathcal{V}$ cannot change this number b_2 anymore. Now $\mathcal{P}$ opens his blob, so $\mathcal{V}$ will know the outcome b_1 of the coin-flip by $\mathcal{P}$ (and will verify it). The outcome of this coin-flipping protocol will be $b_1 \oplus b_2$. This protocol behaves like a fair coin, if at least one of the two persons uses a fair coin.

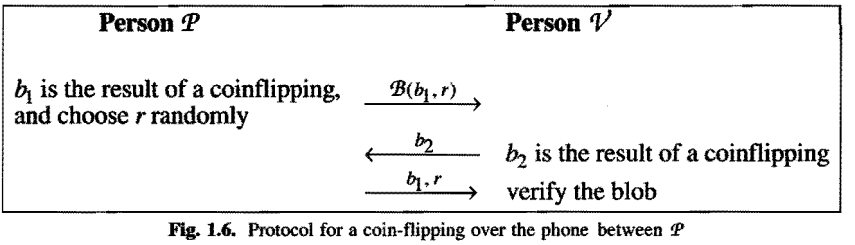


Fig. 1.6. Protocol for a coin-flipping over the phone between $\mathcal{P}$ and $\mathcal{V}$. The outcome of the coin-flip will be $b_1 \oplus b_2$.

Blobs are called *simulatable* if in addition to (i), (ii), and (iii) they satisfy:

(iv') $\mathcal{V}$ can generate blobs for each value with the same probability distribution as $\mathcal{P}$ would.

Blobs are called *chameleon* if in addition to (i), (ii), and (iii) they satisfy:

(iv") $\mathcal{V}$ can generate blobs that he can open in several ways; and for each value the probability distribution is the same as that of $\mathcal{P}$.

Thus chameleon blobs allow $\mathcal{V}$ to do what $\mathcal{P}$ was forbidden to do (property (ii)).

(For the use of chameleon blobs, see Section 1.7.) Blobs are called *unconditionally secure for the prover* if property (iii) holds regardless of $\mathcal{V}$'s computing power, and are called *unconditionally secure for the verifier*† if property (ii) holds regardless of $\mathcal{P}$'s computing power. Several implementations of these different kinds of blobs can be found in [BCC88].

1.7. Zero-Knowledge

Suppose a prover $\mathcal{P}$ wants to convince a verifier $\mathcal{V}$ that $x \in L$, where L is a language. They will use an interactive protocol, in which the two parties are allowed to exchange messages and to toss coins. At the end of this protocol, $\mathcal{V}$ will either accept or reject $\mathcal{P}$'s claim that $x \in L$(of course, no protocol could possibly force $\mathcal{V}$to be convinced).

Two properties are required for this interactive protocol (we denote by $\overline{\mathcal{U}}$ a person who follows the designated protocol and by $\tilde{\mathcal{U}}$ a cheater who can deviate from the protocol in an arbitrary way):

- *Completeness*: if $x \in L$, $\overline{\mathcal{V}}$ accepts $\overline{\mathcal{P}}$'s proof with overwhelming probability.
- *Soundness*: if $x \notin L$, $\overline{\mathcal{V}}$ accepts $\tilde{\mathcal{P}}$'s proof with negligible probability.

Such an interactive protocol is *zero-knowledge* (introduced in [GMR85]) if $\mathcal{V}$ does not learn anything more from the interaction beyond the validity that $x \in L$. More formally

- *Zero-knowledge*: for each $\tilde{\mathcal{V}}$ there exists a probabilistic polynomial-time algorithm (called a *simulator*) that can simulate the communication between $\mathcal{P}$ and $\tilde{\mathcal{V}}$.

We distinguish two types of zero-knowledge. A protocol is called

- (*computationally*) *zero-knowledge* if, to each polynomial-time verifier $\tilde{\mathcal{V}}$, there corresponds a polynomial-time simulator capable of producing $\tilde{\mathcal{V}}$'s view of the protocol that is *polynomially* indistinguishable from the probability distribution, without even talking to the prover.
- *perfectly zero-knowledge* if it produces *exactly* the same probability distribution.

In the literature two types of soundness are distinguished (the names are proposed by Chaum, see [Brass91]):

- *statistically convincing*: $\mathcal{V}$ is convinced of $\mathcal{P}$'s claim, because if that claim had been false, $\mathcal{P}$ would have been caught cheating except with exponentially small probability (e.g., in the protocol of Figure 1.7 this type of soundness is used).

† Note the difference between"unconditionally secure for the verifier" for signatures (see Section 1.3) and for blobs. If these notions are used in this thesis, they are used in the context of signatures or of blobs.

- *computationally convincing*: $\mathcal{V}$ is convinced of $\mathcal{P}$'s claim, provided that he believes that $\mathcal{P}$ did not break a specific instance of the appropriate cryptographic assumption while the protocol was in progress (e.g., in Step 2 of Protocol 4.2 this type of soundness is used).

In this thesis we will be loose: by saying that an interactive protocol is zero-knowledge, we mean that it is complete, sound (i.e., convincing in one of the two ways), and zero-knowledge.

$\mathcal{P}$ and $\mathcal{V}$ may have different kinds of computing power. So we can distinguish the following four cases:

- $\mathcal{P}$ has unlimited computing power and $\mathcal{V}$ has polynomial time computing power (e.g., [GMR85], [GMR89]).
- $\mathcal{P}$ has polynomial time computing power and $\mathcal{V}$ has unlimited computing power (e.g., [BrCr86], [Ch86], and Chapters 5 and 7 of this thesis).
- $\mathcal{P}$ and $\mathcal{V}$ both have polynomial time computing power (e.g., [FFS88] and Chapter 4 of this thesis).
- $\mathcal{P}$ and $\mathcal{V}$ both have unlimited computing power (e.g., quantum blobs [BCC88]).

Consider the protocol for proving the possession of a discrete logarithm of [CEvdG87]: let p be a prime, g a generator of $\mathbb{Z}_p^*$, and $c \in \mathbb{Z}_p^*$, and suppose that $\mathcal{P}$ wants to convince $\mathcal{V}$ that he knows an integer s such that $c \equiv g^s \pmod{p}$. If $\mathcal{P}$ reveals s to $\mathcal{V}$, then $\mathcal{V}$ will be convinced, but this is not in a zero-knowledge way. A zero-knowledge protocol can be found in Figure 1.7, and we prove that this protocol is complete, sound, and perfect zero-knowledge. By $a \in_R S$ we denote the random selection of an element (that will be called a) from the set S according to the uniform probability distribution.

Prover $\mathcal{P}$		**Verifier $\mathcal{V}$**
choose $r \in_R \{0,\ldots,p-2\}$	$\xrightarrow{a \equiv g^r \pmod{p}}$	
	$\xleftarrow{b}$	choose $b \in_R \{0,1\}$
	$\xrightarrow{x \equiv r+sb \pmod{p-1}}$	verify that $g^x \equiv ac^b \pmod{p}$

Fig. 1.7. Perfect zero-knowledge protocol for proving possession of discrete logarithm [CEvdG87].

Complete: This protocol is trivially complete.

Sound: If $\mathcal{P}$ can answer both challenges of $\mathcal{V}$ correctly, then he knows $x_1 \equiv r$ and $x_2 \equiv r+s$. Hence he knows a number s that satisfies $c \equiv g^s$. Thus if $\tilde{\mathcal{P}}$ does not know s, he can answer at most one of the challenges, and thus he will be detected cheating with probability $\geq \frac{1}{2}$. Therefore this protocol will be iterated t times in sequential order

(where t is a number polynomial in $\log p$), so the probability that $\tilde{\mathcal{P}}$ will be detected cheating will be at least $1-2^{-t}$. Hence this protocol is sound.

Perfect zero-knowledge: For each $\tilde{\mathcal{V}}$ the following simulator can be constructed:

repeat t times
  repeat at most $l:=\lfloor \log p \rfloor$ *times*
    choose e at random from $\{0,1\}$,
    choose r at random from $\{0,1,\ldots,p-2\}$,
    compute $a \equiv g^r c^{-e}$,
    receive $b \in \{0,1\}$ *from* $\tilde{\mathcal{V}}$,
    if $b=e$ *then output* $x=r$
  until $b=e$,
  if $b \neq e$ *in all l executions, output "bad luck"*

Note that this simulator has polynomial running time, that the number a is uniformly distributed over $\mathbb{Z}_p^*$, and that a and e are mutually independent. So b is independent of e and thus $b=e$ with probability $\frac{1}{2}$. Hence the probability that this simulator outputs "bad luck" is at most $2/p$. Hence this protocol is perfectly zero-knowledge. But if p is a composite number in stead of a prime, then this protocol is computationally zero-knowledge (see [CEvdG87]).

The protocol of Figure 1.7 is executed t times in sequential order and is zero-knowledge. From a theoretical point of view, a zero-knowledge protocol that is executed in parallel need no longer be zero-knowledge, because $\mathcal{V}$ can choose his challenge $b_1,\ldots,b_t$ to be the outcome of a collision-free one-way function on the numbers $a_1,\ldots,a_t$. By doing this, $\mathcal{V}$ cannot create a transcript himself, and therefore this protocol is not zero-knowledge (each simulator that outputs a transcript with nonnegligible probability has exponential running time). But by using chameleon blobs the protocol of Figure 1.7 can be executed in parallel (see Figure 1.8). Because $\mathcal{V}$ can open these blobs in several ways, it is easy to see that a simulator for this protocol now has polynomial running time.

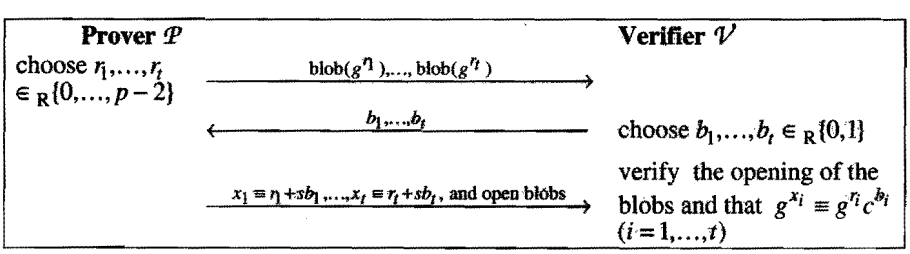


Fig. 1.8. Parallel execution of the protocol of Figure 1.7, by using chameleon blobs. We omit the blinding factors in the blobs.

But if a protocol is not zero-knowledge, that does not mean that the verifier receives "useful information" after the execution. An example of such a protocol is the parallel version of the identification scheme of [FFS88]. They proved that the verifier obtains no useful information for factoring the modulus.

1.8. Summary of the remaining chapters

This thesis consists of six chapters (apart from the introduction), divided into three parts, each dealing with a different aspect of digital signatures.

In Chapter 2 and 3 we give two new theoretical results for the security of large classes of RSA-based signatures.

In Chapter 4 and 5 we present three new kinds of signature schemes and constructions for them: group signatures, undeniable signatures unconditionally secure for the signer, and convertible signatures unconditionally secure for the signer. We also present an efficient construction for fail-stop signatures.

In Chapter 6 and 7 we offer two applications of signature methods: a payment system and a new kind of blob.

The chapters can be read independently of each other, except that

- Chapter 3 is a continuation of Chapter 2, and that
- the soundness of Protocol 4.2 (Section 4.3.1) and an attack on the payment system of Chapter 6 (Cheating 3 of Section 6.10) are studied by using Corollary 2.1.

Below we describe these chapters in more detail.

Chapter 2. Which new RSA signatures can be computed from certain given RSA signatures?

Problem. A signature authority $\mathcal{Z}$ sets an RSA-scheme with modulus N, and issues RSA-signatures $S_1,\ldots,S_s$ of certain types to an individual $\mathcal{A}$. The individual tries, by using these signatures $S_1,\ldots,S_s$, to compute a new RSA-signature S' of a type not issued by $\mathcal{Z}$. The RSA-signatures are products of rational powers of residue classes modulo N, and the residue classes are chosen at random by $\mathcal{Z}$ (e.g., $x_1^{1/3}x_2^{2/5}x_3^{22/15} \pmod N$ for residue classes x_1,x_2,x_3). The rational exponents in the product determine the type of the signature.

Literature. Two related problems are analyzed: computing x^{1/k_1} from $\{x,x^{1/k_2},\ldots,x^{1/k_t}\}$ [Sh83], and computing x^k from $\{x^{k_1},\ldots,x^{k_t}\}$ (with x unknown) [AT83].

Our contribution. We combine and generalize these results from one variable to arbitrarily many variables. Let x_i be residue classes (mod N), uniformly chosen by $\mathcal{Z}$

and let $a_{i,j}, b_j$ be rational numbers. If $\mathcal{A}$ receives the signatures $S_1 \equiv \prod x_j^{a_{1,j}}, \ldots, S_s \equiv \prod x_j^{a_{s,j}}$, our main theorem states that computing $S' \equiv \prod x_j^{b_j}$ from $\{S_1, \ldots, S_s\}$ is polynomial time equivalent to computing a certain RSA-root on random residue classes (mod N).

Applications. We illustrate this result by analyzing two payment systems and one signature scheme, under the assumption that it is infeasible for $\mathcal{A}$ to compute RSA-roots.

Chapter 3. Which new RSA signatures can be computed from RSA signatures, obtained in a specific interactive protocol?

Problem. A signature authority $\mathcal{Z}$ sets an RSA-scheme with modulus N, and issues RSA-signatures $S_1, \ldots, S_s$ of certain types to an individual $\mathcal{A}$, and $\mathcal{A}$ is able to influence the form of these RSA-signatures. The RSA-signatures are products of rational powers of residue classes modulo N; some of these residue classes are chosen by $\mathcal{Z}$ and the others are chosen freely by $\mathcal{A}$. $\mathcal{A}$ wants to choose these residue classes in such a way that he can use these signatures $S_1, \ldots, S_s$ to compute a new RSA-signature S' of a type not issued by $\mathcal{Z}$.

Literature. In Chapter 2 the case was studied in which $\mathcal{A}$ has no influence on the received signature, that is, $\mathcal{A}$ chooses no residue class. In [Dav82], [Denn84] and [DO85] the case was studied in which $\mathcal{Z}$ chooses no residue class, that is, individuals were able to obtain signatures on desired messages.

Our contribution. In this chapter we combine both cases, so some of these residue classes are chosen by $\mathcal{Z}$ and the others are chosen freely by $\mathcal{A}$. In this chapter we make the following two assumptions:

- $\mathcal{A}$ cannot compute RSA-roots of randomly chosen residue classes.
- In his computations, the only operations modulo N that $\mathcal{A}$ uses are multiplications and divisions.

We formulate a necessary and sufficient condition under which $\mathcal{A}$ is able to influence the signatures he receives from $\mathcal{Z}$ in such a way that he can later use these signatures to compute a signature of a type not issued by $\mathcal{Z}$. It turns out that this condition is equivalent to the solvability of a particular quadratic equation in integral matrices.

Applications. In all payment systems, the user chooses blinding factors and can thus influence the signatures he will receive from $\mathcal{Z}$. So in these cases our results can be applied.

Chapter 4. Group Signatures

Problem. A group signature scheme for a group of persons has the following three properties:

- only group members can sign messages;
- the recipient of the signature can verify that it is a valid group signature, but

cannot discover which group member created it;
- if necessary, the signature can be "opened", so that the person who signed the message is revealed.

So a group signature scheme is a signature scheme with one public key and several secret keys; and if several owners of secret keys conspire, they cannot create new secret keys.

Literature. If a person wants to prove that he belongs to a certain group, then there are several protocols known that can be used (see for instance [SKI90], [OOK90] and [CE86]). These schemes cannot be used to construct group signatures, because of several reasons.

Our contribution. We present four different constructions for such schemes and use Corollary 2.1 to prove the security of one of the schemes. These constructions differ, for instance, in complexity theoretic assumption, the need for a trusted authority, the number of computations and the number of bits to be transmitted.

Applications. These schemes can, for instance, be used if the signer does not want to reveal his identity to the recipient, but if later on the recipient wants to know the identity of the signer, he is able to.

Chapter 5. Signatures unconditionally secure for the signer

Problem. A signature scheme is called unconditional secure for the signer if an attacker with unlimited computing power cannot fake signatures of a signer $\mathcal{S}$, i.e., he cannot create the correct signature of $\mathcal{S}$ on an arbitrary message. An exponential small error probability is tolerated, because the attacker can always guess the secret key of $\mathcal{S}$ correctly.

Literature. Undeniable signatures are introduced in [CvA89], fail-stop signatures in [WP89], and convertible signatures in [BCDP90], all as defined in Section 1.3.

Our contribution. We present the first construction of an undeniable signature unconditionally secure for the signer; the first construction of a convertible signature unconditionally secure for the signer; and an efficient construction of fail-stop signatures.

Applications. These kind of signatures have the advantage over ordinary digital signatures that, for instance in an electronic payment system, the customers need not worry about the trusted authority (which normally has more computing power than the customer) being able to break the underlying assumptions of the signature scheme.

Chapter 6. Efficient offline electronic checks

Problem. In an electronic payment system the users have to buy *information* from the bank (such as specially created numbers) instead of special objects (such as coins); during the payment, the construction of these numbers is revealed to the shop. The shop can verify the construction of the numbers and if the constructions are correct, the shop will accept these numbers as valid money (so the number itself does not represent

money, but merely the knowledge of the construction of these numbers). The privacy of the user can be protected unconditionally, which means that even with unlimited computing power, the bank cannot determine where the users have spent their money. Electronic information can be easily duplicated, so the user can give a copy of the information to a second shop. This shop will accept this information as valid money, since the information is still correct as it was before. Hence any user can spend the same money at several different shops without being caught, because his privacy is protected unconditionally. This *double spending* can be prevented if the shop uses an *online* connection with the bank: immediately after receiving the information, the shop contacts the bank in order to verify that the information has not been used before. A system including such online connections is very expensive, so it would be better to have an *offline* connection: the shop contacts the bank, say, once a week. For the offline case to preserve unconditional privacy, the user must have unconditional privacy if the money is spent once; but his identity must be revealed if the money is spent twice.
Literature. The first offline electronic payment system can be found in [CFN88].
Our contribution. We improve the efficiency of the offline electronic payment system of [CFN88], by reducing the number of computations (done by bank, user, and shop) and the number of bits transmitted during the creation and spending phases of the money. Also, some new functionalities are added, and we examine several ways of cheating (one by using Corollary 2.1).

Chapter 7. Chameleon blobs, unconditionally secure for the verifier

Problem. A blob allows a person to commit to a certain bit (or bits) in such a way that he cannot change this bit afterwards, and that the recipient of the blob cannot learn this bit. Usually there is a public function $\mathcal{B}$, and if a person wants to commit himself to bit b, he chooses r randomly and computes the blob $B = \mathcal{B}(b,r)$. Opening this blob consists of revealing b and r.
Literature. See for instance [BCC88] for an overview of blob constructions. In the literature it is proven that it is not possible to have chameleon blobs that are unconditionally secure for the verifier.
Our contribution. We present the first blob in which the person reveals only b and convinces the recipient in a zero-knowledge way that he knows r such that $B = \mathcal{B}(b,r)$. We present several constructions for these blobs; most of these constructions are based on the discrete logarithm.
Applications. These new blobs can for instance be used to create chameleon blobs that are unconditionally secure for the verifier (for our new blobs the definition of chameleon can be modified a little).

2

Which new RSA signatures can be computed from certain given RSA signatures?‡

2.1. Introduction

Several more complicated protocols use as a building block simple protocols in which only one party, called the *signature authority*, can create signatures and issue them to the other parties, called the *individuals*. Such protocols are used, for instance, in credential systems and payment systems, in which a signature represents a credential or money. In fact, in such credential or payment systems, the signature authority issues different types of signatures, corresponding to different credentials or different values of money. A very challenging question in cryptology is what the security of certain protocols will be. The security of these systems depends on whether an individual (or a group of conspiring individuals) is able or unable to compute a signature of a type not issued by the signature authority, by using signatures that were issued by the authority.

In this chapter we consider a generalization of the RSA-scheme of Section 1.5: signature authority $\mathcal{Z}$ chooses at random several residue classes (mod N), computes a number of RSA-signatures that are products of rational powers of these residue classes modulo N, and issues these signatures to individual $\mathcal{A}$, together with the residue classes. The exponents in the product determine the type of signature. It will appear to be useful also to consider the variation in which $\mathcal{Z}$ sends only the signatures but not the

‡ This chapter is based on the papers "Which new RSA signatures can be computed from some given RSA signatures?" by Jan-Hendrik Evertse and Eugène van Heyst, which appeared in *Advances in Cryptology-EUROCRYPT '90*, I.B. Damgård ed., Lecture Notes in Computer Science 473, Springer-Verlag, pp. 83-97; and "Which new RSA signatures can be computed from certain given RSA signatures?" by Jan-Hendrik Evertse and Eugène van Heyst, *J. Cryptology* 5 (1992), pp. 41-52.

residue classes to $\mathcal{A}$ (so that $\mathcal{A}$ cannot verify the signatures). $\mathcal{A}$ may also have received the signatures without the residue classes by eavesdropping. It is conceivable that an individual learns several RSA-signatures issued by $\mathcal{Z}$ (by participating in a signature-issuing protocol or by eavesdropping) and that he uses these to compute useful signatures not issued by $\mathcal{Z}$.

Here is an example of the kind of problems we are facing. Suppose that an individual $\mathcal{A}$ received two randomly chosen residue classes $x_1, x_2 \pmod N$ and a signature $S \equiv x_1^{2/3} \cdot x_2^{1/9} \pmod N$, and that he wants to compute $S' \equiv x_2^{1/9} \pmod N$. $\mathcal{A}$ can easily compute $x_2^{1/3} \pmod N$, since $x_2^{1/3} \equiv x_1^{-2} S^3 \pmod N$. But then $\mathcal{A}$ still has to compute some cube RSA-root. From Theorem 2.1, stated in Section 2.4, it follows that computing $x_2^{1/9}$ from $\{x_1, x_2, S\}$ is just as difficult as computing $x^{1/3} \pmod N$ for each residue class $x \pmod N$. Thus if $\mathcal{A}$ cannot compute RSA-roots, he cannot compute $x_2^{1/9}$ from $\{x_1, x_2, S\}$.

Akl and Taylor [AT83] and Shamir [Sh83] considered related problems. Shamir showed, roughly speaking, that for pairwise coprime integers $k_1, \ldots, k_t$, computing x^{1/k_1} from $\{x, x^{1/k_2}, \ldots, x^{1/k_t}\}$ is just as difficult as computing u^{1/k_1} from u alone, for random u (see Example 2.5 of Section 2.7). Akl and Taylor proved that if $k, k_1, \ldots, k_t$ are integers with $\gcd(k_1, \ldots, k_t) / \gcd(k, k_1, \ldots, k_t) = r$, then computing x^k from $\{x^{k_1}, \ldots, x^{k_t}\}$ (with x unknown) is at least as difficult as computing $u^{1/r}$ from u for random u (see Example 2.3 of Section 2.7). We generalize these results to arbitrarily many variables as in the example above. Our main result is stated in Section 2.4, independently of the context of the protocols mentioned above. Let $S_1 \equiv \prod x_j^{a_{1,j}}, \ldots, S_s \equiv \prod x_j^{a_{s,j}}$, $S' \equiv \prod x_j^{b_j} \pmod N$, where the x_i are uniformly chosen residue classes $\pmod N$ and the $a_{i,j}$, b_j are rational numbers (we are not precise here). Then computing S' from $S_1, \ldots, S_s$ is polynomial-time reducible to computing a certain RSA-root on random residue classes $\pmod N$ and vice versa. Note that all the signatures are modulo the same modulus N. For different moduli see for instance [Has85].

This chapter is organized as follows. In the next section we give a brief overview of Turing machines, that can be used to formalize notions like "computational feasibility". In Section 2.3 we summarize the notations used in this chapter, while our two main theorems are stated in Section 2.4. In Section 2.5 we give two algorithms that simulate the choice of a random element from some set by using coin tosses only. The proofs of our main theorems can be found in Section 2.6, while an important corollary and some practical applications (in particular, two payment systems) of the main theorems are given in Section 2.7. In Section 2.8 we give several remarks on the corollary of the

previous section, and we end with some open problems.

2.2. Deterministic and probabilistic algorithms (Turing machines)

For the convenience of the reader, we recall some notions related to Turing machines. A *Turing machine* (TM) consists of a finite control unit and a two-way infinite tape with a read/write unit. The machine has a finite number of states, among which are two special states called the *initial state* and the *halting state.* The machine uses a finite alphabet S, containing a so-called *blank* symbol.

The tape is divided into cells, and each cell contains a symbol. At each stage, only a finite number of cells on the tape contains nonblanks. The content of the tape in the initial state is called the *input*. The unit can read/write the cell underneath it, and afterwards the unit can remain there or move one cell to the left or right. The machine also has a set of instructions (write symbol, move unit, and go to next stage), which prescribes what the machine should do if it is in some state and has read some symbol. Hence, after an instruction, the content of the tape may be changed. If there is at most one instruction for each state and alphabet symbol, the machine is called a *deterministic Turing machine* (DTM). The machine stops if it arrives at the halting state. The non-blank content of the tape in this state is called the *output*. The *running time* is then defined as the number of steps the unit has made before stopping. A *polynomial-time* TM is a TM for which there is a polynomial with nonnegative coefficients, such that for every input, the running time of that TM is bounded above by the value of that polynomial evaluated in the length of the input (i.e., the number of nonblank symbols).

A *probabilistic Turing machine* (PTM) is a TM that has at most two instructions for each state and alphabet symbol. It is decided by an unbiased coin toss which one of the instructions is followed (cf. [Gill77]). Thus the output of a PTM is no longer uniquely determined by the input, but can be considered as a stochastic variable, whose probability distribution depends on the input. In general, the running time of a PTM can also be considered as a stochastic variable.

We can extend the notion of a PTM by allowing it to issue *oracle requests*. Let S^* be the collection of two-way infinite strings of symbols from S, at most finitely many of which are nonblanks. An *oracle* O is a collection of random variables $\{O(i) \mid i \in I\}$ on S^*, where I is a subset of S^*, such that the number of nonblank symbols of $O(i)$ is bounded polynomially in terms of the number of nonblank symbols of i, for $i \in I$. An *O-using PTM* is a device that has, apart from its usual read-write and random tape, an O-request tape. Whenever the O-using PTM needs to do a request to O, it goes into an O-request state; it writes some $i \in I$ on the O-request tape; then

the oracle O writes some value of $O(i)$ on the O-request tape, and this value is read by the PTM; after this, the PTM continues its ordinary computations. We shall consider the action of O in which it outputs a value of $O(i)$ as one step in the execution of the O-using PTM. Thus, a polynomial-time O-using PTM does, apart from polynomially many ordinary PTM-operations, at most polynomially many O-requests.

The above notions of Turing machines can be used to define formally the notion of "computational feasibility", the complexity classes NP and NP-complete, and lower bounds on the running time of certain problems. But in this chapter we will be loose with Turing machines and we will use the words *deterministic* and *probabilistic algorithms* to indicate DTM and PTM. In the algorithms that we consider, the inputs are tuples of integers and rationals, and the length of such an input is the sum of the lengths of the binary representations of the integers and of the numerators and denominators of the rational numbers in the input. In general, both the output and the running time of a probabilistic algorithm are stochastic variables, depending on the input and the random coin tosses. However, here we consider only probabilistic algorithms whose running time is determined by the input alone. Thus, if a probabilistic algorithm is used to solve a particular problem, then it does not have to output a solution with absolute certainty but only with some probability of success. More generally, the underlying probability space consists of the strings of bits chosen during the execution of the algorithm, with uniform distribution, and of a set J of possible inputs, from which input I is chosen with probability p_I. Thus, if some algorithm solves a problem with conditional probability of success s_I given input I, then its probability of success is $\sum_{I \in J} p_I s_I$. By a *polynomial-time algorithm* we mean a deterministic algorithm whose running time depends polynomially on the length of the input. For instance, there are polynomial-time algorithms for doing the following (where a, b, c are integers):

- computing $\gcd(a,b)$ from a and b (Euclid's algorithm),
- computing the inverse of a (mod b) from a and b, if $\gcd(a,b) = 1$ (Euclid's algorithm),
- computing a^{-b} (mod c) from a, b and c, if $\gcd(a,c) = 1$,
- testing rational vectors for linear independence,
- solving a linear diophantine equation with rational coefficients,
- deciding if a system of rational linear equations has an integral solution, and if so, finding one,
- computing the Smith normal form of an integral matrix ([KaBa79]).

Also $\gcd(a,b)$, where $a,b \in \mathbb{Q}$ can be computed in polynomial time: let d be a positive integer such that da and db are integral, and define $\gcd(a,b) := \frac{\gcd(ad,bd)}{d}$. Note that this definition is independent of the choice of d. For instance, $\gcd(\frac{2}{3},\frac{5}{6}) = \frac{1}{6}$ and $\gcd(\frac{2}{3},1) = \frac{1}{3}$.

2.3. Notation

The following notation will be used in this chapter (some of notation was already introduced in Section 1.5). Boldface characters are used to denote vectors, and the RSA-modulus N used is created by the signature authority.

$\gcd(a_1,\ldots,a_t)$	the greatest common divisor of $a_1,\ldots,a_t \in \mathbb{Q}$ (see Section 2.2).
$\log x$	base-2 logarithm of x.
$\ln x$	natural logarithm of x.
$\mathbf{a} \equiv \mathbf{b} \pmod N$	$N^{-1}(\mathbf{b}-\mathbf{a}) \in \mathbb{Z}^k$; this is defined for $\mathbf{a},\mathbf{b} \in \mathbb{Q}^k$, $k \in \mathbb{N}$.
$\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$	$\{\sum_{i=1}^{s} \xi_i \mathbf{a}_i \mid \xi_1,\ldots,\xi_s \in \mathbb{Z}\}$: the abelian group generated by $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$.
$\mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$	$\{\sum_{i=1}^{s} \xi_i \mathbf{a}_i \mid \xi_1,\ldots,\xi_s \in \mathbb{Q}\}$: the vector space generated by $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$.
$\langle \mathbf{a},\mathbf{b} \rangle$	$a_1b_1+\ldots+a_kb_k$: the scalar product of $\mathbf{a} = (a_1,\ldots,a_k)$ and $\mathbf{b} = (b_1,\ldots,b_k)$.
$\mathbb{Z}_N^*$	the set $\{a \mid a \in \mathbb{N}, 1 \le a \le N, \gcd(a,N) = 1\}$ of $\varphi(N)$ elements.
$\tilde{\mathbb{Q}}_N$	the ring $\{\frac{a}{d} \mid a,d \in \mathbb{Z}, d > 0, \gcd(d,\varphi(N)) = 1\}$.
$x^{1/d} \pmod N$	the d^{th} *RSA-root* of $x \pmod N$: the unique solution $S \in \mathbb{Z}_N^*$ to $S^d \equiv x \pmod N$ for $x \in \mathbb{Z}_N^*$ and $d \in \mathbb{Z}$ with $\gcd(d,\varphi(N)) = 1$.
$\mathbf{x}^{\mathbf{a}} \pmod N$	the number $S \in \mathbb{Z}_N^*$ with $S \equiv x_1^{a_1}x_2^{a_2}\ldots x_k^{a_k} \pmod N$, for $\mathbf{x} = (x_1,\ldots,x_k) \in (\mathbb{Z}_N^*)^k$ and $\mathbf{a} = (a_1,\ldots,a_k) \in (\tilde{\mathbb{Q}}_N)^k$.
$l(n)$	length of the binary representation of $n \in \mathbb{N}$; the length of a negative integer m, a rational number p/q ($q \neq 1$) and a vector $\mathbf{c}$ are defined by: $l(m) = l(-m)+1$, $l(p/q) = l(p)+l(q)+1$ and $l(\mathbf{c}) = \sum_i (l(c_i)+1)$, respectively.
length(a,b)	$l(a) + l(b)$.

2.4. Statement of the theorems

Let $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b} \in (\tilde{\mathbb{Q}}_N)^k$. We consider the problem of computing $\mathbf{x}^{\mathbf{b}}$ for random $\mathbf{x} \in (\mathbb{Z}_N^*)^k$, if $\{\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$ (but not necessarily $\mathbf{x}$) is given as input. We distinguish whether or not $\mathbf{b}$ is an element of $\mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$. To each of these two cases a theorem is devoted, of which the proof can be found in Section 2.6.

In Theorem 2.1 below, $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}$ are vectors in $(\tilde{\mathbb{Q}}_N)^k$, satisfying

$$\left.\begin{array}{l}\mathbf{b}\in \mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\};\ \ \text{length}(N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}) = L;\\ \gcd(d,\varphi(N)) = 1,\ \text{where } d = \min\{\, x\in\mathbb{N} \mid x\mathbf{b}\in\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}\,\}.\end{array}\right\} \tag{2.1}$$

Theorem 2.1. *Let* $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}\in(\tilde{\mathbb{Q}}_N)^k$ *satisfy* (2.1).

(i) *For every probabilistic algorithm* AL *that on input* $\{N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b},\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$ *computes* $\mathbf{x}^{\mathbf{b}}$ *in time* $\le T_{\text{AL}}$ *with probability of success* $\ge \varepsilon_{\text{AL}}$ *for* random $\mathbf{x}\in(\mathbb{Z}_N^*)^k$, *there exists a probabilistic algorithm* $\overline{\text{AL}}$ *that for* arbitrary $u\in\mathbb{Z}_N^*$ *computes* $u^{1/d}$ *in time* $\le T_{\text{AL}} + L^{O(1)}$ *with probability of success* $\ge \frac{1}{2}\varepsilon_{\text{AL}}$.

(ii) *For every probabilistic algorithm* AL *that on input* $\{N,u\}$ *computes* $u^{1/d}$ *in time* $\le T_{\text{AL}}$ *with probability of success* $\ge \varepsilon_{\text{AL}}$ *for* random $u\in\mathbb{Z}_N^*$, *there exists a probabilistic algorithm* $\overline{\text{AL}}$ *that for* arbitrary $\mathbf{x}\in(\mathbb{Z}_N^*)^k$ *computes* $\mathbf{x}^{\mathbf{b}}$ *from* $\{N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b},\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$ *in time* $\le T_{\text{AL}} + L^{O(1)}$ *with probability of success* $\ge \frac{1}{2}\varepsilon_{\text{AL}}$.

Remark 2.1. Theorem 2.1 can be generalized to the case that $\gcd(d,\varphi(N)) > 1$. Let G be the largest subgroup of $\mathbb{Z}_N^*$ whose order is coprime to d. Then for every $u\in G$, there is a unique $x\in G$ with $x^d \equiv u \pmod N$ and we denote this x by $u^{1/d}$. We can prove that for every probabilistic algorithm AL as in Theorem 2.1 there is a probabilistic algorithm $\overline{\text{AL}}$ that for arbitrary $u\in G$ computes $u^{1/d}$ in time $\le T_{\text{AL}} + L^{O(1)}$ with probability of success $\ge \frac{1}{2}\varepsilon_{\text{AL}}$. As the proof of this generalization is the same as that of Theorem 2.1, we shall omit it here.

Remark 2.2. Theorem 2.1 deals with the situation that $\mathbf{x}^{\mathbf{b}}$ has to be computed from $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ while $\mathbf{x}$ itself is not known. We can treat the case that $\mathbf{x}$ is also known. Thus $\mathbf{x}^{\mathbf{b}}$ has to be computed from given $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ *and* $\mathbf{x}$. We can apply Theorem 2.1 with $\mathbf{a}_1,\ldots,\mathbf{a}_s,\ \mathbf{e}_1 = (1,0,\ldots,0),\ \mathbf{e}_2 = (0,1,\ldots,0),\ldots,\ \mathbf{e}_k = (0,\ldots,0,1)$, instead of $\mathbf{a}_1,\ldots,\mathbf{a}_s$. Note that $\mathbf{b}\in\mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ for all $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}\in(\tilde{\mathbb{Q}}_N)^k$. Further, if d is the smallest positive integer x with $x\mathbf{b}\in\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{e}_1,\ldots,\mathbf{e}_k\}$, then d is the gcd of all these integers x. Hence if $\mathbf{b}\in(\tilde{\mathbb{Q}}_N)^k$, then $\gcd(d,\varphi(N)) = 1$.

We can also treat the case in which $\mathbf{x}_1^{\mathbf{b}_1}\ldots\mathbf{x}_s^{\mathbf{b}_s}$ (which is a product of numbers) has to be computed from $\{\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$ for certain $\mathbf{b}_1,\ldots,\mathbf{b}_s\in(\tilde{\mathbb{Q}}_N)^k$, where $\mathbf{x}_1,\ldots,\mathbf{x}_s$ are distinct vectors from $(\mathbb{Z}_N^*)^k$: namely, put $\mathbf{x}' := (\mathbf{x}_1,\ldots,\mathbf{x}_s)$, $\mathbf{a}_1' := (\mathbf{a}_1,\mathbf{0},\ldots,\mathbf{0})$, $\mathbf{a}_2' := (\mathbf{0},\mathbf{a}_2,\mathbf{0},\ldots,\mathbf{0})$,..., $\mathbf{a}_s' := (\mathbf{0},\ldots,\mathbf{0},\mathbf{a}_s)$, $\mathbf{b}' := (\mathbf{b}_1,\ldots,\mathbf{b}_s)$, and apply Theorem 2.1 with $\mathbf{x}',\mathbf{a}_1',\ldots,\mathbf{a}_s',\mathbf{b}'$.

In Theorem 2.2 below, $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}$ are vectors in $(\tilde{\mathbb{Q}}_N)^k$, satisfying

$$\left.\begin{array}{l}\mathbf{b}\notin \mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\};\ \ \text{length}(N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b})=L;\\ d=\min\{x\in\mathbb{N}\mid \exists \xi_1,\ldots,\xi_s\in\mathbb{Z}\colon x\mathbf{b}\equiv\sum_{i=1}^s\xi_i\mathbf{a}_i\ (\mathrm{mod}\,\lambda(N))\}.\end{array}\right\} \tag{2.2}$$

Note that d is the gcd of all the integers x as in (2.2). Hence d divides $\lambda(N)$. We have:

Theorem 2.2. *Let* $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}\in(\tilde{\mathbb{Q}}_N)^k$ *satisfy* (2.2).

(i) *There exists a polynomial (in* L*)-time algorithm that computes a nonzero multiple of* $\lambda(N)/d$ *from* $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}$.

(ii) *For every* $\mathbf{x}\in(\mathbb{Z}_N^*)^k$, *the cardinality of the set* $\{z\in\mathbb{Z}_N^* \mid \exists\,\mathbf{y}\in(\mathbb{Z}_N^*)^k\colon \mathbf{y}^{\mathbf{b}}\equiv z\ (\mathrm{mod}\,N),\ \mathbf{y}^{\mathbf{a}_i}\equiv\mathbf{x}^{\mathbf{a}_i}\ (\mathrm{mod}\,N)$ for $i=1,\ldots,s\}$ *is equal to the number of solutions* $z\in\mathbb{Z}_N^*$ *of* $z^d\equiv 1\ (\mathrm{mod}\,N)$.

For instance, if $d=1$, then from $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}$ we can compute in polynomial (in L) time a multiple of $\lambda(N)$ and from that we can compute in probabilistic polynomial (in length(N)) time the factorization of N [Mil75]. The other extreme situation is that $d=\lambda(N)$ and $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ are given but $\mathbf{x}$ is unknown. In this case $\mathbf{x}^{\mathbf{b}}$ is not uniquely determined by $\{\mathbf{x},\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$; in fact, every number in $\mathbb{Z}_N^*$ is possible for $\mathbf{x}^{\mathbf{b}}$.

2.5. Taking random elements

In general (so $\log N\notin\mathbb{Z}$), there is no known polynomial-time algorithm to choose an element uniformly from $\{0,\ldots,N-1\}$ in which the only possible non-deterministic operations are coin tosses. But we can simulate it for instance in one of the two following ways.

Algorithm 2.1. Let $0<\varepsilon<1$.

Step 1. *Compute the integer* $K:=\lceil\log\frac{N}{\varepsilon}\rceil$.

Step 2. *Choose* c *at random from* $\{0,\ldots,2^K-1\}$ *by doing* K *coin tosses.*

Step 3. *Compute (the unique)* $r\in\{0,\ldots,N-1\}$, *such that* $r\equiv c\ (\mathrm{mod}\,N)$.

Lemma 2.1. *Algorithm* 2.1 *has running time that is polynomial in* $\log\frac{N}{\varepsilon}$, *and the probability distribution is such that* $\frac{1}{N}(1-\varepsilon)\le\Pr(r)\le\frac{1}{N}(1+\varepsilon)$, for $r\in\{0,\ldots,N-1\}$.

Proof. It is easy to see that the running time of this algorithm is polynomial in $\log\frac{N}{\varepsilon}$. For each $r\in\{0,\ldots,N-1\}$, the number of $c\in\{0,\ldots,2^K-1\}$ such that $r\equiv c\ (\mathrm{mod}\,N)$ is

either $\lfloor \frac{2^K}{N} \rfloor$ or $\lfloor \frac{2^K}{N} \rfloor + 1$. Hence:

$$\Pr(r) = \frac{1}{2^K}\lfloor \frac{2^K}{N} \rfloor \quad \text{or} \quad \Pr(r) = \frac{1}{2^K}(\lfloor \frac{2^K}{N} \rfloor + 1).$$

$$\text{So} \begin{cases} \Pr(r) \geq \frac{1}{2^K}(\frac{2^K}{N} - 1) = \frac{1}{N} - \frac{1}{2^K} \geq \frac{1-\varepsilon}{N}, \\ \Pr(r) \leq \frac{1}{2^K}(\frac{2^K}{N} + 1) = \frac{1}{N} + \frac{1}{2^K} \leq \frac{1+\varepsilon}{N}. \end{cases}$$ ❑

Algorithm 2.2.

Step 1. *Compute the integer* $K := \lceil \log N \rceil$ (*so* $N \leq 2^K < 2N$).

Step 2. *Choose r at random from* $\{0,\ldots,2^K-1\}$ *by doing K coin tosses.*

Step 3. *Check if* $r \in \{0,\ldots,N-1\}$.

Lemma 2.2. *Algorithm* 2.2 *has running time polynomial in* $\log N$, *the probability of success is* $\geq \frac{1}{2}$, *and the conditional probability distribution given success is uniform.*

Proof. The probability of success is $\frac{N}{2^K} > \frac{N}{2N} = \frac{1}{2}$; it is easy to see that the running time is polynomial in $\log N$, and that the conditional probability distribution given success is uniform. ❑

Algorithms 2.1 and 2.2 can be modified to take random elements uniformly from $\mathbb{Z}_N^*$ (in stead of $\{0,\ldots,N-1\}$) with probability of success $\geq \frac{1}{2}$ as follows. Check in the last step whether $r \in \{0,\ldots,N-1\}$ and $\gcd(r,N) = 1$. If so, take the residue class r (mod N). Thus, we get an element of $\mathbb{Z}_N^*$ with probability of success $\varphi(N)/2^K \geq 1/(12 \ln\ln N)$, in view of the inequality $\varphi(N) \geq N/(6 \ln\ln N)$ for $N>4$ [RS62]. So after $12\ln\ln N$ repetitions of Algorithm 2.2 we have with probability at least $\frac{1}{2}$ an element from $\mathbb{Z}_N^*$.

There is a constant $c>0$ with $1 - \left(1 - \frac{1}{12 \ln\ln N}\right)^{c \ln k \cdot \ln\ln N} \geq \left(\frac{1}{2}\right)^{1/k}$. Hence, after at most

$$K \cdot k \cdot c \cdot \ln k \cdot \ln\ln N \quad \text{coin tosses}$$

we find a vector $\mathbf{r} \in (\mathbb{Z}_N^*)^k$ with probability of success $\geq \frac{1}{2}$. Moreover, the conditional probability distribution of $\mathbf{r}$, given success, is uniform on $(\mathbb{Z}_N^*)^k$. In the rest of this thesis we will use Algorithm 2.2.

2.6. Proofs of the theorems

We need some lemmas to prove Theorems 2.1 and 2.2.

Lemma 2.3. *There exists a polynomial-time algorithm that computes for every* $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$ *a basis* $\{\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ *of* $\mathbb{Z}^k$ *and positive elements* $d_1,\ldots,d_t \in \mathbb{Q}$ *such that* $\{d_1\mathbf{e}_1,\ldots,d_t\mathbf{e}_t\}$ *is a basis of* $\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$.

Proof. For $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Z}^k$ this follows from the result of Kannan and Bachem [KaBa79] that we can compute in polynomial-time the Smith normal form of an integral matrix. For $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$, one may first compute $d \in \mathbb{N}$ such that $d\mathbf{a}_1,\ldots,d\mathbf{a}_s \in \mathbb{Z}^k$ and then apply the result of Kannan and Bachem. ❑

Lemma 2.4. *For* $a,b \in \mathbb{Z}$, $a,b \neq 0$, *let* $(a\backslash b)$ *denote the largest positive divisor of* $a \in \mathbb{Z}$ *which is not divisible by any prime number dividing* b. *There exists a polynomial-time algorithm that computes* $(a\backslash b)$ *from* $a,b \in \mathbb{Z}$, $a,b \neq 0$.

Proof. Consider the sequence of integers $c_0 = |a|$, $c_1 = c_0 / \gcd(c_0,b)$, $c_2 = c_1 / \gcd(c_1,b),\ldots$. There is an i such that $\gcd(c_i,b) = 1$; let i_0 be the smallest such i. Since $c_1 \leq c_0/2$, $c_2 \leq c_1/2,\ldots$, $c_{i_0} \leq c_{i_0-1}/2$, $c_{i_0+1} = c_{i_0}$, we have $i_0 \leq l(a)$. Hence it takes polynomial time to compute c_{i_0}. For each prime number p and each $a \in \mathbb{Z}$, $a \neq 0$, let $\mathrm{ord}_p(a)$ be the integer such that $a \cdot p^{-\mathrm{ord}_p(a)}$ is an integer not divisible by p. Obviously, $\mathrm{ord}_p(c_{i_0}) = 0$ for each prime p dividing b. Further, if p is a prime dividing a but not b, then $\mathrm{ord}_p(a) = \mathrm{ord}_p(c_0) = \mathrm{ord}_p(c_1) = \ldots = \mathrm{ord}_p(c_{i_0})$. It follows that $c_{i_0} = (a\backslash b)$. ❑

Lemma 2.5. *Let* $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$, $\mathbf{b} \in \mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$ *and let* d *be the smallest positive integer such that* $d\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$. *Then there exists a vector* $\mathbf{r} \in \mathbb{Q}^k$ *such that the denominators of the coordinates of* $\mathbf{r}$ *only have prime factors dividing* d *and such that*

$$<\mathbf{a}_i,\mathbf{r}> \in \mathbb{Z} \text{ for } i=1,\ldots,s, \text{ and } <\mathbf{b},\mathbf{r}> - \tfrac{1}{d} \in \mathbb{Z}. \tag{2.3}$$

Further, there is a polynomial-time algorithm that computes d *and the above vector* $\mathbf{r}$ *from* $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}$.

Proof. Compute a basis $\{\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ of $\mathbb{Z}^k$ and $d_1,\ldots,d_t \in \mathbb{Q}_{>0}$ as in Lemma 2.3 from $\mathbf{a}_1,\ldots,\mathbf{a}_s$. Further, compute $\xi_1,\ldots,\xi_t \in \mathbb{Q}$ with $\mathbf{b} = \sum_{i=1}^t \xi_i d_i \mathbf{e}_i$, e.g., by Gaussian elimination. Then d is the smallest positive integer such that $d\xi_1,\ldots,d\xi_t \in \mathbb{Z}$; hence it can be computed in polynomial time. Let $u_1,\ldots,u_t$ be the numerators of $d_1,\ldots,d_t$, respectively. Compute $(u_1\backslash d),\ldots,(u_t\backslash d)$. Note that $\gcd(d, \xi_1 d(u_1 \setminus d),\ldots,\xi_t d(u_t \setminus d)) = 1$. Now compute $s_1,\ldots,s_t \in \mathbb{Z}$ satisfying

$$\sum_{i=1}^t \xi_i s_i d \cdot (u_i \setminus d) \equiv 1 \pmod{d}, \tag{2.4}$$

with Euclid's algorithm. Finally, compute $\mathbf{r} \in \mathbb{Q}^k$, e.g., by Gaussian elimination, with

$$< \mathbf{e}_i, \mathbf{r} > = \begin{cases} \frac{s_i(u_i \setminus d)}{d_i} & \text{for } i = 1,\ldots,t, \\ 0 & \text{for } i = t+1,\ldots,k. \end{cases} \tag{2.5}$$

Note that the denominators of $s_i \cdot (u_i \setminus d)/d_i$ only have prime factors dividing d for $i = 1,\ldots,t$. Since $\{\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ is a basis of $\mathbb{Z}^k$, this implies that the denominators of the coordinates of $\mathbf{r}$ only have prime factors dividing d. Further, from (2.5) and $\mathbf{a}_i \in \mathbb{Z}\{d_1\mathbf{e}_1,\ldots,d_t\mathbf{e}_t\}$ it follows that $< \mathbf{a}_i, \mathbf{r} > \in \mathbb{Z}$ for $i = 1,\ldots,t$. Finally, from (2.4), (2.5), and $\mathbf{b} = \sum_{i=1}^{t} \xi_i d_i \mathbf{e}_i$, it follows that $< \mathbf{b}, \mathbf{r} > - \frac{1}{d} \in \mathbb{Z}$. It is easy to verify that all the computations mentioned above take polynomial time. This proves Lemma 2.5. ❑

Proof of Theorem 2.2. Without loss of generality we may assume that $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b} \in \mathbb{Z}^k$. Indeed, if $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b} \in (\tilde{\mathbb{Q}}_N)^k$, we can compute in polynomial (in L) time $m \in \mathbb{N}$ with $\gcd(m, \lambda(N)) = 1$ such that $\mathbf{a}_i' := m\mathbf{a}_i$ $(i = 1,\ldots,s)$, $\mathbf{b}' := m\mathbf{b} \in \mathbb{Z}^k$, and we can proceed further with $\mathbf{a}_1',\ldots,\mathbf{a}_s',\mathbf{b}'$. The integer d is also the smallest positive integer x for which $x\mathbf{b}' \equiv \sum_{i=1}^{s} \xi_i \mathbf{a}_i' \pmod{\lambda(N)}$ is solvable in $\xi_1,\ldots,\xi_s \in \mathbb{Z}$ and $x \mapsto x^m$ is 1–1 on $\mathbb{Z}_N^*$. Hence $\{z \in \mathbb{Z}_N^* \mid \exists \mathbf{y} \in (\mathbb{Z}_N^*)^k\colon \mathbf{y}^{\mathbf{b}'} \equiv z,\ \mathbf{y}^{\mathbf{a}_i'} \equiv \mathbf{x}^{\mathbf{a}_i'} \pmod{N}$ for $i = 1,\ldots,s\}$ has the same cardinality as $\{z \in \mathbb{Z}_N^* \mid \exists \mathbf{y} \in (\mathbb{Z}_N^*)^k\colon \mathbf{y}^{\mathbf{b}} \equiv z,\ \mathbf{y}^{\mathbf{a}_i} \equiv \mathbf{x}^{\mathbf{a}_i} \pmod{N}$ for $i = 1,\ldots,s\}$.

(i) Compute a basis $\{\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ of $\mathbb{Z}^k$ and $d_1,\ldots,d_t$ (which are now positive integers) such that $\{d_1\mathbf{e}_1,\ldots,d_t\mathbf{e}_t\}$ is a basis of $\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$. Further, compute integers $\beta_1,\ldots,\beta_k$ such that $\mathbf{b} = \sum_{i=1}^{k} \beta_i \mathbf{e}_i$. Since $\mathbf{b} \notin \mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$, at least one of the integers $\beta_{t+1},\ldots,\beta_k$, say β_{t+1}, is nonzero. There are integers $\eta_1,\ldots,\eta_t$ such that $d\mathbf{b} \equiv \sum_{i=1}^{t} \eta_i d_i \mathbf{e}_i \pmod{\lambda(N)}$. This implies that $d\beta_{t+1} \equiv 0 \pmod{\lambda(N)}$. Hence β_{t+1} is a nonzero multiple of $\lambda(N)/d$. All operations mentioned above can be done in polynomial (in L)-time and so β_{t+1} can be computed in polynomial (in L)-time. This proves (i).

(ii) Let $S_1 = \{z \in \mathbb{Z}_N^* \mid \exists \mathbf{y} \in (\mathbb{Z}_N^*)^k\colon \mathbf{y}^{\mathbf{b}} \equiv z,\ \mathbf{y}^{\mathbf{a}_i} \equiv 1 \pmod{N}$ for $i = 1,\ldots,s\}$. Then $\{z \in \mathbb{Z}_N^* \mid \exists \mathbf{y} \in (\mathbb{Z}_N^*)^k\colon \mathbf{y}^{\mathbf{b}} \equiv z,\ \mathbf{y}^{\mathbf{a}_i} \equiv \mathbf{x}^{\mathbf{a}_i} \pmod{N}$ for $i = 1,\ldots,s\} = \{z \cdot \mathbf{x}^{\mathbf{b}} \mid z \in S_1\}$. Hence it suffices to show, that S_1 is equal to $S_2 := \{z \in \mathbb{Z}_N^* \mid z^d \equiv 1 \pmod{N}\}$.
First take $z \in S_1$. There are $\xi_1,\ldots,\xi_t \in \mathbb{Z}$ such that $d\mathbf{b} \equiv \sum_{i=1}^{s} \xi_i \mathbf{a}_i \pmod{\lambda(N)}$. Together with the fact that $a^{\lambda(N)} \equiv 1 \pmod{N}$ for every $a \in \mathbb{Z}_N^*$, this implies that for some $\mathbf{y} \in (\mathbb{Z}_N^*)^k\colon\ z^d \equiv \mathbf{y}^{d\mathbf{b}} \equiv \prod_{i=1}^{s} (\mathbf{y}^{\mathbf{a}_i})^{\xi_i} \equiv 1 \pmod{N}$. Hence $z \in S_2$. It follows that $S_1 \subseteq S_2$.
Now take $z \in S_2$. We have that $N = PQ$. Put $\delta = \gcd(d, P-1)$. Then δ is the smallest positive integer x such that $x\mathbf{b} \equiv \sum_{i=1}^{s} \xi_i \mathbf{a}_i \pmod{P-1}$ has a solution in

$\xi_1,\ldots,\xi_s \in \mathbb{Z}$, i.e., the smallest positive integer x for which $x\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s,(P-1)\mathbf{e}_1,\ldots,(P-1)\mathbf{e}_k\}$, where $\{\mathbf{e}_1,\ldots,\mathbf{e}_k\}$ is any basis of $\mathbb{Z}^k$. By Lemma 2.5, there is a vector $\mathbf{r} \in \mathbb{Q}^k$, with

$$\left.\begin{aligned} <(P-1)\,\mathbf{e}_j,\mathbf{r}> &\in \mathbb{Z} \quad \text{for } j=1,\ldots,k,\\ <\mathbf{a}_j,\mathbf{r}> &\in \mathbb{Z} \quad \text{for } j=1,\ldots,s,\\ <\mathbf{b},\mathbf{r}> -\tfrac{1}{\delta} &\in \mathbb{Z}\,. \end{aligned}\right\}$$

Put $\mathbf{v}:=(P-1)\mathbf{r}$. Then $\mathbf{v} = (v_1,\ldots,v_k) \in \mathbb{Z}^k$ and

$$\left.\begin{aligned} <\mathbf{a}_j,\mathbf{v}> &\equiv 0 \pmod{(P-1)} \quad \text{for } j=1,\ldots,s,\\ <\mathbf{b},\mathbf{v}> &\equiv \tfrac{(P-1)}{\delta} \pmod{(P-1)}. \end{aligned}\right\} \tag{2.6}$$

Since $z \in S_2$, we have $z^\delta \equiv 1 \pmod P$. Further, the group $\mathbb{Z}_P^*$ is cyclic of order $P-1$. Hence there is a residue class w with $w^{(P-1)/\delta} \equiv z \pmod P$. Put $\mathbf{y}_1 = (w^{v_1},\ldots,w^{v_k})$. Then (2.6) implies that $\mathbf{y}_1^{\mathbf{a}_j} \equiv w^{<\mathbf{a}_j,\mathbf{v}>} \equiv 1 \pmod P$ for $j=1,\ldots,s$ and that $\mathbf{y}_1^{\mathbf{b}} \equiv w^{<\mathbf{b},\mathbf{v}>} \equiv w^{(P-1)/\delta} \equiv z \pmod P$.

In the same way we have for the other prime Q a $\mathbf{y}_2$ such that $\mathbf{y}_2^{\mathbf{a}_j} \equiv 1 \pmod Q$ for $j=1,\ldots,s$ and $\mathbf{y}_2^{\mathbf{b}} \equiv z \pmod Q$. By the Chinese Remainder Theorem, there is a $\mathbf{y} \in (\mathbb{Z}_N^*)^k$ with $\mathbf{y} = \mathbf{y}_1 \pmod P$ and $\mathbf{y} = \mathbf{y}_2 \pmod Q$. This $\mathbf{y}$ satisfies $\mathbf{y}^{\mathbf{a}_j} \equiv 1 \pmod N$ for $j=1,\ldots,s$, and $\mathbf{y}^{\mathbf{b}} \equiv z \pmod N$. Hence $z \in S_1$. We conclude that also $S_2 \subseteq S_1$. Therefore $S_2 = S_1$ and part (ii) of Theorem 2.2 has been proved. ❑

Proof of Theorem 2.1. Assume we are given N and $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b} \in (\tilde{\mathbb{Q}}_N)^k$ satisfying (2.1).

(i) Assume there is a probabilistic algorithm AL which computes $\mathbf{x}^{\mathbf{b}}$ from $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ in time $\le T_{\mathrm{AL}}(L)$ with probability of success $\ge \varepsilon_{\mathrm{AL}}(L)$ for randomly chosen $\mathbf{x} \in (\mathbb{Z}_N^*)^k$. Fix $u \in \mathbb{Z}_N^*$. We describe a probabilistic algorithm $\overline{\mathrm{AL}}$ to compute $u^{1/d}$. The idea is to apply AL to the vector $\mathbf{u} = (u^{t_1},\ldots,u^{t_k})$ for an appropriate vector $\mathbf{t} \in \tilde{\mathbb{Q}}_N$. However, this $\mathbf{u}$ is not a random vector in $(\mathbb{Z}_N^*)^k$, all its coordinates being a power of the same residue class; hence we know nothing about the probability of success when AL is applied to $\mathbf{u}$. We use the well-known trick of applying AL to a vector of the form $\mathbf{x} = (u^{t_1} r_1^m,\ldots,u^{t_k} r_k^m)$ instead, where $\mathbf{r} = (r_1,\ldots,r_k)$ (the blinding factors) is randomly chosen from $(\mathbb{Z}_N^*)^k$ and m is such that $m\mathbf{a}_1,\ldots,m\mathbf{a}_s,m\mathbf{b} \in (\mathbb{Z}_N^*)^k$. Since $\mathbb{Z}_N^*$ is a multiplicative group and the mapping $x \to x^m$ on $\mathbb{Z}_N^*$ is 1-1 in view of $\gcd(m,\varphi(N)) = 1$, this vector $\mathbf{x}$ is uniformly distributed on $(\mathbb{Z}_N^*)^k$.

Below we describe algorithm $\overline{\mathrm{AL}}$ (all congruences are mod N):

Step 1. Compute $\mathbf{t}=(t_1,\ldots,t_k)\in\mathbb{Q}^k$ and $\alpha_1,\ldots,\alpha_k,\beta\in\mathbb{Z}$ such that $\langle\mathbf{a}_i,\mathbf{t}\rangle=\alpha_i$ for $i=1,\ldots,s$, $\langle\mathbf{b},\mathbf{t}\rangle+\beta=\frac{1}{d}$ and the denominators of $t_1,\ldots,t_k$ are composed of primes dividing d. Since $\gcd(m,\varphi(N))=1$, we have $\mathbf{t}\in(\tilde{\mathbb{Q}}_N)^k$. Compute m such that $m\mathbf{a}_1,\ldots,m\mathbf{a}_s,m\mathbf{b}\in\mathbb{Z}^k$.

Step 2. Choose $\mathbf{r}=(r_1,\ldots,r_k)\in_R(\mathbb{Z}_N^*)^k$.

Step 3. Compute $u^{\alpha_i}\mathbf{r}^{m\mathbf{a}_i}\equiv u^{\langle\mathbf{a}_i,\mathbf{t}\rangle}\cdot\mathbf{r}^{m\mathbf{a}_i}\equiv\mathbf{x}^{\mathbf{a}_i}$ for $i=1,\ldots,s$, where $\mathbf{x}\equiv(u^{t_1}r_1^m,\ldots,u^{t_k}r_k^m)$. This computation is easy, since $\alpha_i\in\mathbb{Z}$, $m\mathbf{a}_i\in\mathbb{Z}^k$ for $i=1,\ldots,s$. Note that it need not be feasible to compute $\mathbf{x}$.

Step 4. Apply AL to $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$.

Step 5. If AL outputs $\mathbf{x}^{\mathbf{b}}$, then compute $\mathbf{x}^{\mathbf{b}}\mathbf{r}^{-m\mathbf{b}}u^{\beta}\equiv u^{\langle\mathbf{b},\mathbf{t}\rangle+\beta}\equiv u^{1/d}$. This is possible since $\beta\in\mathbb{Z}$ and $m\mathbf{b}\in\mathbb{Z}^k$.

Choosing $\mathbf{r}\in_R(\mathbb{Z}_N^*)^k$ in Step 2, can be done by using Algorithm 2.2 from Section 2.5. The probability of success $\geq\frac{1}{2}$ and the conditional probability distribution of $\mathbf{r}$ given success is uniform on $(\mathbb{Z}_N^*)^k$.

Steps 1, 2, 3 and 5 of algorithm $\overline{\text{AL}}$ described above have running time $L^{O(1)}$. Further, Step 4 has running time T_{AL}. Hence the running time of $\overline{\text{AL}}$ is $\leq T_{\text{AL}}+L^{O(1)}$. Step 2 has probability of success $\geq\frac{1}{2}$. Given success in Step 2, the conditional probability distribution of $\mathbf{x}$ is uniform on $(\mathbb{Z}_N^*)^k$ and hence the conditional probability of success in Step 4 is $\geq\varepsilon_{\text{AL}}$. Therefore, the unconditional probability of success of $\overline{\text{AL}}$ is $\geq\frac{1}{2}\varepsilon_{\text{AL}}$. This proves (i).

(ii) Assume we are given a probabilistic algorithm AL which from randomly chosen $u\in\mathbb{Z}_N^*$ computes $u^{1/d}$ in time $\leq T_{\text{AL}}$ and probability of success $\geq\varepsilon_{\text{AL}}(L)$. We construct the following algorithm $\overline{\text{AL}}$ (the congruences are mod N):

Step 1. Compute $\xi_1,\ldots,\xi_s\in\mathbb{Z}$ such that $d\mathbf{b}=\sum\xi_i\mathbf{a}_i$.

Step 2. Choose $r\in_R\mathbb{Z}_N^*$.

Step 3. Compute $u\equiv r^d\cdot\prod_i(\mathbf{x}^{\mathbf{a}_i})^{\xi_i}$.

Step 4. Apply AL to u.

Step 5. If AL outputs $u^{1/d}$, then compute $r^{-1}u^{1/d}\equiv\prod_i(\mathbf{x}^{\mathbf{a}_i})^{\xi_i/d}\equiv(\mathbf{x}^{d\mathbf{b}})^{1/d}\equiv\mathbf{x}^{\mathbf{b}}$.

If we choose r in Step 2 as described above, then with essentially the same argument as above, it follows that $\overline{\text{AL}}$ has running time $\leq T_{\text{AL}}+L^{O(1)}$ and probability of success $\geq\frac{1}{2}\varepsilon_{\text{AL}}$ for arbitrary $\mathbf{x}\in(\mathbb{Z}_N^*)^k$. This proves (ii). ❑

2.7. Some practical applications

Let us return to the protocols of Section 2.1. Let N be a composite modulus and let $\mathbf{a}_1,..,\mathbf{a}_s,\mathbf{b} \in (\tilde{\mathbb{Q}}_N)^k$. Signature authority $\mathcal{Z}$ chooses at random $\mathbf{x} \in (\mathbb{Z}_N^*)^k$ and issues the signatures $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ to individual $\mathcal{A}$. Now $\mathcal{A}$ wants to compute $\mathbf{x}^{\mathbf{b}}$. Is he able to do this? Of course, we assume that for $\mathcal{A}$ it is computationally infeasible to compute RSA-roots modulo N, since otherwise he could forge all signatures. According to Assumption 1.3, Theorem 2.1 implies the following:

Corollary. 2.1. *Assume there is an integer d with $(d,\varphi(N))=1$ and $d\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$. Then it is feasible for $\mathcal{A}$ to compute $\mathbf{x}^{\mathbf{b}}$ from $\{N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}, \mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}\}$ for uniformly chosen $\mathbf{x} \in (\mathbb{Z}_N^*)^k$ if and only if $\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$.*

If $\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$, then $\mathbf{x}^{\mathbf{b}}$ can be computed from $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ simply by multiplying and dividing (mod N) the signatures received: if $\xi_1,\ldots,\xi_s \in \mathbb{Z}$ are such that $\mathbf{b} = \xi_1\mathbf{a}_1 + \ldots + \xi_s\mathbf{a}_s$, then $\mathbf{x}^{\mathbf{b}} \equiv \prod_{i=1}^{s}(\mathbf{x}^{\mathbf{a}_i})^{\xi_i} \pmod N$. Hence the corollary means that $\mathcal{A}$ cannot compute RSA-signatures from other signatures, unless he is able to do this using only the obvious operations on RSA-signatures: multiplying and dividing (mod N). This corollary can also be used in situations where $\mathcal{Z}$ also issues $\mathbf{x}$ or where $\mathcal{A}$ receives signatures $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ on distinct vectors $\mathbf{x}_1,\ldots,\mathbf{x}_s$ (see Remark 2.2 of Section 2.4).

We now give five examples to illustrate Corollary 2.1.

Example 2.1. In Chapter 6, a user-anonymous offline payment system is introduced, and in Section 6.10 we will use Corollary 2.1 to discuss a special attack by the user on this payment system (Cheating 3).

Example 2.2. Consider the user-anonymous offline payment system of [OO89] for coins. In this system, the bank uses a signature scheme that we do not specify here. The user makes RSA-signatures using his own modulus N, which factorization he keeps secret; here, then, the user plays the role of a signature authority. Let L be a fixed integer, and define $I \equiv (\mathrm{ID}_{\mathrm{user}} \| R)^L \bmod N$, where R is a number chosen at random by the user and $\|$ denotes concatenation. In Figure 2.1 the basic idea of the withdrawal (in which the user is able to blind messages and the bank to verify the messages and to sign them, (see [OO89]), and the spending protocol of a coin is given. After some time, the shop sends the numbers that it received to the bank and the bank verifies that these numbers have not been used before.

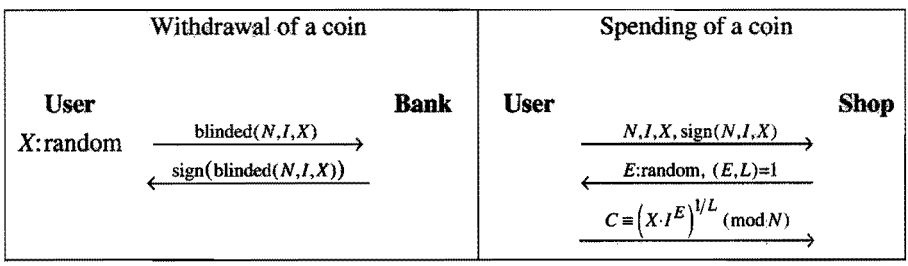


Fig. 2.1. The (simplified) offline payment system of [OO89] for coins.

From Corollary 2.1 it follows that it is not feasible for the shop/bank to compute the identity of the user (i.e., $I^{1/L} \bmod N$) from N,I,X,E,L and $C = X^{1/L} \cdot I^{E/L}$. But if the user spends the same coin at two shops, the bank receives N,I,X,L, sign(N,I,X), two integers E_1,E_2 that are coprime to L, and the signatures $\left(X \cdot I^{E_1}\right)^{1/L} \pmod N$ and $\left(X \cdot I^{E_2}\right)^{1/L} \pmod N$. From Corollary 2.1 it follows that the bank can compute $I^{1/L} \bmod N$ from these numbers (and hence the user's identity) if and only if $\gcd(E_1-E_2,L) = 1$.

A user can spend the same coin at different shops ("double spending"). What is the probability that a double spender will be caught, that is, what is $\Pr(E_1-E_2 \in \mathbb{Z}_L^*)$ for randomly chosen numbers $E_1,E_2 \in \mathbb{Z}_L^*$? Let p be a prime divisor of L. Then $p \nmid E_1$, $p \nmid E_2$, and thus $\Pr(p \nmid (E_1-E_2)) = 1 - \Pr(p \mid (E_1-E_2)) = 1 - \frac{1}{p-1}$. If we write $L = p_1^{a_1} \ldots p_n^{a_n}$ with p_i prime and $a_i > 0$, then $\Pr(E_1-E_2 \in \mathbb{Z}_L^*) = \Pr(p_j \nmid (E_1-E_2) \mid 1 \le j \le n) = \prod_{j=1}^{n} (1 - \frac{1}{p_j - 1}) \approx \varphi(L)/L$. This probability is close to 1 if L is a large prime, and it is close to 0 if L is the product of many small distinct primes. The probability is equal to 0 if 2 is a divisor of L. Therefore, it is unwise to let the user choose L freely himself (which was the original suggestion); rather L should be fixed as a large prime.

Example 2.3. In [AT83] and [McKTMA85] schemes based on cryptography are proposed for controlling access to information within a group of users organized in a hierarchy. Assume a communication system of users U_i (or classes of users) that is partially ordered by the relation $\le$, where $U_i \le U_j$ means that user U_j can have access to information destined to user U_i. Denote the authority by U_0, who chooses an RSA-modulus N and a secret key K. To each user U_i the authority assigns a public key t_i and a secret key $K^{t_i} \pmod N$. These integers t_i are chosen in such a way that

$$t_j \mid t_i \quad \text{if and only if} \quad U_i \le U_j. \tag{2.7}$$

Hence this scheme enables a user U_j at some level to compute from his own secret key

K^{t_j} the secret keys K^{t_i} of the users U_i below him in the organization, because $(K^{t_j})^{t_i/t_j} \equiv K^{t_i} \pmod N$ and t_i/t_j is an integer.

However, if $U_i \not\leq U_j$, then t_i/t_j is not an integer and this computation by U_j is considered to be infeasible. Thus the only remaining question is how to choose the integers t_i. Figure 2.2 shows the Hasse diagram of a poset and shows three different methods for the choices for t_i, which are indicated inside the nodes (representing users). The methods are explained below.

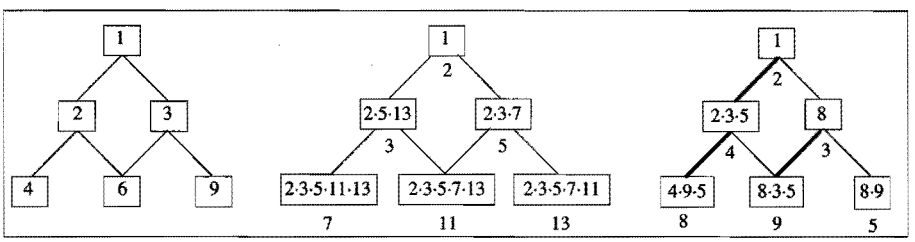


Fig. 2.2.1. **Fig. 2.2.2.** **Fig. 2.2.3.**

The Hasse diagram of a poset with three different methods for the choices of the public keys, which are indicated inside the nodes (representing users).

i) The assignment of Figure 2.2.1 satisfies condition (2.7). But two users may be able to successfully cooperate to discover a secret key to which they are not entitled. For instance, from the keys K^4 and K^9, the secret key K can be computed as $(K^4)^{-2}K^9 \equiv K \pmod N$, and hence all the keys in the system can be computed.

ii) In [AT83] it is proven that it is feasible for a user to compute K^d from $\{K^{a_1},\ldots,K^{a_s}\}$ if and only if $\gcd(a_1,\ldots,a_s) \mid d$. Thus the previous collaborative attack can be prevented by choosing the integers t_i in such a way that they also satisfy

$$\gcd_{\{j \mid U_j \geq U_i\}} (t_j) \not\mid t_i. \tag{2.8}$$

They propose the following choice of the integers t_i: the authority chooses a sequence $\{p_i\}$ of distinct primes (which are indicated in Figure 2.2.2 below the nodes) and computes $t_i = \prod_{U_j \leq U_i} p_j$. It is easy to see that this assignment satisfies (2.7) and (2.8). The disadvantage of this choice is that the used exponents (the t_i's) can get quite large, even for small posets.

iii) In [McKTMA85] another assignment - one that satisfies (2.7) and (2.8)- is proposed (see Figure 2.2.3), called the canonical assignment. The poset is first decomposed into disjoint chains (a *chain* is a totally ordered subset: see the bold lines). Each chain is assigned a distinct prime. For each node i, we define $n_i = p^m$, where i is the m^{th} node from the top in the chain whose prime is p. Then the t_i's are computed as $t_i = \operatorname{lcm}_{j \leq i} n_j$.

The authors also prove that any assignment contains a canonical set of t_i's, and that any effort to keep the t_i's as small as possible will also lead to a canonical set.

Example 2.4. In [Bos92] a new signature scheme is presented, based on the Lamport signatures [DH76]. In this new construction a user cannot compute a new signature on a given message, even if he received RSA-signatures on messages of his choice.

Let k,l be fixed parameters (at least one of which is even) and R a fixed public table of k random elements $(r_1,...,r_k)$. Let S a fixed partition of a $l\times k$ matrix into dominos of size 1×2 (dominos are used for a good visualization). These $\frac{kl}{2}$ dominos have a fixed order, and they all have a "0" side and a "1" side (see Figure 2.3). Each user has his own RSA-modulus.

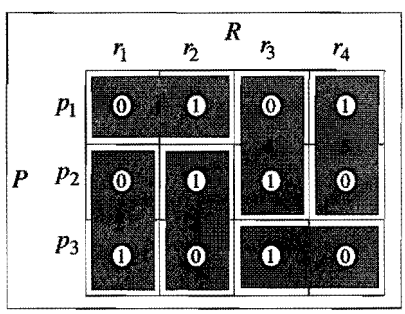


Fig. 2.3. Example of a partition S of a 3×4 matrix into six 0/1-dominos.

If a user wants to sign a $\frac{kl}{2}$ bit message $m=(m_1,...,m_{kl/2})$, he determines a set $P=(p_1,...,p_l)$ of l distinct primes. By using the partition S, the bits of the message determine a subset M of size $\frac{kl}{2}$, because bit m_i determines the m_i-side of domino i. The user's signature on m will be

$$P,\ \prod_{i,j\in M} r_j^{1/p_i} \pmod N.$$

If for example, the message to be signed is (001011), then the signer determines a set (p_1,p_2,p_3) of distinct primes, and by using the partition of Figure 2.3, his signature will be (p_1,p_2,p_3), $r_1^{1/p_1} r_1^{1/p_2} r_2^{1/p_2} r_3^{1/p_1} r_4^{1/p_1} r_3^{1/p_3} \pmod N$. For the next signature, the user must choose a set P that is disjoint from all previous sets of primes used (so for instance all these sets of primes can be fixed and published in advance).

Consider all the signatures that person $\mathcal{A}$ has issued. If somebody else wants to forge a signature of $\mathcal{A}$ on a message m not yet issued by $\mathcal{A}$, he has to compute several $r_j^{1/p_i} \pmod N$, where all the used primes p_i are different from all primes previously used by $\mathcal{A}$. Hence by Corollary 2.1 it follows that he cannot compute any of these

numbers r_j^{1/p_i}. Thus, nobody can forge any signature of $\mathcal{A}$ (except by reusing the primes).

This signature scheme can also be optimized in various ways (see [Bos92]), but this is beyond the scope of this chapter.

Example 2.5. If two parties want to exchange secret messages, they can use the *one-time pad* (invented by Vernam): the secret key is a long sequence of randomly chosen bits and the cipher text is the X-OR of the cleartext and the secret key. The ciphertext thereby obtained contains no Shannon information about the cleartext; but the main drawback of this system is the huge secret key that has to be generated, distributed, and stored by the two parties. Therefore, instead of a truly random key, a *pseudorandom* key will be used, which is created by a generator from an initial seed. The strength of this scheme depends on the strength of the pseudorandom key.

Shamir [Sh83] has proposed the following pseudorandom sequence: the two parties create an RSA-modulus N and choose a random seed S and a sequence of keys $k_1,k_2,k_3,\ldots$, which are coprime to $\varphi(N)$ and pairwise coprime to each other (e.g. the sequence 3,5,7,11,13,...). Then the pseudorandom sequence $R_1,R_2,R_3,\ldots$ will be

$$\begin{aligned} R_1 &\equiv S^{1/k_1} \pmod N, \\ R_2 &\equiv S^{1/k_2} \pmod N, \\ &\vdots \end{aligned}$$

Shamir considers the following two problems ($t,k_1,k_2,\ldots,k_t$ are fixed):

(1) Given N and S, compute R_1.

(2) Given $N,S,R_2,R_3,\ldots,R_t$, compute R_1.

Shamir proves that from $N,a_1,\ldots,a_t$, and $S^{a_1},S^{a_2},\ldots,S^{a_t} \pmod N$, one can compute in polynomial time $S^{a_0} \pmod N$ where $a_0 = \gcd(a_1,\ldots,a_t)$. With this result, he proves (roughly speaking) that if it is feasible for a person to solve some of the instances of problem (1) with probability $\geq \varepsilon$, then that person can solve some of the instances of problem (2) with probability $\geq \varepsilon$.

This chapter is a generalization of Shamir's result.

2.8 Some remarks on Corollary 2.1.

Remark 2.3. From this corollary, the following well-known results can be obtained.

Corollary 2.2. *Let $a,a_1,\ldots,a_s,b,d$ be positive integers coprime to $\varphi(N)$, $c,c_1,\ldots,c_s$ be integers, and x,y be chosen randomly from $\mathbb{Z}_N^*$. Then the following five results hold for $\mathcal{A}$.*

(i) *It is feasible to compute $x^{1/d}$ from $\{x,x^{c/a}\}$* $\quad \Leftrightarrow d \mid \frac{a}{\gcd(a,c)}$.

(ii) *It is feasible to compute $x^{1/d}$ from $\{x,y,(x^{c_1}y^{c_2})^{1/a}\}$* $\quad \Leftrightarrow d \mid \frac{\gcd(c_2,a)}{\gcd(c_1,c_2,a)}$.

(iii) *It is feasible to compute* $x^{1/d}$ *from* $\{x, x^{1/a_1}, \ldots, x^{1/a_s}\}$ $\Leftrightarrow$ $d \mid \mathrm{lcm}(a_1, \ldots, a_s)$
(iv) *It is feasible to compute* $(xy)^{1/d}$ *from* $\{x, y, x^{1/a}, y^{1/b}\}$ $\Leftrightarrow$ $d \mid \gcd(a,b)$.
(v) *It is feasible to compute* x^d *from* $\{x^{c_1}, \ldots, x^{c_s}\}$ $\Leftrightarrow$ $\gcd(c_1, \ldots, c_s) \mid d$
[AT83].

Remark 2.4. Corollary 2.1 can also be proved without Lemmas 2.3, 2.4, and 2.5, by using the next lemma, which is a corollary of Kronecker's Approximation Theorem.

Lemma 2.6. *Let A be a rational matrix and let* $\mathbf{b}$ *be a rational vector. Then the system* $A\mathbf{v} = \mathbf{b}$ *has an integral solution* $\mathbf{v}$ *if and only if* $\mathbf{y}^T\mathbf{b}$ *is an integer for each rational column vector* $\mathbf{y}$ *for which* $\mathbf{y}^T A$ *is integral.*

Proof of Corollary 2.1.
We will prove only the "difficult" part of Corollary 2.1: assume that there is an integer d with $(d, \varphi(N)) = 1$ and $d\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1, \ldots, \mathbf{a}_s\}$; and that there is a probabilistic algorithm AL that on input $\{N, \mathbf{a}_1, \ldots, \mathbf{a}_s, \mathbf{b}, \mathbf{x}^{\mathbf{a}_1}, \ldots, \mathbf{x}^{\mathbf{a}_s}\}$ computes $\mathbf{x}^{\mathbf{b}}$ in time $\leq T_{\mathrm{AL}}$ with probability of success $\geq \varepsilon_{\mathrm{AL}}$ for random $\mathbf{x} \in (\mathbb{Z}_N^*)^k$. Let $u \in \mathbb{Z}_N^*$ be an arbitrary number. Below we describe an algorithm to compute an RSA-root on u.

Step 1. Compute $t = (t_1, \ldots, t_k) \in \mathbb{Q}^k$ and $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}$ such that $\langle \mathbf{a}_i, \mathbf{t} \rangle = \alpha_i$ for $i = 1, \ldots, s$; and the denominators of $t_1, \ldots, t_k$ are composed of primes dividing d. Since $\gcd(d, \varphi(N)) = 1$, we have $\mathbf{t} \in (\tilde{\mathbb{Q}}_N)^k$. Compute m such that $m\mathbf{a}_1, \ldots, m\mathbf{a}_s$, $m\mathbf{b} \in \mathbb{Z}^k$. Write $\langle \mathbf{b}, \mathbf{t} \rangle + \beta = \frac{\gamma}{\delta}$ for some integers $\beta, \gamma > 0$, δ, $\gcd(\gamma, \delta) = 1$.

Step 2. Choose $\mathbf{r} = (r_1, \ldots, r_k) \in_R (\mathbb{Z}_N^*)^k$.

Step 3. Compute $u^{\alpha_i} \mathbf{r}^{m\mathbf{a}_i} \equiv u^{\langle \mathbf{a}_i, \mathbf{t} \rangle} \cdot \mathbf{r}^{m\mathbf{a}_i} \equiv \mathbf{x}^{\mathbf{a}_i}$ for $i = 1, \ldots, s$, where $\mathbf{x} \equiv (u^{t_1} r_1^m, \ldots, u^{t_k} r_k^m)$. This computation is easy, since $\alpha_i \in \mathbb{Z}$, $m\mathbf{a}_i \in \mathbb{Z}^k$ for $i = 1, \ldots, s$. Note that it need not be feasible to compute $\mathbf{x}$.

Step 4. Apply AL to $\mathbf{x}^{\mathbf{a}_1}, \ldots, \mathbf{x}^{\mathbf{a}_s}$.

Step 5. If AL outputs $\mathbf{x}^{\mathbf{b}}$, then compute $\mathbf{x}^{\mathbf{b}} \mathbf{r}^{-m\mathbf{b}} u^{\beta} \equiv u^{\langle \mathbf{b}, \mathbf{t} \rangle + \beta} \equiv u^{\gamma/\delta}$. This is possible because $\beta \in \mathbb{Z}$ and $m\mathbf{b} \in \mathbb{Z}^k$. Because $\gcd(\gamma, \delta) = 1$, we compute $u^{1/\delta}$ from $u^{\gamma/\delta}$.

We have assumed that it is feasible to compute $u^{1/\delta}$ for an arbitrary number u, if and only if $\delta = 1$, i.e., if $\mathbf{y}^T\mathbf{b}$ is an integer. Now, Lemma 2.6 states that the system $[\mathbf{a}_1 \ldots \mathbf{a}_s]\mathbf{v} = \mathbf{b}$ has an integral solution $\mathbf{v}$, which can be computed in polynomial time (for instance by using the Gaussian elimination method). Thus $\mathbf{b} \in \mathbb{Z}\{\mathbf{a}_1, \ldots, \mathbf{a}_s\}$. ❑

Remark 2.5. In the case that $\mathbf{x}^{\mathbf{b}}$ has to be computed from $\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s}$ and $\mathbf{x}$ (see Remark 2.2), we can reformulate Corollary 2.1 by using the next lemma of [Heg1858] (page 111); but it does *not* yield a polynomial-time algorithm.

Lemma 2.7. *Let A be a rational matrix of full row rank with k rows, and let $\mathbf{b}$ be a rational k-dimensional column vector. Then $A\mathbf{v} = \mathbf{b}$ has an integral solution $\mathbf{v}$, if and only if the gcd of all subdeterminants of A of order k divides each subdeterminant of $[A\ \mathbf{b}]$ of order k.*

Corollary 2.3. *Let $\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b}\in(\tilde{\mathbb{Q}}_N)^k$, $\mu_1,\ldots,\mu_m$ be all the subdeterminants of $[\mathbf{a}_1\ \ldots\ \mathbf{a}_s]$ of order between 1 and $\min(k,s)$, and let $\mu_{m+1},\ldots,\mu_n$ be all the subdeterminants of $[\mathbf{a}_1\ \ldots\ \mathbf{a}_s\ \mathbf{b}]$ of order between 1 and $\min(k,s+1)$, containing at least one entry from $\mathbf{b}$. Then the following two statements are equivalent:*

(i) *It is feasible for $\mathcal{A}$ to compute $\mathbf{x}^{\mathbf{b}}$ from $\{N,\mathbf{a}_1,\ldots,\mathbf{a}_s,\mathbf{b},\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_s},\mathbf{x}\}$ for uniformly chosen $\mathbf{x}\in(\mathbb{Z}_N^*)^k$.*

(ii) $\gcd(1,\mu_1,\ldots,\mu_m) = \gcd(1,\mu_1,\ldots,\mu_n)$.

Note that statement (ii) *cannot* be verified in polynomial time, because $m=\binom{s+k}{k}$ and $n = m+\binom{s+k}{k-1}$ (for $s\geq k$).

Proof. Define the matrices $A=[\mathbf{a}_1\ \ldots\ \mathbf{a}_s]$, $I=[\mathbf{e}_1\ \ldots\ \mathbf{e}_k]$ and $B=[A\ \ I]$. Since each column of I has exactly *one* nonzero entry, each subdeterminant of B containing q columns from I is a subdeterminant of A of order $s-q$. Further, $\det(I) = 1$. Similarly, each subdeterminant of $[A\ \ I\ \ \mathbf{b}]$ containing q columns from I and at least one entry from $\mathbf{b}$ is a subdeterminant of $[A\ \ \mathbf{b}]$ of order $s-q$, containing at least one entry from $\mathbf{b}$.

Hence the subdeterminants of B of order k are $\mu_0=\det(I)=1,\mu_1,\ldots,\mu_m$, and the subdeterminants of $[B\ \mathbf{b}]$ of order k, containing at least one entry from $\mathbf{b}$, are $\mu_{m+1},\ldots,\mu_n$. So for every integer δ, the subdeterminants of $[A\ \delta\mathbf{b}]$ of order k are $\mu_0,\ldots,\mu_m,\delta\mu_{m+1},\ldots,\delta\mu_n$. Because B has full row rank, Lemma 2.7 implies that the equation $B\mathbf{v} = \delta\mathbf{b}$ has an integral solution $\mathbf{v}$ if and only if $\gcd(\mu_0,\ldots,\mu_m)\mid\delta\mu_i$ $(i = m+1,\ldots,n)$. This holds if and only if $\gcd(\mu_0,\ldots,\mu_m)$ is a divisor of $\delta\cdot\gcd(\mu_0,\ldots,\mu_m,\mu_{m+1},\ldots,\mu_n)$. By defining d to be the smallest positive integer such that $B\mathbf{v} = d\mathbf{b}$ has an integral solution $\mathbf{v}$, we have that $d=\frac{\gcd(\mu_0,\ldots,\mu_m)}{\gcd(\mu_0,\ldots,\mu_m,\mu_{m+1},\ldots,\mu_n)}$. From Corollary 2.1 it follows that statement (i) is equivalent with $\gcd(\mu_0,\ldots,\mu_m) = \gcd(\mu_0,\ldots,\mu_m,\mu_{m+1},\ldots,\mu_n)$. ❑

Remark 2.6. One can transform the problem of computing a certain RSA-signature from some given RSA-signatures into another problem. But one must be careful to transform the problem into an *equivalent* problem. We will illustrate this with an example: let x,y be chosen randomly from $\mathbb{Z}_N^*$, and suppose an individual wants to compute

$$x^{1/d} \text{ from } \{x, y, (xy^3)^{1/9}\}.$$

(According to Corollary 2.2.(ii)., this is feasible if and only if d=1 or d=3). We can write $xy^3 \equiv x^2 z$, where $z \equiv y^3 / x$, and we can consider the problem of computing

$$x^{1/d} \text{ from } \{x, z, (x^2 z)^{1/9}\}.$$

(According to Corollary 2.2.(ii)., this is feasible if and only if d=1). The second problem is *not* equivalent to the first problem, because it is infeasible to compute y from $\{x, z, (x^2 z)^{1/9}\}$, i.e., it is infeasible to compute $(\frac{z}{x})^{1/3}$ from $\{x, z, (x^2 z)^{1/9}\}$ (Corollary 2.1). It is easy to see that the problem of computing

$$x^{1/d} \text{ from } \{x, z, (z/x)^{1/3}, (x^2 z)^{1/9}\}$$

is equivalent to the first problem.

If we write $xy^3 \equiv x^4 z^3$, where $z \equiv y / x$, then the problems of computing $x^{1/d}$ from $\{x, y, (xy^3)^{1/9}\}$ and from $\{x, z, (x^4 z^3)^{1/9}\}$ are equivalent, because one can compute $\{x,y\}$ from $\{x, y/x\}$ and vice versa.

2.9. Some open problems

Can the same kind of results be found for discrete log-based signatures? Shmuely [Shm85] made the following first start.

Let N be a composite modulus and g a base to generate encryption keys. Each user U has a secret key $s_U \in \{1,\ldots,N\}$ and a public key $c_U \equiv g^{s_U} \pmod N$. If two users $\mathcal{A}$ and $\mathcal{B}$ want to use a common secret key to encrypt messages, they can use the Diffie-Hellman key exchange protocol [DH76]: they can both compute the key $k \equiv g^{s_A s_B} \pmod N$, because $(c_A)^{s_B} \equiv g^{s_A s_B} \equiv (c_B)^{s_A} \pmod N$.

Is it feasible to break this Diffie-Hellman key exchange protocol, i.e., to compute g^{xy} from $\{N, g, g^x, g^y\}$? Shmuely proves (roughly speaking) that any algorithm that will break this composite Diffie-Hellman key exchange protocol for a nonnegligible fraction of bases g can be used to factor the modulus.

Another example is breaking the protocol for *mental poker* [SRA79]: suppose $\mathcal{A}$ wants to send a secret message m to $\mathcal{B}$. Then they choose a composite modulus N (of

which only $\mathcal{A}$ and $\mathcal{B}$ know $\varphi(N)$) and they perform the protocol of Figure 2.4. Shmuely proves that breaking mental poker, i.e., computing m from $\{N, m^x, m^y, m^{xy}\}$ is a special case of breaking the composite Diffie-Hellman key exchange.

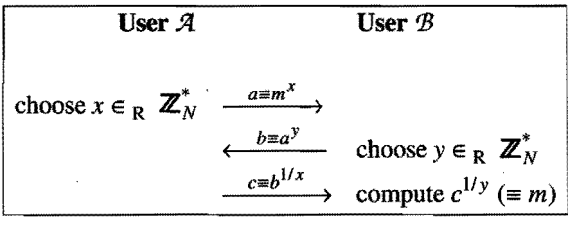


Fig. 2.4. Protocol for Mental poker [SRA79].

3

Which new RSA signatures can be computed from RSA signatures, obtained in a specific interactive protocol?‡

3.1. Introduction

In the previous chapter we studied the case in which a signature authority $\mathcal{Z}$ issues RSA-signatures of certain types to an individual $\mathcal{A}$. The individual tries, by using the signatures he has received, to compute an RSA-signature of a type not issued by $\mathcal{Z}$. The RSA-signatures are products of rational powers of residue classes modulo N, and the residue classes are chosen at random by $\mathcal{Z}$. In this chapter we consider an interactive protocol in which $\mathcal{A}$ may choose some of these residue classes freely.

A class of interactive protocols that will not be considered in this chapter are so-called *ping-pong protocols* (cf. [EGS85]). In such a protocol (which consists of several moves), one party generates a secret message, applies a sequence of operators to it, and sends it to the other party. This party also applies a sequence of operators to the message received, and sends the result back. In each move of the protocol, one of the parties applies a sequence of operators to the last message received, and sends it back. The question would be whether an "active" third party can discover the initial message (by altering messages, impersonating other users, etc.).

In this chapter however, we consider not the entire class of interactive problems, but

‡ This chapter is based on the paper "Which new RSA signatures can be computed from RSA signatures, obtained in a specific interactive protocol?" by Jan-Hendrik Evertse and Eugène van Heyst, which will appear in *Advances in Cryptology-EUROCRYPT '92*.

only problems related to those in Figure 3.1. Initially, $\mathcal{Z}$ chooses two large primes P, Q and computes their product N. Further, $\mathcal{Z}$ chooses two integers a,b coprime to $\varphi(N)=(P-1)(Q-1)$. $\mathcal{Z}$ makes N, a, b public, and keeps P and Q secret. Let c, d also be some integers coprime to $\varphi(N)$. In this protocol, $\mathcal{Z}$ chooses a residue class u, and $\mathcal{A}$ wants to choose h in such a way that after the execution of this protocol, he is able to compute from $\{u,h,u^{1/a}h^{1/b}\}$ a pair $\{t,k\}$ satisfying $t \equiv u^{1/c}k^{1/d} \pmod N$. The reason for considering such problems is that in all payment systems the user chooses blinding factors and can thus influence the signatures he will receive from $\mathcal{Z}$.

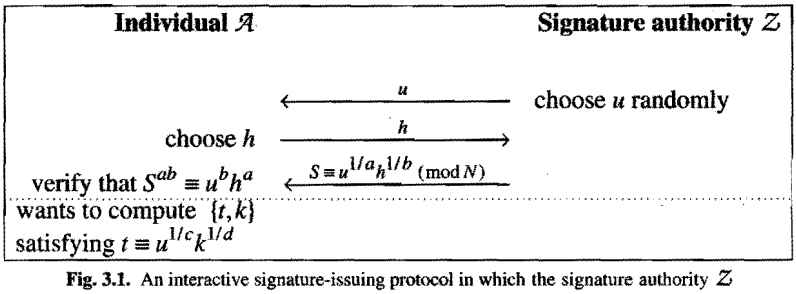


Fig. 3.1. An interactive signature-issuing protocol in which the signature authority $\mathcal{Z}$ issues a signature to individual $\mathcal{A}$.

In Chapter 2 we studied the case in which $\mathcal{A}$ has no influence on the signature received, that is, $\mathcal{A}$ chooses no residue class (i.e., b=1 in Figure 1.1). A necessary and sufficient condition was given for the computation of this new signature to be feasible for $\mathcal{A}$.

In [Dav82], [Denn84] and [DO85] the case is studied in which $\mathcal{Z}$ chooses no residue class, that is, in which individuals were able to obtain signatures on desired messages (i.e., a=1 in Figure 1.1). [Dav82] states that $\mathcal{A}$ can decrypt ciphertext encrypted under $\mathcal{Z}$'s public key and can forge $\mathcal{Z}$'s signature on meaningful messages; [Denn84] can foil this attack by using hashing. The result of [DO85] states that if $\mathcal{A}$ can get enough signatures on carefully chosen residue classes, he is able to sign any message; and they prove that this method is more efficient than the best known algorithms to factor the modulus.

Here, we consider an interactive protocol in which $\mathcal{Z}$ issues a fixed amount of RSA-signatures to $\mathcal{A}$. Generally, these RSA-signatures consist of products of rational powers of residue classes modulo the composite number N of the underlying RSA-scheme; some of these residue classes are chosen by $\mathcal{Z}$ and the others are chosen freely by $\mathcal{A}$. In this chapter we make the following two assumptions:

(i) $\mathcal{A}$ cannot compute RSA-roots of randomly chosen residue classes.

(ii) In his computations, the only operations modulo N that $\mathcal{A}$ uses are multiplications

and divisions.

The problem whether assumption (ii) is necessary remains open. We formulate a necessary and sufficient condition under which $\mathcal{A}$ is able to influence the signatures he receives from $\mathcal{Z}$ in such a way that he can later use these signatures to compute a signature of a type not issued by $\mathcal{Z}$. It turns out that this condition is equivalent to the solvability of a particular quadratic equation in integral matrices.

This chapter is organized as follows: The notation used is introduced in the next section, while in Section 3.3 a small example of the problem considered is given. In Section 3.4 the interactive protocol considered and the problem we are facing are defined, and in Section 3.5 this problem is analyzed by assuming that the individual performs only multiplications and divisions modulo N (this is called an algebraic strategy). Some generalizations of the protocol of Section 3.4 are given in Section 3.6. In Section 3.7 we give some applications, and in Section 3.8 some open problems are stated.

3.2. Notation

The following notation is used throughout this chapter (some of the notation was already mentioned in Sections 1.5, 2.2 and 2.3). Boldface characters are used to denote vectors, and the RSA-modulus N used is created by the signature authority.

$\mathbf{ab}$	$(a_1b_1,\ldots,a_kb_k)$, if $\mathbf{a}=(a_1,\ldots,a_k)$ and $\mathbf{b}=(b_1,\ldots,b_k)$.
$\mathbf{a}\equiv\mathbf{b}\pmod m$	$m^{-1}(\mathbf{b}-\mathbf{a})\in\mathbb{Z}^k$; this is defined for $\mathbf{a},\mathbf{b}\in\mathbb{Q}^k$, $m,k\in\mathbb{N}$, $m>0$.
N	a composite, odd number.
$\mathbb{Z}_N^*$	the set $\{a \mid a\in\mathbb{N},\ 1\le a\le N,\ \gcd(a,N)=1\}$ of $\varphi(N)$ elements.
$\tilde{\mathbb{Q}}_N$	the ring $\{\frac{a}{d} \mid a,d\in\mathbb{Z},\ d>0,\ \gcd(d,\varphi(N))=1\}$.
$x^{1/d}\pmod N$	the d^{th} *RSA-root* of x (mod N): the unique solution $S\in\mathbb{Z}_N^*$ to $S^d\equiv x\pmod N$, for $x\in\mathbb{Z}_N^*$ and $d\in\mathbb{Z}$ with $\gcd(d,\varphi(N))=1$.
$\mathbf{x}^{\mathbf{a}}\pmod N$	the number $S\in\mathbb{Z}_N^*$ with $S\equiv x_1^{a_1}x_2^{a_2}\ldots x_k^{a_k}\pmod N$, for $\mathbf{x}=(x_1,\ldots,x_k)\in(\mathbb{Z}_N^*)^k$ and $\mathbf{a}=(a_1,\ldots,a_k)\in(\tilde{\mathbb{Q}}_N)^k$.
$[\mathbf{a}_1\ \ldots\ \mathbf{a}_l]$	the matrix with columns $\mathbf{a}_1,\ldots,\mathbf{a}_l$.
$\mathbf{x}^A\pmod N$	$(\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_l})\in(\mathbb{Z}_N^*)^l$, for $A=[\mathbf{a}_1\ldots\mathbf{a}_l]\in(\tilde{\mathbb{Q}}_N)^{k,l}$ and $\mathbf{x}\in(\mathbb{Z}_N^*)^k$; so $(\mathbf{x}^A)^B=\mathbf{x}^{AB}$.
$l(n)$	length of the binary representation of $n\in\mathbb{N}$; the length of a negative integer m, a rational number p/q $(q\neq1)$, a vector $\mathbf{c}$, and a matrix $A=(a_{i,j})$ are defined by: $l(m)=l(-m)+1$, $l(p/q)=l(p)+l(q)+1$, $l(\mathbf{c})=\sum_i(l(c_i)+1)$, and $l(A)=\sum_{i,j}(l(a_{i,j})+1)$, respectively.

length(A,B) $l(A)+l(B)$.

$a\in_R S$ denotes the random selection of an element (that will be called a) from S according to the uniform probability distribution; for any set S.

3.3. A small example of the problem under consideration

Let $a,b,c\in\tilde{\mathbb{Q}}_N$ be fixed and assume that the denominators of a,b, and c are coprime to $\varphi(N)$. We analyze the following Protocol 3.1 between $\mathcal{Z}$ and $\mathcal{A}$ (see Figure 3.2). In this protocol, $\mathcal{A}$ receives from $\mathcal{Z}$ the RSA-signature $u^{a+xb} \pmod N$, which $\mathcal{A}$ can verify. Note that $\mathcal{A}$ cannot compute this signature on a randomly chosen residue class u himself, because in general $a+xb\in \mathbb{Q}\setminus\mathbb{Z}$.

The next lemma states when it is feasible for $\mathcal{A}$ to compute u^c after the execution of this protocol.

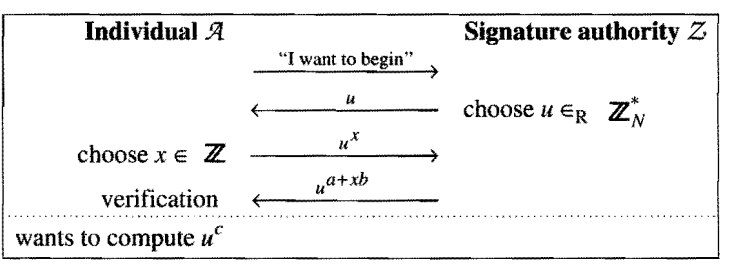


Fig. 3.2. Protocol 3.1.

Lemma 3.1. *$\mathcal{A}$ can choose x in Protocol* 3.1 *in such a way (and in polynomial time) that it is feasible for him to compute u^c after the execution of the protocol if and only if* $\gcd(1,a,b)|c$.

Note that $\gcd(1,a,b)$ is in general not 1, because a and b are rational numbers. This lemma can be proved by using Corollary 2.1 and the following two lemmas.

Lemma 3.2. *Let $a,b,c\in\mathbb{Q}$, $c\neq 0$. Then there exists an integer λ such that* $\gcd(a+\lambda b,c) = \gcd(a,b,c)$, *and this λ can be computed in polynomial (in length(a,b,c)) time.*

Proof. Define $\bar{a}=a/\gcd(a,b,c),\bar{b}=b/\gcd(a,b,c),\bar{c}=c/\gcd(a,b,c)$. Thus $\bar{a},\bar{b},\bar{c}$ are integers with $\gcd(\bar{a},\bar{b},\bar{c})=1$. It suffices to show that we can compute in polynomial time a $\lambda\in\mathbb{N}$ that satisfies $\gcd(\bar{a}+\lambda\bar{b},\bar{c})=1$.

For each prime number p and each $a\in\mathbb{Z}, a\neq 0$, let $\mathrm{ord}_p(a)$ be the integer such that $a\cdot p^{-\mathrm{ord}_p(a)}$ is an integer not divisible by p. Take

$$\lambda = \prod_{p|\bar{c}, p\nmid \bar{a}} p^{\mathrm{ord}_p(\bar{c})}.$$

Let p be a prime dividing $\bar{c}$. If $p|\bar{a}$, then $p\nmid \bar{b}$ (by $\gcd(\bar{a},\bar{b},\bar{c})=1$) and $p\nmid \lambda$ (by definition of λ), hence $p\nmid (\bar{a}+\lambda\bar{b})$. If $p\nmid \bar{a}$, then $p|\lambda$ (by definition of λ and $\mathrm{ord}_p(\bar{c})\geq 1$) and thus also $p\nmid (\bar{a}+\lambda\bar{b})$. We conclude that no prime divides both $\bar{c}$ and $(\bar{a}+\lambda\bar{b})$; therefore $\gcd(\bar{a}+\lambda\bar{b},\bar{c})=1$.
Define the sequence $c_0 := |\bar{c}|$ and $c_{i+1} := c_i/\gcd(\bar{a},c_i)$ for $i=0,1,2,\ldots$. Let i_0 be the smallest integer such that $\gcd(\bar{a},c_{i_0})=1$. It is easy to see that $c_{i_0}=\lambda$ and that $i_0 \leq l(\bar{c})$; thus λ can be computed in polynomial time. ❑

Lemma 3.3. *Let $a,b,c\in\mathbb{Q}$. Then there are $x,y,z\in\mathbb{Z}$ such that $c=(a+xb)y+z$ if and only if $\gcd(1,a,b)|c$. Further, if such $x,y,z\in\mathbb{Z}$ exist, then they can be computed in polynomial (in length(a,b,c)) time.*

Proof. Note that $a,b,1$ are integral multiples of $\gcd(1,a,b)$. Hence, if there exist $x,y,z\in\mathbb{Z}$ such that $c=(a+xb)y+z$, then c is also an integral multiple of $\gcd(1,a,b)$. Hence $\gcd(1,a,b)|c$.
On the other hand, assume that $\gcd(1,a,b)|c$. By Lemma 3.2 we can compute in polynomial time an $x\in\mathbb{Z}$ such that $\gcd(a+xb,1)=\gcd(1,a,b)$. Further we can compute in polynomial time $y,z\in\mathbb{Z}$ with $c=(a+xb)y+z$ (e.g., let $d\in\mathbb{N}$ such that $da,db,dc\in\mathbb{N}$, and use $\gcd(a+xb,1)|c$ and Euclid's algorithm to compute $y,z\in\mathbb{Z}$ with $dc=(da+xdb)y+dz$). This proves Lemma 3.3. ❑

Proof of Lemma 3.1.
(i) Suppose that $\gcd(1,a,b)|c$. According to Lemma 3.3, $\mathcal{A}$ can compute in polynomial time numbers $x,y,z\in\mathbb{Z}$ such that $c=(a+xb)y+z$. $\mathcal{A}$ will use the obtained number x during the execution of Protocol 3.1. Afterwards, $\mathcal{A}$ can compute u^c from $\{N,a,b,c,u,u^{a+xb}\}$ as follows:

$$(u^{a+xb})^y \cdot u^z \equiv u^c \pmod{N}.$$

(ii) Suppose that $\mathcal{A}$ can choose x in Protocol 3.1 in such a way that it is feasible for him to compute u^c after the execution of the protocol. Corollary 2.1 states that computing u^c from $\{N,a,b,c,u,u^{a+xb}\}$ for uniformly chosen $u\in\mathbb{Z}_N^*$ is feasible for $\mathcal{A}$ if and only if $c\in\mathbb{Z}\{1,a+xb\}$. That is, if and only if there are $y,z\in\mathbb{Z}$ such that $c=(a+xb)y+z$. ❑

We generalize Protocol 3.1 to Protocol 3.2 (see Figure 3.3), in which $\mathcal{A}$ initially chooses some residue class, but we will prove that doing so does not influence the

feasibility of computing the signature u^c after the execution of the protocol.

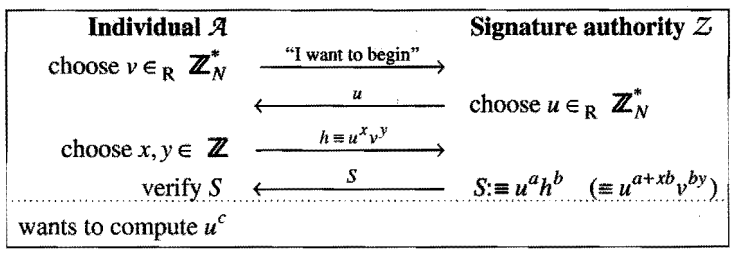


Fig. 3.3. Protocol 3.2.

How must $\mathcal{A}$ choose v,x,y so that it is feasible for him to compute u^c from $\{u, v, u^{a+xb}v^{by}, a, b, c, x, y\}$? According to Corollary 2.1 this computation is feasible if and only if there is an integral solution z_1, z_2, z_3 to

$$\begin{cases} z_1 + z_3(a + xb) = c, \\ z_2 + z_3 by = 0. \end{cases}$$

According to Lemma 3.3, a necessary condition for the first equation is that $\gcd(1,a,b)|c$. But the number z_3 hereby obtained does not need to be a solution of the second equation. If $y=0$, then the z_3 obtained is also a solution of the second equation. Hence a *necessary* condition for the simultaneous solvability of the two equations is that $\gcd(1,a,b)|c$; and if $y=0$, then this condition is also *sufficient*. Thus the best strategy for $\mathcal{A}$ is to choose $y=0$ and to take x according to Lemma 3.1; hence this algebraic strategy "works" if and only if $\gcd(1,a,b)|c$.

In the two protocols above, we restricted the behaviour of $\mathcal{A}$: he has to send u^x or $u^x v^y$ to $\mathcal{Z}$. A more general protocol is Protocol 3.3 of Figure 3.4, but this protocol is difficult to analyze. In the next sections we generalize Protocol 3.3, but we are only able to analyze this new protocol if we restrict the behaviour of $\mathcal{A}$, like in Protocol 3.1 and 3.2.

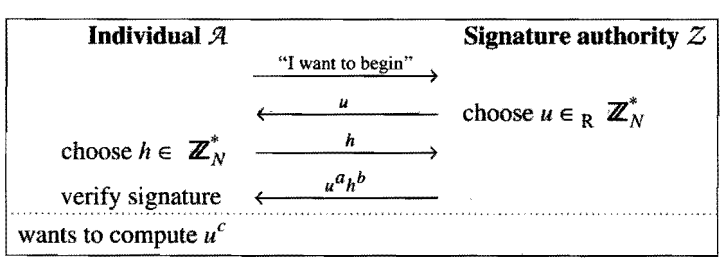


Fig. 3.4. Protocol 3.3.

3.4. The protocol and problem under consideration

In this section we consider the following interactive Protocol 3.4 (see Figure 3.5), which is more general than Protocol 3.3. The signature authority $\mathcal{Z}$ has created an RSA-modulus N, and issues RSA-signatures that will be products of rational powers of residue classes modulo N. Let $M = \{A,B,C,D\}$ be a set of fixed rational matrices $A \in (\tilde{\mathbb{Q}}_N)^{k,l}$, $B \in (\tilde{\mathbb{Q}}_N)^{m,l}$, $C \in (\tilde{\mathbb{Q}}_N)^{k,n}$, $D \in (\tilde{\mathbb{Q}}_N)^{m,n}$. In Protocol 3.4, an individual $\mathcal{A}$ requests $\mathcal{Z}$ to create the RSA-signature (in fact it consists of l RSA-signatures)

$$\mathbf{s}_1 \equiv \mathbf{u}^A \mathbf{h}_1^B \pmod{N},$$

where $\mathbf{u} \in (\mathbb{Z}_N^*)^k$ is chosen by $\mathcal{Z}$, and $\mathbf{h}_1 \in (\mathbb{Z}_N^*)^m$ is chosen by $\mathcal{A}$ ($\mathbf{h}_1$ may depend on N,$M = \{A,B,C,D\}$ and $\mathbf{u}$). But actually, $\mathcal{A}$ wants to have the RSA-signature

$$\mathbf{s}_2 \equiv \mathbf{u}^C \mathbf{h}_2^D \pmod{N},$$

for some $\mathbf{h}_2 \in (\mathbb{Z}_N^*)^m$ and $\mathbf{s}_2 \in (\mathbb{Z}_N^*)^n$. Therefore he wants to choose $\mathbf{h}_1$ in such a way that after the execution of Protocol 3.4, he can compute from $\{N,A,B,C,D,\mathbf{u},\mathbf{h}_1, \mathbf{s}_1 \equiv \mathbf{u}^A \mathbf{h}_1^B\}$ a pair $\{\mathbf{s}_2,\mathbf{h}_2\}$ satisfying $\mathbf{s}_2 \equiv \mathbf{u}^C \mathbf{h}_2^D$. This way of choosing $\mathbf{h}_1$ (which may depend on N,$M = \{A,B,C,D\}$ and $\mathbf{u}$) in order to be able to compute a pair $\{\mathbf{s}_2,\mathbf{h}_2\}$, is called an M-*strategy*. We assume that the running time of an M-strategy is not a stochastic variable, but that it is determined by N and M. This implies that the M-strategy will not with 100% certainty output a pair $\{\mathbf{s}_2,\mathbf{h}_2\}$ satisfying $\mathbf{s}_2 \equiv \mathbf{u}^C \mathbf{h}_2^D$. Hence the problem that we wish to consider is the following:

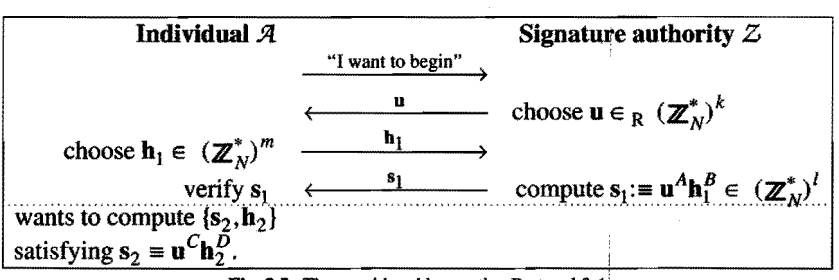


Fig. 3.5. The considered interactive Protocol 3.4.

Problem 3.1. *For which system of matrices* $M = \{A,B,C,D\}$ *does there exist a polynomial (in length(N,A,B,C,D))-time* M*-strategy that outputs with probability* $\geq \frac{1}{2}$, *say, a pair* $\{\mathbf{s}_2,\mathbf{h}_2\}$ *satisfying* $\mathbf{s}_2 \equiv \mathbf{u}^C \mathbf{h}_2^D$?

This problem was solved in Chapter 2 for the special case in which B and D are the all-zero matrices, i.e., for the noninteractive case.

If there are no restrictions on $\mathbf{h}_2$ (e.g., $\mathbf{h}_2$ must be an element from a special subset

of $(\mathbb{Z}_N^*)^m$), then we can restrict ourselves in Protocol 3.4 to the case that $D = [0]$ (the all-zero matrix), according to the next lemma.

Lemma 3.4. *Let* $C \in (\tilde{\mathbb{Q}}_N)^{k,n}$, $D \in (\tilde{\mathbb{Q}}_N)^{m,n}$ and $\mathbf{u} \in (\mathbb{Z}_N^*)^m$. *Then there exists a matrix* $\tilde{C} \in (\tilde{\mathbb{Q}}_N)^{k,n}$ (*which is computable in polynomial time from C and D*), *such that computing a pair* $(\mathbf{s},\mathbf{h}) \in (\mathbb{Z}_N^*)^n \times (\mathbb{Z}_N^*)^m$ *satisfying* $\mathbf{s} \equiv \mathbf{u}^C\mathbf{h}^D$ *is polynomial-time equivalent to computing* $\mathbf{u}^{\tilde{C}}$.

Proof. We can reformulate the identity $\mathbf{s} \equiv \mathbf{u}^C\mathbf{h}^D$ as

$$\mathbf{u}^C \equiv (\mathbf{s},\mathbf{h})^{\begin{bmatrix} I \\ -D \end{bmatrix}},$$

where the first n coordinates of vector $(\mathbf{s},\mathbf{h})$ are those of $\mathbf{s}$, the last m coordinates are those of $\mathbf{h}$, the first n rows of $\begin{bmatrix} I \\ -D \end{bmatrix}$ are those of the identity matrix I, and the last m rows are those of $-D$. According to [KaBa79], we can find in polynomial time unimodular matrices P,Q and a matrix $\begin{bmatrix} G \\ 0 \end{bmatrix}$ in Smith normal form, such that $P\begin{bmatrix} G \\ 0 \end{bmatrix} = \begin{bmatrix} I \\ -D \end{bmatrix}Q$. Because $\begin{bmatrix} I \\ -D \end{bmatrix}$ has full column rank, G is invertible. Define $\tilde{C} = CQG^{-1}$. Then from $(\mathbf{s},\mathbf{h})$ satisfying $\mathbf{s} \equiv \mathbf{u}^C\mathbf{h}^D$ we can compute $\mathbf{u}^{\tilde{C}}$ in polynomial time because

$$\mathbf{u}^{\tilde{C}} = \mathbf{u}^{CQG^{-1}} = (\mathbf{s},\mathbf{h})^{\begin{bmatrix} I \\ -D \end{bmatrix}QG^{-1}} \equiv (\mathbf{s},\mathbf{h})^{P\begin{bmatrix} G \\ 0 \end{bmatrix}G^{-1}} = (\mathbf{s},\mathbf{h})^{P\begin{bmatrix} I \\ 0 \end{bmatrix}}$$
$$= \text{first } n \text{ coordinates of } (\mathbf{s},\mathbf{h})^P.$$

If, on the other hand, $\mathbf{u}^{\tilde{C}}$ is given, then we obtain the pair $\{\mathbf{s},\mathbf{h}\}$ satisfying $\mathbf{s} \equiv \mathbf{u}^C\mathbf{h}^D$ by first computing the vector $\tilde{\mathbf{s}} = (\mathbf{u}^{\tilde{C}},1,\ldots,1)$ of length $n+m$ and then by defining $\mathbf{s},\mathbf{h}$ by $(\mathbf{s},\mathbf{h}) := \tilde{\mathbf{s}}^{P^{-1}}$. ❑

3.5. Algebraic strategies

As shown in the previous section, we may restrict ourselves to the case in which M = $\{A,B,C,[0]\}$. We have no idea how to decide if there exists a polynomial-time M-strategy for given $N,A,B,C,\mathbf{u}$. Therefore we consider only M-strategies belonging to a special class, the so-called *algebraic* M-*strategies*. In an algebraic strategy, $\mathcal{A}$ only applies to u multiplications and divisions mod N in order to compute $\mathbf{h}_1$.

Let $A \in (\tilde{\mathbb{Q}}_N)^{k,l}$, $B \in (\tilde{\mathbb{Q}}_N)^{m,l}$, $C \in (\tilde{\mathbb{Q}}_N)^{k,n}$ be fixed rational matrices. $\mathcal{A}$ is assumed to follow an algebraic M-strategy; hence, in Protocol 3.4, $\mathbf{h}_1$ must consist of products of integral powers of the entries of $\mathbf{u}$, i.e., $\mathbf{h} \equiv \mathbf{u}^X$, for some $X \in \mathbb{Z}^{k,m}$. Thus instead of analyzing the general Protocol 3.4 in this section, we will analyze Protocol 3.5 (see Figure 3.6, in which we write $\mathbf{h}$ in stead of $\mathbf{h}_1$).

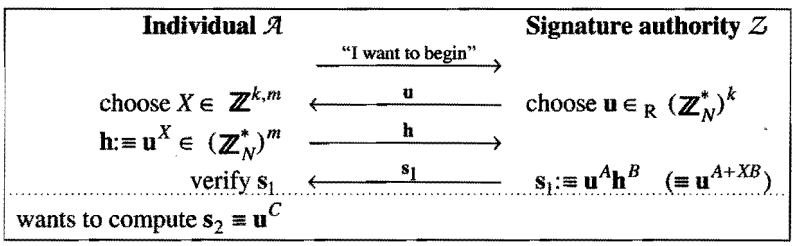


Fig. 3.6. Protocol 3.5, which is equivalent to Protocol 3.4 if $\mathcal{A}$ follows an algebraic M-strategy.

We now also assume that it is computationally infeasible for $\mathcal{A}$ to compute RSA-roots modulo N, since otherwise he could forge all signatures. Under this assumption, Corollary 2.1 implies the following for Protocol 3.5:

Corollary 3.1. *Let A,B,C be fixed rational matrices. Then the following two statements are equivalent for an individual:*

(i) *It is feasible to compute integral matrices X,Y,Z such that $C = (A+XB)Y + Z$.*

(ii) *There is a feasible algebraic* M*-strategy to compute* $\mathbf{h}$ *from* $\{N,A,B,C,\mathbf{u}\}$ *and* $\mathbf{u}^C$ *from* $\{N,A,B,C,\mathbf{u},\mathbf{u}^A\mathbf{h}^B\}$.

According to this result, we are interested in the following problem:

Problem 3.2. *Let* $A \in (\tilde{\mathbb{Q}}_N)^{k,l}$, $B \in (\tilde{\mathbb{Q}}_N)^{m,l}$, $C \in (\tilde{\mathbb{Q}}_N)^{k,n}$ *be rational matrices. Find a polynomial (in length(A,B,C))-time algorithm that decides whether the equation*

$$C = (A+XB)Y + Z$$

is solvable in integral matrices $X \in \mathbb{Z}^{k,m}$, $Y \in \mathbb{Z}^{l,n}$, $Z \in \mathbb{Z}^{k,n}$, *and if so, find a solution X,Y,Z.*

We have not been able to solve Problem 3.2 in full generality. In [Evert90] it is proven that there exists such a polynomial-time algorithm for Problem 3.2 in the case that n=1, but this proof is not included in this chapter. In Section 3.3, we solved Problem 3.2 in the special case that k=l=m=n=1.

3.6. Generalizations

In Protocol 3.4 (see Figure 3.5) the system of matrices used is M = $\{A,B,C,D\}$, so the individual will receive one type of signature (s_1), and wants to compute a second type (s_2).

We now assume that there are $(t+1)$ types of signatures, so $\mathcal{Z}$ creates a public system of matrices $M = \{A_1,B_1,\ldots,A_{t+1},B_{t+1}\}$, where the matrices (A_i,B_i) are used for the i^{th} type. But $\mathcal{Z}$ will only issue signatures of type $1,\ldots,t$ to $\mathcal{A}$, who will try to compute a signature of type $t+1$. Thus the protocol we want to consider is the serial Protocol 3.6 (see Figure 3.7), in which we assume that $\mathcal{Z}$ uses the same $\mathbf{u}$ in every signature, and that the individual chooses $\mathbf{h}_1,\ldots,\mathbf{h}_t$ (where $\mathbf{h}_i$ may depend on $M,\mathbf{u}$ and $\mathbf{s}_1,\ldots,\mathbf{s}_{i-1}$) and receives the signatures

$$\mathbf{s}_i \equiv \mathbf{u}^{A_i}\mathbf{h}_i^{B_i} \quad (i=1,\ldots,t).$$

$\mathcal{A}$ will not receive signatures of type $(t+1)$, so he tries to choose $\{\mathbf{h}_1,\ldots,\mathbf{h}_t\}$ in such a way that after receiving $\{\mathbf{s}_1,\ldots,\mathbf{s}_t\}$, he is able to compute a pair $(\mathbf{s}_{t+1},\mathbf{h}_{t+1})$ such that

$$\mathbf{s}_{t+1} \equiv \mathbf{u}^{A_{t+1}}\mathbf{h}_{t+1}^{B_{t+1}}.$$

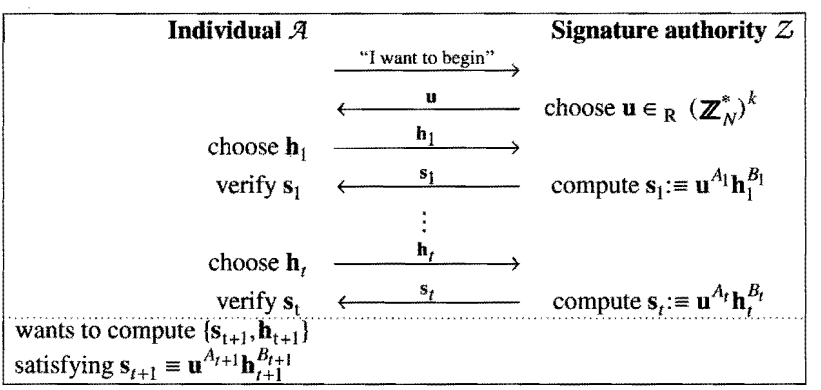


Fig. 3.7. The serial signature-issuing Protocol 3.6, in which $\mathcal{A}$ receives the signatures $s_1,\ldots,s_t$.

If we assume that $\mathcal{A}$ uses an algebraic M-strategy, then we can prove that it suffices to consider algebraic strategies on protocols with $s=1$; that is, we can reduce Protocol 3.6 in polynomial time to Protocol 3.4 as follows:

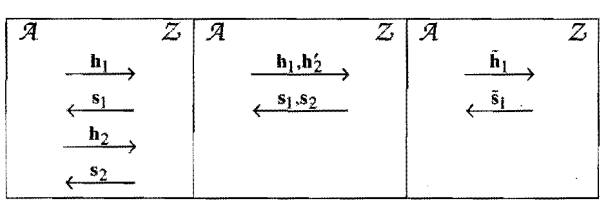


Fig. 3.8. How to modify Protocol 3.6 into Protocol 3.4.

Moves 3 up to 6 of Protocol 3.6 are shown in the left-hand side of Figure 3.8. Let d be the smallest positive integer such that dA_1 and dB_1 are integral matrices (so d can be the lcm of all the denominators of A_1 and B_1). Hence $\mathbf{s}_1^d$ can be computed by $\mathcal{A}$

without knowing $\mathbf{s}_1$, because $\mathbf{s}_1^d \equiv \mathbf{u}^{dA_1}\mathbf{h}_1^{dB_1}$, so the used exponents are integral. In order to create $\mathbf{h}_2$, $\mathcal{A}$ might use $\mathbf{s}_1$. But $\mathcal{A}$ only applies multiplications and divisions on $\mathbf{s}_1$, so $\mathcal{A}$ is able to compute $\mathbf{h}_2' := \mathbf{h}_2^d$ *without* knowing $\mathbf{s}_1$ ($\mathcal{A}$ will only use $\mathbf{s}_1^d$, which he could compute without knowing $\mathbf{s}_1$). By defining the new matrix $B_2' := \frac{1}{d}B_2$, we have that $\mathbf{h}_2'^{B_2'} \equiv \mathbf{h}_2^{B_2}$, so $\mathcal{A}$ does not need to know $\mathbf{s}_1$ in order to compute $\mathbf{h}_2^{B_2}$ (by using matrix B_2'). The possibility that $\mathcal{A}$ can compute $\mathbf{s}_{t+1}$ at the end of the protocol remains the same if we carry out the first four moves in parallel instead of serially (see Figure 3.8, middle). By defining $\tilde{\mathbf{s}}_1 := (\mathbf{s}_1, \mathbf{s}_2)$, $\tilde{\mathbf{h}}_1 := (\mathbf{h}_1, \mathbf{h}_2')$, $\tilde{A}_1 := [A_1\ A_2]$, $\tilde{B}_1 := [B_1\ B_2']^T$, we have that $\tilde{\mathbf{s}}_1 \equiv \mathbf{u}^{\tilde{A}_1}\mathbf{h}_1^{\tilde{B}_1}$; thus we can combine the first four moves into two (see Figure 3.8, right-hand side). In this way we obtain a protocol with 2 moves less. By repeating this argument, we only have to analyze a protocol with 2+2 moves, i.e., Protocol 3.4.

3.7. Some open problems

In the analysis of the protocols we assumed that the individual uses an algebraic strategy, i.e., that he uses only multiplications and divisions modulo N (and also additions and subtractions). The problem whether this assumption is necessary remains open. So the question is whether it is possible to analyze protocols like Protocol 3.3, in which the individual can use any strategy.

Another open problem (as mentioned in Section 3.5) is to find a polynomial time algorithm to verify whether the matrix equation $C = (A+XB)Y + Z$ is solvable, and if so, to find a solution.

4

Group Signatures[‡]

4.1. Introduction

If a person wants to prove that he belongs to a certain group, then there are several protocols known that can be used.

In [CE86] *credential systems* are constructed: if a person belongs to a certain group, an authority will give him a *credential*. If a party wants to verify this credential, the group member transforms his credential to a form that does not reveal his identity. The privacy of the person is protected unconditionally: even with unlimited computing power two parties cannot link credentials together.

In [OOK90] two *membership authentication schemes* are proposed, in which the same secret key is given to each group member. These schemes give an efficient construction for hierarchical situations.

In [SKI90] another membership authentication scheme is proposed. However, this scheme cannot be used for signing messages, and each group member can create other secret keys from his own secret key.

In this chapter we present a new type of signature, which we call a *group signature*: it is a signature scheme for a group of persons that has the following three properties:

(i) only members of the group can sign messages;

(ii) the recipient of the signature can verify that it is a valid signature of that group, but cannot discover which member of the group created it;

(iii) in case of dispute later on, the signature can be "opened" (with the help of the group members or of a trusted authority) to reveal the identity of the signer.

[‡] This chapter is based on the paper "Group Signatures" by David Chaum and Eugène van Heyst, which appeared in *Advances in Cryptology-EUROCRYPT '91*, D.W. Davies ed., LNCS 547, Springer Verlag, pp. 257-265.

The use of group signatures will be illustrated with the following two examples: A company has several computers, each connected to the local network. Each department of that company has its own printer (also connected to the network) and only members of that department are allowed to use their department's printer. Before printing, therefore, the printer must be convinced that the user is working in that department. At the same time, the user wants privacy: the user's name may not be revealed. If, however, someone discovers at the end of the day that a printer has been used too often, the director must be able to discover who misused that printer, to send him a bill.

Alternatively, suppose that any doctor can give a signed note to a patient for an insurance claim. The insurance company can verify the validity of a signature (i.e., that is was signed by a doctor), but cannot discover which doctor signed it. If somebody steals a doctor's key, the identity of the doctor can be revealed in order to trace the thief.

Such a group signature scheme is not just a signature scheme with one public and several secret keys. The scheme must also have the property that it is infeasible for conspiring group members to create a new secret key out of their own secret keys. Otherwise they could sign messages with this new secret key and their identity can never be revealed (so it violates property (iii)).

The three constructions of membership authentication schemes in the papers mentioned above cannot be used to create group signatures for several reasons: [CE86] does not have property (iii), in [OOK90] all group members have the same secret key, and in [SKI90] each group member can construct other secret keys from his own secret key. In this chapter four different constructions of group signature schemes are presented. Persons can be members of more than one group, but in each construction only one group of persons will be considered (so the hierarchical situation will not be treated here). These four constructions can be found in Sections 4.2 up to 4.5. These four constructions are compared (see Table 4.1) on the following five items.

Complexity theoretic assumption. In the first construction, every public key system can be used while the other constructions are based on either Assumption 1.2 or 1.3 (see Sections 1.4.1 and 1.5). In all constructions the privacy of the signer is protected computationally. Care must of course be taken in the selection of the exponents used, in order to protect the anonymity of the signer (see Section 4.6). Not even a group member (other than the signer) can determine who created a certain signature.

Trusted authority. Let $\mathcal{Z}$ be an authority, trusted by the group. $\mathcal{Z}$ sets the group signature scheme, except for the last construction: in this case a group signature scheme can be created from a "normal" setup of a scheme based on the discrete logarithm, without using a trusted authority. Except for the first construction, $\mathcal{Z}$ is no longer needed after the setup.

Creation of the group. In the first two constructions, the group of persons is determined during the setup of the scheme. In the last two constructions, it is assumed that there is already a "normal" setup of the RSA-scheme or of a scheme based on the discrete logarithm. If in one of these last two constructions of group signatures someone wants to sign a message without revealing his identity, he creates *at that moment* some "group" of persons. He can, for instance, do this by picking these persons from a Trusted Public Directory of public keys, and he proves that he belongs to that group. In case of dispute later on, the other "group members" are able to deny that signature (it is not necessary that they know the "group"), while the signer is not able to deny his signature.

Type of signature. In the last three constructions, the signatures created by the group members are undeniable signatures. Therefore in these constructions we have a confirmation protocol (in which the signer can convince the recipient that the signature on the message is correct) and a disavowal protocol (in which the other group members can convince the recipient that they did not create that signature).

It is possible to create digital (i.e., not undeniable) group signatures in these last three constructions (by using the same protocols). This can be realized as in [FFS88], by doing the iterations of the confirmation protocol in parallel and letting the recipient choose the challenge vector not randomly, but as the outcome of a collision-free one-way-function on the received numbers. Because this parallel execution of the protocol is no longer zero-knowledge, the signature together with all the numbers sent during the confirmation protocol will be a digital signature. Still to be proven is that this parallel protocol gives "no useful knowledge" to the recipient.

Costs. In all four constructions the length of the public key (i.e., the number of bits in the group's public key) is linear in the number of group members. The numbers of bits and the number of computations are only compared in the case of the confirmation protocol, because in the disavowal protocols, both these numbers are independent of the number of group members.

Group signature implementation	Complexity theoretic assumption	Trusted authority needed for	Creation of the group	Type of signature	Length of the public key of the group	Number of computations during conf. pr.	Number of bits transmitted during conf. pr.
1	Any	Setup+opening	In advance	Any type	Linear	Independent	Independent
2	1.3	Setup	In advance	Undeniable	Linear	Linear	Independent
3	1.3	Setup	Afterwards	Undeniable	Linear	Linear	Independent
4	1.2	--	Afterwards	Undeniable	Linear	Linear	Linear

Table 4.1. Comparison of the four group signature constructions presented in this chapter. "Independent, linear" means that the number is independent respectively linear in the number of group members.

4.2. First construction of group signatures

The trusted authority Z chooses a public key system, gives each person a list of secret keys (these lists are all disjoint), and publishes the complete list of corresponding public keys (e.g., sorted on name) in a Trusted Public Directory (see Figure 4.1). Thus this TPD is the public key of the group, which itself consists of some public keys.

Each person can sign a message with a secret key from his own list, and also issues to the recipient the corresponding public key. The recipient can verify this signature with the public key and can verify that this public key is from the public list. Each secret key will be used only once, otherwise signatures created with that key are linked. Z knows all the lists of secret keys, so that in case a recipient asks Z to open a signature, he knows who created the disputed signature. Hence Z is both needed for setup and for "opening" a signature.

If each group member gets from the trusted authority the same number of secret keys, then the length of the public key of this group signature construction (i.e., the number of public keys of the Trusted Public Directory) is linear in the number of persons; but the number of messages a person can sign is fixed (because each secret key will be used only once). The number of bits to be transmitted and the number of computations needed to verify a signature are independent of the number of persons (leaving aside the look-up in the TPD).

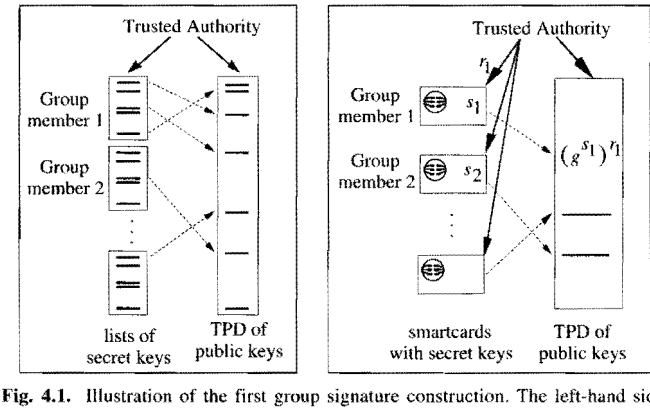


Fig. 4.1. Illustration of the first group signature construction. The left-hand side shows the basic idea and the right-hand side shows the blinded public key construction.

A problem with this construction is that Z knows all the secret keys of all the group members and can therefore fake the signatures of the group members. This can be prevented by using *blinded public keys* (see Figure 4.1). Let the public key system used be a scheme based on the discrete logarithm, for instance the ElGamal scheme [ElG85] or the undeniable signature scheme [CvA89]. Let p be a prime according to

Assumption 1.1 and let g be a generator of the multiplicative group $\mathbb{Z}_p^*=\{1,2,\ldots,p-1\}$. Group member i creates his own secret key s_i and gives the public key $g^{s_i} \pmod p$ to Z. Thus Z has a list of all these public keys together with the group member's name. Each week, Z gives each group member i a randomly chosen number $r_i \in \{1,\ldots,p-1\}$ and publishes the list of all the blinded public keys $(g^{s_i})^{r_i}$ in random order. During this week group member i will use $s_i r_i \pmod{p-1}$ as secret key for signing messages. Opening a group signature is done by contacting Z, just as in the previous scheme.

The advantages of this modification are that Z cannot fake signatures, and that each group member needs only one "really secret key" (for instance in a smart card), which can be blinded in order to make other secret keys. Only the particular week's signatures can be linked, so that each group member need have only a few secret keys in his smart card to prevent this linking. Even if an r_i is accidentally disclosed, no more information about the secret key s_i is revealed.

In another modification, no trusted authority is needed: each user untraceably sends one (or more) public keys to a public list, which will be the public key of the group. But only group members must be able to send public keys to that list.

4.3. Second construction of group signatures

Z chooses two different large primes p, q together with a one-way-function f of which the outcome may be assumed to be coprime to $N=pq$. For security it is essential that the users do not know $\varphi(N)$ (so N cannot be prime). Z issues to group member i a secret key s_i, which is a large prime coprime to $\varphi(N)$ and randomly chosen from the public set $\Phi=\{\lceil\sqrt{N}\rceil, \lceil\sqrt{N}\rceil+1,\ldots, 2\lceil\sqrt{N}\rceil-2\}$. Z publishes N, $v:=\prod s_i$ and f, which will be the public key of the group. If group member i wants to sign message n, his signature will be

$$(f(n))^{s_i} \bmod N,$$

and he has to convince the recipient that s_i divides v and that s_i is an element in Φ, without revealing anything more about s_i. This can be done by using the confirmation protocol of Section 4.3.1. If in case of dispute later on, the recipient wants to know the member who signed the message, he can perform a disavowal protocol with each group member, without the help of Z (see Section 4.3.2). The signer cannot perform this disavowal protocol successfully, and thus his identity will be revealed to the recipient. To prove the security of this group signature construction we need Assumption 1.3.

4.3.1. Confirmation protocol

We first consider the following Instance 4.1 (Figure 4.2), in which $\mathcal{P}$ wants to convince $\mathcal{V}$ that he knows a certain discrete logarithm modulo a composite number, and that this discrete logarithm lies in some interval. $\mathcal{P}$ does not need to know the factorization of the modulus.

secret of $\mathcal{P}$	:	c
public	:	N, x, y, Ω; $x, y \in \mathbb{Z}_N^*$, $\Omega = \{\alpha, \ldots, \alpha+\beta\} \subset \mathbb{N}$
prove to $\mathcal{V}$	:	$x^c \equiv y \pmod N \wedge c \in \Omega$

Fig. 4.2. Instance 4.1.

This instance is solved by [BCDvdG87] by using Protocol 4.1†, which uses computationally secure blobs $\mathcal{B}$ (we will be loose in writing blobs: we write $\mathcal{B}(z)$ in stead of $\mathcal{B}(z,t)$ for some blinding factor t). This protocol has to be iterated several times.

Protocol 4.1. (for Instance 4.1)

Step 1. $\mathcal{P}$ *chooses* $r \in \{0,\ldots,\beta\}$. *He computes blobs on* $z_1 \equiv x^r \pmod N$ *and* $z_2 \equiv x^{r-\beta-1} \pmod N$, *and sends the unordered pair* $\{\mathcal{B}(z_1), \mathcal{B}(z_2)\}$ *to* $\mathcal{V}$.

Step 2. $\mathcal{V}$ *chooses randomly* $b \in \{0,1\}$ *and sends it to* $\mathcal{P}$.

Step 3. $\mathcal{P}$ *sends* $\mathcal{V}$ *in case*

b=0: *the number* r, *and the opening of both blobs.*

b=1: *the number* $(c+r)$ *or* $(c+r-\beta-1)$, *whichever is in the set* Ω (*this number will be called* $\tilde{r}$), *and the opening of respectively the blob on* z_1 *or* z_2 (*this number will be called* $\tilde{z}$).

Step 4. $\mathcal{V}$ *verifies in case*

b=0: *that* $r \in \{0,\ldots,\beta\}$ *and that the blobs contain* x^r *and* $x^{r-\beta-1}$ *in some order.*

b=1: *that* $\tilde{r} \in \Omega$, *that one of the blobs contains* $\tilde{z}$ *and that* $\tilde{z}$ *satisfies* $x^{\tilde{r}} \equiv \tilde{z} y$.

But the protocol of [BCDvdG87] does not really solve Instance 4.1: note that $r \in \{0,\ldots,\beta\}$ and that $(c+r)$ or $(c+r-1-\beta)$ is an element from $\{\alpha,\ldots,\alpha+\beta\}$. So after execution of this protocol, $\mathcal{V}$ will be convinced that $c \in \tilde{\Omega} = \{\alpha-\beta,\ldots,\alpha+2\beta\}$. There is a discrepancy between the requirements for c, namely $c \in \{\alpha,\ldots,\alpha+\beta\}$,

† Hence, by using $\Omega=\{1,\ldots,N\}$, one can prove that he knows a discrete logarithm modulo N, without knowing $\varphi(N)$.
The original protocol of [BCDvdG87] contains a mistake, which is changed in our Protocol 4.1.

and what is actually proven, that $c \in \{\alpha-\beta,\ldots,\alpha+2\beta\}$. In their paper they prove that only if $c \in \{\alpha,\ldots,\alpha+\beta\}$, their protocol reveals no (other) information: Because if $c \in \Omega$, then the distribution of $\tilde{r}$ in Protocol 4.1 is uniform over Ω and this distribution is thus independent of c. Therefore this protocol can be simulated by $\mathcal{V}$.

With this protocol we are able to create a confirmation protocol for the group signature scheme, so let $\mathcal{P}$ be a fixed group member who wants to convince the recipient $\mathcal{V}$ that S is his correct group signature on the message n. Thus the following instance (in which we write m instead of $f(n)$) must be solved:

secret of $\mathcal{P}$	:	s
public	:	$N, v, m, S, \Phi;\quad m, S \in \mathbb{Z}_N^*$
prove to $\mathcal{V}$	:	$S \equiv m^s \pmod N \wedge s \in \Phi \wedge s \mid v$

Fig. 4.3. Instance 4.2.

Protocol 4.2. (for Instance 4.2)

Step 1. *Convince the recipient of having a number s such that* $S \equiv m^s \pmod N$ *and that* $s \in \Phi$ *with Protocol* 4.1, *iterated k times (use the substitution* $\Omega = \Phi$, $x = m$, $y = S$ *and* $c = s$*).*

Step 2. *Convince the recipient that s is a divisor of v, by using the following protocol:*

Prover $\mathcal{P}$		**Verifier** $\mathcal{V}$
	$\xleftarrow{a \equiv S^r}$	choose $r \in_R \{1,\ldots,N\}$
$b :\equiv a^{v/s}$	$\xrightarrow{\mathcal{B}(b)}$	
verify a	$\xleftarrow{r}$	
	$\xrightarrow{\text{open blob}}$	verify opening and that $b \equiv m^{vr}$

We prove that Protocol 4.2 is complete, sound, and zero-knowledge. Note that for all S the probability distributions of $S^r \pmod N$ where $r \in \{1,\ldots,\varphi(N)\}$ or $r \in \{1,\ldots,N\}$ are polynomially indistinguishable ([CEvdG87]). Step 1 of Protocol 4.2 has already been proven to be sound, complete, and zero-knowledge (Protocol 4.1). Step 2 is trivially complete and zero-knowledge (recall that the blobs $\mathcal{B}$ are computationally secure).

So we now only have to prove that Step 2 is sound. In Step 1 it is proven that $\mathcal{P}$ knows an integer s such that $S \equiv m^s$. When is it feasible for him to compute $b \equiv x^v$ from $\{s, v, m, a \equiv x^s\}$, where $x \equiv m^r$? Under the assumption that it is infeasible to compute RSA-roots (so here N is not a prime), it follows from Corollary 2.2.v. that computing x^v from $\{s, v, x^s\}$ is feasible if and only if $s \mid v$. These two problems are equivalent, because we can construct generators of x as follows: randomly choose an

odd number $r' \in \{1,\ldots,N\}$. This number will with high probability be coprime to $\varphi(N)$. By defining $m' \equiv (x^s)^{r'}$, we have a number that generates a group containing x, because sr' is coprime to $\varphi(N)$. Therefore these two problems are equivalent, and so computing $b \equiv (m^r)^v$ from $\{s,v,m,\ a \equiv (m^r)^s\}$ is feasible if and only if $s \mid v$. Hence Step 2 is also sound. ❑

By using Protocol 4.1 as a subprotocol in Protocol 4.2, $\mathcal{P}$ convinces the recipient in fact that the exponent used in the signature is an element of $\{2,3,\ldots,3\lceil\sqrt{N}\rceil-4\}$.

4.3.2. Disavowal protocol

If a recipient $\mathcal{V}$ wants to open a signature S on message m, he has to perform a disavowal protocol with each group member. Only group members that have not created this signature S (i.e., that have an other secret key) must be able to successfully perform this protocol. Hence the following instance has to be solved, in which group member $\mathcal{P}$ has not created signature S.

secret of $\mathcal{P}$	:	s
public	:	$N,v,m,S,\Phi;\quad m,S \in \mathbb{Z}_N^*$
prove to $\mathcal{V}$	:	$S \not\equiv m^s \pmod N \wedge s \in \Phi \wedge s \mid v$

Fig. 4.4. Instance 4.3.

There are no zero-knowledge disavowal protocols known in the literature to convince $\mathcal{V}$ that $\alpha^x \not\equiv \beta^x \pmod N$, for given $\{N,\alpha,\beta,\alpha^x\}$, where $\varphi(N)$ is unknown. Therefore we use the following modification of the disavowal protocol of [Ch90] to solve Instance 4.3. $\mathcal{Z}$ publishes $\{\tilde{g},\tilde{h}\}$, which generates the whole group $\mathbb{Z}_N^*$ (see Section 4.3.3 for how to construct $\tilde{g}$ and $\tilde{h}$), together with a Trusted Public Directory containing the triples {name group member, $\tilde{g}^s,\tilde{h}^s$} where s is the secret key of that group member. Let l be a very small constant such that exhaustive search over $\{0,\ldots,l\}$ is feasible. The disavowal protocol uses the fact that if $S \equiv m^s$, then $\mathcal{P}$ cannot compute a from $(\frac{m^s}{S})^a$, because $(\frac{m^s}{S})^a \equiv 1$. In this case, therefore, he only can guess a.

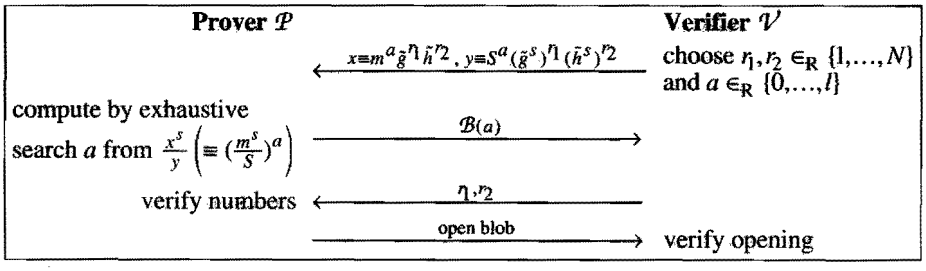


Fig. 4.5. Protocol 4.3 for Instance 4.3.

4.3.3. Some remarks on this construction of group signatures

Remark 4.1.
If all group members except one conspire, the secret key of that one person can be computed, because $\frac{v}{\prod_{i \neq j} s_i} = s_j$. This threat can be easily eliminated if the authority Z makes himself a member of the group, that is, if Z computes v as $v = s_Z \cdot \prod s_i$, where s_Z is a secret key only known to Z. By using this trick, the group can also consist of only two members.

Remark 4.2.
The number of bits in v (v is the only part of the public key that depends on the number of group members) is linear in the number of persons. So raising a number to the power v will take a time linear in the number of group members. Hence the number of computations in Step 2 of confirmation Protocol 4.2 is linear in the number of group members while Step 1 is independent of the number of group members.

The number of bits transmitted during confirmation Protocol 4.2 is independent of the number of group members.

Remark 4.3.
During the setup we defined the set Φ to be $\{\lceil\sqrt{N}\rceil, \lceil\sqrt{N}\rceil+1, \ldots, 2\lceil\sqrt{N}\rceil-2\}$. This is not the only possible choice for the set, from which the public keys are taken. Suppose that $\Phi = \{\varphi_1, \varphi_1+1, \ldots, \varphi_1+\varphi_2\} \subset \mathbb{N}$; then the following conditions must hold:

$$1, \tfrac{N}{2}, \varphi_1^2 \notin \tilde{\Phi} = \{\varphi_1 - \varphi_2,\ \varphi_1 - \varphi_2 + 1, \ldots,\ \varphi_1 + 2\varphi_2\}.$$

The first condition that $1 \notin \tilde{\Phi}$ is necessary to prevent m^1 from being a valid group signature on m. We use the condition $\frac{N}{2} \notin \tilde{\Phi}$ to reduce the size of the secret key (so the secret keys are now smaller than $\varphi(N)$). The last condition $\varphi_1^2 \notin \tilde{\Phi}$ is needed to avoid the following conspiracy attack: if two group members, say i and j, conspire, they can create signatures $S \equiv m^{s_i s_j}$, which they can both disavow later. Because $\varphi_1^2 \notin \tilde{\Phi}$, we have that $s_i s_j \notin \tilde{\Phi}$, so this signature $S \equiv m^{s_i s_j}$ will not be accepted in Step 1 of Protocol 4.2.

Instead of creating the signature m^s on m, a group member can also create the signatures $m^{v/s}$ and m^v. Although v/s and v are divisors of v, both numbers are not elements of $\tilde{\Phi}$, because $v > v/s > \varphi_1^2$. Hence these two signatures are not accepted by confirmation Protocol 4.2.

Remark 4.4.
The order of the elements from $\mathbb{Z}_N^*$ is a divisor of $\varphi(N)$, so if an element is randomly chosen from $\mathbb{Z}_N^*$, it might have small order. In [SS90] it is proven that if the two

(randomly chosen) prime factors p, q of N are of equal size (the same number of bits), then a substantial fraction of the elements in $\mathbb{Z}_N^*$ have high order.

Remark 4.5.
The blob $\mathcal{B}$ can be implemented in the following way: $\mathcal{Z}$ chooses generators g_p and h_q of $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ respectively, and constructs using the Chinese Remainder Theorem numbers $g \equiv \begin{cases} g_p \bmod p \\ 1 \bmod q \end{cases}$ and $h \equiv \begin{cases} 1 \bmod p \\ h_q \bmod q \end{cases}$. It is easy to see that the pair g,h generates $\mathbb{Z}_N^*$ uniformly, that is, each image of the function $f(x,y) \equiv g^x h^y \pmod N$ has the same number of preimages, for $x,y \in \{1,2,\ldots,\varphi(N)\}$.

But if $\mathcal{Z}$ reveals g,h, he also reveals the factorization of N. Therefore $\mathcal{Z}$ chooses integers a_1,a_2,b_1,b_2 satisfying $\gcd(a_1,b_1,p-1) = \gcd(a_2,b_2,q-1) = \gcd(a_1b_2 - a_2b_1, \frac{\varphi(N)}{\lambda(N)}) = 1$, and publishes $\tilde{g} \equiv g^{a_1}h^{a_2}$ and $\tilde{h} \equiv g^{b_1}h^{b_2}$. Remark that $\frac{\varphi(N)}{\lambda(N)} = \gcd(p-1,q-1)$. Below we prove that the pair $\tilde{g},\tilde{h}$ generates the whole group $\mathbb{Z}_N^*$ uniformly, if the exponents are chosen from $\{1,2,\ldots,\varphi(N)\}$. Hence, in order for $\mathcal{P}$ to make $\mathcal{B}(y)$, he chooses $r_1,r_2 \in \{1,2,\ldots,N\}$ and creates $\mathcal{B}(y)$ as $yg^{r_1}h^{r_2} \pmod N$.

We wish to prove that $\tilde{g},\tilde{h}$ generate $\mathbb{Z}_N^*$ uniformly. We know that $\tilde{g},\tilde{h} \in \mathbb{Z}_N^*$, and thus $<\tilde{g},\tilde{h}> \subset \mathbb{Z}_N^*$. Hence it is sufficient to prove that $g,h \in <\tilde{g},\tilde{h}>$, because this implies that $\mathbb{Z}_N^* \subset <\tilde{g},\tilde{h}>$, and thus $\mathbb{Z}_N^* = <\tilde{g},\tilde{h}>$. To prove that $g \in <\tilde{g},\tilde{h}>$, we have to find $c,d \in \mathbb{Z}$ such that $g \equiv \tilde{g}^c\tilde{h}^d \pmod N \equiv g^{ca_1+db_1}h^{ca_2+db_2} \pmod N$. The pair $<g,h>$ generates $\mathbb{Z}_N^*$ uniformly and the order of g,h is $(p-1)$ and $(q-1)$ respectively; hence it is equivalent to finding $c,d \in \mathbb{Z}$ such that $\begin{cases} ca_1+db_1 \equiv 1 \bmod (p-1) \\ ca_2+db_2 \equiv 0 \bmod (q-1) \end{cases}$, or with finding a vector $x \in \mathbb{Z}^4$ such that

$$\begin{pmatrix} a_1 & b_1 & p-1 & 0 \\ a_2 & b_2 & 0 & q-1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

We will apply Lemma 2.7 to find out when this matrix equation has an integral solution. Let A be the matrix $\begin{pmatrix} a_1 & b_1 & p-1 & 0 \\ a_2 & b_2 & 0 & q-1 \end{pmatrix}$ which has full rank. The gcd of all subdeterminants of A of order 2 is

$$\gcd\big(a_1b_2 - a_2b_1,\ -a_2(p-1),\ -b_2(p-1),\ a_1(q-1),\ b_1(q-1),\ (p-1)(q-1)\big) =$$
$$\gcd\big(a_1b_2 - a_2b_1,\ (p-1)\cdot\gcd(-a_2,-b_2,(q-1)),\ (q-1)\cdot\gcd(a_1,b_1,(p-1))\big) =$$
$$\gcd(a_1b_2 - a_2b_1,\ (p-1),\ (q-1)) =$$
$$1.$$

By applying Lemma 2.7, we show that the equation $Ax = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ has an integral solution. This proves that $g \in <\tilde{g},\tilde{h}>$. By using a similar argument we can prove that

$h \in < \tilde{g}, \tilde{h} >$.

4.4. Third construction of group signatures

For this construction of group signatures we assume that there is already a Trusted Public Directory in which each person's RSA-modulus is listed (the person's public RSA-exponent is not needed in this group signature construction). We also assume that the prime factors of each modulus $N_i = p_i q_i$ have the property that $p_i \in \Phi = \{\lceil\sqrt{M}\rceil, \lceil\sqrt{M}\rceil+1, \ldots, 2\lceil\sqrt{M}\rceil-2\}$ and $q_i > 3\sqrt{M}$, where M is some public number. The security of this construction is based on Assumption 1.3.

During the setup, a trusted authority Z chooses an RSA-modulus N, such that both N and $\varphi(N)$ are coprime to all the N_i's. This number N will be the public key of the group and nobody will know $\varphi(N)$. Choosing this modulus is the only task that Z has to perform, because the secret key of group member i will be the factorization of his own RSA-modulus $N_i = p_i q_i$.

So far no group has been created; there is only an already existing TPD. If someone wants to sign a message, he creates *at that moment* some "group" of persons. He can do this by randomly picking some individuals (including himself) from the Trusted Public Directory of public keys; then he proves that he belongs to that group (i.e., that he belongs to that subgroup of members of the TPD). So if person i wants to sign message n, he first chooses randomly some set Γ of persons from the TPD (including himself). His group signature will be

$$\Gamma, \quad (f(n))^{p_i} \bmod N,$$

and he convinces the recipient in a zero-knowledge way that this is his correct signature, that is, that the exponent used in the signature is the smallest divisor of the modulus of somebody in the set Γ. This can be done by performing Protocol 4.2, with $\Omega = \Phi = \{\lceil\sqrt{M}\rceil, \lceil\sqrt{M}\rceil+1, \ldots, 2\lceil\sqrt{M}\rceil-2\}$ and $v = \prod_{j\in\Gamma} N_j$. So the signer convinces the recipient that the exponent used in the signature is

- an element of $\tilde{\Phi} = \{2, 3, \ldots, 3\lceil\sqrt{M}\rceil-4\}$, and
- a divisor of $\prod_{j\in\Gamma} N_j$.

Each person only knows the factorization of his own modulus. Therefore the exponent used is the number 1 or is a product of some of the moduli of $\Gamma \setminus \{i\}$, together with q_i and/or p_i. According to the bounds on the prime divisors of each modulus, we have that $N_j > q_j > 3\sqrt{M}$ (for each j). Hence for all j: N_j and q_j are not elements of $\tilde{\Phi}$ and therefore N_j and q_i cannot be divisors of the exponent used in the signature. Also the number 1 is not an element of $\tilde{\Phi}$, and thus the exponent used in the signature must be p_i.

If a "group member" wants to deny a signature on a message, he can use Protocol 4.3. If he did not sign that message, he can successfully perform this protocol, despite the fact that he did not know he was in such a group.

4.5. Fourth construction of group signatures

Let p be a large public prime and let g, h be public generators of $\mathbb{Z}_p^*$. We assume that there is already a setup for a scheme based on the discrete logarithm (Assumption 1.2), so person i has a secret key s_i and a public key $k_i \equiv g^{s_i} \pmod p$. There is a TPD with all these public keys.

To transform this scheme into a group signature scheme, we need no trusted authority. The "group" is created in the same way as in the previous section, so that if person i wants to sign message $m = f(n)$, he first randomly chooses some set Γ of persons (including himself) from the TPD; his signature will be

$$\Gamma,\ m^{s_i} \pmod p.$$

He must convince the recipient in a zero-knowledge way that the secret exponent used in that signature is also used in the public key of some person in the group Γ. Thus they have to use a protocol that solves the following instance 4.4 of Figure 4.6:

secret of $\mathcal{P}$	:	s_i
public	:	p, g, h, m, S, Γ
to prove to $\mathcal{V}$	:	$S \equiv m^{s_i} \pmod p \wedge g^{s_i} \in \{k_j \mid j \in \Gamma\}$

Fig. 4.6. Instance 4.4.

They use the protocol below, which gives no additional information about the person i and his secret key s_i. In Protocol 4.4 we have compressed the three proofs that

- S is of the correct form, that
- the exponents used in S and in some public key are the same, and that
- the public key is used by somebody in Γ.

Protocol 4.4. (for Instance 4.4)

Step 1. *$\mathcal{P}$ chooses numbers $r_1,\ldots,r_{|\Gamma|}$, t_1, t_2, $t_3 \in \{1,\ldots,p-1\}$ and a permutation τ of Γ. He sends $\mathcal{V}$ the numbers: $x \equiv \left(\frac{g}{m}\right)^{t_1} h^{t_2} \pmod p$, $y \equiv m^{t_3} \pmod p$, and $z_{\tau(j)} \equiv k_j h^{r_j} \pmod p$ (for all $j \in \Gamma$).*

Step 2. *$\mathcal{V}$ chooses $b \in \{0,1\}$ and sends b to $\mathcal{P}$.*

Step 3. $\mathcal{P}$ *sends* $\mathcal{V}$ *in case*

b=0: $r_1,\ldots,r_{|\Pi|}, t_1, t_2, t_3$ *and* τ.

b=1: $t_1+s_i \pmod{p-1}$, $t_2+r_i \pmod{p-1}$, $t_3+s_i \pmod{p-1}$, *and index* $\tau(i)$.

Step 4. $\mathcal{V}$ *verifies in case*

b=0: *that the numbers* $x, y, z_1,\ldots,z_{|\Pi|}$ *are formed correctly.*

b=1: *that* $yS \equiv m^{t_3+s_i} \pmod{p}$ *and that* $xz_{\tau(i)} \equiv Sh^{t_2+r_i}(g/m)^{t_1+s_i} \pmod{p}$.

This protocol is trivially complete. It is also sound, because if $\mathcal{P}$ can answer both questions, he knows the number s_i that satisfies $S \equiv m^{s_i}$ and $k_i \equiv g^{s_i}$. So if this protocol is iterated k times, $\mathcal{V}$ will be convinced with confidence $1-2^{-k}$. This protocol is also zero-knowledge because it can be simulated (with the same probability distributions) by the following algorithm.

Simulator

Step 1. *Choose a permutation* τ *of* Γ, *numbers* $r_1,\ldots,r_{|\Pi|}$, t_1, t_2, $t_3 \in \{1,\ldots,p-1\}$, *and* $e \in \{0,1\}$.

Compute and send the numbers: $z_{\tau(j)} \equiv k_j h^{r_j} \pmod{p}$ (for all $j \in \Gamma$), $y \equiv m^{t_3} / S^e \pmod{p}$, *and* $x \equiv \left(\frac{g}{m}\right)^{t_1} h^{t_2} (S / z_{\tau(i)})^e \pmod{p}$.

Step 2. *Receive* $b \in \{0,1\}$.

Step 3. *In case*

$e=b=0$: *send the numbers* $r_1,\ldots,r_{|\Pi|}, t_1, t_2, t_3$ *and* τ.

$e=b=1$: *send index* $\tau(i)$, *and* t_1, t_2, t_3.

$e \neq b$: *restart this algorithm.*

If a "group member" wants to deny a group signature, he can for instance use the disavowal protocol of [Ch90]. He can successfully perform this protocol despite the fact that he did not know he was in such a group.

Confirmation Protocol 4.4 is zero-knowledge, so it reveals no additional information about the identity of the signer. But what additional information about the signer is revealed by the group signature, in other words, we need also to discuss the anonymity of signers in this construction of group signatures. If we write $p = 2^d q+1$ (for q odd), then, given m and $m^s \pmod{p}$, it is easy to compute the d least significant bits of s(as mentioned in Section 1.4.1). Thus the recipient of the signature can eliminate persons from Γ who cannot have created this signature. To avoid this attack, the d least significant bits of all secret keys must be the same. And we have to use Assumption 1.2.

4.6. Applications

In Section 4.1 we already mentioned two applications of group signatures. Another application of group signatures can be found in [Ped92]. In Section 6.3 of [Ped92] two *distributed signature* schemes are described: in such a scheme there are n agents having secret keys and any k of them can together sign messages, while k–1 agents cannot.

In the first scheme of [Ped92], the recipient of the signature can see which k agents have signed the message. In his second scheme this is not the case, but each agent who participated in signing the message knows who the other k–1 agents were and he can prove who the other k–1 agents were.

By using group signatures, even this knowledge can be eliminated. Suppose we have digital group signatures (see Section 4.1), so each agent can create his signature on a message and the recipient can verity this signature himself. The recipient has a valid distributed signature on a message m only if he has k different digital group signatures on the same message (i.e., k different agents signed the same message). With group signatures not even the group members know who signed a message.

4.7. Some open problems

We have presented four different constructions of group signatures. In the first construction the recipient has to ask Z to open a received signature, while in the three other constructions the recipient has to perform the disavowal protocol with each group member. For the disavowal protocol all group members must be available. It would be nice to create group signature schemes in which for opening a signature another situation holds, such as: a majority of the group members can open a signature.

Is it possible to make digital (i.e., not undeniable) group signatures other than by using [FFS88] on undeniable signatures?

Can the results of [SS90] and [Per85] be applied to show that specific choices of the exponents in the constructions of Sections 4.2-4.4 and 4.5, respectively, protect anonymity in ways equivalent to known computational problems?

Can the trusted authority Z be replaced by a multiparty protocol?

Is it possible to modify the fourth group signature scheme in such a way that the number of transmitted bits during the confirmation protocol is independent of the number of group members?

What knowledge is revealed by releasing $< \tilde{g}, \tilde{h} >$ (see Section 4.3.3)? With this knowledge, is it easier to factor the modulus?

A group signature scheme is a signature scheme with one public and several secret keys, and it is infeasible for conspiring group members to create a new secret key out of their own secret keys. Is it possible to create a complementary scheme with one secret and several public keys, in which it is infeasible for conspiring group members to create a new public key out of their own public keys?

5

Signatures unconditionally secure for the signer[‡]

5.1. Introduction

Digital signatures are intended to provide legal security in digital communication, as handwritten signatures should do in conventional communication: a digital signature guarantees that the presumed signer really is the sender of this message (or has at least authorized it), and the recipient can prove this to third parties. All known conventional signature schemes have a publicly known test predicate *test*, that can evaluate signatures in polynomial time. Each signer has a public key *PK*; a number *S* will be accepted as his signature on message *m* if it satisfies *test*(*PK*,*m*,*S*) = "true". This implies that forging signatures is in the complexity class NP: one can guess a signature and test its correctness in polynomial time.

All these schemes are based on a generally trusted, but unproven, complexity-theoretic assumption (such as the infeasibility of integer factoring or the computation of some discrete logarithm). Hence signatures can be forged if this assumption turns out to be false. The signers thus have only computational security, and the presumed signer is defenseless because the forged digital signature looks exactly like an authentic one and satisfies the test predicate. The recipients of the signatures have unconditional security: if a received signature satisfies the test predicate, then this signature will always be valid, no matter how much computing power the signer has. In order to deal with increasingly powerful computers and better "breaking" algorithms, then, one must

[‡] This chapter is based on the paper "Cryptographically strong undeniable signatures, unconditionally secure for the signer" by David Chaum, Eugène van Heyst and Birgit Pfitzmann, *Advances in Cryptology-CRYPTO '91*, J. Feigenbaum ed., LNCS 576, Springer-Verlag, pp. 470-484; and on the paper "How to make efficient Fail-stop signatures" by Eugène van Heyst and Torben Pedersen, which will appear in *Advances in Cryptology-EUROCRYPT '92*. This chapter contains only a part of the first paper; the rest will appear in the Ph.D. thesis of Birgit Pfitzmann.

increase often enough the so-called *security parameters*, such as the size of the used modulus (which hopefully cannot be factored).

In this chapter we will study the "opposite case", in which the signer has unconditional security and the recipient computational security. Clearly, these signatures cannot be conventional digital signatures. In such a scheme, someone who wants to forge a signature of another person on some message cannot do more than merely guess the signature, and cannot verify locally whether his guess was correct.

An example of such a scheme is the *fail-stop signature* (see [WP89], [BPW90], [PW90] and [PW91]). With such a signature, unforgeability also relies on a complexity theoretic assumption; but even if a signature is forged, the presumed signer can prove that the signature is a forgery: he can prove that the system's underlying assumption has been broken. This proof of forgery may fail (with a very small probability), but the ability to prove a forgery does not rely on any complexity theoretic assumption and is independent of the forger's computing power. Hence, the signer is protected against forgers with unlimited computing power, because after the first forgery, all other participants in the system and the system operator know that the signature scheme has been broken, and the system will be stopped. That is why this system is called "fail-stop". If signatures become invalid after forgery has been proven, the signatures are unconditionally secure for the signer.

Another example of the new signature schemes is the *unconditionally secure signatures* of [CR90]: these are signatures in which both the signer and the recipient have unconditional security, but they differ widely from conventional signatures: each participant has a different test predicate, and this test predicate depends on how many participants have received the signature. Moreover, active attacks on recipients may damage the security of the system.

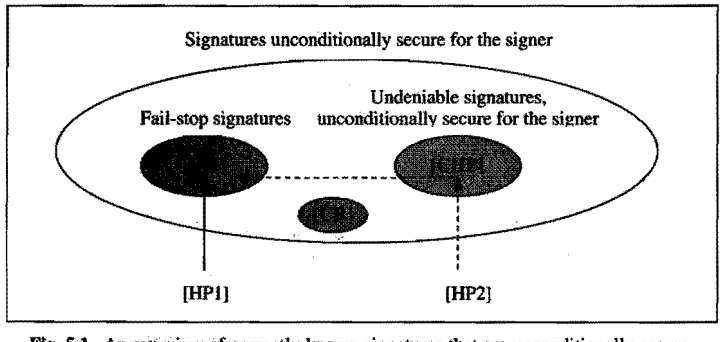


Fig. 5.1. An overview of currently known signatures that are unconditionally secure for the signer.

In this chapter we present three signature schemes that are unconditionally secure for the signer (see Figure 5.1):

(i) A new efficient construction of fail-stop signatures (called [HP1] in Figure 5.1), in

which a signature on a message consists of only two numbers, rather than of signing a message bit-wise as in [BPW90] (see Subsection 5.3.3).

(ii) The first construction of undeniable signatures that are unconditionally secure for the signer (called [CHP] in Figure 5.1; see Section 5.4). We show that these signatures are not fail-stop signatures.

(iii) The first construction of *convertible signatures* that are unconditionally secure for the signer. If the signer reveals some numbers, then these signatures turn into fail-stop signatures (called [HP2] in Figure 5.1; see Section 5.5).

In principle all these signatures use *one-time keys*: this means that for each signature a new public key must be used, in order to provide unconditional security for the signer. But see also Subsection 5.3.4 for efficient constructions how to use the public key several times, without endangering the unconditional security for the signer.

5.2. Notation

Throughout this chapter p and q denote large primes (say, of at least 500 bits) such that q divides $p-1$, and G_q is the unique subgroup of $\mathbb{Z}_p^*$ of order q. As any element $b \neq 1$ of G_q generates the group, the discrete logarithm of $a \in G_q$ with respect to the base b is defined and denoted by $\log_b(a)$.

In this chapter we will use generators g,h of G_q and not generators of $\mathbb{Z}_p^*$ (see for instance also [CvA89]). Note that this is not the usual setup for a scheme based on the discrete logarithm. The reason for doing this is that each nonzero element x has a multiplicative inverse modulo q, while this is not true modulo $p-1$. So in G_q we may rewrite $g^x \equiv h^y \pmod p$ as $g \equiv h^{y/x} \pmod p$ for $x \not\equiv 0 \pmod q$. We assume that computing the discrete logarithm in $\mathbb{Z}_p^*$ is infeasible (Assumption 1.2), and thus also computing the discrete logarithm in a large subgroup, i.e., in G_q.

In this chapter we denote by

SK	the secret key of the signer,
$PK = pub(SK)$	the corresponding public key,
$S = sign(SK,m)$	signature of the signer on message m,
$test(PK,m,S)$	polynomial-time (in a *security parameter* k) computable predicate to verify the signature S on m. A signature is called *valid* if it satisfies this predicate *test*.

5.3. Fail-stop signatures

5.3.1. General introduction

Fail-stop signatures have been formally defined in [PW90]. This Subsection 5.3.1 only gives a brief and rather informal description of the properties of fail-stop signatures.

In a fail-stop signature scheme, exponentially many (in k) secret keys correspond to a given public key, and different secret keys will (with very high probability) give different signatures on the same message. All these signatures satisfy the test predicate and will thus be valid. So note that with fail-stop signatures not only *one* signature satisfies the test predicate. However, the signer knows only one of these secret keys and can therefore construct only one of these signatures on a message.

Furthermore, given the public key and signatures on some messages, a forger must not be able to guess which signature the signer is able to construct on a new message, so that even if the forger (by using his unlimited power) succeeds in making a valid signature, this signature will with very high probability be different from the signer's signature. Given such a forged signature the signer must then be able to prove that it is different from his own signature, thereby proving that it is a forgery. After having discovered such a forgery and proved it, the signer should stop using the scheme.

For fail-stop signatures we require that the predicate *test* must satisfy that for every secret key SK^* corresponding to PK

$$test(PK, m, sign(SK^*,m)) = \text{"}true\text{"}.$$

A scheme is a fail-stop signature scheme if it satisfies the following three requirements:

(i) Let PK and the signature $S = sign(SK,m)$ on m be given. Then there are exponentially many (in k) possible secret keys SK^* corresponding to PK such that $S = sign(SK^*,m)$. Furthermore, if such a secret key SK^* is chosen at random, then the probability that $sign(SK,m^*) = sign(SK^*,m^*)$ is negligible, for every message $m^* \neq m$.
(Informally: it is not possible to compute the signer's signature on a new message, even with unlimited computing power.)

(ii) There is a polynomial-time computable function *proof*, which on input SK, PK, a message m, and a valid, forged signature $S' \neq sign(SK,m)$ on m, outputs a proof that S' is a forgery.
(Informally: the presumed signer is able to supply a proof of the forgery.)

(iii) No signer with polynomial-time computing power is able to construct a valid signature S on a given message m and also construct a proof that S is a forgery.
(Informally: the signer cannot make signatures which he can later prove to be forgeries.)

These requirements are for one-time keys. It is not hard to generalize this definition of fail-stop signatures to comprise schemes in which more than one message can be signed with one secret key.

The first two requirements imply that fail-stop signatures are unconditionally secure for the signer, whereas the third requirement says that the scheme is secure for the recipients of the signatures. Unlike the security for the signer, the security for the recipient depends on a complexity theoretic assumption. The reader is referred to [PW90] for a thorough discussion of the properties of fail-stop signatures. (To avoid confusion, note that [PW90] considers different security parameters for the security of the signer and for the security of the recipient, while in our scheme these are equal.)

5.3.2. The construction of (BPW90), (PW91)

In [BPW90] and [PW91] a construction of a particular fail-stop signature (called a *hiding scheme*) is given. This scheme is based on the assumption of the existence of *claw-free pairs of permutations* ([GMR88]): this means that there are permutations f_0 and f_1 of the same set such that it is infeasible to find a pair (x_0,x_1) that satisfies $f_0(x_0) = f_1(x_1)$ (which is called a *claw*).

This hiding scheme is based on the idea of *Lamport (one-time) signatures* ([DH76]): Let g be a one-way function and k an a priori fixed number indicating how many bits can be signed. Each person chooses random numbers $r_{i,0}, r_{i,1}$ $(i = 1,2,\ldots,k)$ and publishes his public key

$$\begin{cases} g(r_{1,0}),\ g(r_{2,0}),\ldots,\ g(r_{k,0}), \\ g(r_{1,1}),\ g(r_{2,1}),\ldots,\ g(r_{k,1}). \end{cases}$$

If he wants to sign the t^{th} bit with value $b \in \{0,1\}$, he sends the preimage $r_{t,b}$ of $g(r_{t,b})$ to the recipient, who can easily verify its correctness. Thus the signer can use this public key to sign at most k bits.

To transform this Lamport signature into a fail-stop signature, the authors require that g also fulfills the following two conditions:

- for a fixed value $\sigma > 0$, and for each $x \in \mathrm{dom}(g)$, the value $g(x)$ has at least 2^{σ} preimages,
- g is collision-free for the signer.

Such a function g is called a *bundling function*, and this function is *not* chosen by the signer but by the recipient: otherwise this function need not be collision-free for the signer. It can easily be verified that the Lamport signature scheme that uses a bundling function is a fail-stop signature scheme. The following construction of a bundling function g is proposed, that uses a claw-free pair of permutations f_0 and f_1:

$$g(a_0,\ldots,a_{\sigma-1},x) := f_{a_0}(\ldots f_{a_{\sigma-2}}(f_{a_{\sigma-1}}(x))\ldots),$$

where $a_i \in \{0,1\}$ $(i = 1,\ldots,\sigma)$ and x is an element from the permuted set. It is easy to see that finding a g-collision is as hard as finding a claw, and that each image of g has 2^σ preimages; thus this function g is a bundling function.

In [GMR88] efficient claw-free pairs of permutations are constructed as follows. Let m be a large Blum integer (so $m = pq$, where both primes p and q are congruent 3 mod 4) and define for $x \in \{0,1,2,\ldots,m-1\}$:

$$|x|:=\begin{cases} x & \text{if } x \in \{0,1,\ldots,\frac{m-1}{2}\}, \\ m-x & \text{if } x \in \{\frac{m+1}{2},\frac{m+3}{2},\ldots,m-1\}. \end{cases}$$

Then the following two functions are permutations of $D=\{1 \le x \le \frac{m-1}{2} \mid \text{Jacobi symbol } \left(\frac{x}{m}\right)=1\}$:

$$f_0(x) := |x^2 \bmod m|,$$
$$f_1(x) := |4x^2 \bmod m|.$$

In [GMR88] it is proven that finding a claw for these two permutations (i.e., a pair (x_0,x_1) with $f_0(x_0) = f_1(x_1)$) is as hard as factoring m. The bundling function g that uses these two permutations can be rewritten as

$$g(a_0,\ldots,a_{\sigma-1},x) = |4^\alpha x^{2^\sigma} \bmod m|,$$

where $\alpha = \sum_{i=0}^{\sigma-1} a_i 2^i$ and $x \in D$. Thus the application of this bundling function g mainly consists of σ squarings, and the collision-freeness of g depends on the infeasibility of factoring large Blum integers.

This hiding scheme that uses Lamport signatures has the disadvantage that it signs messages bit-wise: for each bit of the message, one signature has to be given. Of course messages can also be hashed before being signed, to reduce the length of the fail-stop signature. In the next subsection we construct a fail-stop signature that does not have this disadvantage: the length of the signature is twice the length of the (hashed) message.

5.3.3. New construction of fail-stop signatures

In this subsection we describe an efficient fail-stop signature (based on Assumption 1.2), that does not sign messages bit-wise. Let g and h be elements of G_q such that no participant knows $\log_g(h)$. Recall that these two elements have order q which is prime. These elements can either be chosen by a trusted authority, when the system is initialized, or by the participants using a coin-flipping protocol. Although (p,q,g,h) is part of the public key (called the *prekey* of the scheme), it will not be mentioned as part of the public key in the following. To give a better idea of the scheme, we will first

assume that a person issues only one signature. Let the secret key of person $\mathcal{A}$ be

$$SK = (x_1,x_2,y_1,y_2) \in \mathbb{Z}_q^4,$$

and $\mathcal{A}$ publishes the corresponding public key

$$PK = pub(SK) = (p_1,p_2) = (g^{x_1}h^{x_2}, g^{y_1}h^{y_2}). \tag{5.1}$$

To sign a message $m \in \mathbb{Z}_q$, $\mathcal{A}$ computes the following numbers:

$$\begin{aligned} S = sign(SK,m) &= (\sigma_1,\sigma_2), \text{ where} \\ \sigma_1 &\equiv x_1+my_1 \pmod q, \\ \sigma_2 &\equiv x_2+my_2 \pmod q. \end{aligned} \tag{5.2}$$

The recipient of this digital signature verifies that

$$p_1p_2^m \equiv g^{\sigma_1}h^{\sigma_2} \pmod p. \tag{5.3}$$

The following three lemmas show that this signature scheme is a fail-stop signature scheme. First note that for every secret key SK^* corresponding to PK, the predicate $test(PK,m,sign(SK^*,m))$ holds, i.e., for every tuple $(x_1,x_2,y_1,y_2) \in \mathbb{Z}_q^4$ that satisfies

$$\begin{aligned} PK &= (g^{x_1}h^{x_2}, g^{y_1}h^{y_2}), \\ \sigma_1 &\equiv x_1+my_1 \pmod q, \\ \sigma_2 &\equiv x_2+my_2 \pmod q, \end{aligned}$$

we have

$$p_1p_2^m \equiv g^{\sigma_1}h^{\sigma_2} \pmod p.$$

Lemma 5.1. *The public key PK, together with the signature sign(SK,m) on m, contain no information about which of q possible secret keys are used for SK.*

Proof. This lemma is a special case of Theorem 4.4 of [Ped91]. Another way to prove it is the following. Define $h := g^a$, $p_1 := g^{e_1}$, $p_2 := g^{e_2}$. This representation is possible because g is a generator of G_q. Then we can write Equations (5.1) and (5.2) as:

$$\begin{cases} e_1 \equiv x_1 + ax_2 \pmod q, \\ e_2 \equiv y_1 + ay_2 \pmod q, \\ \sigma_1 \equiv x_1 + my_1 \pmod q, \\ \sigma_2 \equiv x_2 + my_2 \pmod q. \end{cases}$$

The fact that equation (5.3) holds, follows immediately from Equations (5.1) and (5.2). The forger has to find a solution (x_1,x_2,y_1,y_2) to the equations

$$\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \\ 1 & 0 & m & 0 \\ 0 & 1 & 0 & m \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix} \equiv \begin{pmatrix} e_1 \\ e_2 \\ \sigma_1 \\ \sigma_2 \end{pmatrix} \pmod q.$$

It is easy to see that this matrix has rank 3 (the rank is defined because q is prime), hence, since there is at least one solution, there are exactly q solutions to this equation. ❑

Lemma 5.2. *Let PK, the signature $S = sign(SK,m)$ on m and a valid signature $S' = (\tau_1,\tau_2)$ on m' (so $p_1 p_2^{m'} \equiv g^{\tau_1} h^{\tau_2}$), be given. Then there exists a unique secret key SK* corresponding to PK such that $S = sign(SK^*,m)$* and *$S' = sign(SK^*,m')$*.

Proof As in the proof of Lemma 1, a solution (x_1,x_2,y_1,y_2) to the matrix equation

$$\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \\ 1 & 0 & m & 0 \\ 0 & 1 & 0 & m \\ 1 & 0 & m' & 0 \\ 0 & 1 & 0 & m' \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_1 \\ \sigma_1 \\ \sigma_2 \\ \tau_1 \\ \tau_2 \end{pmatrix}$$

has to be found. It is easy to see that this matrix has rank 4 (because $m' \neq m$), so there is exactly one solution. ❑

Lemma 5.1 says that there are q possible secret keys corresponding to a given public key and one given signature. By Lemma 5.2, each of them will yield a different signature on a message $m' \neq m$. This shows that the first requirement for the security of the signer is satisfied.

Lemma 5.3. *If the presumed signer receives a valid, forged signature $S' = (\tau_1,\tau_2)$ on m (so $p_1 p_2^m \equiv g^{\tau_1} h^{\tau_2}$), but $S' \neq sign(SK,m)$, then he can compute* $\log_g(h)$.

Proof By writing $sign(SK,m) = (\sigma_1,\sigma_2)$, we have that $p_1 p_2^m \equiv g^{\tau_1} h^{\tau_2} \equiv g^{\sigma_1} h^{\sigma_2}$ and thus that $g^{\sigma_1 - \tau_1} \equiv h^{\tau_2 - \sigma_2} \pmod{p}$. If $\sigma_2 = \tau_2$, then we also have $\sigma_1 = \tau_1$ and thus $S' = sign(SK,m)$. This is a contradiction and therefore the presumed signer can compute $\log_g(h)$ as $(\sigma_1 - \tau_1)(\tau_2 - \sigma_2)^{-1} \bmod q$. ❑

Hence, under the assumption that the signer cannot compute $\log_g(h)$, this logarithm is a proof of forgery. Thus we can define

$$proof(SK,S') := (\sigma_1 - \tau_1)(\tau_2 - \sigma_2)^{-1} \bmod q.$$

Lemma 5.3 also implies that the signer cannot compute a valid signature different from $sign(SK,m)$ without being able to compute discrete logarithms. Hence, the signature scheme is computationally secure for the recipients if it is infeasible to compute $\log_g(h)$. So this scheme also has properties (ii) and (iii) of fail-stop signatures.

Note that this secret key is a one-time key: if two different messages are signed using the same secret key, then it is easy to compute the secret key from these signatures.

Remark 5.1. In the above scheme, the public key consists of two numbers modulo q. It is possible to reduce the size of this public key as follows. Let H be a collision-free hash function that maps the elements of G_q (of length $l(q)$, which is the number of bits of q) into numbers of a smaller size. Then the public key will be

$$PK^* = (H(p_1),p_2),$$

where p_1 and p_2 are defined as before. A signature (σ_1,σ_2) on the message m is constructed as before, and it is verified as

$$H(g^{\sigma_1}h^{\sigma_2}p_2^{-m}) = H(p_1).$$

By using this public key, the Lemmas 5.1, 5.2 and 5.3 have to be modified. For instance, Lemma 5.3 has to be modified as follows:

Lemma 5.3.a. *If the presumed signer receives a valid, forged signature* $S' = (\tau_1,\tau_2)$ *on* m *(so* $H(g^{\tau_1}h^{\tau_2}p_2^{-m}) = H(p_1))$*), but* $S' \neq sign(SK,m)$*, then he can compute* $\log_g(h)$ *or he has found a collision for* H.

Long messages can first be hashed into smaller messages before signing, but then Lemma 5.3 has to be modified in a similar way as was done in Lemma 5.3.a. Another modification of our fail-stop signature scheme is the following, in which H, p_1, p_2, σ_1, and σ_2 are defined as before.

$$PK^* = (H(p_1),H(p_2)),$$
$$S^* = sign(SK,m) = (\sigma_1,\sigma_2,p_2).$$

The recipient verifies this signature S^* as

$$H(g^{\sigma_1}h^{\sigma_2}p_2^{-m}) = H(p_1) \quad \text{and} \quad H(p_2) = H(p_2).$$

Even if no hash functions are used in the public key, it is more efficient to verify the signature by computing

$$g^{\sigma_1}h^{\sigma_2}p_2^{-m},$$

and comparing it with p_1. This requires less than $2l(q)$ multiplications, if the products gh, gp_2, hp_2 and ghp_2 are precomputed.

5.3.4. More than one signature per public key

As noted in the previous subsection, a person can use his public key (and secret key) only once. We present three different ways to overcome this problem: the public key can be used to provide k messages with a signature, and the signer still has unconditional security. In the first two ways, the secret key consists of $2k$ numbers, while each signature consists of 2 numbers. These two ways differ in the computations needed. In the third way, the secret key consists of at most k elements, while each signature consists of $\lceil \log k \rceil + 3$ numbers.

Method 1. Person $\mathcal{A}$ chooses as a secret key

$$SK = (x_1, y_1, x_2, y_2, \ldots, x_{k+1}, y_{k+1}),$$

and he publishes the corresponding public key

$$PK = (p_1, \ldots, p_{k+1}) = (g^{x_1}h^{y_1}, \ldots, g^{x_{k+1}}h^{y_{k+1}}).$$

To sign a message $m \in \mathbb{Z}_q^*$, $\mathcal{A}$ computes the following numbers:

$$sign(SK,m) = (\sigma_1,\sigma_2), \text{ where}$$
$$\sigma_1 \equiv x_1 + mx_2 + \ldots + m^k x_{k+1} \pmod q,$$
$$\sigma_2 \equiv y_1 + my_2 + \ldots + m^k y_{k+1} \pmod q.$$

The recipient of this digital signature verifies that

$$p_1 p_2^m \ldots p_{k+1}^{m^k} \equiv g^{\sigma_1} h^{\sigma_2} \pmod p.$$

After issuing signatures on k different messages, the signer still has unconditional security. (This follows from Theorem 4.4 of [Ped91].) The security of the recipient follows from the same arguments as in the proof of Lemma 5.3.

Method 2. ([Pf91]) Person $\mathcal{A}$ chooses the same secret key and public key as in Method 1. In Method 2, the signature on the message depends on the number of messages that $\mathcal{A}$ has signed previously: If $\mathcal{A}$ has signed $i-1$ messages ($1 \le i \le k$), the signature on a message, m, will be

$$sign(SK, m, i) = (i,\sigma_1,\sigma_2), \text{ where}$$
$$\sigma_1 \equiv x_i + mx_{i+1} \pmod q,$$
$$\sigma_2 \equiv y_i + my_{i+1} \pmod q.$$

The recipient of this signature verifies that

$$p_i p_{i+1}^m \equiv g^{\sigma_1} h^{\sigma_2} \pmod p.$$

Hence, at the cost of including a counter in the signatures, the computations of the signer as well as the recipient are easier here than in Method 1. Again, the security of the recipient follows from Lemma 5.3.

In order to prove that the signer has unconditional security after issuing k different signatures, it is sufficient to show that the rank of the following $(3k+1)\times(2k+2)$ matrix A_k is $2k+1$. This will be done by proving that matrix $\tilde{A}_k$ below can be obtained from A_k by elementary row operations, and noting that matrix $\tilde{A}_k$ has rank $2k+1$.

$$A_k = \begin{bmatrix} 1 & 0 & m_1 & & & & & & \\ & 1 & 0 & m_1 & & & & & \\ & & 1 & 0 & m_2 & & & & \\ & & & 1 & 0 & m_2 & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & 0 & m_k & \\ & & & & & & 1 & 0 & m_k \\ 1 & a & & & & & & & \\ & & 1 & a & & & & & \\ & & & & \ddots & & & & \\ & & & & & & & 1 & a \end{bmatrix} \quad \text{and} \quad \tilde{A}_k = \begin{bmatrix} 1 & 0 & m_1 & & & & & & \\ & 1 & 0 & m_1 & & & & & \\ & & 1 & 0 & m_2 & & & & \\ & & & 1 & 0 & m_2 & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & 0 & m_k & \\ & & & & & & 1 & 0 & m_k \\ & & & & & & & 1 & a \end{bmatrix}$$

To see that this matrix has rank $2k+1$, consider the following four rows of matrix A_k

$$\begin{array}{cccccccccc} 0 & \ldots & 0 & 1 & 0 & m_i & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 1 & 0 & m_i & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & a & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 1 & a & 0 & \ldots & 0 \end{array}$$

By Lemma 5.1, this submatrix has rank 3 and the third row can be removed. By using this method for $i = 1,2,\ldots,k-1$, we may delete all rows $(0 \ \ldots \ 0 \ 1 \ a \ 0 \ \ldots \ 0)$ from matrix A_k, except the last row. This results in matrix $\tilde{A}_k$.

Hence, PK and k signatures $sign(SK,m_1,1),\ldots,sign(SK,m_k,k)$ contain no information about which of q possible secret keys are used for SK, and each of these possible secret keys will yield a different signature on a new message m^*.

Method 3. Use one of the two types of tree-authentication (assume that $k=2^n$).
(1) (see [Merk80]) The signer has $k(=2^n)$ pairs of secret key S_i and public key $PK_i = pub(SK_i)$. These numbers PK_i are used to construct a binary tree by using a collision free hash function h, as indicated in Figure 5.2 (left-hand side). The public key PK will be the root of the tree and the secret key SK contains all other nodes of the tree (the signer does not need to store all the elements of SK).

The i^{th} signature on message m_i does not only consists of $sign(SK_i,m_i)$, but it also contains $n+1$ nodes of the tree, so that the recipient can verify the signature (see the right-hand side of Figure 5.2, remember that the recipient only knows PK). So each signature consists of $n+3$ numbers, which is logarithmic in k.

(2) (see [Merk87] and [GMR88]) The signer starts with one pair of a secret and public key at the root. Then he creates two sons, each having a new pair of secret and public key, and he uses the secret key of the father to sign the note saying: "my two sons have public key ... and ...". For each of these two sons he creates two new sons, each having a new pair of secret and public key, and he uses the secret key of each father to sign the note saying what the two new public keys of the sons are. And so on.

He can sign messages by using the secret key that belongs to a leave of the tree, and a complete signature consists of this signature and the corresponding branch of the tree with the signed notes. But the signer does not need to build the complete tree during the setup: the tree is built up from left to right as the signatures are needed.

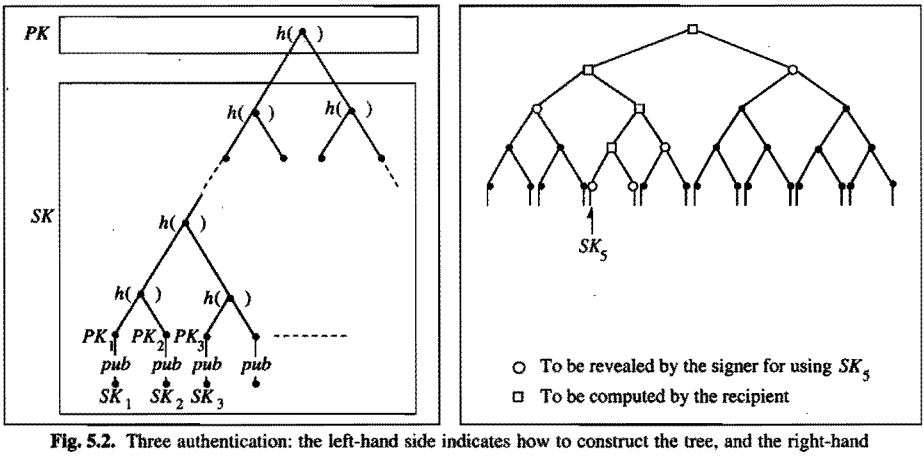


Fig. 5.2. Three authentication: the left-hand side indicates how to construct the tree, and the right-hand side shows which numbers must be revealed by the signer and how the recipient can verify the signature.

5.4. Undeniable signatures unconditionally secure for the signer

5.4.1. Description of the signature scheme

Let g_1, g_2 and g_3 be generators of G_q such that no participant knows $\log_{g_j}(g_i)$ $(i \neq j)$ (in this subsection we will use that $g_1 \neq 1$ and $g_1 \neq g_2$). These elements can either be chosen by a trusted authority, when the system is initialized, or by the participants using a coin-flipping protocol. They also choose security parameters l, l', which will be the number of times that the confirmation/disavowal protocol has to be iterated. The prekey of the scheme is (p,q,g_1,g_2,g_3,l,l'). To give a better idea of our scheme, we will first assume that a person issues only one signature. Let the secret key of person $\mathcal{A}$ be

$$SK = (x_1,x_2,x_3) \in \mathbb{Z}_q^3,$$

and suppose $\mathcal{A}$ publishes the corresponding public key

$$PK = pub(SK) \equiv g_1^{x_1} g_2^{x_2} g_3^{x_3} \pmod{p}. \tag{5.4}$$

To sign a message $m \in \mathbb{Z}_q$, $\mathcal{A}$ computes:

$$S = sign(SK,m) \equiv x_1 + x_2 + mx_3 \pmod{q}. \tag{5.5}$$

This signature is an undeniable signature. Hence the signer ($\mathcal{A}$) has to perform a zero-knowledge confirmation protocol (see Subsection 5.4.3) with the recipient to convince him of the correctness of his signature, i.e., of the correctness of the assertion:

"I know a SK such that PK = pub(SK) and S = sign(SK,m)",

or he has to perform a zero-knowledge disavowal protocol (see Subsection 5.4.4) to convince him that some number S' is not his signature on m, i.e., that

"I know a SK such that PK = pub(SK) and S' ≠ sign(SK,m)".

Remark 5.2. This scheme is not a fail-stop signature scheme: if a person breaks the underlying Assumption 1.2, then he is able to disavow all his previous signatures, and nobody will notice that the underlying assumption has been broken. This is not the case with fail-stop signatures.

5.4.2. Security of this signature scheme

Security for the verifier

By using the Certified Discrete Logarithm Assumption (Assumption 1.2), we will prove that the function used, i.e., *pub*, is collision-free: this means that $\mathcal{A}$ cannot find two different secret keys having the same public key.

Lemma 5.4. *On Assumption 1.2, it is infeasible for $\mathcal{A}$ to have a probabilistic polynomial-time algorithm that on a random triple of generators (g_1,g_2,g_3) of G_q as input, outputs a G_q-collision, i.e., a pair $(x_1,x_2,x_3) \neq (y_1,y_2,y_3) \in \mathbb{Z}_q^3$ satisfying* $g_1^{x_1} g_2^{x_2} g_3^{x_3} \equiv g_1^{y_1} g_2^{y_2} g_3^{y_3} \pmod{p}$.

Proof. Assume that $\mathcal{A}$ has such a probabilistic polynomial-time algorithm AL to compute these collisions. Then $\mathcal{A}$ can compute collisions for random generators g_1,g_2 of G_q as follows:

Step 1. Choose $e_1,e_2 \in_R \mathbb{Z}_q^*$ and $r_1,r_2 \in_R \mathbb{Z}_q$ (r_1,r_2 not both zero). If $g_1^{r_1} g_2^{r_2} = 1$, then $\mathcal{A}$ has found a nontrivial collision for g_1,g_2. So suppose that $g_1^{r_1} g_2^{r_2} \neq 1$; hence it is a generator of G_q.

Step 2. Apply the algorithm AL on the three generators $(g_1^{e_1}, g_2^{e_2}, g_1^{r_1} g_2^{r_2})$. With a probability corresponding to the algorithm AL, it outputs a nontrivial collision $(x_1,x_2,x_3) \neq (y_1,y_2,y_3)$ for $(g_1^{e_1}, g_2^{e_2}, g_1^{r_1} g_2^{r_2})$.

Rewrite the obtained collision as $g_1^{e_1(x_1-y_1)-r_1(y_3-x_3)} \equiv g_2^{-e_2(x_2-y_2)+r_2(y_3-x_3)} \pmod{p}$. If $x_3 = y_3$, then it is easy to verify that we have a nontrivial collision for g_1,g_2, so assume that $x_3 \neq y_3$. If the collision obtained is trivial for g_1,g_2, then we have

$r_1 \equiv \frac{(x_1-y_1)e_1}{y_3-x_3} \pmod q$ and $r_2 \equiv \frac{(x_2-y_2)e_2}{y_3-x_3} \pmod q$. Thus we obtain from the algorithm the trivial collision for only *one* choice of (r_1,r_2), but from $g_1^{r_1}g_2^{r_2}$ the algorithm cannot determine which of q pairs (r_1,r_2) was used. Hence with probability $(1-1/q)$ the output of the algorithm produces a nontrivial collision for g_1,g_2. So the overall probability that this method succeeds is at least $(1-1/q)$ times the probability that the algorithm AL outputs a collision for triples, which was assumed to be infeasible. ❑

The confirmation and disavowal protocol are sound (which will be proved in Subsections 5.4.3 and 5.4.4) and the function *sign* is deterministic, so according to Lemma 5.4, it is infeasible for $\mathcal{A}$ to convince the recipient that S is a valid signature on m and disavow it later (otherwise $\mathcal{A}$ would have found two different secret keys having the same public key, which contradicts this lemma). *So the verifier has computational security.*

Security for the signer

By using the following lemma, we will show that if a forger knows the public key of $\mathcal{A}$ together with his signature on m, and if he forges a signature of $\mathcal{A}$ on $m' \neq m$, then the probability that $\mathcal{A}$ can disavow this forged signature is $1-1/q$.

Lemma 5.5. *Let $PK = pub(SK)$ and the signature $S = sign(SK,m)$ be given. Then for each forged signature S' on m' there exists a unique SK^* such that $PK = pub(SK^*)$, $S = sign(SK^*,m)$, and $S' = sign(SK^*,m')$.*

Proof. Because g_1 is a generator of G_q, we define α,β,γ as $g_2 \equiv g_1^{\alpha} \pmod p$, $g_3 \equiv g_1^{\beta} \pmod p$, and $PK \equiv g_1^{\gamma} \pmod p$. We want to find (x_1,x_2,x_3) that solves the following equations:

$$\begin{cases} \gamma \equiv x_1 + \alpha x_2 + \beta x_3 \pmod q, \\ S \equiv x_1 + x_2 + m x_3 \pmod q, \\ S' \equiv x_1 + x_2 + m' x_3 \pmod q. \end{cases}$$

The matrix of these three equations can be transformed by row operations into

$$\begin{pmatrix} 1 & \alpha & \beta \\ 0 & 1-\alpha & m-\beta \\ 0 & 0 & m'-m \end{pmatrix}.$$

Because we require that $m \neq m'$ and that $\alpha \neq 1$ $(g_1 \neq g_2)$, this matrix has rank 3, and thus there is exactly one solution to our system of equations. ❑

Thus, given the public key of $\mathcal{A}$ and a valid signature on m, this determines exactly q secret keys that satisfy the conditions (5.4) and (5.5) together, and each of these q secret keys determines a different value for the signature on a message m' $(\neq m)$. So

the forger can only guess the secret key used; thus with probability $1-1/q$ he chooses another secret key. Because of the completeness of the disavowal protocol, any signature on m' other than $sign(SK,m')$ can be disavowed by $\mathcal{A}$ with probability exactly 1. *So the signer has unconditional security with exponentially small error probability* (because with probability $1/q$ the forger chooses the correct secret key).

Note that the security of $\mathcal{A}$ only depends on the fact that $g_1 \neq g_2$, and not on the randomness of the generators.

The forger may ask $\mathcal{A}$ several times to disavow (forged) signatures on m', and because there are only q possible signatures for m', we cannot allow him to try them all. If we restrict him to $\sqrt{q}$ attempts, then it is easy to see (by using the previous Lemma 5.5) that the probability that $\mathcal{A}$ can disavow all these $\sqrt{q}$ forged signatures is at least $1-\frac{1}{\sqrt{q}}$.

5.4.3. Confirmation protocol

If a signer wants to convince the recipient of the correctness of his signature $S=sign(SK,m)$ on m with public key $PK=pub(SK)$, they can perform Protocol 5.1 of Figure 5.3 (which has to be iterated l times).

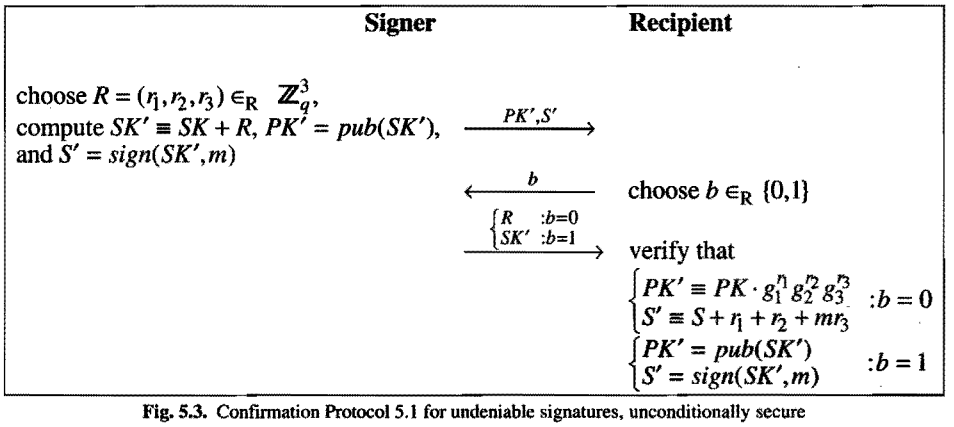


Fig. 5.3. Confirmation Protocol 5.1 for undeniable signatures, unconditionally secure for the signer.

Lemma 5.6. *Protocol 5.1 is a perfect zero-knowledge interactive protocol to convince the recipient that the signer knows a satisfying assignment for*

"I know a SK such that PK = pub(SK) and S = sign(SK,m)".

Proof. This protocol is a special case of the proof system for random self-reducible relations of [TW87]. This can easily be verified by using:
$N=(q,m)$, $X_N=(PK,S)$, $Y_N=SK$,

relation R_N on $X \times Y$: $((PK,S),SK) \in R_N \Leftrightarrow PK = pub(SK)$ and $S = sign(SK,m)$,
blinding: $blind(SK,R) = SK+R$, and
$A(PK,S,R) := (PK',S')$, where $PK' \equiv PK \cdot g_1^{r_1} g_2^{r_2} g_3^{r_3}$ and $S' \equiv S + r_1 + r_2 + mr_3$. ❑

5.4.4. Disavowal protocol

We assume that the presumed signer wants to disavow a forged signature SF on a message m. We will first show a simplified protocol. To do so, the signer and the recipient can perform the disavowal Protocol 5.2, which takes l' rounds. In each round the signer chooses randomly $R = (r_1,r_2,r_3) \in \mathbb{Z}_q^3$, computes

$$\begin{aligned} SK' &\equiv SK + R, \\ PK' &= pub(SK'), \\ S' &= sign(SK',m), \\ SF' &\equiv SF + r_1 + r_2 + mr_3, \end{aligned}$$

and issues commitments on PK',S' and SF' to the recipient. Then the recipient can choose one out of the three challenges:

C1. Signer must open the commitments on PK' and SF', and must reveal R,
C2. Signer must open the commitments on PK' and S', and must reveal SK',
C3. Signer must convince the recipient that $S' \neq SF'$, without revealing these numbers.

This is depicted in Figure 5.4, in which the ⬭ denotes commitments on numbers, and the three possible challenges are indicated.

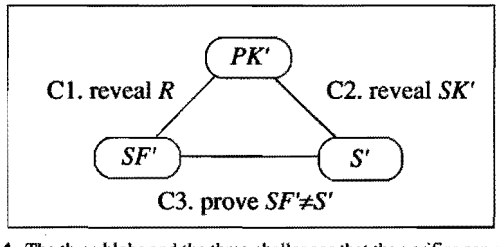


Fig. 5.4. The three blobs and the three challenges that the verifier can ask for.

For blobs, we will use the following efficient commitment scheme, which is unconditionally secure for the signer. Let $b \in \mathbb{Z}_q = \{0,1,\ldots,q-1\}$ be the value to be committed to, and let the blob be

$$\mathcal{B}(b,t) \equiv g^b h^t \pmod{p},$$

where $t \in \{0,1,\ldots,q-1\}$ and g,h are two generators of G_q such that $\log_g(h)$ is not known (this blob construction was simultaneously proposed in [Ped91]). If one created the blobs $\mathcal{B}(b_1,t_1)$ and $\mathcal{B}(b_2,t_2)$, then one can also compute and open the blobs $\mathcal{B}(b_1+b_2,\ t_1+t_2)$ and $\mathcal{B}(b_1-b_2,\ t_1-t_2)$. Hence the committed values in these blobs

can be added and subtracted, and thus it is sufficient in Challenge 3 to prove that the committed value $SF'-S'$ is nonzero. But even this difference may not be revealed, so the protocol needs the following blinding:

$$(\mathcal{B}(b,t))^r \equiv g^{br}h^{tr} \equiv \mathcal{B}(br,tr) \pmod{p}.$$

If $r \in_R \mathbb{Z}_q^* = \{1,\ldots,q-1\}$, then br is uniformly distributed in $\mathbb{Z}_q$ for $b \neq 0$, and br is always 0 if $b=0$. Because $\mathcal{B}(b,t)$ and $\mathcal{B}(br,tr)$ together uniquely determine r, the signer may not reveal both $\mathcal{B}(b,t)$ and the pair br, tr at the same time; otherwise, b is uniquely determined. Hence an additional level of commitments is needed for the disavowal protocol, as shown in Figure 5.5.

The numbers SF', S' and $r(SF'-S')$ are elements from $\mathbb{Z}_q$ and thus we can compute the blobs $\mathcal{B}(SF',*)$, $\mathcal{B}(S',*)$ and $\mathcal{B}(r(SF'-S'),*)$ by using the blob construction above. But these blobs are elements from G_q, and we cannot compute the blobs $\mathcal{B}(\mathcal{B}(SF',*),*)$, $\mathcal{B}(\mathcal{B}(S',*),*)$ and $\mathcal{B}(\mathcal{B}(r(SF'-S'),*),*)$ by the same blob construction. Thus for the outer commitments we need an efficient embedding $\lambda: G_q \mapsto \mathbb{Z}_{q'}$ (for a prime $q' \geq q$). For instance, if we have that $p=2q+1$, $G_q = \mathbb{Z}_p^* / \{\pm 1\}$, which is represented by $\{1,2,\ldots,q\}$, then we can use the embedding:

$$\lambda(x) = \begin{cases} x & \text{if } x \in \{1,\ldots,q-1\}, \\ 0 & \text{if } x = q. \end{cases}$$

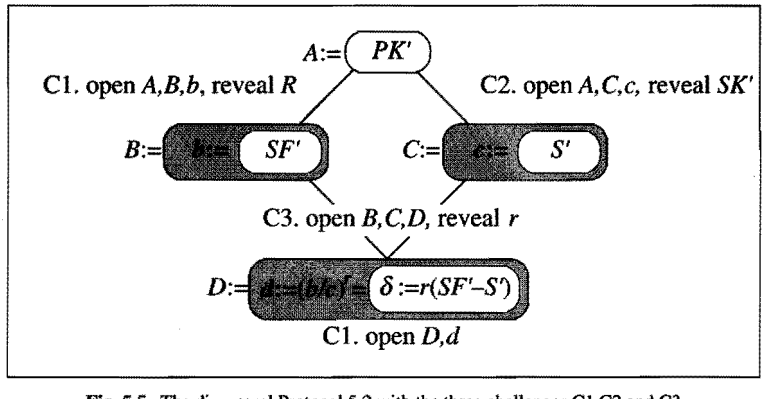


Fig. 5.5. The disavowal Protocol 5.2 with the three challenges C1,C2 and C3.

Disavowal protocol 5.2. (see Figure 5.5)

Repeat l' times:

The signer computes PK', SF' and S' as indicated in the beginning of this subsection, and chooses $r \in_R \{1,\ldots,q-1\}$. He creates commitments A,b,B,c,C as indicated in Figure 5.5, computes $\delta \equiv r(SF'-S') \pmod{q}$, $d \equiv (b/c)^r \pmod{p}$, and creates the commitment D.

The recipient receives A, B, C, D and can choose one of the three challenges:

C1. Signer opens the commitments A, B, b, D, d and reveals R.
Recipient verifies the openings and that $\delta \neq 0$, $SF' \equiv SF + r_1 + r_2 + mr_3 \pmod{q}$, and that $PK' \equiv PK \cdot g_1^{r_1} g_2^{r_2} g_3^{r_3} \pmod{p}$.

C2. Signer opens the commitments A, C, c and reveals SK'.
Recipient verifies the openings and that $PK' = pub(SK')$ and $S' = sign(SK', m)$.

C3. Signer opens the commitments B, C, D and reveals r.
Recipient verifies the openings and that $d \equiv (b/c)^r \pmod{p}$.

Lemma 5.7. *This disavowal Protocol 5.2 is a perfect zero-knowledge interactive protocol to convince the recipient that the presumed signer either knows a satisfying assignment for*

"I know a SK such that PK = pub(SK) and SF ≠ sign(SK,m)",

or can break the commitment scheme.

Sketch of the proof.

Completeness. It can easily be verified that if the signer knows such an SK and behaves correctly, then he can answer all three challenges.

Soundness. We will prove that if the signer answers all three challenges, then the recipient can either open a commitment in two ways, or he knows that $SF' \neq S'$. If the signer opens a commitment in two ways, we have proven the statement. So suppose that he answers everything correctly: he reveals numbers SK', S', SF', d, r, δ, x, y, z that satisfy $b = \mathcal{B}(SF', x)$, $c = \mathcal{B}(S', y)$ and $d = \mathcal{B}(\delta, z)$. If we define $\delta^* := r(SF' - S')$ and $z^* := r(x - y)$, then $\mathcal{B}(\delta^*, z^*) = \mathcal{B}(\delta, z)$. If $(\delta^*, z^*) \neq (\delta, z)$, then the recipient has a commitment that can be opened in two ways, and if $(\delta^*, z^*) = (\delta, z)$, then it is easy to see that $SF' \neq S'$.

Perfect zero-knowledge. We will show that each of the three cases can be simulated.

C1. Choose R, $\delta \neq 0$ and C randomly, and compute $PK' \equiv PK \cdot g_1^{r_1} g_2^{r_2} g_3^{r_3} \pmod{p}$, $SF' \equiv SF + r_1 + r_2 + mr_3 \pmod{q}$, and commitments A, b, B, d, D.

C2. Choose SK', B and D randomly, and compute $PK' = pub(SK')$, $S' = sign(SK', m)$, and commitments A, C, c.

C3. Choose b, c, $r \neq 0$ and A randomly, and compute $d = (b/c)^r$, and commitments B, C, D.

It can be easily shown that in each case this simulator can answer the corresponding challenge, and that in each case the joint probability distribution on the values visible for the recipient is the same for the signer and the simulator. ❑

5.5. Convertible undeniable signatures unconditionally secure for the signer

Convertible undeniable signatures were introduced in [BCDP90]. Briefly, these signatures allow the signer of undeniable signatures to change his signatures to ordinary digital signatures. In Section 5.4 an undeniable signature scheme was presented in which the signer is unconditionally secure. This section combines the ideas of [BCDP90] with Sections 5.3 and 5.4 by constructing an undeniable signature scheme that is unconditionally secure for the signer and has the property that the signer can convert the fail-stop undeniable signatures to plain fail-stop signatures.

Let p, q, g and h be as in Section 5.2. The secret key of person $\mathcal{A}$ is

$$SK = (x_1, x_2, y_1, y_2) \in \mathbb{Z}_q^4,$$

and the corresponding public key is

$$PK = pub(SK) = (p_1, p_2) = (g^{x_1}h^{x_2}, g^{y_1}h^{y_2}).$$

The undeniable signature of $\mathcal{A}$ on the message $m \in \mathbb{Z}_q$ is

$$\sigma_1 = sign(SK,m) \equiv x_1 + my_1 \pmod{q}.$$

This signature is undeniable, because given PK, the signature is just a random number in $\mathbb{Z}_q$. The signature scheme is unconditionally secure for the signer, because given σ_1, it is impossible for a forger with unlimited computing power to construct $\sigma_1' = sign(SK,m')$ for a new message $m' \neq m$. This follows from the same arguments as in Lemma 5.1 and 5.2. We mention that $\mathcal{A}$ can convert this signature to a fail-stop signature by publishing

$$\sigma_2 \equiv x_2 + my_2 \pmod{q},$$

which changes the undeniable signature into the fail-stop signature of Section 5.4.

Confirmation protocol

To verify the signature σ_1 on a message m, $\mathcal{A}$ and the recipient compute $u \equiv p_1 p_2^m g^{-\sigma_1} \pmod{p}$, and $\mathcal{A}$ convinces him with a zero-knowledge protocol that he knows a number σ_2 such that $u \equiv h^{\sigma_2} \pmod{p}$. Perfect zero-knowledge protocols for this problem are well known (e.g. [CEvdG87]), so it can be proven that:

Theorem 5.8. *There is a perfect zero-knowledge protocol for convincing someone of having σ_2 satisfying $u \equiv h^{\sigma_2} \pmod{p}$.*

Disavowal protocol

Disavowal of these signatures is slightly more complicated. A given number $\sigma \in \mathbb{Z}_q$ is not $\mathcal{A}$'s signature on m if

$$p_1 p_2^m \not\equiv g^{\sigma} h^{\sigma_2} \pmod{p},$$

where $\sigma_2 \equiv x_2 + m y_2 \pmod{q}$. $\mathcal{A}$ can therefore convince someone that σ is not his signature by convincing him that he knows numbers s and t such that

$$p_1 p_2^m \equiv g^s h^t \pmod{p}, \quad \text{and} \quad \sigma \neq s.$$

(because if σ was his signature this would mean that $\mathcal{A}$ knew $\log_g h$). The perfect zero-knowledge protocol presented in Subsection 5.4.4 can be used as follows:

$$\begin{array}{rlrl} SK &= (s,t), & SK' &= (s+r_1, t+r_2), \\ PK &\equiv p_1 p_2^m \pmod{p}, & PK' &\equiv PK \cdot g^{r_1} h^{r_2} \pmod{p}, \\ S &= s, & S' &\equiv s + r_1 \pmod{q}, \\ SF &= \sigma, & SF' &\equiv \sigma + r_1 \pmod{q}. \end{array}$$

(Note that in this disavowal protocol we use the definition $sign(SK,m) = sign((s,t),m) := s$.)

5.6. Applications

In [PW91] an application of fail-stop signatures is suggested, called the *3-phase protocol*. Customers in an electronic payment system would use fail-stop signatures when signing a request to the bank (for example for the withdrawal of money). These signatures have the advantage over ordinary digital signatures that the customers need not worry about the bank (which normally has more computing power than the customer) being able to break the underlying assumptions of the signature scheme. In this 3-phase protocol for making such requests, the customer only has to sign a single bit with a fail-stop signature.

If the signature scheme of Subsection 5.3.3 is used, this 3-phase protocol can be avoided and the customer needs only to send a single message to the bank in order to sign the request.

5.7. Some open problems

For each message, the public key of the fail-stop signature presented in Subsection 5.3.3 is approximately 1000 bits (assuming a modulus of size 500 bits), the secret key is approximately 2000 bits, and the signature of a 500-bit message is 1000 bits (long messages can first be hashed into 500-bit messages). By using a hash function, we can reduce the size of the public key (see Remark 5.1 in Subsection 5.3.3). Unknown is

whether our construction for fail-stop signatures is the most efficient construction, and if not, what the most efficient construction will be.

6

Efficient offline electronic checks[‡]

6.1. Introduction

In a "traditional" payment system, the bank creates special objects (such as coins, paper cash, checks, or credit cards) that are worth money: they can be used to pay a shop in return for some goods, or to pay a person. The money can also be stored in some account at the bank. It is assumed that it is infeasible for users to create these objects themselves. In this system, the user's privacy as regards the bank is not very high: the bank can trace paper cash in principle by serial numbers; and by using checks or credit cards, the bank knows who spent what amount and where.

In an electronic payment system, information (in the form of specially created numbers) is used as money, and this information is stored by the users in their handhold computer. These numbers must first be created either by the bank or by the bank and the user together, and it is assumed that it is infeasible for the users to create electronic money without the cooperation of the bank. But a user can copy numbers easily; and because a copy of valid electronic money is still valid electronic money, he can spend this copy at another shop (this is called *double spending*). This second shop cannot detect that this money has already been used. So before the shop accepts the money, it contacts the bank, which can verify that that electronic money has not been used before (this is called an *online* connection). Only then will the shop accept the money. Electronic coins are called *unconditionally untraceable* (the user has *unconditional privacy*), if for each point in time the bank has no information (in the Shannon sense) to link any payment until then to any creation of electronic money until then, even with

[‡] This chapter is based on the paper "Efficient Offline Electronic Checks" by Bert den Boer, David Chaum, Eugène van Heyst, Stig Mjølsnes and Adri Steenbeek, which appeared in *Advances in Cryptology-EUROCRYPT '89*, J-J. Quisquater and J. Vandewalle eds., Lecture Notes in Computer Science 434, Springer-Verlag, pp. 294-301.

unlimited computing power (except when only one person has created and spent money).

So in an electronic payment system three types of parties are involved: a *bank*, a *user* (to be called *Alice*), and a *shop*, and there are three possible transactions between them:

- *Withdrawal* transaction between the bank and Alice: this is the creation phase of the money and is done partly by Alice and partly by the bank. The bank decreases Alice's account by the value of the electronic money created.
- *Payment* transaction between Alice and the shop: Alice uses the electronic money to pay at a shop.
- *Deposit* transaction between the shop and the bank: the shop transmits the electronic numbers it has received from Alice to the bank, which verifies that these numbers are correct and that they have not yet been spent. If so, the bank increases the balance of the shop.

An online connection between the shop and the bank is very expensive, so we want a payment system that replaces this online connection by an *offline* connection (this means that the shop contacts the bank only occasionally, say once a day or once a week, to transmit the numbers received). The first such system can be found in [CFN88], which has the following properties:

- The shop does not need an online connection with the bank.
- During payment the user does not have to do computations.
- The withdrawal and payment of a check are unconditionally unlinkable.
- The system can be used for electronic *checks*. This means that the money can be used for many values up to a certain maximum.

These properties seem to cause two major problems. The first is that a user can reuse valid money and the second shop cannot detect this cheating (because of the offline connection). After both deposits the bank detects this double spending, but it cannot trace the user because her privacy was protected unconditionally. This problem is ingeniously solved in [CFN88] by creating the money in such a way that the privacy of the user is protected unconditionally if she uses the money only once (as she is supposed to), and that the bank can trace the user (with high probability) if she reuses that money (see Figure 6.1).

The second major problem results from the use of checks. The amount for which a check is spent, cannot be paid later to the bank (because of the unconditional privacy requirement), and this amount is not known at the time of withdrawal. Therefore the check has to be issued for the maximum value and must be able to be spent for many lesser values. Later on the user must request the bank to *refund* the unspent part of the check, and this is the fourth transaction needed in their payment system (see Figure 6.2).

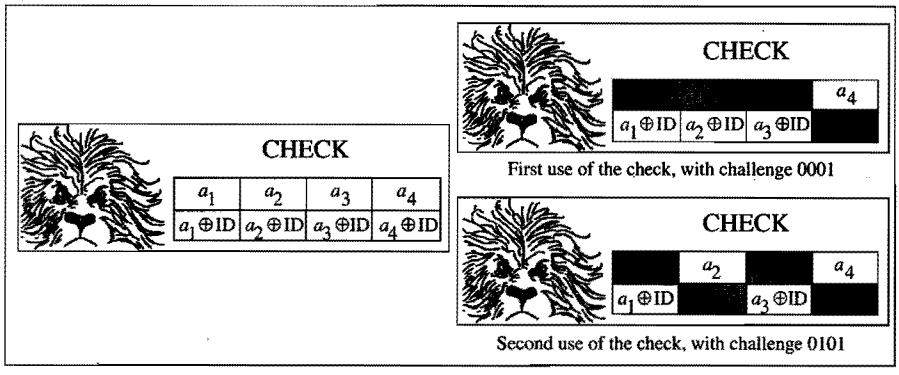


Fig. 6.1. An illustration of how to prevent double spending in the offline case. A check is created as a 2x4 matrix, in which the numbers a_i are randomly chosen by Alice and in which ID is her account number. During a payment, she will receive from the shop a random 4-bit string and she will reveal from each column of the check only one number, according to the challenge. Hence the identity of the user is still protected unconditionally.

But suppose that she spends this check twice, and that the first shop gives challenge 0001 and the second shop gives challenge 0101. The numbers revealed are indicated in both cases. After both deposits the bank can trace Alice because $a_2 \oplus (a_2 \oplus ID) = ID$. In this small example the second shop will generate with probability 15/16 a different challenge than the first shop, and thus the bank will be able to trace Alice with probability 15/16. This probability can be made arbitrarily close to 1 by using a wider matrix.

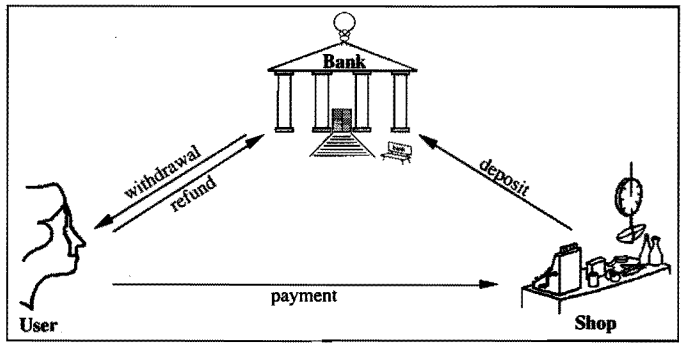


Fig. 6.2. The four transactions in an electronic payment system for checks.

In this chapter we present a payment system that improves the efficiency of [CFN88]; also, some functionalities are added. In the next section we give the setting of our new payment system and the notation used. In Section 6.3 we describe the four transactions, which are compared with [CFN88] in Section 6.4. In Sections 6.5 up to 6.9 we discuss the possibility of passing a check from person to person, the storage of the checks, a demo of this payment system for an Apple Macintosh computer, the suggested size of the parameters used, and some improvements of our scheme. In Section 6.10 we

analyze several ways of cheating. We end with a brief overview of offline payment systems that appeared in the literature after this paper, and some open problems.

6.2. Setting and notation

The underlying scheme of this payment system is the RSA-scheme. All calculations are done modulo N, the factorization of which only the bank knows (in our formulas we will omit this modular reduction). The check is a product of $k/2$ terms and these terms are ordered lexicographically: the first j terms (called *denomination part*) are used for the amount to be spent in a shop and the last $k/2-j$ (called *challenge part*) to prevent the check from being spent twice. The j terms in the denomination part have denominations \$1 (one dollar), \$2,..., $\$2^{j-2}$, $\$2^{j-1}$ respectively (or 1¢, 2¢,..., \$10.24,..., $\$2^{j-1}/100$), and thus each check can be spent for each integral value up to $\$(2^j-1) = \$(2^0+2^1+\ldots+2^{j-1})$. As in the original paper [CFN88], let

- k be a security parameter that is an even number,
- j be the number of denomination bits in a check (we assume that $5 \le j \le 14$; for other values of j see Section 6.8),
- f,g be two-argument, collision-free one-way functions, with g such that if its first argument is fixed, the mapping is q-to-1 from the second argument onto the range (for some integer q),
- h be an injective one-way function with one argument,
- l be an injective k-argument one-way function,
- ID_i be Alice's account number concatenated with a counter,
- $\oplus$ denote bitwise exclusive-or, and
- $\|$ denote concatenation.

Essentially such a check is constructed as follows: for each term she randomly chooses integers a,b,c, and d, and computes the term as $f(\,g(a\|b,c),\ g(a\oplus ID,d))$, thus each of these terms can be represented by the tree of Figure 6.3.

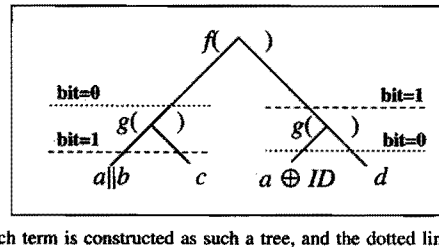


Fig. 6.3. Each term is constructed as such a tree, and the dotted lines indicate the numbers that must be revealed if the corresponding bit is "1" or "0".

During payment, Alice will receive one bit for each term (challenge and

denomination) of her check, and according to this bit she has to reveal some numbers of how this term is constructed, see Figure 6.3. For instance, if this bit is "1", then Alice reveals the numbers $a\|b$, c, and $g(a \oplus ID, d)$. If later on she wants to request refund for a denomination bit, she has to reveal the corresponding number b.

Thus, if Alice wants to both spend and request refund for the same denomination bit, she reveals the same number b twice and will be caught cheating by the bank. And if Alice reveals the answers to both challenges 0 and 1, then her identity can be computed as $a \oplus (a \oplus ID) = ID$. This is a mathematical implementation of the idea of Figure 6.1.

Alice creates these terms herself, so a *cut-and-choose protocol* will be included in the withdrawal transaction: Alice creates twice as many terms as needed and issues them to the bank. The bank randomly chooses half of these terms and Alice has to reveal their construction (i.e., she has to reveal $a\|b$, c, $a \oplus ID$ and d for each of these terms). The bank verifies these constructions of the opened terms, and if they are all correct the bank will assume that (nearly) all the other terms are also correct. She will sign these unopened terms, so that they become valid money.

For *coins* (i.e., checks with only one denomination), the user has unconditional privacy, because

- The numbers a and $g(a \oplus ID, d)$ together contain no Shannon information about ID, that is, each number is equally likely to be ID (because of the definition of the function g).
- The numbers $a \oplus ID$ and $g(a\|b, c)$ together contain no Shannon information about ID.
- If all users withdraw money at the same time, and spend it at the same time, the bank cannot link a payment with a withdrawal, because of the blinding used. (If one person withdraws money and spends it immediately, the bank can link withdrawal to payment, despite the blinding.)

For checks the privacy of each user is protected "almost" unconditionally: a user can request refund for several checks at once, to keep the bank from learning the amount spent for each check. Payments and refunds are unlinkable, except by the little that can be learned from the total number of each type of unspent denomination.

In the next section the construction of the check and the four transactions will be explained in more detail.

6.3. Transactions

As mentioned, the payment system consists of three parties (bank, user Alice and a shop) and four transactions: withdrawal, payment, deposit, and refund. Each of these transactions is a protocol between two of the three parties. The transactions do not need

to be in this order: after withdrawal and payment, the refund of the check involved can occur earlier or later than its deposit. Alice can also first request refund for part of the check and later spend the rest of the check in a shop (see Figure 6.4).

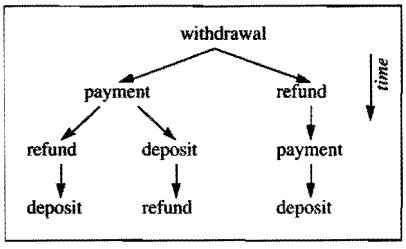


Fig. 6.4. The three possible sequences of four transactions for one check.

In figures 6.5, 6.6, 6.7, and 6.8, describing these four transactions, the symbols, , , , and are used to indicate computation, trivial computation (e.g. table lookup), verification, transmission, and writing in an archive list, respectively.

6.3.1. Withdrawal Transaction

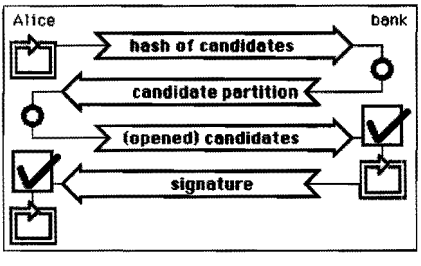


Fig. 6.5. Withdrawal of a check.

(1) Alice chooses at random: r_i, a_i, b_i, c_i, d_i, e_i, $(1 \le i \le k)$ and computes:

$$\begin{aligned} x_i &= g(a_i \| b_i, c_i), \\ y_i &= g(a_i \oplus ID_i, d_i), \\ M_i &\equiv f(x_i, y_i) && \text{(called } \textit{major term}\text{)}, \\ m_i &\equiv h(g(b_i, e_i)) && \text{(called } \textit{minor term}\text{)}, \\ \alpha_i &\equiv M_i^{3^j} \cdot m_i^{17} \cdot r_i^{17 \cdot 3^j} && \text{(called } \textit{blinded candidate}\text{)}. \end{aligned}$$

These blinding factors r,a,b,c,d,e are used as follows:

- r: to make the withdrawal and payment unlinkable,
- a: to hide the identity of the user in a check unconditionally,
- c,d,e: to hide the first argument in the function g unconditionally,
- b: to prevent users from spending a denomination bit in a shop *and* requesting for refund for the same denomination bit.

All these computations can be made before connection with the bank, and during the connection with the bank only the hash value $l(\alpha_1,\alpha_2,\ldots,\alpha_k)$ is sent.

(2) The bank partitions the integers $1,\ldots,k$ randomly between two unordered sets S_o and S_c, both of $k/2$ elements. The partitioning is then sent to Alice.

(3) Alice orders the elements in the partition S_c by the M_i value of the corresponding candidate, forming the ordered set T_c (so T_c is a permutation of S_c). This set T_c, the α_i's of the elements in T_c, and the r_i, a_i, b_i, c_i, d_i, e_i of the candidates in S_0 are sent to the bank. The candidates in S_0 are said to be "*opened*".

(4) The bank verifies that the hash of the blinded candidates revealed equals $l(\alpha_1,\alpha_2,\ldots,\alpha_k)$ and that every element of the opened partition is correctly formed. (The opened partition can now be discarded by both parties, because it was only used for the cut-and-choose protocol.) The bank takes a random integer R and computes

$$D \;:\equiv\; \prod_{i=1}^{j}(\alpha_{t(i)})^{1/17\cdot 3^{j+1-i}} \cdot \prod_{i=j+1}^{k/2}(\alpha_{t(i)})^{1/17}\cdot R^{1/3^j},$$

where $t(i)$ is the i^{th} element of T_c; sends D and R to Alice; and reduces Alice's account by $\$(2^j-1)$, the maximum value of the check. The bank also stores R with Alice's account number in some list.

(5) Alice verifies the validity of this signature D by testing whether

$$D^{17\cdot 3^j} \;\stackrel{?}{\equiv}\; \prod_{i=1}^{j}(\alpha_{t(i)})^{3^{i-1}} \cdot \prod_{i=j+1}^{k/2}(\alpha_{t(i)})^{3^j}\cdot R^{17}.$$

Let α,β be such that $3^j\alpha + 17\beta = 1$. From this signature D she computes the following numbers:

$$\begin{aligned}
\tilde{D} \;&:\equiv\; D\cdot\prod_{i=1}^{j}(r_{t(i)})^{-3^{i-1}}\cdot\prod_{i=j+1}^{k/2}\left((r_{t(i)})^{-3^j}\cdot(m_{t(i)})^{-1}\right)\\
&\equiv\; \prod_{i=1}^{j}\left((M_{t(i)})^{3^{i-1}/17}\cdot(m_{t(i)})^{1/3^{j+1-i}}\right)\cdot\prod_{i=j+1}^{k/2}(M_{t(i)})^{3^j/17}\cdot R^{1/3^j},\\
C \;&:\equiv\; \left((\tilde{D})^{3^j}\cdot\prod_{i=1}^{j}(m_{t(i)})^{-3^{i-1}}\cdot R^{-1}\right)^{\alpha}\cdot\prod_{i=1}^{j}(M_{t(i)})^{\beta\cdot 3^{i-1}}\cdot\prod_{i=j+1}^{k/2}(M_{t(i)})^{\beta\cdot 3^j}\\
&\equiv\; \prod_{i=1}^{j}(M_{t(i)})^{3^{i-1}/17}\cdot\prod_{i=j+1}^{k/2}(M_{t(i)})^{3^j/17},\\
C' \;&:\equiv\; (\tilde{D})^{17}\cdot\prod_{i=1}^{j}(M_{t(i)})^{-3^{i-1}}\cdot\prod_{i=j+1}^{k/2}(M_{t(i)})^{-3^j}\\
&\equiv\; R^{17/3^j}\cdot\prod_{i=1}^{j}(m_{t(i)})^{17/3^{j+1-i}}.
\end{aligned}$$

Number C is called *signed check* and will be used during payment, and number C' is called *refund part* and will be used during refund. The computations of C and C' can be done more efficiently than the formulas suggest and can be accomplished any time before payment.

6.3.2. Payment Transaction

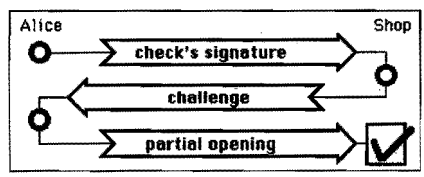


Fig. 6.6. Payment of a check.

Alice can use the signed check C to pay in a shop. The first j terms of a check (remember that a check is the product of $k/2$ *ordered* elements and each shop can verify this ordering) have denominations \$1, \$2,..., $\$2^{j-2}$, $\$2^{j-1}$ respectively. So, every integral amount smaller than $\$(2^j-1)$ can be paid with C; let $(w_1,\ldots,w_j)$ be the binary representation of the amount of payment (so the amount is $\$ \sum_{i=1}^{j} w_i 2^{i-1}$).

(1) Alice gives C to the shop.

(2) The shop generates a binary challenge-vector $(w_{j+1},\ldots,w_{k/2})$ and sends it to Alice.

(3) Alice gives the following partial opening of the check C $(1 \le i \le k/2)$ (see also Figure 6.3):

if w_i=1, she reveals the corresponding $a_i \| b_i$, c_i, y_i;

if w_i=0, she reveals x_i, $a_i \oplus ID_i$, d_i.

(4) The shop verifies the partial opening, the check's signature, and the ordering of the M_i's in C. If these are "OK", then he accepts this check as valid money.

6.3.3. Deposit Transaction

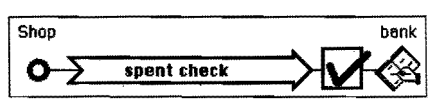


Fig. 6.7. Deposit of a check.

(1) The shop sends to the bank: C, the vector $\mathbf{w}=(w_1,..,w_{k/2})$ (amount and challenge vector), and the partial opening.

(2) The bank verifies the signature, the partial opening, and the ordering of the M_i's in C, just as the shop did.

(3) The bank has two searchable archive lists: one of spent checks and one of revealed minor terms. In the first list the bank stores the number C and the corresponding partial opening a_i or $a_i \oplus ID_i$ $(1 \le i \le k/2)$. In the last list he stores the revealed b_i's. The bank consults the searchable lists to be sure, perhaps by sorting them, that no b_i has already been refunded and that no check has been spent twice. When double spending of a check is found, the ID_i can be reconstructed from any difference in the corresponding vectors $\mathbf{w}$ and $\mathbf{w}'$, thereby revealing the cheater's

account number (see Figure 6.1). There will be a difference in the last $k/2-j$ bits of $\mathbf{w}$ and $\mathbf{w}'$ with probability $1-2^{j-k/2}$.

6.3.4. Refund Transaction

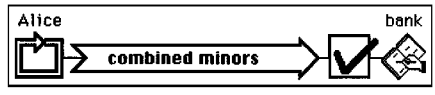


Fig. 6.8. Refund of a check.

(1) After each payment, but before the refund, the minor terms of several checks are accumulated by Alice and $g(b_i,e_i)$ is computed for each $w_i=1$. Alice sends the bank the product of the C''s, the R's, and in addition the $g(b_i,e_i)$ for each denomination spent, and the b_i,e_i for the denominations not spent.
(2) The bank verifies the opened minor terms and their signature C', much in the way that checks are verified. The bank also verifies that the R's were stored with Alice's account number.
(3) The bank verifies that the b_i's are not listed and stores them on its list of revealed minor terms together with Alice's account number, to prevent Alice from spending a denomination bit after the refund.

Note that in case of requesting refund for multiple checks at once, Alice keeps the bank from learning the amount spent for each check; only the total number of each type of unspent denomination is revealed.

6.4. Comparison with [CFN88]

In our payment system three basic changes from [CFN88] were made to improve efficiency:

- During the withdrawal transaction Alice initially sends to the bank *not* the candidates for the check, but the value of the one-way function l with the candidates as its arguments. Hence she avoids sending half of the candidates.
- All the minor terms are signed together (still with different exponents) rather than separately. Hence the bank has only to give *one* signature D per check, while in [CFN88] the bank has to give $j+1$ signatures per check.
- In [CFN88] each major and minor term is blinded separately, while in our scheme Alice sends a blinded product of the major and minor terms; hence she needs only half as many blinding factors, only half as many bits are transmitted, and the bank makes only one signature. When Alice receives the signed check, though, she must do some calculations to obtain the signed check and the refund part (i.e., to separate the major and minor terms). An additional one-way

function h is introduced for this.

If we compare our new scheme with [CFN88] for the values k=40 and j=10, then our new system saves 91% on the number of the signatures, saves 41% on the (other) multiplications, saves 73% on the divisions, and saves 33% on the bit transmissions.

6.5. Transferability

Transferability means that a person Alice (payer) can pay a person Bob (called *payee*) electronic money, and Bob can use this money to pay another person or shop. In [CFN88] this concept is not mentioned, and in [OO89] the first transferable offline electronic payment system is presented. Their idea is that Bob gives Alice a challenge (as usual) and receives the numbers corresponding to this challenge and a note signed by Alice saying that that money now belongs to Bob. When Bob wants to spend this money in a shop, he first transmits the transcript of the protocol with Alice, and after receiving the challenge from the shop, he transmits the corresponding response.

But this system can easily be cheated: Alice can give the same money to several persons, by asking the second payee (who cooperates with Alice) to issue her the same challenge as the first payee. Both payees have a correct signed note, so they both can spend the money without difficulty. The bank later detects this cheating, but can trace neither Alice nor the payees.

In [Ant90] another way of constructing transferable offline cash is presented. Here the basic concept needed to realize transferability is that of "checks" with no denomination terms in them, that is, checks with no value. These "empty checks" can be obtained freely from the bank. The challenge Bob issues to Alice is the outcome of the one-way function f on Bob's empty check. In this way the check paid to Bob is connected to one of his empty checks. When he pays a shop with his new "check" (or rather, coin, since it can only be spent for the amount for which he received it), he not only issues the check data received during the previous payment transaction, but also uses the empty check to answer a challenge issued by the shop (see Figure 6.9). Hence Bob cannot double-spend this new check, or else he will with high probability reveal his identity to the bank.

Suppose that Alice wants to give the check C_1 to two different payees, instead of only one. Then both payees must use the same empty check C_2 (and thus the same challenge ch_1), otherwise Alice's identity will be revealed to the bank after both deposits (see Figure 6.9). Hence both payees must know the construction of this empty check C_2 in order to be able to respond to the shop's random challenge. But this empty check is created correctly, so it contains the identity of one of the payees. Thus if they both spend this check, they will each receive a different challenge (with high probability), and the payee's identity will be revealed to the bank. Therefore no-one is

likely to cooperate in this way.

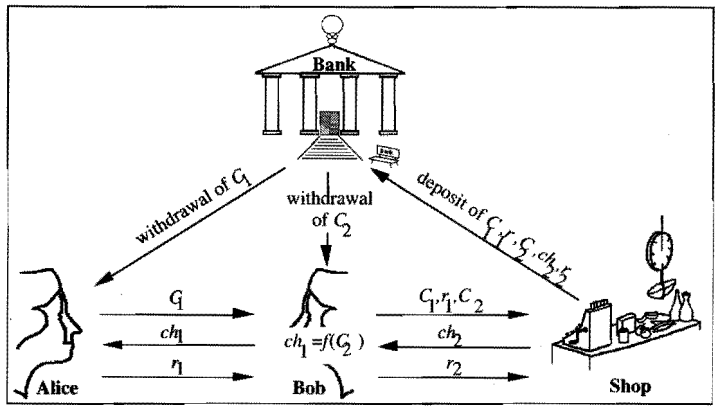


Fig. 6.9. Transferability in our payment system.

6.6. Storage

For each check Alice has to store a lot of numbers in her computer. The numbers she will store depend on which transactions already took place, and on the fact that memory can be exchanged for computation. For instance, if Alice stores a_i, b_i, c_i, d_i, then she can compute M_i as $f(g(a_i \| b_i, c_i), g(a_i \| b_i, c_i))$, but she can also store M_i. At the cost of one memory location, then, she need not do computations. In the following overview we use the "normal" time order (so refund after payment, see Figure 6.4), and we indicate only the numbers that really need to be stored in her computer. Other numbers can be recomputed:

- before withdrawal: r_i, a_i, b_i, c_i, d_i, e_i for each candidate (i=1,...,k);
- after withdrawal: C, C', R and a_i, b_i, c_i, d_i, e_i for each unopened candidate ($i = t(1),\ldots,t(k/2)$);
- after payment: C', R and b_i, e_i for each unopened candidate ($i = t(1),\ldots,t(k/2)$).

By storing these numbers, Alice has to recompute the integers x_i, y_i, and $g(b_i, e_i)$ before payment or refund.

The memory that becomes available after a transaction can be used to store the numbers of the candidates needed for a new check. This allows Alice to do all computations before connecting the bank for a withdrawal.

6.7. Demo

Hans Beuze and Peter Sliepenbeek implemented the payment system described in the previous sections on an Apple Macintosh (see Figure 6.10), and Adri Steenbeek wrote the numerical part of it. Diskettes containing this demo are available from the authors. There is also an earlier version available for an IBM-PC.

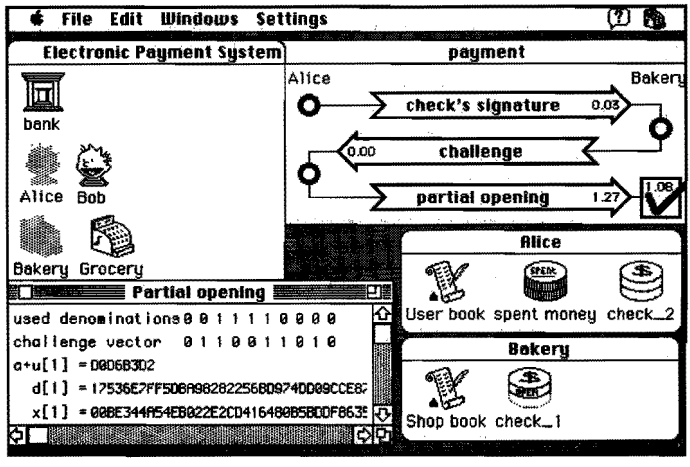


Fig. 6.10. A screen dump of the Macintosh demo of this payment system.

6.8. Parameter values

We assume that the RSA-modulus N has 512 bits. We can of course take all integers to have 512 bits, but this is not necessary for the security or the efficiency.

The variables used for unconditionally hiding information (c,d,e,r) are taken to be 512 bits. The size of b depends on the number of users: if twice in the lifetime of this payment system the same b is generated, then the second user loses the money of the related denomination bit. So if b has 128 bits, then the probability of this happening is very close to zero. The identity ID can have 32 bits, and thus also its blinding factor a. The outcome of the collision-free function g can be 128 bits (64 bits is too small). The system will be used for low-value payments, so we can use $j \leq 15$ (allowing check values $\leq$ \$327.68). By taking k=100, the shop has a challenge of 35 bits.

Thus for a practical and reliable system we suggest the following parameter values:

k	:	100 bits,
ID_i,a_i	:	32 bits,
b_i	:	128 bits,
c_i,d_i,r_i,e_i,N	:	512 bits,
f	:	128×128→512 bits,

g : 128×512→128 bits,
h : 128→512 bits,
j : 15 bits.

6.9. Improvements

6.9.1. Combining challenge and denomination bits

In our scheme each denomination bit and each challenge bit are in separate terms, but a challenge bit and a denomination bit can be combined into one term. The advantage of doing this is that fewer computations have to be performed, and fewer bits transmitted, during the protocols. If $j = k/4$, we can create a symmetric check: it no longer consists of a separate denomination and a challenge part. To do so, Alice chooses during the withdrawal transaction at random: r_i, a_i, b_i, c_i, d_i, e_i ($1 \le i \le 3k/4$), and computes:

$$\begin{aligned} x_i &= g(b_i, c_i), \\ y_i &= g(b_i, d_i), \\ v_i &= g'(a_i, x_i), \\ w_i &= g'(a_i \oplus ID_i, y_i), \\ M_i &\equiv f(v_i, w_i), \\ m_i &\equiv h'(h(b_i, e_i)), \\ \alpha_i &\equiv M_i^{3^j} \cdot m_i^{17} \cdot r_i^{17 \cdot 3^j}. \end{aligned}$$

During the cut-and-choose operation $k/2$ blinded candidates have to be opened, so a check consists of $k/4$ terms. During the payment transaction, let $(\pi_1,\ldots,\pi_{k/4})$ be the binary representation of the amount of payment, and $(\chi_1,\ldots,\chi_{k/4})$ be the binary challenge-vector. Then Alice gives a partial opening of the check, according to Table 6.1.

χ_i	π_i	revealed numbers
0	0	$v_i, a_i \oplus ID_i, y_i$
0	1	$v_i, a_i \oplus ID_i, b_i, d_i$
1	0	a_i, x_i, w_i
1	1	a_i, b_i, c_i, w_i

Table 6.1. The revealed numbers for the payment and challenge bits, if these are combined into one term.

The major term M_i can be expressed by the tree in Figure 6.11 (the indices i are omitted), in which the dotted lines indicate the numbers revealed during the payment transaction: start in each of the main subtrees at a certain level (which depends on the challenge bit), and if the payment bit is a one, go one level deeper in one of the main subtrees.

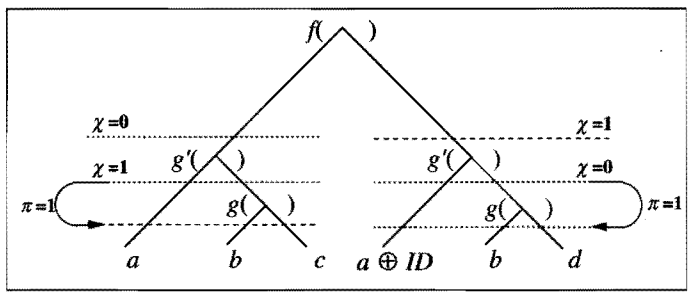


Fig. 6.11. An expression of the major term and its opening, which depends on the payment bit π and the challenge bit χ.

6.9.2. More denomination bits per check

By using the same idea as in Subsection 6.9.1, the number of denomination bits in the check can be doubled with little additional cost. The minor term is constructed as $m \equiv h(g(b_1,e_1), g(b_2,e_2))$ and the major term as the tree in Figure 6.12. Thus number b_1 is revealed if payment bit$_1$ is 1, and b_2 is revealed if payment bit$_2$ is 1.

Using this method one may want to put all denomination bits into one term for efficiency. But then there are problems with the security of the system, due to the cut-and-choose protocol: suppose Alice creates k' terms and exactly one of them is wrongly created. If she has to open $k'-1$ terms, then with probability $1/k'$ she receives the signature on the wrongly created term, so she can both spend and request refund for this check, without being caught cheating.

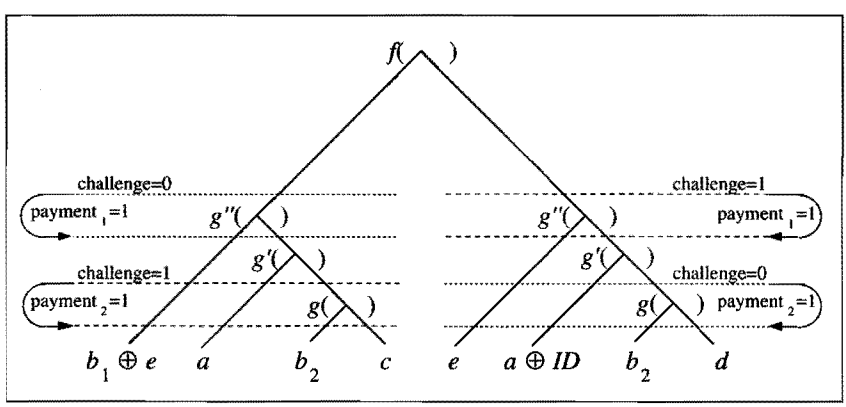


Fig. 6.12. An expression of the major term with two denominations (represented by b_1 and b_2). The way to open this major term, which depends on the payment bits and the challenge bit, is also indicated.

6.9.3. Anonymous refund

In the payment system presented here, the bank puts a random number R in the user's check in order to link the withdrawal to the refund, or, better: to link the refund to the user's correct identity. In order to have *anonymous refund*, the payment system can be changed the following way: at the beginning of the withdrawal transaction, Alice chooses at random r_i, a_i, b_i, c_i, d_i, e_i, z_i $(1 \le i \le k)$ and computes:

$$\begin{aligned} x_i &= g(a_i \| b_i \| z_i, c_i), \\ y_i &= g(a_i \oplus ID_i, d_i), \\ M_i &\equiv f(x_i, y_i), \\ m_i &\equiv h(g(b_i \| (z_i \oplus ID_i), e_i)), \\ \alpha_i &\equiv M_i^{3^j} \cdot m_i^{17} \cdot r_i^{17 \cdot 3^j}. \end{aligned}$$

The bank will compute the signature

$$D \equiv \prod_{i=1}^{j} (\alpha_{t(i)})^{1/17 \cdot 3^{j+1-i}} \cdot \prod_{i=j+1}^{k/2} (\alpha_{t(i)})^{1/17},$$

without the number R. The refund can be anonymous, because if Alice tries to request refund for an already spent denomination, her identity will be revealed to the bank: from $a_i \| b_i \| z_i$ and $b_i \| (z_i \oplus ID_i)$ the bank can compute ID_i.

6.10. Several ways Alice, the shop, and the bank can try to cheat

In this section we discuss several ways Alice may try to cheat, how a shop and Alice can conspire, and how the bank may try to cheat (Cheating 1, 4, and 5 can be found in [CFN88]). Assume that Alice tries to spend $\$a$ at the shop and to get back $\$b$ from the bank, where $a+b > (2^j-1)$. Note that she cannot show the same b_i both to the shop when buying and to the bank when requesting for refund; if she does, she is caught cheating.

Cheating 1: by Alice.
Alice constructs the blinded candidate α_1 improperly: she uses b_1 to create x_1 and b_1' to create m_1. She will create the other candidates $\alpha_2,\ldots,\alpha_k$ properly and in such a way that M_1 is lexicographically the smallest of $M_2,\ldots,M_k$. With probability 1/2 the candidate α_1 will not need to be opened during the cut-and-choose protocol. After computing C and C' from the signature received from the bank, she can spend the check C for the maximum amount; and by using C', she can still request for a refund for b_1', because $b_1' \neq b_1$. The number b_1' will have denomination \$1 because it belongs to the lexicographically first major term.

Instead of improperly creating α_1, she can also create α_i improperly in the same way (for some i). The advantage of cheating in α_i is that it will belong to a higher

denomination than \$1 (so for instance, take $i=j$).

The original payment system [CFN88] is vulnerable to the same attack.

This attack is *not* prevented by using an ordering other than lexicographic in the check; that only reduces the probability of success from 1/2 to j/k, since in a check only j of the k terms are used for the denominations. A better way to prevent this attack is to allocate several minor terms per denomination. Then, if a denomination is used, Alice has to reveal several b's of several terms. Thus in order to both spend and request for refund a denomination, all the b's in these terms must be created improperly.

Cheating 2: by Alice.

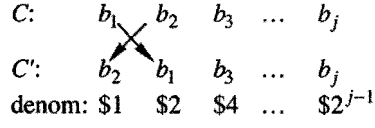
Alice withdraws a check by using her name, but she requests the bank to refund this check for the maximum value by not using her own name, and using Bob's name instead. After receiving this refund, she spends the check at the shop for the maximum value. The bank detects this cheating only after the check is deposited, but it only knows the name Bob and thus cannot trace Alice.

This attack is prevented in the payment system presented here, because during the withdrawal the bank randomly generates for each refund term a number R and stores this number with the user's name (during withdrawal Alice has to use her own name). Another way to prevent this attack is to require the user to identify herself during the refund.

For anonymous refund (see Subsection 6.9.3), the system can be cheated by a combination of Cheating 1 and 2: Alice creates α_1 improperly by using different b's in x_1 and m_1 and also using another account number than ID_i. She will create $\alpha_2,\ldots,\alpha_k$ properly and in such a way that M_1 is lexicographically the smallest of $M_2,\ldots,M_k$. With probability 1/2 the candidate α_1 will not need to be opened. Now she requests the bank to refund the check for the maximum amount by using C', and afterwards she goes to a shop to spend check C for the denomination of b_1 only. After deposit the bank detects this cheating, but cannot trace Alice. The same method of allocating several minor terms per denomination can be used to prevent this attack.

Cheating 3: by Alice.

The order of b_i's in C and C' is the same. But Alice may want to change the order of the terms in either C or C', so that some of the b_i's are no longer correlated, as for instance:



If this is possible, then Alice can spend this check C for $\$\sum_{i=1}^{j-1} 2^i = \$(2^j - 2)$ by

revealing $b_2, b_3, \ldots, b_j$; and she can request refund for \$2 by using C' and revealing b_1. But $\$(2^j-2) + \$2 = \$(2^j-1) + \$1$, so now she has gained \$1, without being caught.

One way for Alice to accomplish this, therefore, is to order the elements of T_c wrongly so that the terms in C will be also ordered wrongly, and she has to reorder them lexicographically. So she needs to compute $C_\sigma \equiv \left[M_1^{3^{\sigma(0)}} M_2^{3^{\sigma(1)}} \ldots M_j^{3^{\sigma(j-1)}} P^{3^j}\right]^{1/17}$, where σ is a non-identical permutation of $\{0,\ldots,j-1\}$ and $P = \prod_{i=j+1}^{k/2}(M_{t(i)})$. Alice could cheat with this signature C_σ as follows: she looks for j_1 such that $\sigma(j_1) = j_2 < j_1$; she shows C_σ to the shop and spends $\$(\sum_{i \neq j_2} 2^i)$ using only $\$(\sum_{i \neq j_1} 2^i)$ from her check; and gets back from the bank $\$(2^{j_1})$ by showing it C'. So she can spend the check of value $\$(2^j-1)$ for the amount $\$(\sum_{i \neq j_2} 2^i) + \$(2^{j_1}) = \$(2^j - 1) + \$(2^{j_2}(2^{j_1-j_2} - 1))$, so she gained $\$(2^{j_2}(2^{j_1-j_2} - 1))$. If we assume that Alice cannot compute RSA-roots (mod N) of random numbers, and that f,g are good pseudo-random functions, then from Corollary 2.1 follows:

Corollary 6.1. *Let $P, M_1, \ldots, M_j$ be random numbers from $\mathbb{Z}_N^*$. Then it is feasible for Alice to find a non-identical permutation σ of $\{0,\ldots,j-1\}$ and it is feasible to compute $C_\sigma \equiv \left[M_1^{3^{\sigma(0)}} M_2^{3^{\sigma(1)}} \ldots M_j^{3^{\sigma(j-1)}} P^{3^j}\right]^{1/17}$ from $C \equiv \left[M_1^{3^0} M_2^{3^1} \ldots M_j^{3^{j-1}} P^{3^j}\right]^{1/17}$ if and only if there is an i_0 $(1 \le i_0 \le j)$ such that $3^{i_0} \equiv 1 \pmod{17}$.*

The proof, which is not so difficult, is omitted here. By using j=10, it is easy to see that there are no numbers $1 \le i_0 \le j$ that satisfy $3^{i_0} \equiv 1 \pmod{17}$. Hence this kind of attack fails. But if the number of denomination bits (j) varies, the exponent used (17) must also be changed according to Table 6.2, to prevent this kind of attack.

j+1	1,...,3	4,5	6,...,15	16,17	18,...,27
exponent	5	7	17	19	29

Table 6.2. List of the exponent used for the banks signature on the check, for several amounts of denomination bits (j).

A second way for Alice to cheat is to perform the withdrawal protocol properly, but after receiving C and C', to change the order of the elements in C'. For this purpose she needs to compute $C'_\sigma \equiv \left[(Rm_1)^{1/3^{\sigma(j)}} m_2^{1/3^{\sigma(j-1)}} \ldots m_j^{1/3^{\sigma(1)}}\right]^{17}$, where σ is a non-identical permutation of $\{1,\ldots,j\}$. Then Alice can cheat as follows: she looks for i_1 such that $\sigma(i_1) = i_2 > i_1$; she spends $\$(\sum_{i \neq i_1} 2^i)$ at the shop by using C; and when requesting refund, she shows C'_σ and b_{i_1}, e_{i_1} to the bank, in order to get $\$(2^{i_2})$ as refund

instead of $\$(2^i)$. But from Corollary 2.1 it follows that Alice can compute C'_σ if and only if $\exists_{a\in \mathbf{Z}} \forall_{0\le i\le j-1}[3^{-i} - a3^{-\sigma(i)} \in \mathbf{Z}]$. If, however, σ is not the identity permutation, then there is an i such that $\sigma(i)-i < 1$ and thus $a \notin \mathbf{Z}$. Hence Alice cannot compute C_σ,where σ is a non-identical permutation. So this second way does not work.

Cheating 4: collusion between Alice and a shop

Alice spends her check at some shop and reuses this check for the same amount at the colluding shop, which will issue the same challenge to Alice. The bank knows that with very high probability one of the two shops is lying, but cannot determine which one, and cannot trace Alice's account. This attack can be avoided if the shop's challenge has to be generated by a pseudo-random device, the output of which cannot be modified by the shop.

Another way to prevent this attack is for each shop always to use the same challenge; and each of these different challenges is a word from a code with large Hamming distance. Each shop stores all the checks it has ever received, so that a user cannot reuse a check at the same shop. A combination of these two techniques can also be used.

Cheating 5: by the bank.

The bank can generate valid money with Alice's account number in it, and can double-spend this check. To prevent this, Alice will use her own public-key system as follows. During the withdrawal she generates integers z'_i, z''_i $(i = 1,\ldots,k)$ and creates ID_i as $ID \| z'_i \| z''_i$. Along with the blinded candidates she sends her signature on $g(z'_1,z''_1)\|\ldots\|g(z'_k,z''_k)$. After the cut-and-choose protocol the bank knows $k/2$ pairs z'_i, z''_i and after the reuse of this check it will receive another pair. Hence having $k/2+1$ pairs z'_i, z''_i and Alice's signature, the bank has a valid proof that Alice reused her check.

We have assumed that no user can invert the function g and that even if the bank can invert g, it cannot know which of the preimages was used by Alice. Hence with high probability Alice can show another preimage and can therefore prove that the bank has broken g. This, therefore, is a fail-stop signature (see Section 5.3).

6.11. Other offline payment systems

This scheme was presented in '89; since that time other offline payment systems have also been proposed. We will mention four of them very briefly.

[OO89]: (see also Sections 2.7 and 6.5) The authors propose a payment scheme in which the tradeoff between the degree of efficiency and the degree of untraceability can be chosen. If the user has unconditional privacy, the

efficiency is the same as in our scheme.

[Hay90]: This is a variant of [CFN88], and it has the property that if a cheater is revealed, then also all his previously used checks are revealed (called hot-listing).

[Ant90]: This is a variant of [CFN88]: neither the check nor the refund part are one single number, but each can be split into separate parts. During payment therefore, deposit and refund, the amount of numbers to be transmitted is fewer (for instance, one does not need to send x_i or y_i). This scheme is more efficient than ours.

[OO91]: In our scheme a check can only be used once: the part of the check that has not been spent can only be refunded. The authors propose a scheme in which the user can subdivide the amount of the check into many pieces in any way she chooses and each piece can be spent.

6.12. Some open problems

The conditions under which this payment system is reliable are not known. What can be said about these conditions?

In the withdrawal transaction we have a cut-and-choose protocol, but it is better to replace this protocol by another protocol, that is more efficient and easier to analyse.

In Section 6.5, [Ant90] proposed a method to build transferability into this scheme, but the amount of numbers needed to represent the money grows each time the money is transferred. An open question is whether there exist more efficient methods for establishing transferability.

7

Chameleon blobs, unconditionally secure for the verifier

7.1. Introduction

A *bit commitment scheme* is a scheme that allows a person to commit to a certain value in such a way that he cannot change the committed value afterwards, and that the recipient of this commitment (called *blob*) cannot learn its content. When these blobs are created using cryptography, there is a public function $\mathcal{B}$; if person $\mathcal{P}$ wants to commit himself to value b, he chooses randomly a number r and his blob will be $B = \mathcal{B}(b,r)$. Opening this blob to the recipient $\mathcal{V}$ consists of revealing b and r, who will verify that $B = \mathcal{B}(b,r)$ (see Section 1.6).

In this chapter we present the first blobs that use a new method of opening: in stead of revealing b and r to $\mathcal{V}$, $\mathcal{P}$ only reveals b and convinces $\mathcal{V}$, by using a zero-knowledge protocol, that he knows an r that satisfies $B = \mathcal{B}(b,r)$. For these new blobs we may modify the definition of chameleon (see Section 7.2). Therefore, by using our way of opening blobs, we are able to construct chameleon blobs that are unconditionally secure for the verifier, which was proven in the literature to be impossible.

This chapter is organized as follows: in Section 7.2 we analyze why "normal" chameleon blobs cannot be unconditionally secure for the verifier, we define two types of chameleon (active and passive), and we argue why we use passive chameleon for our new blobs. In Section 7.3 and 7.4 we create an efficient blob, based on the Certified Discrete Log Assumption, which opens in this new way and we prove that it is a passive chameleon blob that is unconditionally secure for the verifier (with an exponentially small error probability). In Section 7.5 we give other blob constructions, and we also

show that each "normal" blob can be transformed into a chameleon blob that uses a zero-knowledge protocol for opening. Hence each simulatable blob that is unconditionally secure for the verifier can be transformed into a chameleon blob that is unconditionally secure for the verifier (with an exponentially small error probability). We end in Section 7.6 with some open problems.

7.2. Can chameleon blobs be unconditionally secure for the verifier?

Consider the following two quotations. [BCC88] states (p. 175): "... *blobs that are unconditionally secure for $\mathcal{V}$. (...) Of course, none of these blobs are chameleon.*" The reason for this statement is that unconditional security for the verifier is achieved by requiring that for any blob there cannot exist a matching pair; this means that there cannot simultaneously exist $b_1 \neq b_2$, r_1, r_2 such that $\mathcal{B}(b_1,r_1) = \mathcal{B}(b_2,r_2)$. But "chameleon" implies that $\mathcal{V}$ can construct numbers $b_1 \neq b_2$, r_1, r_2 such that $\mathcal{B}(b_1,r_1) = \mathcal{B}(b_2,r_2)$. Therefore a blob that is unconditionally secure for the verifier cannot be chameleon.

[FS89] states (p. 532): "*If furthermore, $\mathcal{V}$ has secret trapdoor information which allows $\mathcal{V}$ (unlike $\mathcal{P}$) to compute matching pairs (...), the scheme is termed trapdoor commitment scheme (or chameleon in [BCC88]).*" The reason for this statement is that in order to have blobs that $\mathcal{P}$ can only open in one way but $\mathcal{V}$ in several ways, $\mathcal{V}$ must have more (trapdoor) information than $\mathcal{P}$.

According to these two quotations, chameleon blobs (trapdoor commitment schemes) cannot be unconditionally secure for $\mathcal{V}$.

In Section 1.6 a blob was called *chameleon* if $\mathcal{V}$ can generate a blob that he can open in several ways. We define (for the first time) two kinds of chameleon:

- *Active chameleon*: this means that $\mathcal{V}$ can generate $b_1 \neq b_2$, r_1, r_2 such that $\mathcal{B}(b_1,r_1) = \mathcal{B}(b_2,r_2)$, and thus he can successfully *perform* the blob-opening protocol with anybody.
- *Passive chameleon*: this means that $\mathcal{V}$ can generate a blob and that for this blob he can *create several transcripts* of the blob-opening protocol for several committed values.

(The same distinction can be made between passively and actively simulatable, but in this chapter we will take "simulatable" to mean only actively simulatable.) For "normal" blobs these two kinds of chameleon are the same, but for our new blobs they are different. In this chapter we will use the second kind of chameleon; the reason for doing so is expressed in [BCC88], page 167: "*The advantage of chameleon blobs is that they allow $\mathcal{V}$ to simulate in a straightforward way his entire conversation with P, without ever encountering failures. Again, this remains true even if $\mathcal{V}$ deviates*

arbitrarily from his stipulated behaviour."

We show that by using our way of opening a blob, we are able to construct (passive) chameleon blobs that are unconditionally secure for the verifier and in which $\mathcal{P}$ has some secret information instead of $\mathcal{V}$. Hence these blobs can be opened in parallel.

7.3. Description of a new blob that uses an opening protocol

In this section we will first describe a new efficient blob, and in the next section we will prove that it is a passive chameleon blob that is unconditionally secure for the verifier (with an exponentially small error probability).

Let p be a large prime and let g be a generator of the multiplicative group $\mathbb{Z}_p^*$. We assume that $p-1 = 2q$, where q is an odd number of which each divisor is larger than $2\log p$, and we use Assumption 1.2 (Certified Discrete Log Assumption).

Person $\mathcal{P}$ has a secret key $s \in \{1,2,\ldots,p-2\}$ and the corresponding public key will be $g^s \pmod p$. If he wants to commit himself to value $b \in \{0,1,\ldots,\lfloor \log p \rfloor\}$, he randomly chooses a number $r \in \mathbb{Z}_p^*$, $r \not\equiv \pm 1 \pmod p$ and issues to $\mathcal{V}$ the blob

$$(r, r^{s+2b}).$$

$\mathcal{V}$ verifies that $r \not\equiv \pm 1 \pmod p$. To open this blob, $\mathcal{P}$ reveals b and convinces $\mathcal{V}$ that the exponents used in r^{s+2b}/r^{2b} and g^s are the same, by using the (computational) zero-knowledge confirmation protocol of [Ch90]. The number of moves in this opening protocol is 5, because $\mathcal{P}$ first has to issue b to $\mathcal{V}$, and then they have to perform the 4-move confirmation protocol. But this opening protocol can be reduced to 4 moves, as indicated in Figure 7.1.

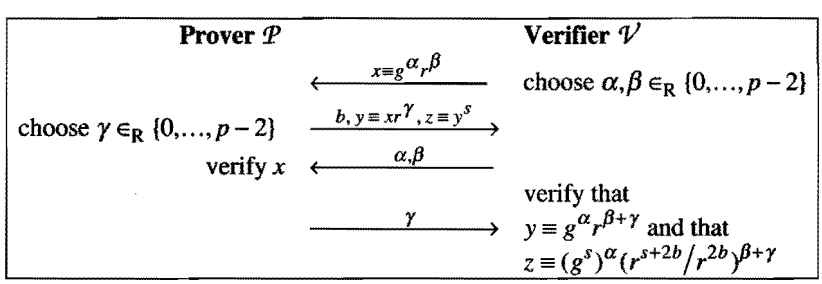


Fig. 7.1. The 4-move blob-opening protocol.

7.4. Properties of the blob of Section 7.3

We will prove that this new blob of Section 7.3 is a passive chameleon blob that is unconditionally secure for the verifier (for the blob's properties see Section 1.5).

- *Property* (i): This new blob has property (i) trivially.

- *Property* (ii): During the confirmation protocol $\mathcal{P}$ can only change the committed value in the blob with exponentially small probability $1/p$ (see [Ch90]); hence $\mathcal{P}$ can open the blob in two different ways, only by choosing numbers $r \in \mathbb{Z}_p^*$ and $b_1, b_2 \in \{0,\ldots,\lfloor \log p \rfloor\}$ such that $b_1 \neq b_2$ and $r^{s+2b_1} \equiv r^{s+2b_2} \pmod{p}$. Hence $r^{2\cdot|b_1-b_2|} \equiv 1 \pmod{p}$. It is easy to see that the number of solutions to the equation $r^d \equiv 1 \pmod{p}$ is equal to $\gcd(d,p-1)$. Because $b_1 \neq b_2 \in \{0,\ldots,\lfloor \log p \rfloor\}$, we have that $2\cdot|b_1-b_2| \leq 2\lfloor \log p \rfloor <$ each divisor of q. Thus $\gcd(2\cdot|b_1-b_2|, p) = \gcd(2\cdot|b_1-b_2|, 2q) = 2$, and so the number of solutions to the equation $r^{2\cdot|b_1-b_2|} \equiv 1 \pmod{p}$ is two: namely $r \not\equiv \pm 1 \pmod{p}$. But these choices of r were ruled out in the definition of the blob, and thus it is not possible for $\mathcal{P}$ to change the committed value in the blob.

- *Property* (iii): The loglog p successive higher-order bits after the least significant bit of a discrete logarithm are simultaneously hard [Per85] (see also Subsection 1.4.1). So for this property we use that the exponent in the blob is $(s+2b)$: if the exponent would be $(s+b)$, then $\mathcal{V}$ can compute in polynomial time the least significant bit of b.

- *Simulatable*: $\mathcal{V}$ can simulate blobs for each value $b \in \{0,\ldots,\lfloor \log p \rfloor\}$ by choosing $t \in \{1,2,\ldots,p-2\} \setminus \{q\}$ and creating the pair $(g^t, (g^s g^{2b})^t)$ (i.e., the blob is (r, r^{s+2b}), where $r \equiv g^t$). This blob is simulatable because $\mathcal{V}$ can *perform* the opening protocol of this blob (although he does not know s) with a verifier $\mathcal{V}'$, and $\mathcal{V}'$ cannot distinguish between an original blob and a simulated blob. Thus $\mathcal{V}$ performs the protocol of Figure 7.2 instead of that of Figure 7.1, while $\mathcal{V}'$ cannot detect the difference between these two protocols[‡]. So it also satisfies property (iv).

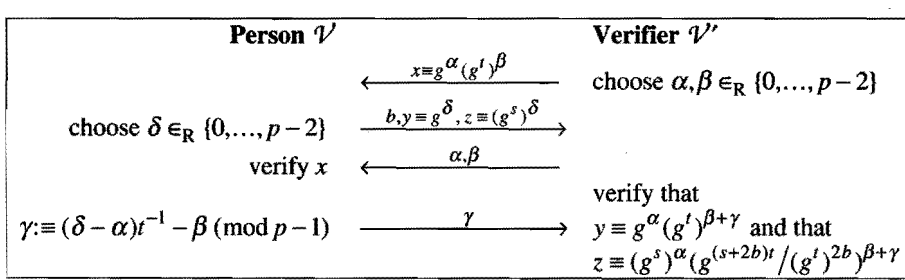


Fig. 7.2. The opening protocol for a simulated blob.

[‡] Thus for normal undeniable signatures also, a forger can create on message g^t the valid signature g^{ts}, and he can perform the confirmation protocol succesfully, even without knowing s. This can be avoided if a hash value of the message is signed.

- *Passive chameleon*: each person $\mathcal{V}$ can simulate a transcript: To create a blob, he randomly chooses two numbers $r_1,r_2 \in \mathbb{Z}_p^*$ and the blob will be (r_1,r_2). To open this blob, $\mathcal{V}$ randomly chooses the "committed" number $b \in \{0,\ldots,\lfloor \log p \rfloor\}$ and creates a transcript of the confirmation protocol that uses these numbers. This is possible because the confirmation protocol is zero-knowledge.

- *Unconditionally secure for the verifier* (with an exponentially small error probability): because the committed value in the blob is uniquely determined by the blob and the public key (remember that $r \not\equiv \pm 1 \pmod p$); moreover, during the confirmation protocol $\mathcal{P}$ can change the committed value in the blob only with probability $1/p$ (which is exponentially small in $\log p$), even if $\mathcal{P}$ has unlimited computing power [Ch90].

In the description of this blob we required that $p-1 = 2q$ where q is odd, but this restriction is not necessary. Write $p = 2^u q+1$ for q odd, and require that each divisor of q is larger than $2\log p$. To generate a blob, $\mathcal{P}$ randomly chooses $r \in \mathbb{Z}_p^*$ such that $r^{2^u} \not\equiv 1 \pmod p$. If $\mathcal{P}$ wants to commit himself to value $b \in \{0,\ldots,\lfloor \log p \rfloor\}$, his blob will be

$$(r, r^{s+2^u b}),$$

and $\mathcal{V}$ verifies that $r^{2^u} \not\equiv 1 \pmod p$. It is easy to verify that $\mathcal{P}$ cannot open this blob in two ways: there do not exist numbers $r \in \mathbb{Z}_p^*$, $r^{2^u} \not\equiv 1 \pmod p$ and $b_1,b_2 \in \{0,\ldots,\lfloor \log p \rfloor\}$ such that $r^{s+2^u b_1} \equiv r^{s+2^u b_2} \pmod p$.

7.5. Other constructions of blobs that use an opening protocol

The new blob presented in Section 7.3 is efficient. Other methods are known for constructing chameleon blobs that are unconditionally secure for the verifier, but some constructions turn out to be less efficient than the one in Section 7.3.

One construction is based on [BCC88] (with the Certified Discrete Log Assumption 1.2). Let $p = 2^u q+1$ be a large prime, q odd, g a generator of $\mathbb{Z}_p^*$, and let $l = \lfloor \log\log p \rfloor$. If $\mathcal{P}$ wants to commit himself to value $b \in \{0,1,\ldots,2^l-1\}$, he randomly chooses a number $r \in \mathbb{Z}_p^*$ such that the $u+l$ least significant bits of r are zero, and his blob will be

$$g^{r+b2^u} \pmod p.$$

To open this blob, $\mathcal{P}$ reveals b and performs the zero-knowledge protocol of

[BCDvdG87] (which must be iterated several times, see also Protocol 4.1 in Subsection 4.3.1) to convince the recipient that he knows an integer s such that

$$0 \le s < \frac{q}{2^l} - 1 \quad \text{and} \quad \left(g^{2^{l+u}}\right)^s \equiv g^{r+b2^u} g^{-b2^u} \pmod{p}. \tag{7.1}$$

This blob is passive chameleon and unconditionally secure for the verifier: Suppose that $\mathcal{P}$ can open this blob in two ways. Then Equation (7.1) implies that he can find numbers B, s_1, b_1, s_2, b_2 such that $g^{s_1 2^{l+u}} g^{b_1 2^u} \equiv g^{s_2 2^{l+u}} g^{b_2 2^u} \equiv B \pmod{p}$, which is equivalent to (because g is a generator of $\mathbb{Z}_p^*$):

$$s_1 2^{l+u} + b_1 2^u \equiv s_2 2^{l+u} + b_2 2^u \pmod{p-1}. \tag{7.2}$$

Because $p-1=2^u q$ and q odd, this equation is equivalent to:

$$(s_1 - s_2)2^l \equiv b_2 - b_1 \pmod{q}. \tag{7.3}$$

By using the fact that $0 \le s_i < \frac{q}{2^l} - 1$, $s_i \in \mathbb{Z}$ and $b_i \in \{0,\ldots,2^l-1\}$ $(i = 1,2)$, we have that (using interval notation mod q):

$$\begin{cases} (s_1 - s_2)2^l \pmod{q} & \in [0] \cup [2^l, q-2^l), \\ b_2 - b_1 \pmod{q} & \in [0, 2^l - 1] \cup [q-2^l+1, q). \end{cases} \tag{7.4}$$

By combining Equations (7.3) and (7.4), we see that the only solution to Equation (7.1) is $s_1 = s_2$ and $b_1 = b_2$. Hence this blob is unconditionally secure for $\mathcal{V}$. It is easy to verify that this blob also satisfies the other blob properties and that it is chameleon (the blob is (actively) simulatable, because no secret key is used in its construction).

We show a construction such that each simulatable blob that is constructed by using a public function $\mathcal{B}$, and for which the opening is the revealing of several numbers, can be modified into a blob B' that uses a zero-knowledge protocol for opening. Consider the predicate

"Blob B contains the value b".

Whenever this predicate is true, there exists a certificate r such that $B = \mathcal{B}(b,r)$; knowing r, it is easy to verify that $B = \mathcal{B}(b,r)$. Now $\mathcal{P}$ and $\mathcal{V}$ can efficiently build a Boolean formula satisfiable if and only if this predicate is true (for instance by using Cook's constructive theorem [Cook71]). Because the satisfiability of Boolean formulas is NP-complete, and if $\mathcal{P}$ knows an r such that $B = \mathcal{B}(b,r)$, then $\mathcal{P}$ can use the basic protocol of [BCC88] to convince $\mathcal{V}$ that this Boolean formula is satisfiable, and hence that he knows r. In this protocol for proving the satisfiability of Boolean formulas, other blobs have to be used, which we will call C (and that use function $\mathcal{C}$). We assume that the blobs B and C are simulatable.

The properties of this blob B' are:

- *Property* (i): trivial.
- *Property* (ii): If $\mathcal{P}$ tries to open the blob B' in another way than the one to which he committed himself, he does not know a satisfying assignment for the Boolean formula (according to property (ii) of blob B). Hence the best strategy that $\mathcal{P}$ can follow is to guess the challenge he will receive from $\mathcal{V}$, which means that he will be correct with probability 1/2. Hence the probability that he will not be caught cheating after k iterations of the protocol is $1/2^k$, and thus blob B' has property (ii).
- *Simulatable*: Because the blobs B and C are simulatable, this blob B' is (actively) simulatable, and thus it also has property (iv).
- *Passive chameleon*: In [BCC88] it is proven that the *sequential* execution of the iterations of this protocol is zero-knowledge, because of property (iii) and (iv) of blob C. Thus $\mathcal{V}$ can simulate his entire sequential conversation with $\mathcal{P}$, without knowing a satisfying assignment. Hence the blob B' is passive chameleon and has property (iii).

If blob B is chameleon, then the blob B' is active chameleon, independent of whether the blobs C are chameleon or not. If the blobs B and C are unconditionally secure for the verifier, then the blob B' is also unconditionally secure for $\mathcal{V}$ (except for an exponentially small error probability).

We have now shown a construction to transform a simulatable blob B that is unconditionally secure for the verifier, into a blob B' that is both passive chameleon and unconditionally secure for the verifier (with an exponentially small error probability).

If the iterations of the protocol of [BCC88] have to be carried out in parallel and if the blob B' has to be both chameleon and unconditionally secure for the verifier, then the blobs C used must also be chameleon and unconditionally secure for the verifier (otherwise $\mathcal{V}$ cannot simulate his entire parallel conversation with $\mathcal{P}$). But now the blobs C are also of this new type that requires an opening protocol, so we are form the frying-pan into the fire.

7.6. Some open problems

In the previous sections we constructed chameleon blobs unconditionally secure for the verifier to commit to values of length loglog p. Can we construct these new kind of blobs that can be used to commit to full length messages (or at least a hash value of, say, size 128 bits)?

Can there be found some applications for these new kind of blobs, other than constructing a counter example?

References

[Ant90] Hans van Antwerpen, Electronic cash, master thesis, Dept. of Math. & Comp. Sc., Technical University of Eindhoven, July 1990.

[AT83] S. Akl and P. Taylor, Cryptographic solution to a problem of access control in a hierarchy, *ACM Trans. Comput. Systems* **1** (1983), pp. 239-248.

[BCC88] Gilles Brassard, David Chaum and Claude Crépeau, Minimum disclosure proofs of knowledge, *J. of Comp. and System Sc.* **37** (1988), pp. 156-189.

[BCDP90] Joan Boyar, David Chaum, Ivan Damgård and Torben Pedersen, Convertible Undeniable Signatures, *Advances in Cryptology—CRYPTO '90*, A.J. Menezes and S.A. Vanstone eds., LNCS 537, Springer Verlag, pp. 189-205.

[BCDvdG87] Ernest Brickell, David Chaum, Ivan Damgård and Jeroen van de Graaf, Gradual and verifiable release of a secret, *Advances in Cryptology—CRYPTO '87*, C. Pomerance ed., LNCS 293, Springer-Verlag, pp. 156-166.

[Blu82] M. Blum, Coin flipping by telephone, *Proc. IEEE Compcon* (1982), pp. 133-137.

[Bos92] Jurjen Bos, Practical privacy, PhD Thesis, Dept. of Math. & Comp. Sc., Technical University Eindhoven, March 1992.

[BPW90] Gerrit Bleumer, Birgit Pfitzmann and Michael Waidner, A remark on a signature scheme where forgery can be proved, *Advances in Cryptology—EUROCRYPT '90*, I.B. Damgård ed., LNCS 473, Springer Verlag, pp. 441-445.

[Brass88] Gilles Brassard, Modern cryptology, LNCS 325, Springer-Verlag, Berlin.

[Brass91] Gilles Brassard, Cryptology column: How convincing is your protocol?, *Sigact News*, **22** (1991), pp. 5-12.

[BrCr86] Gilles Brassard and Claude Crépeau, Non-interactive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond, *FOCS* 1986, pp. 188-195.

[CE86] David Chaum and Jan-Hendrik Evertse, A secure and privacy protecting

protocol for transmitting personal information between organizations, *Advances in Cryptology—CRYPTO '86*, A.M. Odlyzko ed., LNCS 263, Springer-Verlag, pp. 118-167.

[CEvdG87] David Chaum, Jan-Hendrik Evertse and Jeroen van de Graaf, An improved protocol for demonstrating possession of discrete logarithms and some generalizations, *Advances in Cryptology—EUROCRYPT '87*, D. Chaum, W. Price eds., LNCS 304, Springer-Verlag, pp. 127-141.

[CFN88] David Chaum, Amos Fiat, and Moni Naor, Untraceable electronic cash, *Advances in Cryptology—CRYPTO '88*, S. Goldwasser ed., LNCS 403, Springer-Verlag, pp. 319-327.

[Ch86] David Chaum, Demonstrating that a public predicate can be satisfied without revealing any information about how, *Advances in Cryptology—CRYPTO '86*, A.M. Odlyzko ed., LNCS 263, Springer-Verlag, pp. 195-199.

[Ch90] David Chaum, Zero-knowledge undeniable signatures, *Advances in Cryptology—EUROCRYPT '90*, I. Damgård ed., LNCS 473, Springer-Verlag, pp. 458-464.

[Cook71] S.A. Cook, The complexity of theorem proving procedures, *Proceedings 3rd annual ACM symposium on the theory of computing (STOC)*, 1971, pp. 151-158.

[CR90] David Chaum and Sandra Roijakkers, Unconditionally secure digital signatures, *Advances in Cryptology—CRYPTO '90*, A.J. Menezes and S.A. Vanstone eds., LNCS 537, Springer-Verlag, pp. 206-214.

[CvA89] David Chaum and Hans van Antwerpen, Undeniable signatures, *Advances in Cryptology—CRYPTO '89*, G. Brassard ed., LNCS 435, Springer-Verlag, pp. 212-216.

[Dav82] George Davida, Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem, Tech. rept. TR-CS-82-2, Dept of Electrical Engineering and Computer Science, Univ. of Wisconsin, October 1982.

[Denn84] Dorothy Denning, "Digital signatures with RSA and other public-key cryptosystems", *Comm. of the ACM*, **27** (1984) pp. 388-392.

[DH76] Whitfield Diffie and Martin Hellman, New directions in cryptography, *IEEE IT* **22** (1976), pp. 644-654.

[DO85] Yvo Desmedt and Andrew Odlyzko, "A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes", *Advances in Cryptology—CRYPTO 85*, H.C. Williams ed., LNCS 218, Springer-Verlag, pp. 516-522.

[EGS85] Shimon Even, Oded Goldreich and Adi Shamir, "On the security of ping-pong protocols when implemented using the RSA", *Advances in Cryptology—CRYPTO 85*, H.C. Williams ed., LNCS 218, Springer-Verlag, pp. 58-72.

[ElG85] Taher ElGamal, A public key cryptosystem and a signature scheme

based on discrete logarithm, *IEEE* IT-**31** (1985), pp. 469-472.

[Evert90] Jan-Hendrik Evertse, unpublished notes, 1990.

[FFS88] Uriel Feige, Amos Fiat and Adi Shamir, Zero-knowledge proofs of identity, *J. Cryptology* **1** (1988), pp. 77-94.

[FS89] Uriel Feige and Adi Shamir, Zero-knowledge proofs of knowledge in two rounds, *Advances in Cryptology—CRYPTO '89*, G. Brassard ed., LNCS 435, Springer-Verlag, pp. 526-544.

[Gill77] John Gill, Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6** (1977), pp. 675-695.

[GMR85] Shafi Goldwasser, Silvio Micali and Charles Rackoff, Knowledge complexity of interactive proof systems, *STOC* 1985, pp. 291-304.

[GMR88] Shafi Goldwasser, Silvio Micali and Ronald Rivest, A digital signature scheme secure against adaptive chosen-message attacks, *SIAM J. Comput.* **17** (1988), pp. 281-308.

[GMR89] Shafi Goldwasser, Silvio Micali and Charles Rackoff, The knowledge complexity of interactive proof systems, *SIAM J. Comput.* **18** (1989), pp. 186-208.

[Has85] Johan Hastad, On using RSA with low exponent in a public key network, *Advances in Cryptology—CRYPTO '85*, H.C. Williams ed., LNCS 218, Springer-Verlag, pp. 403-408.

[Hay90] Barry Hayes, Anonymous one-time signatures and flexible untraceable electronic cash, *Advances in Cryptology—AUSCRYPT '90*, J. Seberry and J. Pieprzyk eds., LNCS 453, Springer-Verlag, pp. 294-305.

[Heg1858] I. Heger, Über die Auflösung eines Systemes von mehreren unbestimmten Gleichungen des ersten Grades in ganzen Zahlen, *Denkschriften der Königlichen Akademie der Wissenschaften (Wien), Mathematisch-naturwissenschaftliche Klasse* **14** (2. Abth.) (1858), pp. 1-122.

[KaBa79] R. Kannan and A. Bachem, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM J. Comput.* **8** (1979), pp. 499-507.

[McC88] Kevin McCurley, A key distribution system equivalent to factoring, *J. Cryptology* **1** (1988), pp. 95-105.

[McKTMA85] Stephen MacKinnon, Peter Taylor, Henk Meijer and Selim Akl, An optimal algorithm for assigning cryptographic keys to control access in a hierarchy, *IEEE Trans. Comp. Systems* **34** (1985), pp. 797-802.

[Merk80] Ralph Merkle, Protocols for public key cryptosystems; *Proceedings of the 1980 symposium on security and privacy*, April 1980, Oakland, California, pp. 122-134.

[Merk87] Ralph Merkle, A digital signature based on a conventional encryption function, *Advances in Cryptology—CRYPTO '87*, C. Pomerance ed.,

LNCS 293, Springer-Verlag, pp. 369-378.

[Mil75] J.C.P. Miller, On factorization, with a suggested new approach, *Math. Comp.* **29** (1975), pp. 155-172.

[OO89] Tatsuaki Okamoto and Kazuo Ohta, Disposable zero-knowledge authentications and their applications to untraceable electronic cash, *Advances in Cryptology—CRYPTO '89*, G. Brassard ed., LNCS 435, Springer-Verlag, pp. 481-497.

[OO91] Tatsuaki Okamoto and Kazuo Ohta, Universal electronic cash, *Advances in Cryptology —CRYPTO '91*, J. Feigenbaum ed., LNCS 576, Springer-Verlag, pp. 324-337.

[OOK90] Kazuo Ohta, Tatsuaki Okamoto and Kenji Koyama, Membership authentication for hierarchical multigroup using the extended Fiat-Shamir scheme, *Advances in Cryptology—EUROCRYPT '90*, I. Damgård ed., LNCS 473, Springer-Verlag, pp. 446-457.

[Ped91] Torben Pedersen, Non-interactive and information-theoretic secure variable secret sharing, *Advances in Cryptology—CRYPTO '91*, J. Feigenbaum ed., LNCS 576, Springer-Verlag, pp. 129-140.

[Ped92] Torben Pedersen, Distributed provers and verifiable secret sharing based on the discrete logarithm problem, PhD thesis, Computer Science Department, Aarhus University, 1992.

[Per85] René Peralta, Simultaneous security of bits in the discrete log, *Advances in Cryptology—EUROCRYPT '85*, F. Pichler ed., LNCS 219, Springer-Verlag, pp. 62-72.

[Pf91] Birgit Pfitzmann, personal communication.

[PW90] Birgit Pfitzmann and Michael Waidner, Formal aspects of fail-stop signatures, Interner Bericht 22/90, Fakulät für Informatik, Universität Karlsruhe, December 1990.

[PW91] Birgit Pfitzmann and Michael Waidner, Fail-stop signatures and their applications, *SECURICOM '91*; 9th worldwide congress on computer and communications security and protection, Paris, 1991, pp. 145-160.

[PolHel78] S.C. Pohlig and M.E. Hellman, An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE IT* **24** (1978), pp. 106-110.

[RS62] J. Rosser and L. Schoenfeld, Approximate formulas for some functions of prime numbers, *Illinois J. Math.* **6** (1962), pp. 64-94.

[RS84] Ronald Rivest and Adi Shamir, How to expose an eavesdropper, *Comm. ACM* **27** (1984), pp. 393-395.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, *Comm. ACM* **21** (1978), pp. 120-126.

[Sh83] Adi Shamir, On the generation of cryptographically strong pseudorandom sequences, *ACM Trans. Computer Systems* **1** (1983), pp.

38-44.

[Shm85] Zahava Shmuely, Composite Diffie-Hellman public key generating systems are hard to break, Technical report no. 356, Computer science department, Technion-Israel Institute of Technology, February 1985.

[SKI90] Hiroki Shizuya, Kenji Koyama and Toshiya Itoh, Demonstrating possession without revealing factors and its applications, *Advances in Cryptology—AUSCRYPT '90*, J. Seberry and J. Pieprzyk eds., LNCS 453, Springer-Verlag, pp. 273-293.

[SRA79] A. Shamir, R.L. Rivest, and L. Adleman, Mental poker, MIT/LCS/ TM-125, February 1979.

[SS90] Schrift and Shamir, The discrete log is very discreet, *Proc. 22nd STOC 1990*, pp. 405-415.

[TW87] Martin Tompa and Heather Woll, Random self-reducibility and zero knowledge interactive proofs of possession of information, *28th FOCS*, IEEE Computer Society, 1987, pp. 472-482.

[WP89] Michael Waidner and Birgit Pfitzmann, The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability, *Advances in Cryptology—EUROCRYPT '89*, LNCS 434, Springer Verlag, p. 690.

Notation

$\oplus$	bitwise exclusive-or.
$\|$	concatenation.
$\mathbb{N},\mathbb{Z},\mathbb{Q}$	the sets of positive integers, all integers and rational numbers respectively.
$\gcd(a_1,\ldots,a_t)$	the greatest common divisor of $a_1,\ldots,a_t$; defined for rational numbers by $\gcd(a_1,\ldots,a_t):=\frac{\gcd(a_1 d,\ldots,a_t d)}{d}$, where $d\in\mathbb{N}$, $d\neq 0$, is such that $a_1 d,\ldots,a_t d\in\mathbb{Z}$; this definition is independent of the choice of d.
$\mathrm{lcm}(a_1,\ldots,a_t)$	the least common multiple of $a_1,\ldots,a_t\in\mathbb{Q}$ (this is defined for rational numbers analogously to the gcd).
$a\|b$	there is an integer c such that $ac=b$; also defined for $a,b\in\mathbb{Q}$.
$a\nmid b$	there is no integer c such that $ac=b$; also defined for $a,b\in\mathbb{Q}$.
$\mathrm{ord}_p(a)$	the integer such that $a\cdot p^{-\mathrm{ord}_p(a)}$ is an integer not divisible by p, for p prime and $a\in\mathbb{Z}, a\neq 0$.
$(a\backslash b)$	the largest positive divisor of a which is not divisible by any prime number dividing b, for $a,b\in\mathbb{Z}$, $a,b\neq 0$.
$\log x$	base-2 logarithm of x.
$\ln x$	natural logarithm of x.
$\lfloor x\rfloor$	the largest integer $\leq x$.
$\lceil x\rceil$	the smallest integer $\geq x$.
$\xrightarrow{x}$	transmission of the value x from one party to another party.
$a\in_R S$	denotes the random selection of an element (that will be called a) from S according to the uniform probability distribution; for any set S.
$a\equiv b\pmod m$	$(b-a)/m\in\mathbb{Z}$; this is defined for $a,b\in\mathbb{Q}$, $m\in\mathbb{N}$, $m\neq 0$; we shall omit the suffix (mod m), if no confusion is likely to arise.
S^k	the set of vectors $(a_1,\ldots,a_k)$ with $a_1,\ldots,a_k\in S$, for any set S.
$\mathbf{a}$	row vector $(a_1,\ldots,a_k)$; if $\mathbf{a}\in S^k$, then $a_1,\ldots,a_k\in S$.
$[\mathbf{a}_1\ \ldots\ \mathbf{a}_l]$	the matrix with columns $\mathbf{a}_1,\ldots,\mathbf{a}_l$.
$\mathbf{a}\equiv\mathbf{b}\pmod m$	$m^{-1}(\mathbf{b}-\mathbf{a})\in\mathbb{Z}^k$; this is defined for $\mathbf{a},\mathbf{b}\in\mathbb{Q}^k$, $m,k\in\mathbb{N}$, $m>0$.
$\mathbb{Z}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$	$\{\sum_{i=1}^{s}\xi_i\mathbf{a}_i \mid \xi_1,\ldots,\xi_s\in\mathbb{Z}\}$: the abelian group generated by $\mathbf{a}_1,\ldots,\mathbf{a}_s\in\mathbb{Q}^k$.

$\mathbb{Q}\{\mathbf{a}_1,\ldots,\mathbf{a}_s\}$	$\{\sum_{i=1}^{s}\xi_i\mathbf{a}_i \mid \xi_1,\ldots,\xi_s \in \mathbb{Q}\}$: the vector space generated by $\mathbf{a}_1,\ldots,\mathbf{a}_s \in \mathbb{Q}^k$.
$\langle a\rangle,\langle a,b\rangle$	the groups generated by a and generated by a,b respectively.
$\langle \mathbf{a},\mathbf{b}\rangle$	$a_1b_1 + \ldots + a_kb_k$: the scalar product of $\mathbf{a} = (a_1,\ldots,a_k)$ and $\mathbf{b} = (b_1,\ldots,b_k)$.
$\mathbf{ab}$	$(a_1b_1,\ldots,a_kb_k)$, if $\mathbf{a} = (a_1,\ldots,a_k)$ and $\mathbf{b} = (b_1,\ldots,b_k)$.
N	a composite, odd number; so $N = p_1^{k_1}\ldots p_t^{k_t}$ with $p_1,\ldots,p_t$ distinct odd primes and $k_1,\ldots,k_t \in \mathbb{N}$; usually $N = PQ$,for large primes P,Q.
$\mathbb{Z}_N^*$	the set $\{a \mid a \in \mathbb{N}, 1 \le a \le N, \gcd(a,N) = 1\}$.
$\mathbb{Z}_p$	the set $\{0,1,\ldots,p-1\}$, with p prime.
$\mathbb{Z}_p^*$	the set $\{1,2,\ldots,p-1\}$, with p prime.
G_q	the unique subgroup of $\mathbb{Z}_p^*$ of order q, for $q\mid(p-1)$ and p,q prime.
$\varphi(N)$	Euler's Totient function: $\varphi(N) = \mid \mathbb{Z}_N^* \mid = \prod_{i=1}^{t} p_i^{k_i-1}(p_i-1)$.
$a^{-1} \pmod N$	the number $b \in \mathbb{Z}_N^*$ with $ab \equiv 1 \pmod N$; for $a \in \mathbb{Z}_N^*$.
$\tilde{\mathbb{Q}}_N$	the ring $\{\frac{a}{d} \mid a,d \in \mathbb{Z}, d > 0, \gcd(d,\varphi(N)) = 1\}$.
$\lambda(N)$	Carmichael's function: $\lambda(N) = \mathrm{lcm}\left(p_1^{k_1-1}(p_1-1),\ldots,p_t^{k_t-1}(p_t-1)\right)$ (for N odd).
$x^{1/d} \pmod N$	the d^{th} *RSA-root* of $x \pmod N$: the unique solution $S \in \mathbb{Z}_N^*$ to $S^d \equiv x \pmod N$ for $x \in \mathbb{Z}_N^*$ and $d \in \mathbb{Z}$ with $\gcd(d,\varphi(N)) = 1$.
$x^{a/d} \pmod N$	the number in $\mathbb{Z}_N^*$ which is congruent $(x^{1/d})^a \pmod N$ for $a,d \in \mathbb{Z}$ with $\gcd(d,\varphi(N)) = 1$ and $x \in \mathbb{Z}_N^*$; if $a/d = a'/b'$, then $x^{a/d} = x^{a'/b'}$.
$\mathbf{x}^{\mathbf{a}} \pmod N$	the number $S \in \mathbb{Z}_N^*$ with $S \equiv x_1^{a_1}x_2^{a_2}\ldots x_k^{a_k} \pmod N$, for $\mathbf{x} = (x_1,\ldots,x_k) \in (\mathbb{Z}_N^*)^k$ and $\mathbf{a} = (a_1,\ldots,a_k) \in (\tilde{\mathbb{Q}}_N)^k$.
$\mathbf{x}^A \pmod N$	$(\mathbf{x}^{\mathbf{a}_1},\ldots,\mathbf{x}^{\mathbf{a}_l}) \in (\mathbb{Z}_N^*)^l$, for $A = [\mathbf{a}_1\ldots\mathbf{a}_l] \in (\tilde{\mathbb{Q}}_N)^{k,l}$ and $\mathbf{x} \in (\mathbb{Z}_N^*)^k$; so $(\mathbf{x}^A)^B = \mathbf{x}^{AB}$.
$l(n)$	length of the binary representation of $n \in \mathbb{N}$; the length of a negative integer m, a rational number p/q $(q \neq 1)$, a vector $\mathbf{c}$, and a matrix $A = (a_{i,j})$ are defined by: $l(m) = l(-m) + 1$, $l(p/q) = l(p) + l(q) + 1$, $l(\mathbf{c}) = \sum_i (l(c_i) + 1)$, and $l(A) = \sum_{i,j}(l(a_{i,j}) + 1)$, respectively.
$length(A,B)$	$l(A) + l(B)$.

Samenvatting

Dit proefschrift gaat over digitale (electronische) handtekeningen. Een handtekening is een bewijs dat een bepaalde boodschap van de ondertekenaar afkomstig is. Een digitale handtekening onder (een beter woord is "op") een boodschap is dit ook, en heeft de volgende eigenschappen:

- De handtekening bestaat uit één of meerdere getallen, die verkregen zijn door het toepassen van een wiskundige functie op de te tekenen boodschap. Hierdoor verschilt de handtekening van boodschap tot boodschap.
- Iedereen kan controleren of de handtekening bij een boodschap correct is.
- Het is zo goed als onmogelijk om andermans handtekening bij een nieuwe boodschap te maken. Dus als iemand een oude boodschap (een beetje) verandert klopt de handtekening niet meer.

Door deze drie eigenschappen is een digitale handtekening dus *niet* de PIN-code van je bankpas, waarmee je in sommige winkels kunt betalen. Ook niet het wachtwoord voor de toegang tot een computer. En het is ook niet het faxen van een document waar je handtekening onderstaat.

Digitale handtekeningen kunnen door computers verwerkt worden en over een telefoonlijn verstuurd worden. Het is niet zo dat er maar *één* methode is om digitale handtekeningen te construeren. Er zijn verschillende constructies bekend en deze verschillen in de gemaakte aannames en de (extra) eigenschappen die die handtekening kan hebben, zoals: de ontvanger van een boodschap met handtekening kan die handtekening wel controleren, maar hij weet niet wie het ondertekend heeft (zie ook Hoofdstuk 4).

In het eerste hoofdstuk van dit proefschrift worden in het kort enkele bekende cryptografische systemen behandeld (het RSA-systeem en het systeem gebaseerd op de discrete logaritme), en enkele basisbegrippen uit de moderne cryptografie (zoals blobs en zero-knowledge). In de overige zes hoofdstukken worden drie verschillende aspecten van digitale handtekeningen behandeld, namelijk nieuwe theoretische resultaten, constructies van handtekeningen en toepassingen. Deze zes hoofdstukken kunnen onafhankelijk van elkaar gelezen worden, behalve dat

- Hoofdstuk 3 een vervolg is van Hoofdstuk 2, en dat
- in een bewijs in Hoofdstuk 4 en in Hoofdstuk 6 een stelling uit Hoofdstuk 2 gebruikt wordt.

In Hoofdstuk 2 en 3 geven we enkele nieuwe theoretische resultaten voor de veiligheid van klassen handtekeningen systemen, die gebaseerd zijn op RSA.

In Hoofdstuk 2 initialiseert een bepaalde instantie een RSA-systeem met modulus N en geeft s handtekeningen genaamd $H_1,\ldots,H_s$ van bepaalde types aan een individu. Elk zo'n handtekening is niet de wortel van een bepaald residu (zoals gebruikelijk is in RSA), maar is het product van rationale machten van residuen modulo N, en de gebruikte machten bepalen het type van de handtekening. Het individu probeert met de verkregen handtekeningen een nieuwe handtekening H' te berekenen. Onze hoofdstelling zegt dat als de gebruikte residuen willekeurig door die bepaalde instantie gekozen zijn, dat dan het berekenen van H' uit $H_1,\ldots,H_s$ van dezelfde moeilijkheidsgraad is als het berekenen van een bepaalde RSA-wortel op een willekeurig gekozen residu modulo N.

In Hoofdstuk 3 hebben we dezelfde situatie als in Hoofdstuk 2, maar nu worden enkele van de in de handtekeningen gebruikte residuen door die instantie gekozen en de overigen door het individu zelf (dus nu heeft het individu wel invloed op de verkregen handtekeningen). Ook in dit geval wil het individu een nieuwe handtekening H' berekenen uit de verkregen handtekeningen $H_1,\ldots,H_s$. Als we aannemen dat:

- het individu zelf geen RSA-wortels op willekeurig gekozen residuen kan berekenen, en dat
- het individu gebruikt in zijn berekeningen modulo N alleen vermenigvuldigingen en delingen,

dan zegt onze hoofdstelling dat het berekenen van H' uit $H_1,\ldots,H_s$ van dezelfde moeilijkheidsgraad is als het oplossen van een bepaalde tweede graads vergelijking in geheeltallige matrices.

In Hoofdstuk 4 en 5 geven we constructies voor drie nieuwe types van handtekeningen, en een nieuwe, efficiente constructie voor de zogenaamde Fail-stop handtekeningen.

In Hoofdstuk 4 introduceren we de zogenaamde "group signature" en we geven er vier verschillende constructies voor. Zo'n handtekening is bedoeld voor een groep personen en heeft de volgende drie eigenschappen:

- alleen personen uit die groep kunnen een handtekening zetten,
- de ontvanger van de handtekening kan verifiëren dat het een geldige handtekening is, maar hij weet niet welk persoon die handtekening gezet heeft,
- indien noodzakelijk kan een handtekening "geopend" worden, zodat de naam

van de persoon die die handtekening gemaakt heeft bekend wordt.

Handtekeningen die de eerste twee eigenschappen hebben zijn al in de literatuur bekend, maar de derde eigenschap is nieuw.

In Hoofdstuk 5 construeren we drie digitale handtekening systemen, waarbij niemand de handtekening van iemand anders op een nieuwe boodschap kan maken, zelfs niet met onbeperkte rekencapaciteit.

Er zijn constructies bekend voor de volgende drie soorten handtekeningen:

- *undeniable*: een handtekening die niemand kan controleren, tenzij in overleg met de ondertekenaar (op een zero-knowledge manier). Daarom kan iemand die een goede handtekening heeft gekregen niemand anders hiervan overtuigen. En een veronderstelde ondertekenaar kan een vervalste handtekening ontkennen.
- *convertible*: een undeniable handtekening met de extra eigenschap dat de ondertekenaar deze kan veranderen in een gewone digitale handtekening door enige getallen bekend te maken.
- *fail-stop*: als iemand een handtekening vervalst, dan kan de veronderstelde ondertekenaar bewijzen dat het een vervalsing is.

We geven constructies zodat de eerste twee soorten handtekeningen voor het eerst de extra eigenschap krijgen dat niemand de handtekening van iemand anders op een nieuwe boodschap kan maken, zelfs niet met onbeperkte rekencapaciteit. Bovendien geven we een nieuwe constructie voor fail-stop handtekeningen die veel efficienter is dan de bestaande constructie.

In Hoofdstuk 6 en 7 geven we twee toepassingen van digitale handtekening technieken: een nieuw electronisch betalingssysteem en een nieuw soort blob.

In Hoofdstuk 6 presenteren we een nieuw electronisch betalingssysteem, dat het systeem van Chaum, Fiat en Naor vele malen efficienter maakt, en bovendien nieuwe eigenschappen bezit. Betalen met je bankpas en je PIN-code in een winkel is wel electronisch, maar het is niet wat wij bedoelen met een electronisch betalingssysteem. Niemand kan de ingetoetste PIN-code controleren dan alleen de betaalautomaat, waar de geheime sleutel van de bank in zit. Bovendien heeft de gebruiker geen privacy: de bank weet precies waar, wanneer en voor hoeveel je hebt gekocht.

In een electronisch betalingssysteem zoals wij die hier beschouwen heeft iedere gebruiker een kleine computer op zak. Hij gaat er eerst mee naar de bank om deze computer "met geld te laten vullen", dit wil zeggen met speciaal geconstrueerde getallen waar de bank haar digitale handtekening op zet. Bij het betalen laat hij de constructie van deze getallen en de handtekening van de bank zien, en de winkel kan *zelf* controleren of het goed is, zonder een rechtstreeks contact met de bank. In dit systeem kan de bank er niet achterkomen waar, wanneer en voor hoeveel je hebt gekocht.

In Hoofdstuk 7 presenteren we een nieuw commitment systeem: dit is een systeem waarbij je je kunt binden aan een bepaald getal, zonder dat de ontvanger van dit commitment (ook wel blob genaamd) weet welk getal het was, en zonder dat je achteraf dit getal nog kunt veranderen (je geeft dit getal als het ware in een gesloten envelop). De gebruikelijke constructie is met een wiskundige functie $\mathcal{B}$, die de twee bovenstaande eigenschappen moet hebben (in vaktermen: collision-free voor de maker en one-way voor de ontvanger). Als iemand zich wil binden aan een getal b, kiest hij willekeurig een getal r en zijn commitment wordt $B=\mathcal{B}(b,r)$, en hij geeft het getal B aan de ander. Om later dit commitment B te openen maakt hij b en r bekend, zodat de ontvanger B kan controleren.

Wij presenteren een nieuw soort blob: de maker maakt alleen b bekend en bewijst dat hij r weet zonder r bekend te maken. Wij geven verschillende constructies en toepassingen, en we construeren een "tegenspraak" met een in de literatuur bekend lemma.

Acknowledgements

I would like to thank everyone who has contributed—directly or indirectly—to the contents of this thesis. I would like to mention several persons in particular.

First of all, I would like to thank my promotor Henk van Tilborg for all his useful comments and suggestions during the research work. Also I would like to thank my promotor prof. dr. J.H. van Lint for all his suggestions to improve this thesis.

Then I would like to thank David Chaum for introducing me to cryptology, especially to its practical aspects. Several of the problems that are treated in this thesis were posed by him.

I am also very grateful to Bert den Boer, Jurjen Bos, Stefan Brands, Matthijs Coster, Jan-Hendrik Evertse, Maarten van der Ham, Torben Pedersen and Birgit Pfitzmann for their scientific support and encouragements. They listened to all my ideas, which were not always formulated well. It is their patience, criticism, and enthusiasm that purified my ideas. They also proofread my papers and (parts of) this thesis.

Finally, I would like to thank my housemates Aad, Erwin, Gerard, Henk, Herman, Jan, Paul, Peter, Pieter and Theo for so much patience they had with me. They taught me the relativity of science, how to care for other persons, and how to "vivere l'altro".

Index

Stellingen behorende bij het proefschrift

Special Signature Schemes

door

Eugène J.L.J. van Heijst

1.

Het protocol voor "Gradual and verifiable release of a secret" dat gepresenteerd is in het gelijknamige artikel is niet correct, en daarom is Lemma 2 van dat artikel ook niet correct.

Zie: Dit proefschrift, Sectie 4.3.1, en
Ernest Brickell, David Chaum, Ivan Damgård and Jeroen van de Graaf, Gradual and verifiable release of a secret, *Advances in Cryptology—CRYPTO '87*, C. Pomerance ed., LNCS 293, Springer-Verlag, pp. 156-166.

2.

In de nederlandse taal worden begrippen gebruikt met betrekking tot handtekeningen, die niet adequaat weergeven wat er met digitale handtekeningen gebeurt, zoals: *hand-tekening*, *onder*tekenen, *na*maken.

3.

De volgende visualisatie van een blob
is aanschouwelijker dan een dichtgeplakte envelop.

4.

Het kraken van een cryptosysteem houdt nog niet het afkraken van de auteurs in.

5.

Een cryptische uitspraak blijft beter bij.

6.
Het mensbeeld in de cryptologie getuigt van een zodanige argwaan tegenover elkaar, dat, als dit het enige beschikbare mensbeeld zou zijn, de samenleving onleefbaar zou worden. Daarom is het noodzakelijk dat er vanuit verschillende disciplines andere mensbeelden worden gepresenteerd.

7.
Het is terecht dat met behulp van de cryptologie het eigen bezit maximaal beschermd wordt. Zo heeft men volledig de gelegenheid om zijn goederen uit vrije wil in gemeenschap te brengen.

8.
Het religieuze leven met als centrum de Eucharistie heeft ten volle zin als ze uiteindelijk gericht is op eenheid onder de mensen.

9.
Tegen de schijn in tendeert de wereld naar de eenheid.

10.
De uitspraak "toets de theorie aan de praktijk" geldt niet alleen voor onze studie, maar ook voor ons leven.

11.
Ik bemin, dus ik ben.

Zie: De uitspraak "cogito ergo sum" van René Descartes, *Discours de la Méthode* (1637).

12.
Het "hora est" kan niet alleen verlossend werken bij een promotie, maar eveneens bij een homilie, daarbij in het midden latend of dit voor spreker of luisteraar geldt.