Digital Signature Scheme Variations

by

Fiona Emer Siobhan Dunbar

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Combinatorics and Optimization

Waterloo, Ontario, Canada, 2002

Abstract

A digital signature scheme is the process of signing an electronic message that can be transmitted over a computer network. A digital signature provides message authentication that can be proved to a third party. With the rise of electronic communications over the Internet, digital signatures are becoming increasingly important, especially for the exchange of messages of legal significance. In 1988, Goldwasser, Micali and Rivest (GMR) [31] defined a signature scheme as a collection of algorithms: key generation, signature generation and signature verification. They defined a signature scheme as secure if it was existentially unforgeable against a chosen-message attack. These general definitions suited most signatures at the time, however, over the last decade digital signatures have emerged for which the GMR definitions are unsuitable. These signature schemes, together with their applications and security and efficiency considerations, will be explored in this thesis. These signature scheme variations have been classified by the additional services they provide to ordinary signature schemes, namely increased efficiency, increased security, anonymity, and enhanced signing and verifying capabilities.

Acknowledgements

I would like to thank Alfred Menezes for his continuous support in helping me complete this thesis. I would also like to thank the Department of Combinatorics and Optimization and OGS for their financial support over the course of my graduate work. Thanks also to Edlyn Teske and Doug Stinson for reviewing this thesis.

I owe many thanks to my undergraduate mathematics professors, specifically Dr. Ed Wang, Dr. Sydney Bulman-Fleming and Dr. Anthony Bonato for encouraging me to pursue a Master's degree in Mathematics.

Thanks to my family and friends for supporting me over the last few years, particularly Eithne, Tony, Kevin, Siobhan and Aileen Dunbar, Matt Brodie, Karen Priestman, Robin Caicco and Vivian Ibeas.

Contents

Chapter 1

Introduction

Handwritten signatures attached to a document provide us with the means to be certain that the person who signed the document is in fact responsible for it. It is the aim of digital signatures to transfer this property from the physical realm to the electronic realm. Like a handwritten signature, a digital signature provides message authentication that can be proved to a third party. This is known as non-repudiation. Although we assume that it is impossible to copy someone's handwritten signature, a digital signature could be easily copied from one message to another. Thus, it is imperative that one person's digital signature on two different messages be distinct. This implies that the signature must be a function of the signer and the message itself.

With the rise of electronic communications over the Internet, digital signatures are becoming an important addition to, if not substitute for, the written signature. Specifically, digital signature schemes must be in place for all applications of electronic commerce such as digital payment schemes, the purchase of products on-line, and the authentication of commercial software.

A signature scheme is the method of creating and confirming digital signatures on a message to be transmitted over an electronic channel. Not to be confused with encryption, signatures do not need to be sent over a secure channel, since signature schemes do not aim to provide confidentiality, but authenticity. Another common misconception is that digital signatures are a form of identification. With a handwritten signature, the sequence of letters contained in the signature reveals the signer's identity. A digital signature, however, is simply a sequence of 0's and 1's that tell us nothing about the identity of the signer . We will see shortly that identification is an issue of key distribution rather than the signature itself, and that digital signatures aim to authenticate the message rather than the individual.

In 1988, Goldwasser, Micali and Rivest (GMR) [31] defined a signature scheme as a collection of three algorithms: key generation, signature generation and signature verification. In the key generation stage, the signer randomly selects a pair of keys; one will be kept secret, and the other will be published. The secret key or signing key is used in the signature generation algorithm, where the signer signs a message and sends the resulting signature and the message to the verifier. The public key or verification key is used in the signature verification phase, where the verifier checks that the signature on the given message is authentic. If the verification algorithm returns true, the verifier accepts the signature, otherwise it is rejected. Note that all algorithms are public, and only the signing key is kept secret.

Some important aspects of digital signature which will not be discussed further in this paper, but are worth mentioning here, are the problems involved with key generation and key distribution. We indicated that each user must select a secret key in order to create signatures. To prevent others from discovering any information about this secret key, it should be chosen randomly. Generating truly random sequences can be quite time-consuming, so in practice a pseudo-random number generator is often used. It was also mentioned that each user must select a public key. If users are to verify that a signature is valid, they must first be sure that the public key they have obtained is indeed the correct one. This is the problem of key distribution, where the users themselves must be authenticated. A solution is certification, whereby a trusted third party called a *certification authority* binds together a user's identity and her public key. To minimize storage, authentication trees can be used to authenticate several public keys at once using one-way hash functions.

The most crucial requirement of signature schemes is that given some message, it should be infeasible for anyone other than the true signer to compute a signature such that the verification algorithm returns the value "true". If an adversary creates such a signature, it is called a *forgery*. There are three different attack models to consider when defining the information available to an adversary who wishes to forge a signature. In a *key-only attack*, the adversary knows only the public key. For a *known-message attack*, the adversary has a list of valid message-signature pairs at her disposal. In a *chosen-message attack*, the adversary requests the signatures from the true signer on some messages of the adversary's choice. We will consider three possible goals of the adversary when launching an attack on a signature scheme. In a *total break*, the adversary either uncovers the secret key of the signer, or creates an efficient algorithm for forging valid signatures. With *selective forgery*, the adversary can create a valid signature on a particular message of her choice. By *existential forgery*, we mean there exists at least one message for which the adversary can forge a signature.

According to the GMR definition of security [31], an ordinary signature scheme is *secure* if it resists a chosen-message attack by an adversary with computationally bounded resources, whose goal is existential forgery. This is also known as *unforgeability.* In practice, it is unlikely that a signer would sign every message for an adversary, since otherwise there would be no need for forgery. However, by assuming the most powerful adversary with the weakest goal of attack in our notion of security, we also protect against the other forms of attacks.

We say that a signature scheme is *unconditionally secure* if it is secure against an adversary with infinite computational resources. In practice, signature schemes cannot be unconditionally secure since an attacker with infinite resources can always try all signatures up to a given length (assuming there is some publicly known bound on the length of possible signatures) for a given message until a valid one is found. Since we assume an adversary's resources are computationally bounded, this is infeasible provided the bounds are large enough. Thus, signature schemes can only be *computationally secure.* As a result, the security of digital signatures must rely on some underlying, usually unproven mathematical assumptions. These assumptions generally state there is no method for solving a particular problem without unlimited computational power. In terms of digital signatures, if a computationally bounded adversary tries to forge a signature, she will be faced with an instance of some problem for which no feasible solution is known. A signature scheme is *provably secure* if it can be proven secure assuming the underlying problem is infeasible.

Some well-known problems that are believed to be infeasible are factoring an integer n that is the product of two large primes, and finding the discrete logarithm of an element in a cyclic group G. Of course, these problems can be solved by exhaustive search if n is small or if G has small order, so minimum bounds are set on the length of these parameters. We call these bounds *security parameters*. Increasing the security parameters makes the key and signature length increase, which in turn produces a less efficient signature generation and verification. But, it is believed that forging signatures becomes substantially harder with only a slight increase in the security parameters.

To get an idea of what has been discussed so far, we will present a simple and widely used signature scheme called RSA, which is based on the difficulty of factoring.

RSA Key Generation

To generate a public and private key pair, the signer does the following:

1. Randomly selects two large distinct primes p and q and computes $N = pq$ and $\phi(N) = (p-1)(q-1)$.

2. Selects a random integer e such that $1 < e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$.

3. Finds d such that $ed \equiv 1 (\mathrm{mod} \phi(N))$ using the Extended Euclidean algorithm [40].

Then the signer's public key is (N, e) and private key is d.

RSA Signature Generation and Verification

- In order to sign the message $m \in \mathbb{Z}_N$, the signer computes $S = m^d \bmod N$.
- To verify that S is a valid signature on the message m, the verifier computes $S^e = m' \bmod N$, and accepts S iff $m = m'$.

As we've described RSA above, it is always possible for an adversary to forge the signer's signature on random messages. This is accomplished by computing $m = S^e \bmod N$ for some S. Then S is the signer's signature on the message m. There are two ways to prevent such an attack. We can add redundancy or apply a hash function to the message m.

To add redundancy, we require that the message m be padded with a bit-string having some prescribed structure before it is signed. Then, when the signature is verified, it must be checked that $m' = S^e \bmod N$ has the prescribed redundancy. We can see that the attack described above will fail, since the probability that $S^e \bmod N$ has the prescribed form for a random S is very small if the redundancy is defined suitably.

A *hash function* $H : \{0,1\}^* \to \{0,1\}^l$ maps messages of arbitrary length to outputs of fixed bit-length l. In practice, $l = 160$ is common. A *one-way hash function* has the property that given $M = H(m)$, it is computationally infeasible to find a corresponding m. If we assume that a one-way hash function H is applied to the message $m \in \{0,1\}^*$ before it is signed, the attack described above will yield the signer's signature on $M = H(m)$, from which it is infeasible to obtain the original message m. In our subsequent references to RSA, we will employ one-way hash functions in the signature generation and verification protocols as follows:

RSA Signature Generation and Verification with Hash Functions

- To sign the message $m \in \{0,1\}^*$, the signer computes $M = H(m)$ and $S = M^e \bmod N$ and sends the signature S and the original message m to the verifier.
- To verify the signature S on the message $m \in \{0,1\}^*$, the verifier computes $M = H(m)$ and $M' = S^e \bmod N$, and accepts S as a valid signature iff $M' = M$.

In our discussion so far, we have been concerned with the general framework of signature schemes. However, the scope of signature schemes has broadened significantly over the last decade to contain schemes that do not naturally fit into the signature scheme and security definitions discussed earlier. These new schemes were designed for specific applications and are equipped with additional features that cater to the signers and verifiers. These signature scheme variations are the main focus of this thesis.

We will present examples of 19 such signature schemes, including a discussion of their applications, security requirements, and efficiency considerations where appropriate. In our discussions of security, we will provide heuristic arguments rather than formal proofs of security. We will see that in some cases, there are no existing proofs, however, if such a proof is available, we will refer the reader to the appropriate paper.

As we stated, the signature schemes we are interested in do not satisfy the general definitions. In 1993, Pfitzmann [46] rewrote the definition of signature schemes to include all types of schemes. Rather than altering the general digital signature definition, we will present new signatures in comparsion to the well-known GMR definition. To this end, we have classified them into four categories based on their relationship to ordinary signatures. The schemes that supply an increased level of efficiency over ordinary schemes are *On-line/Off-line*, *Batch*, *Server-Aided*, *Incremental*, and *Identity-Based*. *Fail-Stop*, *Arbitrated*, *Threshold*, *Proactive*, and *Forward-Secure* signatures provide an enhanced degree of security. *Blind*, *Partially Blind*, *Fair Blind*, *Group*, and *Ring* signature schemes allow for users to remain anonymous. Finally, schemes which explore the limitations and possibilities of the signer and verifier are *Proxy*, *Undeniable*, *Convertible Undeniable*, and *Designated Confirmer* signatures.

Chapter 2

Schemes with Increased Efficiency

RSA is a very popular scheme on its own as well as being the basis for numerous signature scheme variations. However, RSA involves the computationally expensive modular exponentiation operation in its signature generation stage. On-line/off-line signatures break up the workload of signature generation to minimize the work done on-line. For some computational devices such as a smart card, verification of RSA signatures can also be costly. Batch verification allows for several signatures to be verified at once. Server-aided mechanisms provide another way for smart cards to verify quickly with the help of a computationally powerful source. Incremental signatures use a special type of hash function which can be used for the rapid signing of several closely related messages. Identity-based signatures simplify public key management, improving the overall efficiency of the scheme.

2.1 On-Line/Off-Line

RSA-based signature schemes are very popular in practice because their security lies in the problem of factoring large integers, which is considered infeasible. However, these schemes require performing modular exponentiations with a large modulus in their signature generation stage, which can be very time consuming. In many applications of digital signature schemes, once a message is presented to the signer, a signature should be generated very quickly. An on-line/off-line signature scheme allows for fast signature generation by dividing the computations into two stages: *off-line* and *on-line*. The first stage is performed off-line, before the message to be signed is known to the signer. This stage is often an RSA-based signature scheme, involving the tedious modular exponentiations and, as a result, is relatively slow. The second stage is much faster and is done on-line after the message to be signed is presented to the signer. Here, a one-time signature scheme is often used because it provides a much faster signing process than ordinary signature schemes. One-time signatures can be used to sign at most one message before a new public key is chosen. When an authentication tree is used in combination with a one-time signature scheme, many messages can be signed. For a complete description of how an authentication tree can be used in conjunction with the one-time signature scheme employed in the following example, see [41].

The following example of an on-line/off-line signature scheme uses RSA as the ordinary signature scheme in the off-line portion of the scheme and Merkle's one-time signature scheme [43] during the on-line phase. We will denote the verification and signing keys of the ordinary signature scheme and the one-time signature scheme as (VK, SK) and (vk, sk), respectively. Note that the signing key is the private key and the verification key is the signer's public key.

Key Generation

The key generation for our scheme corresponds to the key generation of RSA. Thus $VK = (N, e)$ and $SK = d$.

Off-Line Computation

The off-line computation consists of the key generation stage of the one-time scheme of keys vk and sk and the signing of vk with RSA. Thus, in order to sign messages of bit-length n, the signer first does the following:

1. Chooses $t = n + \lfloor \log n \rfloor + 1$ random secret strings $k_1, k_2, ..., k_t$ of bit-length l.

2. Computes $v_i = H(k_i)$, $1 \leq i \leq t$, where $H : \{0,1\}^* \to \{0,1\}^l$ is a pre-image-resistant hash function.

Then $vk = (v_1, v_2, ..., v_t)$ and $sk = (k_1, k_2, ..., k_t)$.

Now, the signer signs vk with the RSA signing key, SK, obtaining $\Sigma = (H(vk))^d \bmod N$ and the signer stores (vk, sk, Σ).

On-line Signing

To sign the message $m \in \{0,1\}^n$, the signer retrieves the stored, pre-computed, unused triple (vk, sk, Σ) and computes the one-time signature of m, using sk as follows:

1. Computes c, the binary representation for the number of 0's in m.

2. Forms the bit-string $w = m\|c = (a_1, a_2, ..., a_t)$.

3. Determines the coordinate positions $i_1 < i_2 < \cdots < i_u$ in w such that $a_{i_j} = 1, 1 \leq j \leq u$.

4. Sets $s_j = k_{i_j}, 1 \leq j \leq u$.

Then the one-time signature on the message m is $\sigma = (s_1, s_2, \ldots, s_u)$ and the full on-line/off-line signature on the message m is (vk, Σ, σ).

Note that the one-time signature σ reveals part of the secret key. Suppose two different messages m and m' were signed using the same public key, then $w' = m'\|c'$ will have a 1 in some position where $w = m\|c$ has a 0. This will result in another private key element being revealed to the requester, who could then possibly use the acquired portion of the private key to sign a new message. This explains why the on-line portion of the scheme is a one-time signature scheme.

Signature Verification

To verify that (vk, Σ, σ) is a valid signature on the message m, the verifier does the following:

1. Checks that Σ is a valid RSA signature of vk with respect to VK, i.e., $\Sigma^e \equiv H(vk) \pmod{n}$.

2. Checks that σ is a valid one-time signature of m with respect to vk as follows:

   (a) Computes c, the binary representation for the number of 0's in m.

(b) Forms the string $w = m \| c = (a_1, a_2, ..., a_t)$.

(c) Determines the coordinate positions $i_1 < i_2 < \cdots < i_u$ in w such that $a_{i_j} = 1, 1 \leq j \leq u$.

(d) Accepts σ iff $v_{i_j} = h(s_j)$ for all $1 \leq j \leq u$.

If steps 1 and 2 are verified, then (vk, Σ, σ) is a valid on-line/off-line signature on the message m.

Security

The security of the on-line/off-line signature scheme depends on the security of the one-time signature scheme and the RSA signature scheme against chosen message attacks. The security of RSA relies on the factoring of large integers, which is considered infeasible.

Suppose an attacker launches a chosen-message attack on Merkle's one-time signature scheme. That is, suppose $\sigma = (s_1, s_2, ..., s_u)$ is a one-time signature on the message m, and $w = m \| c$, where c is the binary representation for the number of 0's in m. Let $w' = m' \| c'$, where c' is the binary representation for the number of 0's in $m' \neq m$. An adversary has access only to the portion of the signer's private key which consists of $\sigma = (s_1 = k_{i_1}, s_2 = k_{i_2}, ..., s_u = k_{i_u})$. Thus, the set of coordinate positions in m' having the value "1" must be a subset of the coordinate positions in m having the value "1", in order to be able to use the known portion of the private key. This implies that m' has more 0's than m and $c' > c$, as integers. Then c' will have a "1" in a position where c has a "0". Therefore, the adversary needs the private key element corresponding to this position, which was not contained in σ. We have shown that Merkle's one-time signature scheme is secure against a chosen-message attack. We conclude that the on-line/off-line signature scheme described above is secure.

Efficiency

The off-line computation requires $n + \lfloor \log n \rfloor + 1$ hash function evaluations to generate vk, and one hash evaluation and one modular exponentiation to the d-th power to obtain Σ. The on-line portion of signature generation requires virtually no computation. Signing an n-bit message m requires $l(n + \lfloor \log n \rfloor + 1)$ bits of storage for vk and $l(n + \lfloor \log n \rfloor + 1)$ bits of storage for sk. Suppose the message m contains r 1's. Then c is the binary representation of $n - r$; the number of 0's in m. Let r' be the number of 1's in c. Then the signature σ requires $l(r + r')$ bits of storage. Step 1 of signature verification requires one hash evaluation and one modular multiplication. If we take $e = 3$, which is common in practice, this is a fairly quick computation. Step 2 of signature verification requires no more than $n + \lfloor \log n \rfloor + 1$ hash function evaluations. Since the evaluation of hash functions is very fast, we can conclude that this scheme is very efficient.

2.2 Batch

As we mentioned in the previous section, modular exponentiation produces the greatest computational expense in many digital signature schemes. Batch cryptography allows for many signatures to be generated or verified together. This is very useful in electronic commerce where many customers are interacting with the same banking server. Batching was introduced by Fiat [26] to assist in the signing and decryption operations of RSA. Fiat proposed a procedure for signing whereby messages are first batched together, then a single modular exponentiation is performed, and finally the batch is split apart into individually signed messages. This is possible because of the homomorphic property of RSA. For some variants of Fiat's RSA batch verification, see [26].

Batch cryptography can also be used for signature verification. A technique called the *small exponents test* was developed by Bellare, Garay, and Rabin [6] for batch verification of exponentiation. Before we describe this test, we will show how batch verification of exponentiation works.

Batch Verification of Exponents

Suppose we have n elements $y_1, y_2, ..., y_n$ in a multiplicative group G of prime order q, n exponents $x_1, x_2, ..., x_n \in Z_q$, and a generator g of G. We want to verify that $y_i = g^{x_i}$ for each i, meaning the batch is correct, without performing n full exponentiations. If the batch is correct, then the following equation holds:

$$\prod_{i=1}^{n} y_i = g^{\sum_{i=1}^{n} x_i}. \tag{2.1}$$

Note that the converse is not true. To see this, note that adding a constant to one of the x_i values and subtracting the same constant from another of the x_i values will satisfy equation (2.1), but the batch is incorrect. To make equation (2.1) useful in batch verification, random values must be introduced. By multiplying the x_i values by small random values, which must also be incorporated as small exponents for the y_i's, an attacker needs to correctly guess these random values in order to fool the verifier into thinking an incorrect batch is acceptable.

Small Exponents Test

The small exponents test proceeds as follows, where l is a security parameter:

1. Choose $s_1, s_2, ..., s_n \in \{0,1\}^l$ randomly.
2. Compute $x = \sum_{i=1}^{n} x_i s_i \bmod q$ and $y = \prod_{i=1}^{n} y_i^{s_i}$.
3. If $g^x = y$, accept. Otherwise, reject.

We will now show how the small exponents test can be used for batch verification of RSA signatures. We cannot directly apply the small exponents test to RSA since in step 2, we reduced modulo the group order, and the verifier of an RSA signature does not know the order $\phi(N)$ of the group Z_N^*.

The technique we will describe was used by Yen and Laih [37] for a modification of ElGamal signatures in 1995. Recall that an RSA signature S satisfies $S = M^d \bmod N$, where d is the RSA private key, N is the RSA modulus, and $M = H(m)$, where H is a one-way hash function and m is the message.

Batch Verification of RSA

Suppose that $S_1, S_2, ..., S_n$ are proposed RSA signatures on the hash values $M_1, M_2, ..., M_n$, respectively. The signer sends the verifier the pairs $(m_1, S_1), (m_2, S_2), ..., (m_n, S_n)$. The verifier then proceeds as follows, using the batch verification test to decide whether $S_i = M_i^d \bmod N$ for $i = 1, ..., n$.

1. Computes $H(m_i) = M_i$ for all $i = 1, 2, ..., n$.

2. Chooses $s_1, s_2, ..., s_n \in \{0, 1\}^l$ randomly.

3. Computes $x = (\prod_{i=1}^{n} S_i^{s_i})^e \bmod N$ and $y = \prod_{i=1}^{n} M_i^{s_i} \bmod N$, where e is the signer's RSA public key.

4. If $x = y$, accept. Otherwise, reject.

Security

The method of RSA batch verification described above was chosen for the sake of simplicity and is not at all a secure verification algorithm. In order to be secure, verification must be *complete*, meaning it is always possible for a valid signature to be proved valid and an invalid signature to be proved invalid, and *sound*, which means no valid signature can be proved invalid and no invalid signature can be proved valid. It is clear that the described method of RSA batch verification is complete. However, verification is not sound, since an invalid signature can be proven valid in the following way. Suppose a dishonest signer replaces S_{i_0} by $-S_{i_0}$. Then the batch will be correct with probability 1/2 depending on the parity of s_{i_0}.

Bellare et al. introduced a weaker notion of verification security called *screening*. If this screening condition is satisfied, then one is convinced that the signatures were originally signed by the true signer, even if none of the individual proposed signatures are valid. The batch verification we described above passes the screening test because one cannot create incorrect signatures that satisfy batch verification without knowing the original valid signatures of the signer. Thus, even if the individual signatures are incorrect, the messages M are unchanged. Depending on the application, screening can be a sufficient security condition. Suppose, for example, that a user wants to batch verify many certificates. Here, it is not necessary to obtain the valid signature, but merely to be assured that the messages are authentic. Bellare et al. [6] provide a proof that the RSA batch verification algorithm described above satisfies the screening condition.

Batch verification can also be used for DSA signature schemes. Harn [33] proposed a scheme for batch verifying DSA signatures that is also based on the small exponents test. Boyd and Pavlovski [10] describe a modified small exponents test which they use as a basis for a more secure version of DSA batch verification than the one proposed by Harn.

Efficiency

The small exponents test uses one full exponentiation in Z_N^* and n small exponentiations to obtain x and then n small exponentiations to obtain y. Note that if a public exponent value of $e = 3$ is used, as is often the case in practice, batch verification is no more efficient than individual verification, since the 'small' exponents s_i are most likely greater than 3. However, there are some applications, such as communication between users of varying computational abilities, in which a larger public key and a smaller private key is desired. For example, suppose a smart card with limited computing power generates a signature that is to be verified by a large banking server. Then a small signing key and a large verification key will provide an optimal trade-off. For an in-depth look at applications of larger verification keys, see [36].

2.3 Server-Aided

Server-aided signature schemes allow the client to borrow computing power from an untrusted server without revealing any secret data of the client. Often, the client is a smart card, which is a small, inexpensive secure device that can store and protect data, but which has poor computing power. The untrusted server is a banking terminal having high computational power. The expensive computing power can be put into a few large servers rather than in many small smart cards.

Since the server is an untrusted device, the smart cards must protect their secret information and verify that the computations done by the server are correct. There are two types of attacks on server-aided schemes. In a *passive attack*, the attacker does not participate in the scheme, but rather obtains information about the secret data by simply observing it. An *active attack* involves a malicious server who returns false values during its computation phase in order to find the secret information contained in the smart card.

To secure the secret information transmitted during server-aided computation, this data is blinded using random numbers, or decomposed so that only some of the pieces are revealed to the server. In RSA, where server-aided signature generation is most common, the secret information is the private exponent d. Since this exponent is a 1024-bit integer, the signing of the hashed message M, $S = M^d \bmod N$ should be computed by the server in such a way that the server obtains no knowledge of d. We will see how this is accomplished in the following example due to Béguin and Quisquater [5].

The signature scheme presented uses a method due to Brickell, Gordon, McCurley and Wilson [12] for performing fast exponentiation. In order to compute M^d, one precomputes and stores $M^{x_0}, M^{x_1}, ..., M^{x_{r-1}}$ for some integers $x_0, x_1, ..., x_{r-1}$, and then finds a decomposition $d = \sum_{i=0}^{r-1} a_i x_i$, where $0 \leq a_i \leq h$ for $0 \leq i \leq r-1$. Then one can compute

$$M^d = \prod_{t=1}^{h} c_t^t,$$

where $c_t = \prod_{a_i=t} M^{x_i}$.

To see that this computation does in fact compute M^d, note that

$$\prod_{t=1}^{h} c_t^t = \prod_{t=1}^{h} \prod_{a_i=t} M^{x_i t} = M^{a_0 x_0} M^{a_1 x_1} \cdots M^{a_{r-1} x_{r-1}} = M^{\sum_{i=0}^{r-1} a_i x_i} = M^d.$$

The following algorithm based on the method described above computes M^d using $r+h-2$ multiplications. For a detailed analysis of the efficiency of this algorithm, see [12].

```
$b \leftarrow 1$
$a \leftarrow 1$
**for** $g \leftarrow h$ to 1 by $-1$ **do**
    **for** each $i$ such that $a_i = g$ **do**
        $b \leftarrow b * M^{x_i}$
    **end for**
    $a \leftarrow a * b$
**end for**
return $a$.
```

We are now ready to present the protocol for server-aided RSA signature generation.

Key Generation

The smart card does the following:

1. Computes $N = pq$, where p, q are two large primes, and $\phi(N) = (p-1)(q-1)$.
2. Finds integers e, d such that $ed \equiv 1 \pmod{N}$, $1 < e, d < \phi(N)$.
3. Using the Extended Euclidean Algorithm, finds integers w_p and w_q such that $w_p + w_q = 1, w_p \bmod p = 0, w_q \bmod q = 0$ and $0 \leq |w_p|, |w_q| \leq N$. That is, integers k, l such that $pk + ql = 1, 1 \leq |pk|, |ql| \leq N$, then $w_p = pk$ and $w_q = ql$ are found.

Signature Generation

The following protocol creates a signature on the hashed message M:

1. The smart card randomly chooses integers $a_0, a_1, ..., a_{r-1}$, where $a_i \in \{0, ..., h\}$, and $x_0, x_1, ..., x_{r-1}$, where $l(x_i) \leq j - \log_2(rh) - 2$. ($l(x_i)$ is the bit-length of x_i and $j = \max(l(p), l(q)) - 1$.)
2. The card computes $s_1 = \sum_{i=0}^{r-1} a_i x_i$.
3. The card sends M, N and $x_0, x_1, ..., x_{r-1}$ to the server.
4. The server returns $z_0, z_1, ..., z_{r-1}$, where $z_i = M^{x_i} \bmod N$.
5. The card computes $z_p = \prod_{i=0}^{r-1} z_i^{a_i} \bmod p$ and $z_q = \prod_{i=0}^{r-1} z_i^{a_i} \bmod q$ using the algorithm above. Note that $z_p = \prod_{i=0}^{r-1} M^{x_i a_i} \bmod p = M^{s_1} \bmod p$.

6. The card computes $s_2 = d - s_1$, then blinds s_2 by $\sigma_p = s_2 \bmod (p-1) + \rho_p(p-1)$, where $\rho_p \in_R \{0, 1, ..., q-2\}$ and $\sigma_q = s_2 \bmod (q-1) + \rho_q(q-1)$, where $\rho_q \in_R \{0, 1, ..., p-2\}$.
7. The card sends σ_p and σ_q to the server.
8. The server computes and sends $y_p = M^{\sigma_p} \bmod N$ and $y_q = M^{\sigma_q} \bmod N$ to the smart card.
9. The card computes $s_p = y_p z_p \bmod p$ and $s_q = y_q z_q \bmod q$.
10. The card computes $S = w_q s_p + w_p s_q \bmod N$.
11. The card verifies that $S^e \bmod N = M$.

Then, as long as step 11 is verified, the smart card transmits S as the signature on the message M. To see that this will produce a valid signature, note that

$$\begin{aligned}
S &= w_q s_p + w_p s_q \bmod N \\
&= (w_q(y_p z_p \bmod p) + w_p(y_q z_q \bmod q)) \bmod N \\
&= (w_q M^{\sigma_p}(M^{s_1} \bmod p) + w_p M^{\sigma_q}(M^{s_1} \bmod q)) \bmod N \\
&= (w_q M^{s_2 \bmod (p-1) + \rho_p(p-1)}(M^{s_1} \bmod p) + w_p M^{s_2 \bmod (q-1) + \rho_q(q-1)} \\
&\quad \times (M^{s_1} \bmod q)) \bmod N \\
&= (w_q M^{s_2 \bmod (p-1)} M^{\rho_p(p-1)}(M^{s_1} \bmod p) + w_p M^{s_2 \bmod (q-1)} M^{\rho_q(q-1)} \\
&\quad \times (M^{s_1} \bmod q)) \bmod N \\
&= (M^{s_1} w_q M^{s_2 \bmod (p-1)} M^{\rho_p(p-1)} \bmod p + M^{s_1} w_p M^{s_2 \bmod (q-1)} M^{\rho_q(q-1)} \bmod q) \\
&\quad \bmod N
\end{aligned}$$

$$\begin{aligned}
&= (M^{s_1} w_q M^{s_2 \bmod (p-1)} \bmod p + M^{s_1} w_p M^{s_2 \bmod (q-1)} \bmod q) \bmod N \\
&= M^{s_1} M^{s_2} (w_q + w_p) \bmod N \\
&= M^{s_1} M^{s_2} (1) \bmod N \\
&= M^d \bmod N
\end{aligned}$$

Security

There is no proof that this server-aided signature scheme is secure. Béguin and Quisquater [5] present several possible passive attacks including exhaustive search. However, these attacks are all shown to be ineffective. They also show that an active attack previously presented by Pfitzmann and Waidner [48] is impossible because of the random values chosen by the smart card.

Efficiency

Suppose we have a 1024-bit RSA modulus. To minimize the number of modular multiplications done by the card, and to ensure that the attacks proposed by Béguin and Quisquater remain ineffective, we take $h = 10$ and $r = 19$ in the above protocol. Then the number of modular multiplications done by the card is 25, as compared to 516 multiplications using the Chinese Remainder Theorem to compute an RSA signature without the aid of the server. This is an acceleration factor of 20.6.

2.4 Identity-Based

Identity-based cryptography was introduced by Shamir in 1984 for the simplification of key management in e-mail systems. For identity-based signature schemes, Shamir proposed that each participant use their e-mail address as their public key. The corresponding secret keys must be computed by a key generation centre rather than the users themselves since if a user could compute her own secret key, she could compute any other user's secret key. Suppose user A signs a message with her secret key given to her by the key generation centre. Then user B can easily obtain A's e-mail address and use it to verify the signature.

Identity-based signature schemes provide a way for users to sign and verify each others signatures without the exchange of private and public keys. There is also no need for the storage of keys in a central directory nor the online assistance of a third party. Instead, a trusted key generation centre issues each user a smart card containing unique identification information such as the user's email address. This information is the user's public key. The trusted centre computes the corresponding secret key which is also embedded in the smartcard. Once the cards are issued, the trusted centre is no longer required for the remainder of the scheme. New users can be added at any time and previous cardholders do not need to update their cards as a result.

We will describe an identity-based signature scheme due to Fiat and Shamir [27] that is a variation of Shamir's original scheme [52].

Key Generation

Before the trusted centre begins issuing smart cards, it selects and publishes a modulus $N = pq$ and a pseudorandom function $f : \{0,1\}^* \to [0, N)$. In this signature scheme, only the trusted centre knows the factors of N, so the same modulus can be used for all signers requesting cards. When a signer requests a smart card, the centre produces a string I which contains information about the signer such as her name, address, and ID number and about the smart card such as the expiration date. The trusted centre does the following:

1. Computes $v_l = f(I, l)$ for small values of l.
2. Chooses k distinct values of l, say $l_1, l_2, ..., l_k$, for which v_l is a quadratic residue modulo N and compute the smallest square root s_l of v_l^{-1} modulo N.
3. Issues a smart card which contains I, the k s_l values, and their indices.

Then the signer's public key is I and the indices $(l_1, l_2, ..., l_k)$ and her secret key is $(s_{l_1}, s_{l_2}, ..., s_{l_k})$. Note that this scheme is not strictly identity-based, as the indices $l_1, ..., l_k$ are included as part of the public key.

Signature Generation

In order to sign the message m, the signer does the following:

1. Selects random integers $r_1, ..., r_t \in [0, N)$ and computes $x_i = r_i^2 \bmod N$.
2. Computes $b = f(m, x_1, ..., x_t)$ and defines $e_{i,j}$ for $1 \leq i \leq t, 1 \leq j \leq k$, as the first kt bits of b.

3. Computes $y_i = r_i \prod_{e_{i,j}=1} s_{l_j} \bmod N$ for $1 \leq i \leq t$.

Then $(e_{i,j}, y_i)$ for all $1 \leq i \leq t, 1 \leq j \leq k$ is the signature on the message m. Note that the $e_{i,j}$'s can be arranged in matrix form for simplicity.

Signature Verification

To verify that $(e_{i,j}, y_i)$ is a valid signature on the message m, the verifier does the following:

1. Computes $v_l = f(I, l_j)$ for $1 \leq j \leq k$.

2. Computes $z_i = y_i^2 \prod_{e_{i,j}=1} v_{l_j} \bmod N$ for $1 \leq i \leq t$.

3. Computes $b' = f(m, z_1, ..., z_t)$ and verifies that the first kt bits of b' are $e_{i,j}$ for $1 \leq i \leq t, 1 \leq j \leq k$.

To see that signature verification works, note that

$$z_i \equiv y_i^2 \prod_{e_{i,j}=1} v_{l_j} \equiv r_i^2 \prod_{e_{i,j}=1} (s_{l_j}^2 v_{l_j}) \equiv r_i^2 \equiv x_i \pmod{N}.$$

This implies that $f(m, z_1, ..., z_t) = f(m, x_1, ..., x_t)$, and thus the first kt bits of b' are the same as the first kt bits of b, which is equal to $e_{i,j}$.

Security

The security of this scheme is based on the difficulty of computing square roots modulo N which is equivalent to factoring N. In order to forge signatures without attempting to find the secret key, a forger could try to guess the $e_{i,j}$ values for all

$1 \leq i \leq t, 1 \leq j \leq k$. This guess will be correct with probability 2^{-kt}. In order to complete this attack, the forger computes $x_i = r_i^2 \prod_{e_{i,j}=1} v_{l_j} \bmod N$ for t random values of $r_i \in [0, N)$ and let $y_i = r_i$. The forger then sends the $e_{i,j}$ matrix and y_i to the verifier. Then the verification condition holds since $z_i = y_i^2 \prod_{e_{i,j}=1} v_{l_j} \bmod N = r_i^2 \prod_{e_{i,j}=1} v_{l_j} \bmod N = x_i$ and the verifier accepts the forged signature.

Efficiency

For the attack mentioned above to be considered infeasible, we require that $kt \geq 80$. If we choose $k = 10$ and $t = 8$, we achieve a security level of 2^{80}. The private key is comprised of k integers modulo N, where N is a 1024-bit modulus. This takes up $(1024/8) \times 10 = 1280$ bytes of storage. If we assume a signature is comprised of the kt $e_{i,j}$ bits and the t 1024-bit y_i values, a signature is $(80/8) + (1024/8)8 = 1034$ bytes in length. The average number of modular multiplications in computing the x_i's and the y_i's in generating the signature is $(tk/2) + t = 8(12)/2 = 48$ if we assume roughly 1/2 of the $e_{i,j}$'s are equal to 1.

Note that there is a trade-off between the length of the private key and the length of the signature. As the length of the key increases, the signature length decreases and vice versa. Also, the number of modular multiplications decreases with t, so, depending on the amount of storage and computing power available, one can adjust the values of k and t accordingly. Fiat and Shamir also propose that by computing the y_i's in parallel, we can further reduce the number of multiplications to 4% of that required in the RSA signature scheme.

2.5 Incremental

Incremental signatures, introduced by Bellare, Goldreich and Goldwasser in 1994 [7], are signatures that are easily updated after the message is slightly modified. The time it takes to update the signature is proportional to how much the message is modified. Incremental signatures use a variation of the hash-then-sign mechanism. The message m is hashed using a collision-free hash function H to obtain $H(m)$. Then $H(m)$ is signed using an ordinary signature scheme obtaining $\sigma(H(m))$. The final signature on the message m is $\sigma(H(m)$. To update this signature on a modified message m' of m, the hash of m' is computed according to the incremental hashing algorithm and then this hash value is signed from scratch.

As an application, suppose a signer wishes to send the same long message to many different users so that the text is identical except for the heading information which specifies the intended recipient. Rather than computing the hash value of each message from scratch, only one message needs to be completely hashed and the others can be hashed quickly using only the altered portion of the original message. Note that incremental hashing is only useful when signing long messages, since ordinarily computing a signature is more expensive than computing a hash function for inputs of the same length.

Recall that a collision-free hash function maps arbitrarily long inputs to outputs of a fixed length so that it is computationally infeasible to find two distinct inputs which map to the same hash value. Thus in order to devise an incremental signature scheme, we first need a collision-free hash function H which satisfies the following: Let $m = m_1, m_2, ..., m_n$ be a message viewed as a sequence of blocks. Suppose block i is modified to m'_i and m' is the modified message. Given $H(m), i, m_i$ and m'_i, computing $H(m')$ is easy.

Most hash functions employed in practice involve iterations, in which each successive block in the message contributes to the final hash value. Clearly, these hash functions are useless for incrementality, where we only want to recompute the hash of the altered block of the message. Merkle [42] describes a tree structure in which adjacent blocks of the message are hashed together to obtain a string half the length of the original, then the process is repeated until a final hash value is computed. Here, a single block of the tree can be modified and only the hash computations from this point of the tree down need to be done. However, the entire tree must be stored, and for incremental hashing we wish to store only the final hash value. The following incremental scheme due to Bellare and Micciancio [8] accomplishes this by using a new method for constructing collision-free hash functions called *randomize-then-combine.*

Randomize

1. The message m is written as a sequence of blocks $m_1, m_2, ..., m_n$, where each block is b bits long.

2. For each block $i = 1, 2, ..., n$, concatenate the binary encoding $< i >$ of the block index i to the message block m_i. We will let $l = \log(n) + b$ denote the bit-length of the largest such concatenated string $< n > m_n$.

3. The "randomizing" function H is applied to each concatenated string as fol-

lows:

$$H(< i > m_i) = y_i.$$

The function H in this construction must be a collision-free compression function, where $H : \{0,1\}^l \rightarrow \mathbb{Z}_p^*$ and p is a large prime.

Combine

The outcomes y_i are then combined to yield the final hash value

$$y = \prod_{i=1}^{n} y_i \bmod p.$$

Since the "combining" operation is multiplication, we call our collision-free hash function MuHASH.

Incrementality

Now suppose block i of the message m changes from m_i to m_i'. Given the final hash value y of the original message m, one can simply recompute the hash value y' of the altered message m' as follows:

$$y' = y \cdot H(< i > m_i)^{-1} \cdot H(< i > m_i') \bmod p.$$

Once the hash value is computed on the message m, an ordinary signature scheme such as RSA can be applied to the hash value.

Security

We are really only concerned with the security of the hash function construction and not the signature scheme. In particular, we require that the hash function we've constructed is collision-free. An interesting result in [8] shows that, if p is chosen large enough so that the discrete logarithm problem in $\mathbb{Z}_p^*$ is hard, and H is an 'ideal' hash function, then MuHASH is collision-free. This result is perhaps surprising since MuHASH does not appear to have any relation to discrete logarithms. The result can be equivalently stated as: If there is any attack that finds collisions in MuHASH then there is an efficient way to compute discrete logarithms in $\mathbb{Z}_p^*$. In practice, this statement is stronger still since even if the discrete logarithm problem were easy, we still don't know a method for finding collisions in MuHASH.

Efficiency

Computing MuHASH takes one multiplication per block of m. To speed up this computation, the block size can be increased. The increment operation takes one inverse operation and two multiplication operations in $\mathbb{Z}_p^*$ and two evaluations of H. Thus, we can see that incrementing is faster than separately hashing long messages using a regular hash function such as SHA-1. Note that because the randomizing function H is not iterative, it is possible to apply H to the blocks of the message in parallel. The multiplications in computing y can also be accomplished in parallel. Given sufficient hardware, this can greatly reduce the cost of computing this hash function.

Chapter 3

Schemes with Increased Security

As we mentioned in the introduction, ordinary signature schemes can only be computationally secure. Thus, an attacker with exceptionally large computational power could potentially break these schemes. If an adversary does succeed in forging a signature, is there any way to distinguish between a forged valid signature and an authentic valid signature? This is an important question in legal terms because it introduces the problem of liability. In the case of forgery, the supposed signer will still be held responsible. Another breach of security occurs if the signer falsely denies her own signature. Fail-stop signature schemes prevent against both situations by means of a disavowal protocol. Another object of attack is the secret key of the signer. This key can be shared with a third party in arbitrated signature schemes so that no one can forge without the help of the arbitrator. Rather than sharing the entire secret key with another user, one can divide the key into shares which can be distributed among many users. Threshold signatures are generated in this way and maintain that a minimum prescribed number of users must collaborate to reconstruct the secret key. Proactive signatures also use the idea of distribution together with the notion of key share renewal. This provides an even higher level of security as shares of the secret are updated over time further restricting the ability of an attacker to discover the secret. Forward secure schemes contain an update algorithm for the private keys, so that in case the current key is exposed, previous signatures are still valid.

3.1 Fail-Stop

Fail-stop signature schemes provide an improved level of security over ordinary signature schemes in the sense that if someone succeeds in forging a signature, the supposed signer can prove it is a forgery. If a forged signature is shown to the signer, she can prove that the underlying assumption upon which the fail-stop signature scheme is based has been broken. More precisely, the signer's proof of forgery is not actually a mathematical proof, but a string that should be infeasible to construct (even by the signer) based on the underlying assumption. This string provides evidence that the assumption has been broken, and thus a forgery has occurred. Once such a forgery is detected, the signing algorithm is no longer used; hence the term "fail-stop" [47].

The fail-stop signature scheme involves a trusted third party as well as a proof of forgery algorithm. With this algorithm, a forger is unable to construct signatures that could be verified by the verification algorithm without performing an exponential amount of computation, and a signer is unable to construct signatures that could later be declared forgeries. We will now provide an example of a particular fail-stop signature scheme due to van Heyst and Pedersen [54] that relies on the intractability of the discrete logarithm problem.

Key Generation

The trusted third party (TTP) does the following:

1. Selects primes p and q, where q divides $p-1$ and q is large enough so that the discrete logarithm problem is intractable in the subgroup H_q of order q in $\mathbb{Z}_p^*$.
2. Randomly selects $g \in \mathbb{Z}_p^*$ and computes $\alpha = g^{(p-1)/q} \bmod p$. If $\alpha = 1$, the TTP chooses a new g.
3. Randomly selects a secret integer a, $1 \le a \le q-1$, and computes $\beta = \alpha^a \bmod p$.
4. Sends the public values (p, q, α, β) to the signer.

The signer then does the following:

1. Randomly selects secret integers $x_1, x_2, y_1, y_2 \in \{0, 1, ..., q-1\}$.
2. Computes $\beta_1 = \alpha^{x_1}\beta^{x_2} \bmod p$ and $\beta_2 = \alpha^{y_1}\beta^{y_2} \bmod p$.

Then the signer's public key is $(\beta_1, \beta_2, p, q, \alpha, \beta)$ and her private key is $\bar{x} = (x_1, x_2, y_1, y_2)$.

Note here that our assumption that the discrete logarithm problem is intractable in H_q maintains that only the TTP has knowledge of a. The mathematical assumption in this particular fail-stop signature scheme is that the discrete logarithm problem is infeasible in H_q. Hence, evidence of forgery would be the integer a. We will describe the proof of forgery algorithm momentarily, but first we will present the algorithms for signature generation and verification.

Signature Generation

To sign a message $m \in \{0, 1, ..., q-1\}$, the signer computes

$$\begin{aligned} S_{1,m} &= x_1 + my_1 \bmod q, \\ \text{and } S_{2,m} &= x_2 + my_2 \bmod q. \end{aligned}$$

Then $(S_{1,m}, S_{2,m})$ is the signature on the message m.

Note that this is a one-time signature scheme, so each time a signature is generated, the signer should choose a new secret $\bar{x}$. Otherwise, if two different messages were signed with the same $\bar{x}$, this secret $\bar{x}$ could be uncovered from the two systems of linear equations in two unknowns for $S_{1,m}$ and $S_{2,m}$.

Signature Verification

To verify the signer's signature on the message m, the verifier does the following:

1. Obtains an authentic copy of the signer's public key $(\beta_1, \beta_2, p, q, \alpha, \beta)$.
2. Computes $v_1 = \beta_1 \beta_2^m \bmod p$ and $v_2 = \alpha^{S_{1,m}} \beta^{S_{2,m}} \bmod p$.
3. Accepts the signature $(S_{1,m}, S_{2,m})$ iff $v_1 = v_2$.

To see that signature verification works, note that

$$v_1 \equiv \beta_1 \beta_2^m$$

$$\begin{aligned}
&\equiv (\alpha^{x_1}\beta^{x_2})(\alpha^{y_1}\beta^{y_2})^m \\
&\equiv \alpha^{x_1+my_1}\beta^{x_2+my_2} \\
&\equiv \alpha^{S_{1,m}}\beta^{S_{2,m}} \\
&\equiv v_2 \pmod p.
\end{aligned}$$

Proof of Forgery

To prove the signature $S' = (S'_{1,m}, S'_{2,m})$ on a message m is a forgery, the signer does the following:

1. Computes a signature pair $S = (S_{1,m}, S_{2,m})$ for m using the private key $\bar{x}$ (see the signature generation stage).

2. If $S = S'$ then stop - the forgery is undetectable; otherwise proceed to step 3.

3. Computes $a = (S_{1,m} - S'_{1,m})(S_{2,m} - S'_{2,m})^{-1} \bmod q$.

To see that this algorithm works, from the verification algorithm we have

$$\begin{aligned}
\alpha^{S_{1,m}}\beta^{S_{2,m}} &\equiv \alpha^{S'_{1,m}}\beta^{S'_{2,m}} \pmod p \\
\Leftrightarrow \alpha^{S_{1,m}-S'_{1,m}} &\equiv \alpha^{a(S'_{2,m}-S_{2,m})} \pmod p \\
\Leftrightarrow S_{1,m} - S'_{1,m} &\equiv a(S'_{2,m} - S_{2,m}) \pmod q \\
\Leftrightarrow a &\equiv (S_{1,m} - S'_{1,m})(S'_{2,m} - S_{2,m})^{-1} \pmod q.
\end{aligned}$$

Security

In order for a fail-stop signature scheme to be considered secure, it must be resist existential chosen-message attacks. Before we present our main theorem of security, which states that an adversary can only compute a valid signature with negligible probability, we require some preliminary results regarding the keys of the scheme.

Lemma 3.1.1 *There are q^2 different quadruples $\bar{x} = (x_1, x_2, y_1, y_2)$ that yield the same pair (β_1, β_2) as part of the signer's public key.*

Proof Recall that $\beta_1 \equiv \alpha^{x_1}\beta^{x_2} \equiv \alpha^{x_1}\alpha^{ax_2} \equiv \alpha^{x_1+ax_2} \pmod{p}$, where α, β, p, q, and a are fixed by the TTP. Fix β_1. Now there are q ways to choose x_1, and for each x_1 there is one choice for x_2 which produces the same β_1. Thus, there are q pairs (x_1, x_2) that give the same β_1. Similarly, there are q pairs (y_1, y_2) that produce the same β_2. This implies that there are q^2 quadruples $\bar{x} = (x_1, x_2, y_1, y_2)$, which yield the same pair (β_1, β_2).

Lemma 3.1.2 *Suppose $\bar{x} = (x_1, x_2, y_1, y_2)$ and $\bar{x}' = (x'_1, x'_2, y'_1, y'_2)$ are two distinct quadruples that yield the same portion of the signer's public key (β_1, β_2). Suppose $\bar{x}$ produces a valid signature $(S_{1,m}, S_{2,m})$ on a message m. Then $\bar{x}'$ also produces a valid signature on m.*

Proof Note that $\beta_1 \equiv \alpha^{x_1}\beta^{x_2} \equiv \alpha^{x'_1}\beta^{x'_2} \pmod{p}$ and $\beta_2 \equiv \alpha^{y_1}\beta^{y_2} \equiv \alpha^{y'_1}\beta^{y'_2} \pmod{p}$. Also, note that if m is signed using $\bar{x}'$, then $S'_{1,m} = x'_1 + my'_1 \bmod q$ and $S'_{2,m} = x'_2 + my'_2 \bmod q$. We need to show that $(S'_{1,m}, S'_{2,m})$ is a valid signature for m. This is true because

$$\begin{aligned} v_2 &\equiv \alpha^{S'_{1,m}}\beta^{S'_{2,m}} \pmod{p} \\ &\equiv (\alpha^{x'_1+my'_1}\beta^{x'_2+my'_2}) \pmod{p} \\ &\equiv \alpha^{x'_1}\beta^{x'_2}(\alpha^{y'_1}\beta^{y'_2})^m \pmod{p} \end{aligned}$$

$$\begin{aligned} &\equiv \beta_1\beta_2^m \pmod p \\ &\equiv v_1 \pmod p. \end{aligned}$$

Lemma 3.1.3 *Let T be the set of q^2 quadruples that yield the same portion of the signer's public key (β_1, β_2). For each $m \in \mathbb{Z}_q$, there are exactly q quadruples in T that give the same signature $(S_{1,m}, S_{2,m})$.*

Proof We want to determine the number of quadruples (x_1, x_2, y_1, y_2) that satisfy the following congruences:

$$\begin{aligned} \beta_1 &\equiv \alpha^{x_1}\beta^{x_2} \pmod p \\ \beta_2 &\equiv \alpha^{y_1}\beta^{y_2} \pmod p \\ S_{1,m} &\equiv x_1 + my_1 \pmod q \\ S_{2,m} &\equiv x_2 + my_2 \pmod q. \end{aligned}$$

Since α is a generator of the cyclic subgroup H_q of order q in $\mathbb{Z}_p^*$, we can write

$$\begin{aligned} \beta_1 &\equiv \alpha^{c_1} \pmod p \\ \beta_2 &\equiv \alpha^{c_2} \pmod p \\ \beta &\equiv \alpha^{a} \pmod p. \end{aligned}$$

Combining these two systems of equations, we obtain

$$c_1 \equiv x_1 + ax_2 \pmod q$$

$$\begin{aligned} c_2 &\equiv y_1 + ay_2 \pmod q \\ S_{1,m} &\equiv x_1 + my_1 \pmod q \\ S_{2,m} &\equiv x_2 + my_2 \pmod q. \end{aligned}$$

In terms of matrices over $\mathbb{Z}_q$, we have

$$\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \\ 1 & 0 & m & 0 \\ 0 & 1 & 0 & m \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ S_{1,m} \\ S_{2,m} \end{pmatrix}.$$

Row-reducing, we can simplify the coefficient matrix to

$$\begin{pmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & a \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We end up with three equations in terms of y_2:

$$\begin{aligned} x_1 &\equiv c_1 + ay_2 \pmod q \\ x_2 &\equiv c_2 - my_2 \pmod q \\ y_1 &\equiv s_{1,m} - ay_2 \pmod q. \end{aligned}$$

Each choice for y_2 gives a unique solution for x_1, x_2, and y_1. Since there are q

possibilities for $y_2 \in \{0, 1, ..., q-1\}$, this gives q solutions for (x_1, x_2, y_1, y_2).

Corollary 3.1.4 *Let $m' \in \mathbb{Z}_q$, $m \neq m'$. Then the q quadruples in T that yield the signer's signature $(S_{1,m}, S_{2,m})$ for m, yield q different signatures for m'.*

Theorem 3.1.5 *Given that $(S_{1,m}, S_{2,m})$ is a valid signature for m and $m' \neq m$, an adversary can compute a valid signature $(S_{1,m'}, S_{2,m'})$ for m' with probability $1/q$.*

Proof Using Lemma 3.1.3 and Corollary 3.1.4, we note that given that $(S_{1,m}, S_{2,m})$ is a valid signature for m, there are q possible quadruples (x_1, x_2, y_1, y_2) that give the same signature $(S_{1,m}, S_{2,m})$ for m. But for any message $m' \neq m$, these q quadruples will produce q different signatures on m'. One of these will be the valid signature for m'.

We have shown that the probability of a successful forgery is $1/q$ if an adversary has access to a message and a valid signature created by the signer. Suppose an adversary only has access to the signer's public key. Then, by Lemma 3.1.1, there are q^2 quadruples which give rise to the public key (β_1, β_2), q of which produce a valid signature. Once again, the probability of a successful forgery is $q/q^2 = 1/q$. Note that this probability does not depend on the computational power of the adversary. The unconditional security arises from the fact that the adversary has no idea which secret quadruple is being used by the signer.

Since we have achieved a level of unconditional security, given a valid signature on a message m, it is infeasible for an adversary to compute the signer's signature on a different message m'. It is, however, still possible for an adversary to forge her own signature on the message m' that will pass the verification condition. Nevertheless, as we mentioned previously, if the signer is given this forged signature, she can produce a proof of forgery with probability $1 - 1/q$. This probability arises from step 2 in the proof of forgery algorithm. The probability that $S = S'$ and the proof of forgery fails is $1/q$, by Lemma 3.1.2. The evidence of forgery is the value $a = \log_\alpha \beta \bmod q$, which was supposed to have been known only to the trusted third party.

Efficiency

The described fail-stop signature scheme has a complexity comparable with RSA signature schemes. The construction of a fail-stop signature requires two multiplications and two additions modulo a prime number. Signature verification can be accomplished in less than two modular exponentiations modulo a prime. Thus, fail-stop signature generation is very efficient, while fail-stop signature verification is only slightly slower than RSA signature verification.

3.2 Arbitrated

Arbitrated signatures were introduced as a method of employing symmetric key encryption in digital signature schemes. Because public key cryptography was relatively new at the time, it was believed that a signature scheme based on symmetric key encryption would offer more security than its public key counterpart. With symmetric key encryption, the ability to sign and verify could be limited to those with whom a secret key was shared. To prevent users with knowledge of the secret key from signing on behalf of other users, a trusted third party, called the *arbi-*

trator, is required in the signature generation and verification stages. Since the arbitrator's cooperation is involved in both these stages, the arbitrator can control who is is capable of signing and verifying signatures.

The scheme begins with each possible signer and verifier, sharing their secret key with the arbitrator. This secret key will be used as a means of authenticating users. The scheme we will present below uses a message authentication code algorithm to accommodate the authentication requirement rather than an encryption function, as is employed by Davies and Price [24]. Since encryption was designed to achieve confidentiality, it is not desirable to use it to obtain authenticity.

Key Generation

The arbitrator selects a secret key $k_T \in K$, where K is the key space. Each user A selects a random secret key $k_A \in K$ and sends it to the arbitrator over a secure and authentic channel.

Signature Generation

In order to generate a signature for a message m, A does the following:

1. Computes $M = H(m)$, where $H : \{0,1\}^* \rightarrow \{0,1\}^l$ is a one-way hash function.

2. Computes $u = MAC_{k_A}(M)$, where $MAC_k : \{0,1\}^l \rightarrow \{0,1\}^l$ is a message authentication code algorithm.

3. Sends u, M and a public identification string I_A to the arbitrator.

Upon receiving u, M and I_A, the arbitrator does the following:

1. Verifies that $u = MAC_{k_A}(M)$.

2. Computes $S = E_{k_T}(M \parallel I_A)$.

3. Sends S to A.

Then A's signature on the message m is S. Suppose Alice wants to send the authenticated message m to Bob. She sends the pair (m, S).

Signature Verification

In order to verify Alice's signature on the message m, Bob does the following:

1. Computes $v = MAC_{k_B}(S)$.

2. Sends v, S and his public identification string I_B to the arbitrator.

Upon receiving v, S and I_B, the arbitrator does the following:

1. Verifies that $v = MAC_{k_B}(S)$.

2. Computes $E_{k_T}^{-1}(S)$ to get $M \parallel I_A$.

3. Computes $w = MAC_{k_B}(M \parallel I_A)$.

4. Sends w and $M \parallel I_A$ to Bob.

Upon receiving w and $M \parallel I_A$, Bob does the following:

1. Verifies that $w = MAC_{k_B}(M \parallel I_A)$.

2. Computes $M' = H(m)$.

3. Accepts signature S iff $M' = M$.

Security

The security of this arbitrated signature scheme is dependent on the particular symmetric key encryption function chosen. The authenticity of the scheme depends on the security of the message authentication code algorithm. The arbitrator provides increased security because no one can forge a signature without the cooperation of the arbitrator. Since each users' private key is shared authentically and secretly with the arbitrator prior to signature generation, only the authentic signers will be given signatures. An additional feature of this scheme is that the arbitrator never sees the message m shared between users A and B. However, the arbitrator must be unconditionally trusted with the secret keys of the two users and there is nothing to stop the arbitrator from forging signatures herself.

Efficiency

Because symmetric-key encryption is faster than public-key encryption, the process of creating and verifying arbitrated digital signatures is quite efficient. However, there are many message and key exchanges involved between users and the arbitrator in order to sign a single message.

3.3 Threshold

Threshold cryptography involves the distribution of some secret information into shares such that a minimum predetermined number, or threshold, of shares are required in order to reconstruct or apply the secret. Each share is then given to a group of servers called shareholders. Some threshold schemes involve a trusted dealer, whose task is to choose and distribute the secret information into shares. The secret information is often the private key of the signer in signature schemes. Thus, a signature can be generated without anyone but the dealer knowing the signing key. There are two types of threshold protocols depending on the application. *Threshold secret sharing* is used for single-use data. The secret information can be derived at most once from a set of secret shares without being revealed. *Threshold function sharing* is used for multiple-use keys such as the private key of a certified authority. The distributed key can be used repeatedly without compromising security.

Threshold cryptography is an important concept which is used widely by certification authorities. A certification authority (or CA) is a trusted third party whose signature binds an entity to their public key. Since a CA signs many public keys, its signing key is a target for attackers who may wish to certify their own invalid public keys. If the CA's signing key is kept in one place, once this site is broken into, the secret key is exposed. However, if the key is distributed among n sites using a (t, n) threshold signature scheme, an attacker must break into t sites in order to reveal the secret key. Threshold cryptography is also the idea behind key escrow, whereby two or more escrow agents hold part of the private key. As needed, the escrow agents are called upon to use their part of the key to compute the entire secret key.

The two important requirements of threshold systems are data secrecy and data integrity. Suppose only data secrecy was desired. Then the secret information could be divided among all n shareholders and all users must cooperate to reveal the secret. However, if one shareholder alters her part of the secret, the secret information is destroyed. On the other hand, suppose data integrity was the only requirement. To satisfy this condition, n copies of the secret key could be made so that each shareholder obtains a copy of the entire key. This would prevent against up to $n-1$ shareholders collaborating maliciously in attempting to alter the secret key. However, any single shareholder could easily give away the entire secret key to an adversary. To achieve both data secrecy and data integrity, a (t, n) threshold protocol involves a group of t shareholders such that any t shareholders can combine their shares to obtain the secret information s, and no $t-1$ shares will yield any significant information about s.

The following threshold secret sharing RSA protocol was designed by Gennaro et al. [29]. This (t, n) threshold signature scheme has two phases. In the first phase called the *Dealing Phase*, the signing key d is distributed by the dealer among n shareholders or players $\{P_1, P_2, \ldots, P_n\}$ so that each player P_i has a share d_i of d. These shares must be chosen carefully so that during the second phase called the *Signature Phase*, given the message m, any subset of t players can collaborate to produce a valid RSA signature S on a message m from their partial signatures $S_i = m^{d_i} \bmod N$. We will discuss the formation of the d_i's and how the resulting signature is computed using Lagrange interpolation below, but we can note here that it is imperative that all partial signatures be correct in order to generate a valid signature. This scheme contains a protocol for detecting an improper partial signature. This is accomplished through the use of public verification data to ensure that each player is providing the correct partial signature. The verification data is generated during the dealing phase and is comprised of a public sample message w and its corresponding partial signature $w^{d_i} \bmod N$ for each of the shares d_i. These sample signatures are made public to all the players so that during the signature phase, each player P_i can verify the partial sample signatures computed by any other player P_j.

Before we describe the dealing phase, signature phase and the interactive verification protocol for detecting invalid partial signatures, we will present the following observation which yields a method for computing polynomials.

Let $p(x) = \sum_{i=0}^{t} a_i x^i$ be a polynomial of degree t with coefficients in a finite field F having order greater than t. Then the following conditions hold:

1. Values $\{p(x_i)\}_{i=1}^{t+1}$ for distinct values x_i determine p on all points by Lagrange's interpolation formula:

$$p(x) = \sum_{i=1}^{t+1} \left(p(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right).$$

2. t or fewer values $\{p(x_i)\}_{i=1}^{t}$ for distinct values of x_i yield no information about $p(y)$ for any single value of $y \notin \{x_1, x_2, \ldots, x_t\}$.

De Santis et al. [51] show how to extend this notion from a finite field F to the ring $\mathbb{Z}_{\phi(N)}$. For simplicity, in the following discussion, we will assume that Lagrange interpolation works in the ring $\mathbb{Z}_{\phi(N)}$, even though some technical modifications are required.

Dealing Phase

The dealer selects an RSA modulus N, a public key e, and a secret key $d \in \mathbb{Z}^*_{\phi(N)}$, such that $ed \equiv 1(\mathrm{mod}\phi(N))$. The dealer selects the key shares $d_i \in \mathbb{Z}^*_{\phi(N)}$, and sends share d_i to player P_i for each $i = 1, 2, \ldots, n$. Once the d_i's are distributed, the dealer discards the secret key d.

The d_i's are chosen as follows:

- The dealer chooses a random polynomial $p(x)$ of degree at most $t-1$ with coefficients in $\mathbb{Z}_{\phi(N)}$, such that $p(0) = d$. That is, $p(x) = \sum_{i=0}^{t-1} a_i x^i$, where $a_0 = d$ and $a_i \in_R \mathbb{Z}_{\phi(N)}$ for $1 \leq i \leq t-1$.

- Each d_i is then computed as $d_i = p(i)$.

Using Lagrange interpolation, it is possible to reconstruct d from the d_i's given any t shares d_j in the following way: $d = \sum_{j=1}^{t}(d_j \prod_{l \neq j} \frac{0-l}{j-l}) \bmod \phi(N)$. However, we do not want a single user to recompute the secret key d since this would counteract the purpose of a threshold signature scheme. Rather, we would like the signature to be jointly created by all players without any of them knowing the secret key d.

Signature Phase

To compute the signature S on the message m, each player P_i computes and publishes their partial signature $S_i = m^{d_i} \bmod N$. Given t of these shares, say $S_1, S_2, \ldots, S_t$, anyone can compute $S = \prod_{j=1}^{t} S_j^{\prod_{l \neq j} \frac{0-l}{j-l}} \bmod N$. To see that this collaboration generates an RSA signature, note that

$$S \equiv \prod_{j=1}^{t} S_j^{\prod_{l \neq j} \frac{0-l}{j-l}} \equiv \prod_{j=1}^{t} m^{d_j \prod_{l \neq j} \frac{0-l}{j-l}} \equiv m^{\sum_{j=1}^{t} (d_j \prod_{l \neq j} \frac{0-l}{j-l})} \equiv m^d \pmod{N}.$$

Note that any subset of t partial signatures S_j will generate the signature for m.

In order for the resulting signature to be correct, each of the partial signatures S_i must be correct. A solution is provided for detecting an improper partial signature. The public exponents e_i corresponding to d_i are unknown to even the signer P_i, since otherwise, the signer could use e_i and d_i to find $\phi(N)$ and factor N. Since only the dealer knows the e_i's, users are unable to verify the partial RSA signatures in the usual manner. Rather, the signers must generate some new information which will allow the other users to verify these partial signatures. The idea is that in order to verify a partial signature on the message m under the key share d_i, it suffices to know a single sample message w and its corresponding correct partial signature $w^{d_i} \bmod N$. The following two phases comprise a protocol for detecting incorrect partial signatures.

Generation of Verification Data

The dealer chooses a random value $w \in \mathbb{Z}_N^*$ and publishes it. Each player P_i, $i = 1, 2, \ldots, n$, computes their partial signature on the sample message w, $w_i = w^{d_i} \bmod N$, and publishes w_i after it has been verified by the dealer.

Verification of Partial Signatures

Each player P_i must verify that the partial signatures produced by each other player P_j is correct. For simplicity, we will examine the protocol between two players P_1 and P_2. Suppose P_1 wishes to verify that P_2's claimed partial signature on the message m is correct. Recall that P_1 has access to the sample message w, its partial signature $w_2 = w^{d_2} \bmod N$ as signed by P_2, the message m, and P_2's claimed partial signature S_2. The interactive verification protocol is as follows:

1. P_1 randomly chooses $i, j \in \{1, 2, \ldots, N\}$, computes $R = m^i w^j \bmod N$ and sends R to P_2.

2. P_2 computes $S_R = R^{d_2} \bmod N$ and sends S_R to P_1.

3. P_1 verifies that $S_R = S_2^i w_2^j \bmod N$. If this holds, then P_1 accepts S_2 as the correct partial signature on m. Otherwise, P_1 rejects it, and P_2 is exposed as the malicious shareholder.

This interactive verification works since

$$\begin{aligned}
S_R &\equiv S_2^i w_2^j \pmod N \\
\Leftrightarrow R^{d_2} &\equiv S_2^i w^{d_2 j} \pmod N \\
\Leftrightarrow (m^i w^j)^{d_2} &\equiv S_2^i w^{d_2 j} \pmod N \\
\Leftrightarrow m^{i d_2} &\equiv S_2^i \pmod N \\
\Leftarrow m^{d_2} &\equiv S_2 \pmod N.
\end{aligned}$$

Note that the partial signature will be accepted as valid if $S_2 = bm^{d_2} \bmod N$, where b is a square root of unity in $\mathbb{Z}_N$. However, $b \equiv \pm 1 \pmod{N}$ is the only possibility since if P_2 could find S_2 such that $b \equiv S_2 m^{-d_2} \pmod{N}$ is a non-trivial root of unity, then she could factor N, which was assumed to be infeasible. Thus, S_2 is accepted as valid if and only if $S_2 \equiv \pm m^{d_2} \pmod{N}$. This may cause both resulting signatures S and $-S$ to be accepted as valid. This problem can be quickly overcome by verifying the final signature using the public exponent e.

Security

The scheme outlined above is both robust and unforgeable. Robustness is satisfied for $t \leq \lceil n/2 \rceil$, meaning that even if up to $t-1$ players behave maliciously, the correct signature will still be generated. This is because of the ability to detect malicious players. The d_i values belonging to the malicious players can simply be left out of the signature generation stage, and a subset of size t can be chosen which does not contain any of the faulty d_i's, since $t \leq \lceil n/2 \rceil$. Unforgeability is met since a subset of less than t players is unable to produce a valid signature because in order to perform polynomial interpolation, t shares S_j are required.

The final security requirement is the completeness, soundness and zero-knowledge property of the interactive verification protocol. We showed above that completeness is satisfied when we proved that verification works. Gennaro et al. [29] show that a cheating player P_2 can only convince P_1 to accept an invalid signature $\tilde{S}_m \neq m^{d_2} \bmod N$ with probability $\frac{O(1)}{p}$. Thus, the protocol is sound. The protocol as it stands above is not zero-knowledge. That is, some information can be leaked as a result of a cheating P_1, who could simply choose an R which does not comply with the protocol and learn $S_R = R^{d_2} \bmod N$, which could not be computed by P_1 alone. However, through the use of a commitment function, which is collision-resistant and one-way, P_2 can be assured that the R as chosen by P_1 was formed correctly. This is accomplished by sending $commit(S_R)$ to P_1 instead of S_R. Only after P_1 reveals the values i and j used to compute R and P_2 checks that $R = m^i w^j \bmod N$, does P_2 send S_R to P_1. This technique provides the desired zero-knowledge property and thus the scheme is secure.

Efficiency

This threshold RSA signature scheme involves a small constant number of modular exponentiations in order to sign a message m. The interactive protocol increases the complexity of an ordinary threshold signature scheme because of the series of checks between pairs of players. However, it is only necessary to carry out this error detection procedure if the final signature $m^d \bmod N$ is first found to be incorrect. That is, the resulting signature can be checked using the public key e, and if the signature is verified, there is no need to carry out the verification of the partial signatures.

3.4 Proactive

Like threshold cryptography, the goal of proactive cryptography is to protect the secrecy of the private keys used in signature schemes and encryption schemes. Using a threshold signature scheme, an attacker must break into many sites before the secret key is exposed. This provides a high level of security as long as the scheme is used only for a short period of time. However, keys that remain unchanged for a long time, such as the signing keys of a certified authority or a bank, run the risk of being revealed by an attacker who has a considerable amount of time to uncover all shares of the key. Proactive schemes combine the concept of key distribution used in threshold schemes with the notion of key share renewal. Instead of distributing fixed key shares to the players, the key shares are updated at frequent intervals to help prevent against the attack mentioned above. The idea is to fix the secret key, but change its representation into key shares from one interval to the next. Proactive signature schemes use threshold signature schemes as a basis, but the entire lifetime of the scheme is split up into time periods such that in each time period, it is assumed that no more than t players will follow the protocol incorrectly. A scheme is *t-proactive* if it is secure and robust even if up to t players are corrupted in any given time period. Therefore, an attacker who wants to learn the secret key must corrupt $t+1$ players in a relatively short amount of time.

The concept of proactive security was introduced by Ostrovsky and Yung [45] in 1991. The first proactive signature scheme for RSA was developed by Jakobsson et al. [35], but was inefficient. There are many simple proactive RSA signature schemes which are secure against passive attacks, but most do not prevent against active adversaries such as the players themselves refusing to follow the protocol correctly. The scheme that we will present is due to Rabin [49], and includes a protocol for reconstructing shares of the secret key in the case that a player will not cooperate in the signature generation and share renewal phases.

This scheme is comprised of a key distribution stage, signature generation stage and a share-refreshing protocol. This proactive scheme is an RSA based scheme where (N, e) is the public key, d is the private key to be shared, g is an element of high order in $\mathbb{Z}_N^*$, n is the number of players, $L = n!$, and t is the threshold.

Key Distribution

During the key distribution stage, the dealer shares the key d in additive form using an (n, n) threshold scheme. Each share d_i is then further shared using a (t, n) threshold sharing scheme. This back-up procedure allows for a corrupted player's share d_i to be reconstructed by t other players. The dealer then computes and publishes all the partial signatures on a sample message g. These *witness* signatures $w_i = g^{d_i} \bmod N$ enable the players to verify each others partial signature on the message m during the signature generation stage.

The complete protocol the dealer executes for sharing the secret key $d \in \mathbb{Z}_{\phi(N)}^*$ is the following:

1. Select random $d_i \in [-nN^2, ..., nN^2]$, for each $1 \leq i \leq n$, send d_i to player P_i, and set $d_{public} = d - \sum_{i=1}^{n} d_i$.

2. Compute the witness or sample value $w_i = g^{d_i} \bmod N$, for each $1 \leq i \leq n$.

3. Share d_i using the Sharing Protocol (see below).

Signature Generation

The signature to be generated on the message m is $m^d \bmod N$. Each player must use their partial key d_i to generate this signature. Since $d = d_{public} + \sum_{i=1}^{n} d_i$, then

$m^d \equiv m^{d_{public}+\sum_{i=1}^{n} d_i} \equiv m^{d_{public}} \prod_{i=1}^{n} m^{d_i} \pmod{N}$. Thus, each player must publish their partial signature $\sigma_i = m^{d_i} \bmod N$ on the message m in order to obtain the resulting signature $S = m^{d_{public}} \prod_{i=1}^{n} \sigma_i \bmod N$.

Now suppose a player generated their partial signature incorrectly. Then, the resulting signature will not be verified and the players proceed in checking each of the partial signatures so the corrupted user can be detected. Recall that for each share d_i, we have a public sample partial signature w_i. Each player will be asked to prove that their partial signature on the message m is correct by showing that $\log_m \sigma_i = \log_g w_i$; this can be accomplished using the partial signature verification protocol for threshold signature described in Section 3.3. Since $\log_g w_i = d_i$, if equality holds we can be assured that the partial signature σ_i was generated correctly. If equality does not hold, this player's share d_i will be reconstructed using the Reconstruction procedure which we will describe momentarily.

Before describing the protocol for share renewal, we will pause and give an explanation of the protocols for sharing and reconstructing secrets, which are used throughout this scheme. This *verifiable secret sharing protocol* is based on a scheme due to Feldman [25].

Sharing Protocol

In order to distribute some secret $s \in [-nN^2, ..., nN^2]$ among the n players, the dealer does the following:

1. Randomly chooses $a_1, a_2, \ldots, a_t \in [-nL^2N^3, nL^2N^3]$ and defines the polynomial $f(x) = sL + a_1x + a_2x^2 + \cdots + a_tx^t$.

2. Computes $f(i) \in \mathbb{Z}$ for $1 \leq i \leq n$ and $b_0 = g^{sL} \bmod N, b_1 = g^{a_1} \bmod N, \ldots, b_t = g^{a_t} \bmod N$.

3. Gives player P_i the value $f(i)$ and publishes the values $b_0, b_1, \ldots, b_t$.

To verify that the dealer has executed the protocol correctly, the following verification protocol is performed between each player and the dealer.

1. Each player P_i verifies that $g^{f(i)} \equiv \prod_{j=0}^{t} (b_j)^{i^j} \pmod N$. If this not hold, she requests that the dealer publishes $f(i)$.

2. The dealer publishes the shares that were requested by the players in step 1. If she refuses to do so, she is disqualified.

3. Each player P_i repeats step 1 for all shares published by the dealer in step 2. If any verification fails, the dealer is disqualified. If a player's published share is valid, then that player is disqualifies.

To see that verification works, note that

$$g^{f(i)} = g^{sL+a_1 i+a_2 i^2+\ldots+a^t i^t} = g^{sL} g^{a_1 i} g^{a_2 i^2} \cdots g^{a_t i^t} \equiv b_0 b_1^i b_2^{i^2} \cdots b_t^{i^t} \pmod N$$

Reconstruction Protocol

In the case that a player refuses to cooperate, a threshold value of t players can collaborate to reconstruct the secret value s as follows:

1. Each player P_i publishes $f(i)$.

2. Player P_j finds a set I of size $t+1$ of indices such that for all $i \in I$, the following equation holds: $g^{f(i)} \equiv \prod_{j=0}^{t}(b_j)^{i^j} \pmod{N}$. That is, the shares $f(i)$ for $i \in I$ are genuine.

3. P_j selects a prime $P > 2nN^2$ and computes the secret s using polynomial interpolation: $s = \sum_{i \in I} f(i) \prod_{j \in I, j \neq i} \frac{-j}{i-j}/L$ smod P, where

$$x \text{ smod } P = \begin{cases} x & \text{if } x < P/2, \\ x - P & \text{if } x > P/2. \end{cases}$$

Share-refreshing Protocol

The share-refreshing protocol allows us to change the representation of the secret key as shares while maintaining the value of the key. We must also change the representation of the back-up of the shares. This is accomplished as follows: Each player splits her share d_i into sub-shares $d_{i,j}$, gives each player one of the sub-shares, and sets $d_{i,public} = d_i - \sum_{j=1}^{n} d_{i,j}$. Then the new shares are formed by adding together all the subshares received from the other players. For example, player i receives the subshares $d_{1,i}, d_{2,i}, ..., d_{n,i}$ and forms her new share $d_i^{new} = \sum_{j=1}^{n} d_{j,i}$. The player also sets $d_{public}^{new} = d_{public} + \sum_{i=1}^{n} d_{i,public}$. We can see that this new sharing is still an additive sharing of $d = \sum_{i=1}^{n} d_i + d_{public}$ since

$$\begin{aligned} \sum_{i=1}^{n} d_i^{new} + d_{public}^{new} &= \sum_{i=1}^{n}\sum_{j=1}^{n} d_{j,i} + d_{public} + \sum_{i=1}^{n} d_{i,public} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j} + \sum_{i=1}^{n} d_{i,public} + d_{public} \end{aligned}$$

$$\begin{aligned} &= \sum_{i=1}^{n}\left(\sum_{j=1}^{n} d_{i,j} + d_{i,public}\right) + d_{public} \\ &= \sum_{i=1}^{n} d_i + d_{public} = d. \end{aligned}$$

Now suppose that P_i does not generate her subshares correctly, i.e. $\sum_{j=1}^{n} d_{i,j} \neq d_i$. This can be detected by each player generating public sample values $g_{i,j} = g^{d_{i,j}} \bmod N$ for the new additive sub-shares $d_{i,j}$. Then each other player can check that $g^{d_{i,public}} \prod_{j=1}^{n} g_{i,j} \pmod N = w_i$.

We also need to ensure that the sharing of the new values d_i^{new} is correct. From the previous verification we have a witness signature d_i^{new} of the form $w_i^{new} = g^{d_i^{new}} \bmod N$. Also, when the d_i^{new}'s are shared using the sharing protocol, the value $(g^{d_i^{new}})^L \bmod N = b_0$ is computed as a by-product. To verify that the sharing protocol was carried out correctly, we need only check that $b_0 = (w_i^{new})^L \bmod N$.

The protocol for updating the shares of the secret key d is summarized:

1. P_i randomly chooses $d_{i,j} \in [-N^2, N^2]$ for all $1 \leq j \leq n$, computes and publishes $g_{i,j} = g^{d_{i,j}} \bmod N$, and sets $d_{i,public} = d_i - \sum_{j=1}^{n} d_{i,j}$.

2. P_i sends $d_{i,j}$ to P_j.

3. P_j verifies that $d_{i,j} \in [-N^2, N^2]$ and $g_{i,j} = g^{d_{i,j}} \bmod N$. If not, P_j requests that the $d_{i,j}$ chosen by P_i be made public and then P_j computes $g_{i,j} = g^{d_{i,j}} \bmod N$.

4. If P_i refuses to cooperate in step 3, P_i's share of the key d_i is reconstructed by the other players using the Reconstruction protocol.

5. P_j verifies that $w_i = g^{d_{i,public}} \prod_{j=1}^{n} g_{i,j} \bmod N$. If not, P_i's share of the key d_i is reconstructed by the other players using the Reconstruction protocol.

6. P_i computes her new share $d_i^{new} = \sum_{j=1}^{n} d_{j,i} + \sum_{i=1}^{n} d_{i,public} + d_{public}$, and shares it with the other players using the Sharing protocol. Note that as a result of this protocol, the value $g^{sL} \bmod N$ is revealed, where $s = d_i^{new}$.

7. If P_i refuses to share her new secret or $(g^{d_i^{new}})^L \neq g^{sL} \bmod N$, then P_j publishes $d_{i,j}$. If P_j refuses to publish $d_{i,j}$, then P_j's share d_j is reconstructed by the other players using the Reconstruction protocol.

Security

The scheme described above is robust since even if t players are corrupted, the correct signature will still be generated for $t < n/2$ because this leaves t honest players to collaborate and reconstruct the corrupted players shares. This feature provides the optimal resilience characteristic which is so often unattained in previous RSA proactive schemes. This scheme also prevents against the exposure of the secret key even in the case that a player's share of the key is reconstructed and revealed. As long as we assume that not all the players' shares are compromised, which is a realistic assumption, the security is intact. For a detailed proof of the security of the verifiable secret sharing protocols, see [49] and [25].

Efficiency

The sharing of the key in additive form allows for simple signature generation as well as an efficient share renewal process. To generate a signature, each player must perform one exponentiation as long as the players all follow the protocol correctly. If a fault occurs, the faulty player must be detected and then their shares must be recovered using the verifiable secret sharing Reconstruction Phase, which is an efficient protocol. In some previous RSA proactive schemes, the size of the shared key increases linearly with the number of players involved or the run time increases linearly with the number of faults. The scheme we have described runs in constant time even when faults occur.

3.5 Forward Secure

Forward secure signature schemes were first proposed by Anderson in 1997 [3] as a way of preserving the validity of previously signed messages even after the secret signing key has been compromised. Ordinary digital signature schemes do not have this property. In fact, once the secret key of an ordinary scheme is lost, all past and future signatures are meaningless. A malicious signer could also post the secret signing key k anonymously on the Internet and claim that her key was lost. Then the signer can claim that she never signed messages that she did in fact sign before the key was published. We can see that this can be disastrous in terms of achieving non-repudiation.

The idea behind forward secure schemes is that T, the total time for which a public key is valid, is split into time periods where each period uses a different secret key. The public key remains fixed while each subsequent secret key is computed from the previous key using a key update algorithm. The update algorithm is public, so there is nothing stopping an adversary from forging future secret keys after discovering the current secret key. However, once a forgery is detected, the public key can be revoked to prevent against this attack. Even if the current secret key is compromised, previous secret keys cannot be computed from this revealed key. The time period during which a message was signed is included in the signature so that it can be verified that the signature was created in the time indicated.

Previous forward secure signature schemes contain efficient signing and key update algorithms but very slow verification. We will present the first scheme where both signing and verifying are as efficient as the underlying ordinary scheme upon which it is based. This forward secure scheme due to Itkis and Reyzin [34] is based on the ordinary Guillou-Quisquater signature scheme [32]. We will begin by giving a brief description of the GQ signature scheme. In the following, k and l are security parameters and H is a hash function. Typical values of k and l are $k = 1024$ and $l = 160$.

Guillou-Quisquater Signature Scheme

To generate the signing key and verification key, the signer does the following:

1. Generates random $\lceil k/2 \rceil$-bit primes p_1, p_2 and computes their product $N = p_1 p_2$.
2. Randomly selects $s \in \mathbb{Z}_N^*$ and $e \in [2^l, 2^{l+1})$ such that $\gcd(e, \phi(N)) = 1$.
3. Computes $v = 1/s^e \bmod N$.

Then the signing key is (s, e, N) and the verifying key is (v, e, N). To sign the message m, the signer does the following:

1. Randomly selects $r \in \mathbb{Z}_N^*$.
2. Computes $y = r^e \bmod N$, $\sigma = H(y, m)$, and $z = rs^\sigma \bmod N$.

Then the signature on the message m is (z, σ).

To verify that (z, σ) is a valid signature on the message m, the verifier does the following:

1. Checks that $z \not\equiv 0 \pmod N$. If $z \equiv 0 \pmod N$, reject.
2. Computes $y' = z^e v^\sigma \bmod N$.
3. Verifies that $\sigma = H(y', m)$. If not, reject.

Note that the security of this scheme is dependent on the assumption that computing e-th roots modulo N is infeasible without knowledge of the factors of N.

The idea of the forward secure version of the GQ scheme is for the signer to generate distinct $(l+1)$-bit primes $e_1, e_2, \ldots, e_T$ such that $\gcd(e_i, e_j) = 1$ and $\gcd(e_i, \phi(N)) = 1$ for all $1 \leq i, j \leq T$, where T is the lifetime of the fixed public key. The e_i's are generated by a deterministic algorithm. Then the signer selects $s_1, s_2, \ldots, s_T$ such that $s_i^{e_i} \equiv 1/v \pmod N$ for $1 \leq i \leq T$. At time period i, the signer uses the GQ signature scheme with secret key (s_i, e_i, N) to sign the message m and the verifier uses the GQ verification procedure with public key (v, e_i, N) to verify that the signature is valid. Then forward security is obtained because if a forger breaks in during time b and learns the secret key s_b, it is possible to compute the e_b-th, e_{b+1}-th,...,e_T-th roots of v modulo N from $s_{b+1}, ..., s_T$ because of the public update algorithm, however, they will be unable to compute the e_j-th root of v for $j < b$.

During key generation, the signer computes the e_i's and the s_i's as we mentioned above. However, the signer does not store all these values since this would require storage linear in T. This problem is addressed by defining $f_i = e_i \cdot e_{i+1} \cdots e_T \bmod \phi(N)$ and introducing t_i such that $t_i^{f_i} \equiv 1/v \pmod{N}$. Then, during time period j, the signer stores only s_j and t_{j+1}. When it comes time to update the secret key, the signer computes $s_{j+1} = t_{j+1}^{f_{j+2}} \bmod N$ and $t_{j+2} = t_{j+1}^{e_{j+1}} \bmod N$. This requires only two values modulo N to be stored at any time.

Note that the f_i's and the e_i's are not stored by the signer. Rather, the e_j for the current time period j, as well as a small fixed string p from which the e_i's were generated, are stored as part of the secret key sk_j. During the update algorithm, the new e_i's for $i > j$ can be regenerated using the seed p.

Key Generation

In order to generate the public key for the total time T and the initial secret key sk_1, the signer does the following:

1. Generates random $(\lceil k/2 \rceil - 1)$-bit primes q_1, q_2 such that $p_i = 2q_i + 1$ are both prime and computes $N = p_1 p_2$.

2. Selects t_1 randomly from $\mathbb{Z}_N^*$.

3. Using a deterministic algorithm such as sequential operations from a fixed starting point p, generates primes e_i such that $2^l(1 + (i-1)/T) \leq e_i < 2^l(1 + i/T)$ for $1 \leq i \leq T$. (The bounds on e_i ensures that the e_i are pairwise distinct.)

4. Sets $f_2 = e_2 \cdot e_3 \cdots e_T \bmod \phi(N)$.

5. Computes $s_1 = t_1^{f_2} \bmod N$, $v = 1/s_1^{e_1} \bmod N$, and $t_2 = t_1^{e_1} \bmod N$.

Then the secret key for the initial time period is $sk_1 = (1, s_1, t_2, e_1, p, N)$ and the public key pk_1 is (N, v, T).

Update Algorithm

To generate the secret key for time $j+1$ from the previous secret key $sk_j = (j, s_j, t_{j+1}, e_j, p, N)$, the signer does the following:

1. If $j = T$, return to key generation protocol.

2. Regenerates $e_{j+1}, ..., e_T$ using the fixed starting point p.

3. Computes $s_{j+1} = t_{j+1}^{e_{j+2} \cdot e_{j+3} \cdots e_T} \bmod N$ and $t_{j+2} = t_{j+1}^{e_{j+1}} \bmod N$.

Then the new secret key is $sk_{j+1} = (j+1, s_{j+1}, t_{j+2}, e_{j+1}, p, N)$.

As we mentioned, the message m is signed according to the GQ signature scheme in time period j with secret key sk_j. Included in the signature is the current value e_j used to sign the message. This is an important feature which speeds up signature verification by enabling the verifier to obtain e_j without having to recompute the e_i's from scratch for $i \leq j$. Since a dishonest signer could easily include an incorrect e_j as part of the signature, there are several things that need to be checked by the verifier to be sure that the e_j contained in the signature is the true e_j. These checks are carried out during signature verification. We will call the supposed e_j included in the signature e and the actual value e_j for simplicity.

Firstly, it must be verified that $e > 2^l$ and $\gcd(e, \phi(N)) = 1$ as in the original GQ signature scheme. It is easy for the verifier to check that $e > 2^l$, and as long as e is odd and $e < 2^{\lceil k/2 \rceil - 1}$, then e is relatively prime with $\phi(N) = 4q_1q_2$.

Secondly, the verifier must check that e is relatively prime with $e_b, ..., e_T$, where b is the break-in time period. This is to ensure that the e_1-th,...,e_j-th roots of v for $j < b$ cannot be computed by the attacker even though she knows the e_b-th,...,e_T-th roots of v. However, it is not a simple task for the verifier to check that e is relatively prime with e_b,...,e_T since she does not know b, the time the attack occurs, nor does she know the values e_b,...,e_T. The solution to this problem is for the interval between 2^l and 2^{l+1} to be split into T consecutive subintervals of size $2^l/T$, where e_i is a prime from the i-th subinterval. Then the verifier can be assured that the true values e_{j+1},...,e_T are primes greater than or equal to $2^l + \frac{2^l j}{T}$. So if the e included in the signature for time period j is less than $2^l + \frac{2^l j}{T}$, then it must be relatively prime with the true values $e_{j+1}, ..., e_T$ and thus with $e_b, ..., e_T$, since $b > j$.

Signature Generation

In order to sign the message m using the current signing key $sk_j = (j, s_j, t_{j+1}, e_j, p, N)$, the signer does the following:

1. Selects a random integer r from $\mathbb{Z}_N^*$.

2. Computes $y = r^{e_j} \bmod N$.

3. Computes $\sigma = H(j, e_j, y, m)$, where $H : \{0,1\}^* \rightarrow \{0,1\}^l$ is a hash function.

4. Computes $z = rs_j^\sigma \bmod N$.

Then (z, σ, j, e_j) is the signature on the message m.

Signature Verification

To verify that (z, σ, j, e_j) is a valid signature on the message m, the verifier does the following:

1. If $e_j \geq s^l(1 + j/T)$ or $e_j < 2^l$ or e_j is even, reject.

2. If $z \equiv 0 \pmod{N}$ then reject.

3. Compute $y' = z^{e_j} v^\sigma \bmod N$.

4. If $\sigma = H(j, e_j, y', m)$ then accept. Otherwise, reject.

Security

A formal proof of security is given in [34], so we will only informally describe how forward security is obtained in this scheme.

Suppose an attacker breaks in during time b and uncovers the signer's secret key s_b. Using the public update algorithm, the attacker can easily compute $s_{b+1}, ..., s_T$. Note that $s_b^{-1} = \sqrt[e_b]{v}$, $s_{b+1}^{-1} = \sqrt[e_{b+1}]{v}$,...,$s_T^{-1} = \sqrt[e_T]{v}$. Since the e_i's are relatively prime for $1 \leq i \leq T$, knowing the e_b-th, e_{b+1}-th,..,e_T-th roots of v will not assist the attacker in computing the e_j-th root of v for $j < b$. Thus, an attacker will be unable to compute the signer's secret key s_j for any previous time period.

Efficiency

As noted, the key feature of this scheme is the efficiency of both signing and verifying, which is equivalent to that of the underlying GQ scheme. Signing and verifying each take two modular exponentiations, one modular multiplication and an application of H. This gives a total time of $O(k^2 l)$ plus the time to apply H. This time can be improved upon by performing one of the two modular exponentiations for signing off-line before the message to be signed is known. Key generation requires $O(k^5 + l^4 T + k^2 l + klT)$ bit operations and the key update algorithm takes $O(k^2 lT + l^4 T)$ bit operations. These running times are explained in [34].

Chapter 4

Schemes with Anonymity Services

The primary goal of ordinary signature schemes is authenticity rather than confidentiality. However, in some applications of digital signatures, a degree of privacy is desirable. In a blind signature scheme, the message to be signed is hidden by a blinding function so that the signer cannot see the message she signs. This prevents the signer from linking the signed message to the user who requested the signature. A partially blind signature contains some additional information to prevent against fraud that can occur in a fully blind signature. Fair blind signature schemes involve a trusted third party who can link signed messages to users in specific situations. So far, the signatures mentioned hide the message to be signed. Alternatively, the signer's identity can be hidden among an ensemble of other users. Group signatures prevent the receiver of a signature from linking the signature to a specific member of the collection of users who collaborated to sign it. There is a group manager who can revoke the identity of the signer in case of a dispute. Ring signatures provide an even stronger degree of anonymity since there is no such group manager.

4.1 Blind

Blind signature schemes were first introduced by David Chaum in 1982 [18]. Chaum developed the notion of an electronic coin, which maintains the privacy features of an actual coin, namely untraceability and unlinkability. Electronic coins are bit strings produced and signed by the bank using a blind signature scheme to ensure authenticity. The blind signature scheme ensures that the bank does not see the actual coin it is signing, and consequently, cannot link the buyer of the coin to the coin itself. For example, suppose the buyer of the coin uses it to purchase an item at a retail store. Eventually the coin is deposited back to the bank by the retail store. The bank cannot trace the coin back to the owner because it did not see the coin to begin with, since it was signed using a blind signature scheme. With the use of blind signature schemes, spending patterns of consumers cannot be monitored.

There are two parties involved in the signature generation stage of this scheme. There is a *requester* A and a *signer* B. A sends a message to B, which is then signed by B and returned to A. A uses this signature to compute B's signature on another message of A's choice. Hence, the signer issues a blind signature without knowing the message she signs and therefore is unable to link the requester's signed message with the requester.

The blind signature scheme contains an ordinary signature scheme as a subroutine of the signature generation stage. The signature generation stage consists of three actions: *blinding*, *signing* and *unblinding*. During the blinding stage, performed by A, the message m is given some predetermined redundancy and then a function f called the *blinding function* is applied to the message m. The signing portion of the signature generation stage uses some general signature scheme, which the signer B uses to sign a blinded message $f(m)$. B then sends her signature on the blinded message, $S_B(f(m))$, to A. In the unblinding stage, the requester A applies an *unblinding function* g to $S_B(f(m))$ defined by $g(S_B(f(m)) = S_B(m)$. Thus, B's signature on the message m is obtained with B knowing neither m nor her resulting signature on m. Note that the choice of blinding and unblinding functions will be determined by the choice of ordinary signature scheme used.

To illustrate the process of a blind signature scheme, we will describe an example from Chaum [19] of a blind signature scheme that uses the RSA signature scheme in the signing stage.

Key Generation

We will define S_B according to the definition of an RSA signature. Thus, the signer B will choose a public key (N, e) and private key d. The requester A will select a fixed secret integer k such that $\gcd(N, k) = 1$.

Signature Generation

Since we will be working with the RSA signature scheme, appropriate choices for the blinding function and unblinding function are as follows:

Blinding: $f : \mathbb{Z}_N \to \mathbb{Z}_N$ is defined by $f(m) = mk^e \bmod N$.

Unblinding: $g : \mathbb{Z}_N \to \mathbb{Z}_N$ is defined by $g(m) = k^{-1}m \bmod N$.

Suppose requester A wishes to obtain B's signature on a message $m \in \mathbb{Z}_N$. This can be accomplished using the following blind signature protocol based on RSA:

Blinding: A computes $f(m) = mk^e \bmod N$ and sends this to B.

Signing: B computes $s^* = S_B(f(m)) = (f(m))^d \bmod N$ and sends it to A.

Unblinding: A computes $s = g(s^*) = k^{-1}s^* \bmod N$.

Then

$$\begin{aligned} s &\equiv k^{-1}(f(m))^d \pmod N \\ &\equiv k^{-1}(mk^e)^d \pmod N \\ &\equiv k^{-1}m^d k^{ed} \pmod N \\ &\equiv k^{-1}m^d k \pmod N \\ &\equiv m^d \pmod N \\ &= S_B(m) \end{aligned}$$

as required.

Signature Verification

Suppose a third party C wishes to verify the signature s on the message m. Then C does the following:

1. Computes $m' = s^e \bmod N$.

2. Accepts s iff m' has the predetermined redundancy.

Security

A blind signature scheme has the following two notions of security: blindness and unforgeability. *Blindness* refers to a signer being unable to link her signature s^* on the message $f(m)$ to the resulting message-signature pair (m, s). In the example above, the signer needs to know k in order to view $m = (s^*)^e k^{-1} \bmod N$ and the resulting signature $s = k^{-1}s^* \bmod N$. Since k was chosen secretly by the requester A, the signer can only guess k with probability $1/\phi(N)$, since $\gcd(k, N) = 1$. *Unforgeability* (often called one-more unforgeability) means that after receiving l valid blind message-signature pairs from the signer, it is infeasible for the attacker to compute a new valid blind message-signature pair. In our example using RSA above, assuming the supposed attacker was the sender herself, she would need to know the signer's private key d in order to compute $s^* = (f(m))^d \bmod N$. This is equivalent to computing d-th roots modulo N, which is considered to be infeasible. For a more formal detailed analysis of the security of other blind signature schemes see [17] and [2].

Efficiency

The scheme presented above is very efficient. To generate a blind signature, the signer must compute one modular exponentiation. The requester must compute two modular multiplications on-line. The rest of the requester's modular operations can be computed off-line before the message m is known.

4.2 Partially Blind

An obvious drawback of blind signature schemes arises from the fact that the signer cannot see the message she signs. Potentially, a requester could ask for the bank's signature on an electronic coin whose actual monetary value is greater than what the requester claims. A simple solution to this problem is to introduce distinct signing keys for each coin denomination. Suppose the requester claims a coin is \$1, when it is in fact \$100. The signer signs the blinded \$100 coin with the private key corresponding to \$1. Then the requester can only verify this signature with the \$1 public key, giving the coin a monetary value of \$1 rather than \$100. However, this proposed solution requires users to have several public keys at their disposal. This can be difficult, since in electronic payment schemes, customers often interact with the bank using a smart-card, which has a limited memory capacity.

Another disadvantage of blind signature schemes is that since the coins themselves are untraceable, there is nothing to prevent users from making several copies of a signed electronic coin and spending it at different locations. One remedy is for the bank to keep a record of all coins that have been previously spent. However, the size of this list would increase without end, which is highly inefficient.

A more effective solution to the two aforementioned problems is a modification of blind signature schemes called *partially blind signature schemes*, proposed by Abe and Fujisaki in 1996 [1]. In this scheme, instead of having no control over the contents of the signature, the signer is able to include some information in the signature, like a monetary value or an expiration date. With a coin value embedded in the signature, only one signing key is needed for all coin values. A signature scheme that contains an expiration date in the signature allows the banks to remove previously spent coins from their database once the expiry date has passed.

We will give an example of such a scheme based on RSA, where the signer and requester (or client) agree on some information that will be contained in the signature [23].

Key Generation

The signer selects an RSA public key (N, e) and corresponding private key d. The signer also selects a secure one-way hash function H.

Signature Generation

To obtain a signature containing some common information a on a message m, the requester does the following:

1. Randomly chooses $r, u \in \mathbb{Z}_N^*$.
2. Computes $\alpha = r^e H(m)(u^2 + 1) \bmod N$.
3. Sends (a, α) to the signer.

Upon receiving (a, α), the signer does the following:

1. Verifies the common information a.
2. Chooses a random positive integer $x < N$.
3. Sends x to the requester.

Upon receiving x, the requester does the following:

1. Selects a random integer $r' \in \{1, ..., N\}$.
2. Computes $b = rr' \bmod N$ and $\beta = b^e(u - x) \bmod N$.
3. Sends β to the signer.

Upon receiving β, the signer does the following:

1. Computes $\beta^{-1} \bmod N$.

2. Computes $t = H(a)^d(\alpha(x^2+1)\beta^{-2})^{2d} \bmod N$.

3. Sends (β^{-1}, t) to the requester.

Upon receiving (β^{-1}, t), the requester does the following:

1. Computes $c = (ux+1)\beta^{-1}b^e = (ux+1)(u-x)^{-1} \bmod N$.

2. Computes $s = tr^2r'^4 \bmod N$.

Then (a, c, s) is the partially blind signature on the message m.

Signature Verification

One can verify that (a, c, s) is the correct signature on the message m by checking whether

$$s^e \equiv H(a)H(m)^2(c^2+1)^2 \pmod{N}. \tag{4.1}$$

To see that signature verification works, note that:

$$\begin{aligned}
s^e &\equiv (tr^2r'^4)^e \\
&\equiv H(a)^{de}(\alpha^e(x^2+1)^e\beta^{-2e})^{2d}r^{2e}r'^{4e} \\
&\equiv H(a)(r^eH(m)(u^2+1))^{2ed}(x^2+1)^{2ed}\beta^{-4ed}r^{2e}r'^{4e} \\
&\equiv H(a)r^{2e}H(m)^2(u^2+1)^2(x^2+1)^2r^{-4e}r'^{-4e}(u-x)^{-4}r^{2e}r'^{4e} \\
&\equiv H(a)H(m)^2(u^2x^2+x^2+u^2+1)^2(u-x)^{-4}
\end{aligned}$$

$$\begin{aligned}
&\equiv H(a)H(m)^2[(ux+1)^2+(u-x)^2]^2(u-x)^{-4}\\
&\equiv H(a)H(m)^2[(ux+1)^4+2(ux+1)^2(u-x)^2+(u-x)^4](u-x)^{-4}\\
&\equiv H(a)H(m)^2[(ux+1)^4(u-x)^{-4}+2(ux+1)^2(u-x)^{-2}+1]\\
&\equiv H(a)H(m)^2(c^2+1)^2 \pmod{N}.
\end{aligned}$$

Security

In order for a partially blind signature scheme to be considered secure, it must pass the unforgeability and unlinkability conditions, as well as the partially blind requirement. The scheme described above contains another security property, namely *randomization*. The signer randomizes the blinded message by choosing the random value x which is returned to the requester. This prevents chosen plaintext attacks whereby an attacker chooses a message and requests a signature on it. Since the random value x is incorporated into the blinded message, the attacker has no control over the blinded message that is signed. Now, suppose an attacker attempts to remove the random value x from the blinded message. In order for the signature to pass the verification condition, the attacker would have to compute t that does not incorporate x. This would require the attacker to compute β' such that $\beta'^2 \equiv x^2+1 \pmod{N}$ so that when β' is sent to the signer in place of β and substituted into the equation for t, the (x^2+1) term cancels out. That is,

$$\begin{aligned}
t \equiv\ & H(a)^d(\alpha(x^2+1)\beta'^{-2})^{2d}\\
\equiv\ & H(a)^d(\alpha(x^2+1)(x^2+1)^{-1})^{2d}\\
\equiv\ & H(a)^d\alpha^{2d} \pmod{N}.
\end{aligned}$$

Without the factorization of N, an attacker would be unable to compute square roots modulo N, and thus it would be infeasible to compute such a β'.

Partial blindness is the inability of the requester to alter the common information a embedded in the signature. In the scheme described above, the requester would have to compute α' such that $\alpha'^2 = H(a)^{-1} \bmod N$, and send this α' to the signer in place of α. Then, when the signer computes t, the common information a is removed from the equation as follows:

$$\begin{aligned} t &\equiv H(a)^d(\alpha'(x^2+1)\beta^{-2})^{2d} \\ &\equiv H(a)^d\alpha'^{2d}(x^2+1)^{2d}\beta^{-4d} \\ &\equiv H(a)^d(H(a)^{-1})^d(x^2+1)^{2d}\beta^{-4d} \\ &\equiv (x^2+1)^{2d}\beta^{-4d} \pmod{N}. \end{aligned}$$

Similarly, the requester could try to compute β' such that $\beta'^4 \equiv H(a) \pmod{N}$. In either case, the common information a is removed from the signature generation stage, and so the requester can publish the tuple (a', c, s) as a valid signature, where a' is some information chosen by the requester without the signer's cooperation. However, in order to compute such values α' or β', one needs to compute square roots and quartic roots modulo N. This task requires the requester to know the factorization of N, which is considered infeasible.

To show that the scheme is unforgeable, we will assume that an attacker is given a valid signature (a, c, s) on a message m, and is trying to derive a valid signature (a', c', s') for another message m'. Firstly, an attacker could attempt to find an $m' \neq m$ such that $H(m') = H(m)$. Then the same signature (a, c, s) could be used on the message m'. However, finding such an m' is difficult since H is a secure one-way hash function. Now suppose $H(m) \not\equiv H(m') \pmod{N}$. Since s is a valid signature on m, s satisfies the verification equation 3.1. We can rewrite this equation by raising both sides to the power d:

$$s = H(a)^d H(m)^{2d} (c^2 + 1)^{2d} \bmod N.$$

To forge a valid signature, the attacker needs to find an s' that satisfies

$$\begin{aligned} s' &\equiv H(a)^d H(m')^{2d} (c^2 + 1)^{2d} \\ &\equiv H(a)^d H(m')^{2d} (c^2 + 1)^{2d} H(m)^{2d} H(m)^{-2d} \\ &\equiv s H(m)^{-2d} H(m')^{2d} \pmod{N}. \end{aligned}$$

However, the attacker would need to know the signer's secret key d, which it does not. In order to compute a valid c' that satisfies (3.1), the attacker would have to solve

$$c'^2 \equiv (s^e H(a)^{-1} H(m')^{-2})^{1/2} - 1 \pmod{N}.$$

But in order to compute a square root modulo N, the attacker would need to know the factorization of N.

Finally, we will show that the scheme presented above passes the unlinkability requirement. That is, the signer is unable to link the final signature (a, c, s, m) to the particular instance it was signed. That is, if the signer signs many partially blinded messages, she is unable to distinguish which final signature corresponds to each of her previous signing processes. The following theorem gives us the desired result.

Theorem 4.2.1 *Suppose a signer wishes to find a particular previous signing process instance corresponding to a given valid signature (a, c, s, m). Suppose the signer has previously signed k partially blinded messages with the same common information a (eg. within a valid time period). For each of the previous instances, represented by $(a, \alpha_i, x_i, \beta_i, t_i)_{1 \le i \le k}$, the signer can find the unique b_i, r_i, and u_i that were chosen by the requester during the signature generation stage of the scheme from the following equations:*

$$\begin{aligned} \alpha_i &\equiv (r_i)^e H(m)(u_i^2 + 1) \pmod{N} && (4.2) \\ \beta_i &\equiv b_i^e (u_i - x_i) \pmod{N} && (4.3) \\ c &\equiv (u_i x_i + 1)(u_i - x_i)^{-1} \pmod{N}. && (4.4) \end{aligned}$$

Proof From equation (4.4), we can solve for u_i, obtaining

$$u_i \equiv (cx_i + 1)(c - x_i)^{-1} \pmod{N}. \qquad (4.5)$$

If we substitute equation (4.5) into (4.2), we have

$$\alpha_i \equiv r_i^e H(m)((cx_i + 1)^2 (c - x_i)^{-2} + 1) \pmod{N}.$$

Then solving for r_i, we obtain

$$r_i \equiv \alpha_i^d H(m)^{-d}((cs_i+1)^2(c-x_i)^{-2}+1)^{-d} \pmod N. \tag{4.6}$$

Substituting (4.5) into (4.3), we have

$$\beta_i \equiv b_i^e((cx_i+1)(c-x_i)^{-1}-x_i) \pmod N$$

and finally solving for b_i,

$$b_i \equiv \beta_i^d((cx_i+1)(c-x_i)^{-1}-x_i)^{-d} \pmod N. \tag{4.7}$$

From equations (4.5), (4.6), and (4.7), we can see that for each of the signer's views of the signing process, there is a unique triple (b_i, r_i, u_i) which was chosen by the requester in the signing process. Thus, every instance of the signing process is a possibility for the given signature (a, c, s).

This theorem tells us that the signer cannot eliminate any instance of the signing process as a possibility of being the correct one for the given signature.

Efficiency

In most applications of blind signature schemes, the client or requester is a smart card of limited computational power. Thus, it is important that the work done by the smart card in blind signature schemes is minimal. In the scheme described above, the client must compute 21 modular multiplications and 2 hashing computations. Compared with a previous partially blind scheme due to Abe and Fujisaki [1], this scheme reduces the amount of computation done by the client by 98%.

4.3 Fair Blind

The unlinkability requirement of blind signature schemes provides an opportunity for criminals to misuse payment systems. In this "perfect crime" scenario, a customer could be blackmailed by the criminal and forced to withdraw digital coins from her account. If a blind signature was used to sign the coins, there is no way to trace the coins deposited by the blackmailer to the coins withdrawn from the victim's account. A fair blind signature scheme has the property that a trusted third party such as a judge has some additional information which can be used to revoke anonymity of suspected criminals. That is the judge is able to link the signer's view of the signing process to the resulting message-signature pair.

A physical analog of blind signature schemes is the following: The requester writes a message on a piece of paper, places it with a carbon paper into an envelope, seals it and sends it to the signer. The signer signs the envelope and because of the carbon, the signature is copied onto the paper with the message. Since the envelope is sealed, no one can tamper with the signed message inside. The signer returns the envelope to the requester who can open the envelope to obtain the message and signature.

Similarly, we can give a physical analog of fair blind signatures. The signer puts a blank piece of paper and a carbon paper in an envelope and seals it. The requester writes the message to be signed on the envelope using "magic ink". This magic ink is visible only after it is "developed". The signer signs the envelope. Because of the carbon paper, the message and the signature are inside the sealed envelope. The signer keeps the envelope and the requester gets the internal paper. In case of discrepancy, the signer can develop the magic ink on the envelope obtaining the message and signature.

There are two classes of fair blind signatures called Type I and Type II. The classes differ based on the information supplied to the judge by the signer. In a Type I signature, the judge is given the signer's view of the signing process, and provides the signer with information that enables her to find the corresponding message-signature pair. In a Type II signature, the judge is given the message-signature pair, and provides the signer with the corresponding view of the signing process.

The following example of a fair blind signature scheme described by Stadler, Piveteau and Camenisch [14] is of Type I and Type II. In this scheme, E_J denotes the enciphering function of a judge's public-key cryptosystem, H is a one-way hash function, and k is a security parameter.

Key Generation

The signer chooses an RSA public key (N, e) and corresponding private key d. The requester and the signer agree on a session identifier ID, where each instance of the signing process corresponds to a different ID.

Signature Generation

To generate a fair blind signature, the following procedure is carried out by the requester and the signer:

1. The requester does the following for $i = 1, 2, ..., 2k$:

   (a) Randomly chooses $r_i \in \mathbb{Z}_N^*$ and strings α_i, β_i.

   (b) Computes $u_i = E_J(m \| \alpha_i)$ and $v_i = E_J(ID \| \beta_i)$.

   (c) Computes $m_i = r_i^e H(u_i \| v_i) \bmod N$ and sends it to the signer.

2. Upon receiving m_i, the signer randomly chooses a subset S, where $S \subset \{1, 2, ..., 2k\}$, $|S| = k$ and sends S to the requester.

3. Upon receiving S, the requester sends r_i, u_i, and β_i for $i \in S$ to the signer.

4. The signer then does the following:

   (a) Checks whether $m_i \equiv r_i^e H(u_i \| E_J(ID \| \beta_i)) \pmod N$ for each $i \in S$.

   (b) Computes $b = (\prod_{i \notin S} m_i)^{1/e} \bmod N$ and sends b to the requester.

5. Upon receiving b, the requester computes $s = b/(\prod_{i \notin S} r_i) \bmod N$.

Then s along with the set of pairs $T = \{(\alpha_i, v_i) | i \notin S\}$ is the resulting fair blind signature on the message m.

Signature Verification

The signature can be verified by checking that

$$s^e \equiv \prod_{(\alpha,v)\in T} H(E_J(m\|\alpha)\|v) \pmod{N}.$$

To see that signature verification works, note that:

$$\begin{aligned}
s^e &\equiv \left(\frac{b}{\prod_{i\notin S} r_i}\right)^e \\
&\equiv \frac{((\prod_{i\notin S} m_i)^{1/e})^e}{\prod_{i\notin S} r_i^e} \\
&\equiv \frac{\prod_{i\notin S} r_i^e H(u_i\|v_i)}{\prod_{i\notin S} r_i^e} \\
&\equiv \prod_{i\notin S} H(u_i\|v_i) \\
&\equiv \prod_{i\notin S} H(E_J(m\|\alpha_i)\|v_i) \\
&\equiv \prod_{(\alpha,v)\in T} H(E_J(m\|\alpha)\|v) \pmod{N}.
\end{aligned}$$

We will show that this scheme is a fair blind signature scheme of both Type I and Type II. Suppose the judge is given a value u_i for $i \in S$. Then, since $u_i = E_J(m\|\alpha_i)$ and E_J is the judge's encryption function, the judge can decrypt one of these values with her private key and obtain $(m\|\alpha_i)$ for some $i \in S$. Since the length of m is known, the judge can pick off the message m. Thus, given the signer's view of the protocol, the judge can obtain the corresponding message-signature pair. On the other hand, suppose the judge is given the signature (s, T). By decrypting one of the v_i's in T, the judge can obtain the session identifier ID. Thus, the judge can find the instance of the signing process corresponding to a given message-signature pair.

This scheme can be modified to be solely of Type I or Type II as follows. Suppose the requester computes $v_i = H(ID \| \beta_i)$ instead of $v_i = E_J(ID \| \beta_i)$. Then the judge is no longer able to recover the session identifier ID, since H is a one-way hash function. The scheme is now of Type I only. Alternatively, suppose the requester computes $u_i = H(m \| \alpha_i)$ rather than $u_i = E_J(m \| \alpha_i)$. Then the judge is unable to disclose the message m, since H is a one-way hash function. The scheme is now of Type II only.

Security

To show that this scheme is secure, we must first discuss the security requirements of an ordinary blind signature scheme. Since the message m and the session identifier ID are encrypted using the judge's public key, it cannot be decrypted without the judge's private key. Therefore, the signer cannot associate the message m with her view of the signing protocol. Thus, this scheme passes the blindness requirement.

Suppose an attacker is given a valid message signature pair and wishes to compute a valid signature on a different message m. Recall that a signature is composed of s and the set of pairs T. Since each $v_i \in T$ depends on the unique session identifier ID, an attacker would be unable to obtain any information from previously signed messages to generate a new message-signature pair. Further, in order to pass the verification condition, an attacker would need to compute e-th roots modulo N which is considered infeasible. Thus, the scheme described above is unforgeable.

Suppose a dishonest requester tries to attack the fairness property of this scheme.

Since the message is recovered from the u_i's and the session identifier from the v_i's, the requester could attempt to choose u_i's or v_i's that are not formatted according to the protocol. Suppose the scheme is of Type I. Then the cheating requester might try to alter the u_i's. The requester must correctly format the half of the u_i's which correspond to the indices $i \notin S$ to satisfy the verification condition. The requester could guess with probability 1/2 an index $i \in S$ and form this u_i incorrectly. But even so, these u_i's for which $i \in S$ are thrown out and unused. Thus, if the requester alters any of the u_i's, it will either be detected during verification or it will be harmless.

Now, suppose the scheme is of Type II. Then the cheating requester could attempt to alter one of the v_i's. The probability that $i \notin S$ for this altered v_i is 1/2. Suppose $i \notin S$ for this v_i. Thus, when the judge is given the v_i's for all $i \notin S$, one of the v_i's will not contain the encryption of the session identifier ID as directed by the protocol. To solve this, the judge should take the majority of the ID's as the correct one.

Efficiency

Unfortunately the scheme we described above is not at all efficient. There is a great deal of information exchanged between the signer and the sender during signature generation. Furthermore, the signature that is produced is quite long. For a more efficient fair blind signature scheme, see [15].

4.4 Group

A group signature scheme allows members of a group to sign messages on behalf of the entire group. The signature can be verified with a single group public key without revealing the identity of the signer. Each group has a group manager who is responsible for generating the group public key and corresponding secret key known only to the group manager. The group manager is capable of "opening" signatures, which uncovers the identity of the signer in case of dispute.

Introduced by Chaum and van Heyst [22] in 1991, group signatures are highly applicable for online voting and bidding. Suppose the group is several bidders who each place an anonymous bid using the group signature. The group manager chooses the highest bid and reveals the winner, while keeping the other bidders' identities secret.

The group signature scheme that we will present is one of the simplest group signatures from Chaum and van Heyst [22]. It relies on the intractability of the discrete logarithm problem and uses the ElGamal signature scheme for signing and verifying.

Let g be a generator of $\mathbb{Z}_p^*$, where p is a prime. Suppose there are n group members and each week, group members are numbered randomly from 1 to n. Suppose I_j is group member j's identity, $1 \leq j \leq n$.

Key Generation

The group members' private and public keys are generated as follows:

1. Each group member j, $1 \leq j \leq n$ selects a secret integer a_j, computes $z_j =$

$g^{a_j} \bmod p$, and sends z_j to the group manager.

2. The group manager stores the list of $(z_j, I_j), 1 \leq j \leq n$.

3. The group manager selects a randomly chosen number $b_j \in \{1, ..., p-1\}$, sends b_j to group member j, and computes $y_j = (g^{a_j})^{b_j} \bmod p$.

4. Group member j computes $s_j = a_j b_j \bmod (p-1)$.

Then, group member j's private key is s_j and corresponding public key is y_j.

Signature Generation

To sign the message $m \in \{0,1\}^*$, group member j does the following:

1. Selects a random secret integer $k \in \{1, ..., p-2\}$ such that $\gcd(k, p-1) = 1$.

2. Computes $M = H(m)$.

3. Computes $R = g^k \bmod p$.

4. Computes $S = k^{-1}(M - s_j R) \bmod (p-1)$.

Then the group signature is (R, S, j).

Signature Verification

To verify that (R, S) is a valid group signature on the message m, the verifier does the following:

1. Obtains y_j from the list of group public keys.

2. Checks that $1 \leq R \leq p-1$. If not, reject.

3. Computes $M = H(m)$.

4. Checks that $y_j^R R^S \equiv g^M \pmod{p}$ and accepts if and only if this equation holds.

To see that signature verification works, note that

$$y_j^R R^S \equiv (g^{s_j})^R g^{kS} \equiv (g^{s_j})^R g^{M-s_jR} \equiv g^M \pmod{p}.$$

Opening Signatures

In case of a dispute, the group manager can identify the signer from the number j contained in the public key in question, by searching the list (z_j, I_j), and obtaining the group member's identity I_j.

Security

Note that each secret key s_j can only be used once by member j to sign a single message. Otherwise, signatures can be linked to group member j. Similarly, group members must be renumbered randomly each time a message is signed. Even though her identity I_j is not revealed, important information is leaked. Thus, the group manager sends a new b_j to member j, who is then given a new numeric value $r \in \{1, ..., n\}$, once a message has been signed. If the group decides to sign only one message a week, the group manager can send out a new $b_j, 1 \leq j \leq n$ to each group member every week. Even if these b_j's are revealed during the transfer, no information about s_j is revealed since a_j is still a secret. Lastly, the group manager is unable to forge a signature on behalf of a group member, since the manager cannot uncover a_j from $g^{a_j} \bmod p$ by the intractability of the discrete logarithm problem.

There is no proof of security provided for this scheme. For a provably secure group signature scheme, see [4].

Efficiency

The length of the public key of this group signature scheme is linear in the number of group members. Thus, this scheme is not very efficient. For more efficient group signature schemes, see [16], [13] and [11].

4.5 Ring

Ring signatures are closely related to group signatures. The main difference is that there is no group manager present. As a result, there is no procedure for obtaining a membership certificate or generating a secret membership key and there is no way to revoke anonymity of the signer. In a ring signature scheme, the signer does not require the consent of the other ring members to create a ring signature. This provides an excellent opportunity for leaking secrets. For example, suppose a member of an internal organization wishes to expose information to an outside source about the organization without being identified. If a ring signature scheme is used, the outside source is convinced that the information came from a member of the organization but has no way of determining which member. Leaking secrets can be viewed as an important concept in maintaining a free society. Suppose a member of the government feels that the public deserves to know something that is being kept from them. This member may fear retribution by those in power, so signing the important information with a ring signature will keep her identity a secret.

We will assume that the members of the ring already have a public key corresponding to some ordinary signature scheme. It is possible for different users to use different public key signature schemes, however, for simplicity we will assume that all ring members are using the same signature scheme.

To produce a ring signature on a message, the signer selects an arbitrary set of signers including herself and computes a signature using her own secret key and the other signers' public keys. Note that the other users may be completely unaware that their public key is being used to sign a message.

Rivest, Shamir and Tauman [50] describe a ring signature scheme in which all ring members use a trapdoor one-way permutation as their individual signature scheme. We will present the scheme using the RSA signature scheme as the permutation.

Suppose a signer wishes to sign the message m with a ring signature for a ring of r individuals $A_1, ..., A_r$, where the signer is $A_s, 1 \leq s \leq r$. Each ring member A_i has an RSA public key (N_i, e_i) which can be used to verify signatures using their trapdoor one-way permutation $f_i(x) = x^{e_i} \bmod N_i$ of $\mathbb{Z}_{N_i}$. Only A_i can compute the inverse permutation or signature efficiently.

Now, since each user can have different sized domains N_i, it will be difficult to combine these individual signatures to obtain a ring signature. Thus, the domains of the permutations are extended to a single common domain $\{0,1\}^b$, such that $2^b > N_i$ for all $1 \leq i \leq r$. Then any b-bit message m is rewritten as $m = q_iN_i + r_i$ for nonnegative integers q_i, r_i where $0 \leq r_i < N_i$. Finally a new permutation g_i is computed as follows:

$$g_i(m) = \begin{cases} q_iN_i + f_i(r_i) & \text{if } (q_i+1)N_i \leq 2^b, \\ m & \text{otherwise.} \end{cases}$$

We can see that g_i operates by applying f_i to the least significant digit of the N_i-ary representation of m, except in the case that this produces a result larger than $2^b - 1$, in which case m remains unchanged. However, if b is chosen large enough, the latter case occurs with negligible probability. Note that each g_i is a permutation over $\{0,1\}^b$ and further it is a one-way trapdoor permutation because with high probability, f_i, a one-way trapdoor permutation, is computed for an iteration of g_i.

Before we present the signing and verifying protocols of this scheme, we will define a family of combining functions $C_{k,v}(y_1, y_2, ..., y_r) = z \in \{0,1\}^b$ where $y_1, y_2, ..., y_r \in \{0,1\}^b$, k is a key corresponding to a symmetric key encryption function E_k which is used as a subroutine, and v is an initialization value.

We will assume the following properties regarding this family of functions:

1. For each s, $1 \leq s \leq r$, and for any fixed y_i, $i \neq s$, $C_{k,v}$ is a one-to-one mapping from y_s to the output z.

2. For each s, $1 \leq s \leq r$, given z and all the y_i values except y_s, one can efficiently compute y_s such that $C_{k,v}(y_1, y_2, ..., y_s, ..., y_r) = z$.

3. Given $g_1, \ldots, g_r, k, v$ and z it is infeasible to solve the equation

$$C_{k,v}(g_1(x_1), g_2(x_2), ..., g_r(x_r)) = z$$

for $x_1, x_2, ..., x_r$ without the ability to invert any of the trap-door functions $g_1, g_2, ..., g_r$.

To satisfy the above conditions, we will define $C_{k,v}$ as follows:

$$C_{k,v}(y_1, y_2, ..., y_r) = E_k(y_r \oplus E_k(y_{r-1} \oplus E_k(y_{r-2} \oplus E_k(\cdots \oplus E_k(y_1 \oplus v) \cdots)))),$$

where $y_i = g_i(x_i)$.

We can see that this function satisfies the three assumptions above. First of all $C_{k,v}$ is a permutation since XOR, g_i and E_k are permutations. Secondly, knowledge of k allows one to apply the function in the forward direction starting with the initial value v and backwards from the final output z to compute any single missing value y_i. It is slightly more complicated to see that the third condition is met, but a thorough explanation is given by Rivest et al. in [50]. In order to use this function for signature verification, we will set $z = v$ and call the resulting function the *ring equation.*

In the following description of the procedures involved in generating a ring signature, $H : \{0,1\}^* \rightarrow \{0,1\}^l$ is a publicly defined collision-resistant hash function.

Key Generation

The signer A_s obtains the public keys $P_1, P_2, ..., P_s, ..., P_r$ of all members of the ring as well as her own secret key S_s.

Signature Generation

To compute a ring signature on the message $m \in \{0,1\}^b$, the signer does the following:

1. Computes $k = H(m)$.
2. Selects an initialization value v randomly from $\{0,1\}^b$.
3. Selects random $x_i \in \{0,1\}^b$ for each of the other ring members $1 \leq i \leq r, i \neq s$ and computes $y_i = g_i(x_i)$.
4. Solves the following ring equation for y_s:

$$C_{k,v}(y_1, y_2, \cdots, y_s, \cdots, y_r) = v.$$

5. Computes $x_s = g_s^{-1}(y_s)$, which is possible since user A_s knows the inverse of the trapdoor permutation g_s.

Then the ring signature on the message m is the $2r{+}1$-tuple $(P_1, P_2, ..., P_r; v; x_1, x_2, ..., x_r)$.

Signature Verification

To verify that a proposed signature $(P_1, P_2, ..., P_r; v; x_1, x_2, ..., x_r)$ came from a ring member, the verifier does the following:

1. Computes $y_i = g_i(x_i)$ for $1 \leq i \leq r$.

2. Computes $k = H(m)$.

3. Checks that the y_i's obtained in step 1 satisfy the ring equation $C_{k,v}(y_1, y_2, ..., y_r) = v$.

Security

The scheme described above is unforgeable against chosen message attacks. Rivest et al. [50] show that forging a ring signature on the message m using numerous ring signatures on chosen messages $m_j \neq m$ is equivalent to inverting one of the trapdoor one-way functions g_i on random inputs y. The scheme also provides irrevocable anonymity. For each input k and v, there are $r-1$ values x_i chosen randomly from $\{0,1\}^b$. Therefore, the ring equation has $(2^b)^{(r-1)}$ solutions for any given k and v. Each of these solutions is equally likely and independent of the signer's identity, since the x_i's are chosen randomly by the signer.

Efficiency

The ring signature scheme we described above is very efficient. If we assume the individual public key of each ring member is $e_i = 3$, as is often the case in practice for RSA signature schemes, the signer must compute a modular cubic operation (or two modular multiplications), on behalf of each of the $r-1$ non-signers. Then, the signer must compute one modular exponentiation to invert her own trapdoor permutation. Thus, signing requires one modular exponentiation and $2(r-1)$ modular multiplications. Verifying this ring signature requires a modular cubic operation for each of the r ring members. Equivalently stated, signature verification requires $2r$ modular multiplications.

Chapter 5

Schemes with Enhanced Signing and Verifying Capabilities

There are some applications of digital signatures where the standard characteristics of signature generation and verification are inappropriate. For example, ordinary protocols for signature generation assume the participation of the original signer, however this is not always possible. A proxy signature scheme allows the signer to delegate her ability to sign to another user of her choice. Another protocol standard is that the verifier alone carries out the verification procedure. Verification of undeniable signatures requires the signer's cooperation in the verification process, thereby giving the signer control over who can authenticate her signatures. A convertible undeniable signature is an undeniable signature that can be transformed into an ordinary signature that can be verified universally. A designated confirmer signature scheme is an undeniable signature scheme where the signer can select another user to aid in the verification process in her place.

5.1 Proxy

A proxy signature scheme allows a designated party to sign on behalf of the original signer. The receiver of the signature must be able to verify the proxy signature in a similar way to the original signature. Ordinary digital signature schemes are non-transferable since their security relies on the secret signing key which only the signer knows. If this secret key is simply given to another user, the underlying security assumption of digital signature schemes has been broken. A proxy signature scheme maintains the original signer's public key and provides the proxy signer with a functioning secret key that is different from the original signer's. This produces distinct signatures for the original signer and the proxy signer, which is important for non-repudiation. That is, neither the proxy signer nor the original signer can deny their signature, claiming the other to be the actual signer.

Proxy signatures are useful in situations where the original signer is unavailable to sign a document. Suppose, for instance, a company executive is on holidays. She could designate her secretary to sign in her place while she is away by giving the secretary a proxy signing key.

Mambo, Usuda and Okamoto [39] describe a method for proxy signing key delegation based on discrete logarithms which can be used in conjunction with any ordinary signature scheme to create a proxy signature scheme. Here, we will use an ElGamal signature scheme as the original signer's scheme. We will briefly present this scheme used by the original signer A before we describe the process whereby A delegates signing capability to the proxy signer B.

Suppose p is a large prime, q is a prime factor of $p-1$, $g \in \mathbb{Z}_p^*$ is an element of order q, and $H : \{0,1\}^* \to \{0,1\}^p$ is a cryptographically strong hash function. Then $x \in \mathbb{Z}_q$ is A's private key and $y = g^x \bmod p$ is A's public key. To sign a message $m \in \{0,1\}^*$, the original signer selects a random $k \in \mathbb{Z}_q$ and computes

$$\begin{aligned} M &= H(m), \\ R &= Mg^k \bmod p, \\ \text{and} \quad S &= Rx + k \bmod q. \end{aligned}$$

Then (R, S) is the signature of the message M. Signature verification is done by checking if $M \equiv g^{-S} y^R R \pmod{p}$. This works since

$$g^{-S} y^R R \equiv g^{-Rx-k} g^{xR} M g^k \equiv M \pmod{p}.$$

Key Delegation

1. A selects a random $\bar{k} \in \mathbb{Z}_q$, computes $\bar{r} = g^{\bar{k}} \bmod p$, and sends $\bar{r}$ to B.

2. B selects a random $a \in \mathbb{Z}_q$, computes $r = g^a \bar{r} \bmod p$, and sends r to A.

3. A computes $\bar{s} = rx + \bar{k} \bmod q$ and sends $\bar{s}$ to B.

4. B computes $s = \bar{s} + a \bmod q$ and accepts s as a valid proxy signing key if $g^s \equiv y^r r \pmod{p}$.

Verification of the proxy signing key works since

$$y^r r \equiv g^{xr} r \equiv g^{\bar{s}-\bar{k}} g^a \bar{r} \equiv g^{\bar{s}} g^{-\bar{k}} g^a g^{\bar{k}} \equiv g^{\bar{s}} g^a \equiv g^s \pmod{p}.$$

The proxy signer's secret signing key is $s \in \mathbb{Z}_q$ and the corresponding public key is $y' = y^r r \bmod p$.

Proxy Signature Generation

To sign a message $m \in \{0,1\}^*$, the proxy signer does the following:

1. Computes $M = H(m)$, where $H : \{0,1\}^* \to \mathbb{Z}_p$.
2. Selects a random $k \in \mathbb{Z}_q$.
3. Computes $R = Mg^k \bmod p$ and $S = Rs + k \bmod q$.

Then (R, S) is the signature of the message m.

Signature Verification

To verify this signature, the verifier computes $M = H(m)$ and checks if $M \equiv g^{-S}y'^R R \pmod{p}$. If this equation holds, the signature is accepted, otherwise it is rejected.

Verification of the proxy signature works since

$$g^{-S}y'^R R \equiv g^{-(Rs+k)}(y^r r)^R R \equiv g^{-Rs}g^{-k}(g^s)^R M g^k \equiv M \pmod{p}.$$

Security

The difficulty of finding the proxy signature key s knowing $\bar{r}, \bar{k}, r$ and x is equivalent to the discrete logarithm problem. Thus, it is computationally infeasible for the original signer to forge a proxy signature. On the other hand, computing $x, \bar{k}$

or forging a valid message-signature pair that satisfies $M \equiv g^{-S}y'^{r}r \pmod{p}$ is infeasible under the assumption that the ElGamal signature scheme is secure. Then it is infeasible for the proxy signer to forge the original signer's signature on another message. Clearly, the scheme is unforgeable since it is even more difficult for a third party who is not the proxy signer to forge the original signer's signature, and for a third party who is not the original signer to forge the proxy signer's signature.

Lee, Hwang and Wang [38] claim that a dishonest proxy signer could cheat in the key delegation protocol and obtain the original signer's signature on any message. The attack is mounted by sending the original signer $r = M\bar{r} \bmod p$ instead of $r = g^a\bar{r} \bmod p$ for a message $M = H(m)$ of the proxy's choice. Then when the original signer returns $\bar{s} = rx+\bar{k} \bmod q$, the proxy has the original signer's signature $(r, \bar{s})$ on the message m, since

$$g^{-\bar{s}}y^r r \equiv g^{-\bar{s}}g^{xr}M\bar{r} \equiv g^{-\bar{s}}g^{\bar{s}-\bar{k}}M\bar{r} \equiv g^{-\bar{k}}Mg^{\bar{k}} \equiv M \pmod{p}.$$

This attack can be prevented by adding a cheating detection protocol to the proxy signature scheme. The protocol designed by Ghodosi and Pieprzyk [30] works as follows: Suppose the original signer A suspects her proxy signer B of cheating. Then, A finds the instance of the key delegation protocol corresponding to the message in question. If A was indeed cheated in the second step of the protocol, she can detect it by checking if $r \equiv m\bar{r} \pmod{p}$. To prove that B is a cheating proxy signer, A asks B to sign a message using her proxy signature. If B is able to create a valid proxy signature, A is convinced that B is not cheating. Otherwise, B is a cheater. To see that this works, note that if a cheating proxy B tries to sign the message as requested by A, then she must find an a so that $g^a = M \bmod p$, which is infeasible assuming the intractability of the discrete logarithm problem. As a consequence, B cannot compute $s = \bar{s} + a \bmod q$ and therefore cannot compute the proxy signature.

5.2 Undeniable

Undeniable signatures were introduced by Chaum and van Antwerpen in 1989 [21] as a way to limit the ability of third parties to verify the validity of signatures binding parties to a confidential agreement. For example, suppose user A sends a signed message to user B that she owes her money. A may want to keep private that she borrowed money from B, so a scheme is required that stops B from showing the promissory note with A's signature on it to someone else.

An undeniable signature, sometimes called an invisible signature, provides additional privacy to the signer because a signature cannot be verified without the cooperation of the signer. Thus, only those parties with whom the signer wishes to communicate are able to verify signatures. However, this gives a lot of power to the signer, who could easily renounce one of her own valid signatures, claiming it to be a forgery and refusing to aid in verifying the signature, or alter her part of the verification process so that the signature will not be validated. To avoid this situation, an undeniable signature scheme incorporates a disavowal protocol whereby if the signer claims a signature is a forgery, she has the means to prove this is so. If she refuses to prove a signature is a forgery, this is evidence that the signature must be valid. In this sense, the signature is undeniable.

The following example of an undeniable signature scheme from Chaum and van Antwerpen [21] is based on the discrete logarithm problem.

Key Generation

Each user A does the following:

1. Selects a random prime $p = 2q + 1$, where q is also a prime.
2. Selects a random element $\beta \in \mathbb{Z}_p^*$ and computes $\alpha = \beta^{(p-1)/q} \bmod p$. If $\alpha = 1$, choose a new β.
3. Selects a random integer $a \in \{1, 2, ..., q-1\}$ and computes $y = \alpha^a \bmod p$.

Then A's public key is (p, α, y) and A's private key is a. Note that computations are done in the multiplicative subgroup G of order q in $\mathbb{Z}_p^*$, generated by α. It is desirable to take $p = 2q + 1$ because it ensures that the subgroup G of order q is as large as possible, which is important since all messages and signatures will be elements of G.

Signature Generation

User A computes $S = m^a \bmod p$. Then A's signature on $m \in G$ is S.

Signature Verification

To verify this signature, the following protocol must be carried out by signer A and verifier B:

1. B randomly selects secret integers $x_1, x_2 \in \{1, 2, ..., q-1\}$, computes $z = S^{x_1} y^{x_2} \bmod p$ and sends z to A.

2. Upon receiving z, A computes $w = (z)^{a^{-1} \bmod q} \bmod p$ and sends w to B.

3. Finally, B computes $w' = m^{x_1} \alpha^{x_2} \bmod p$ and accepts the signature S iff $w = w'$.

In the following proof that signature verification holds, note that all exponents are computed modulo q.

$$\begin{aligned} w &\equiv z^{a^{-1}} \pmod{p} \\ &\equiv (S^{x_1} y^{x_2})^{a^{-1}} \pmod{p} \\ &\equiv ((m^a)^{x_1} (\alpha^a)^{x_2})^{a^{-1}} \pmod{p} \\ &\equiv m^{x_1} \alpha^{x_2} \pmod{p} \\ &= w'. \end{aligned}$$

Disavowal Protocol

To ensure that the signer has performed the verification protocol correctly, the following protocol is carried out between the signer and the verifier:

1. Verifier B randomly selects secret integers $x_1, x_2 \in \{1, 2, ..., q-1\}$, computes $z = S^{x_1} y^{x_2} \bmod p$ and sends z to A.

2. Upon receiving z, signer A computes $w = z^{a^{-1} \bmod q} \bmod p$ and sends w to B.

3. Upon receiving w, B does the following:

   (a) If $w \equiv m^{x_1}\alpha^{x_2} \pmod{p}$, accepts the signature S and halts; otherwise proceeds.

   (b) Randomly selects secret integers $x_1', x_2' \in \{1, 2, ..., q-1\}$, computes $z' = S^{x_1'}y^{x_2'} \bmod p$, and sends z' to A.

4. Upon receiving z', A computes $w' = z'^{a^{-1} \bmod q} \bmod p$ and sends w' to B.

5. Finally B does the following:

   (a) If $w' \equiv m^{x_1'}\alpha^{x_2'} \pmod{p}$, B accepts signature S and the protocol stops; otherwise proceeds to step (b).

   (b) Computes $c = (w\alpha^{-x_2})^{x_1'} \bmod p$ and $c' = (w'\alpha^{-x_2'})^{x_1} \bmod p$. If $c = c'$, then B concludes that S is a forgery; otherwise B concludes that the signature is valid and A is trying to deny S.

To see that the disavowal protocol works, we have to show that the signer A can convince the receiver B that an invalid signature is a forgery.

Theorem 5.2.1 *[53] Suppose S is a forgery ($S \not\equiv m^a \pmod{p}$) and A and B follow the disavowal protocol correctly. Then $c = c'$ and so the receiver's conclusion that S is a forgery is correct.*

Proof We have

$$\begin{aligned} c &\equiv (w\alpha^{-x_2})^{x_1'} \equiv ((z)^{a^{-1}}\alpha^{-x_2})^{x_1'} \pmod{p} \\ &\equiv ((S^{x_1}y^{x_2})^{a^{-1}}\alpha^{-x_2})^{x_1'} \pmod{p} \\ &\equiv S^{x_1a^{-1}x_1'}y^{x_2a^{-1}x_1'}\alpha^{-x_2x_1'} \pmod{p} \end{aligned}$$

$$\begin{aligned}
&\equiv S^{x_1 a^{-1} x_1'} \alpha^{x_2 x_1'} \alpha^{-x_2 x_1'} \pmod p \\
&\equiv S^{x_1 a^{-1} x_1'} \pmod p.
\end{aligned}$$

Similarly,

$$\begin{aligned}
c' &\equiv (w' \alpha^{-x_2'})^{x_1} \equiv ((z')^{a^{-1}} \alpha^{-x_2'})^{x_1} \pmod p \\
&\equiv ((S^{x_1'} y^{x_2'})^{a^{-1}} \alpha^{-x_2'})^{x_1} \pmod p \\
&\equiv S^{x_1' a^{-1} x_1} y^{x_2' a^{-1} x_1} \alpha^{-x_2' x_1} \pmod p \\
&\equiv S^{x_1' a^{-1} x_1} \alpha^{x_2' x_1} \alpha^{-x_2' x_1} \pmod p \\
&\equiv S^{x_1' a^{-1} x_1} \pmod p.
\end{aligned}$$

Therefore, $c = c'$ and the conclusion that S is a forgery is correct. Note that if $S \equiv m^a \pmod p$, c and c' would still be equal, but the protocol would have stopped before reaching this checking stage.

Security

The security of an undeniable signature scheme relies on *unforgeability*, *invisibility*, *completeness and soundness of verification*, and *non-transferability*. *Unforgeability* means the scheme is existentially unforgeable under a chosen message attack. *Invisibility* means that there is no efficient algorithm which, given the public key (p, α, y), a message m and a possible signature S, can decide, with probability greater than guessing, whether or not S is valid. *Completeness* means that valid signatures can be proven valid and invalid signatures can be proven invalid. *Sound-*

ness means that no valid signature can be proven invalid and no invalid signature can be proven valid. *Non-transferability* means that during signature verification, the verifier does not obtain any information that could be used to convince a third party about the correctness of a signature.

The unforgeability of this scheme relies on the intractability of the discrete logarithm problem. That is, suppose an attacker is given a valid message-signature pair (m, S) and wishes to forge A's signature on a different message m'. To obtain a valid signature S', the attacker needs to compute $S' = m'^a \bmod p$. In order to recover A's private key a, the attacker would have to solve $S \equiv m^a \pmod{p}$ for a, which is infeasible.

Invisibility or the ability to verify is also secured by the intractability of the discrete logarithm problem. If one could compute a from $y \equiv \alpha^a \pmod{p}$, they could compute a^{-1}, from which they could compute w, and then could easily verify signatures without the cooperation of the signer.

We showed that this scheme is complete in the proof that signature verification works (valid signatures can be proved valid) and the proof of Theorem 5.2.1 (invalid signatures can be proved invalid).

In the discussion that follows, we will prove the soundness of verification. Theorem 5.2.2 shows that an invalid signature cannot be accepted as valid except with negligible probability and Theorem 5.2.3 shows that a valid signature cannot be proven invalid except with negligible probability.

As we mentioned earlier, a scheme that requires the signer's cooperation to verify gives a lot of power to the signer. Let us now consider the case where the signer is the attacker. Suppose the signer forges a signature S on a message m. That is, $S \not\equiv m^a \pmod{p}$. We will show that the signer cannot fool the receiver into accepting a forged signature except with a very small probability. This result is independent of any computational assumptions.

Theorem 5.2.2 *[53] If $S \not\equiv m^a \pmod{p}$, then the receiver will accept S as a valid signature for m with probability $1/q$.*

Proof Note that each possible $z \equiv S^{x_1} y^{x_2} \pmod{p}$ computed by the receiver corresponds to exactly q pairs (x_1, x_2) since $S, y \in G$, where G is a subgroup of order q. When the signer obtains z, she does not know which of these q pairs the receiver used. Now, if $S \not\equiv m^a \pmod{p}$, when the signer computes $w = (z)^{a^{-1} \bmod q} \bmod p$, we claim that $w \equiv m^{x_1} \alpha^{x_2} \pmod{p}$ for exactly one of the q possible pairs (x_1, x_2).

Since α is a generator of G, we can write any element of G as a unique power of α. We will write $z = \alpha^i, w = \alpha^j, m = \alpha^k$, and $S = \alpha^l$, where $i, j, k, l \in \{0, 1, ..., q-1\}$. Now, consider the system of congruences:

$$\begin{aligned} z &\equiv S^{x_1} y^{x_2} \pmod{p} \\ w &\equiv m^{x_1} \alpha^{x_2} \pmod{p}. \end{aligned}$$

Expressed as powers of α, the equations become

$$\begin{aligned} \alpha^i &\equiv \alpha^{l x_1} \alpha^{a x_2} \pmod{p} \\ \alpha^j &\equiv \alpha^{k x_1} \alpha^{x_2} \pmod{p}. \end{aligned}$$

Equivalently,

$$\begin{aligned} i &\equiv lx_1 + ax_2 \pmod{q} \\ j &\equiv kx_1 + x_2 \pmod{q}. \end{aligned}$$

The coefficient matrix of this system is $\begin{pmatrix} l & a \\ k & 1 \end{pmatrix}$, and since $S \not\equiv m^a \pmod{p} \Leftrightarrow \alpha^l \not\equiv \alpha^{ka} \pmod{p} \Leftrightarrow l \not\equiv ka \pmod{q}$, this matrix has non-zero determinant and thus there is a unique solution for (x_1, x_2). Therefore, every $w \in G$ gives the result $w = w'$ for exactly one of the q possible pairs (x_1, x_2). So the probability that the signer has forged a signature S where the verification condition still holds is $1/q$.

On the other hand, a dishonest signer could attempt to deny a valid signature. We will show that the probability that the signer succeeds in renouncing a valid signature by refusing to follow the disavowal protocol is $1/q$.

Theorem 5.2.3 *[53] Suppose $S \equiv m^a \pmod{p}$ and verifier B follows the disavowal protocol correctly. If $w \not\equiv m^{x_1}\alpha^{x_2} \pmod{p}$ and $w' \not\equiv m^{x_1'}\alpha^{x_2'} \pmod{p}$, then the probability that $c \neq c'$ is $1 - 1/q$.*

Proof Suppose to the contrary that $c = c'$. Then we have the following congruences:

$$\begin{aligned} S &\equiv m^a \pmod{p} \\ w &\not\equiv m^{x_1}\alpha^{x_1} \pmod{p} \\ w' &\not\equiv m^{x_1'}\alpha^{x_2'} \pmod{p} \end{aligned}$$

$$c \equiv (w\alpha^{-x_2})^{x_1'} \equiv (w'\alpha^{-x_2'})^{x_1} \equiv c' \pmod{p}.$$

This last congruence can be rewritten as $w' \equiv m_0^{x_1'}\alpha^{x_2'} \pmod{p}$, where $m_0 = w^{1/x_1}\alpha^{-x_2/x_1}$. To see this, note that if we substitute m_0 into the expression for w', we obtain:

$$\begin{aligned} w' &\equiv (w^{1/x_1}\alpha^{-x_2/x_1})^{x_1'}\alpha^{x_2'} \pmod{p} \\ \Leftrightarrow w'\alpha^{-x_2'} &\equiv w^{x_1'/x_1}\alpha^{-x_2x_1'/x_1} \pmod{p} \\ \Leftrightarrow (w'\alpha^{-x_2'})^{x_1} &\equiv w^{x_1'}\alpha^{-x_2x_1'} \pmod{p}. \end{aligned}$$

From Theorem 5.2.2, we can conclude that the probability that S is a valid signature for m_0 is $1 - 1/q$. But we also assumed that S is a valid signature for m. Thus, with probability $1 - 1/q$, we have

$$m^a \equiv m_0^a \pmod{p},$$

which implies that $m = m_0$.

However, above we have that

$$\begin{aligned} w &\not\equiv m^{x_1}\alpha^{x_2} \pmod{p} \\ \Rightarrow m &\not\equiv w^{1/x_1}\alpha^{-x_2/x_1} \pmod{p} \\ \Rightarrow m &\neq m_0. \end{aligned}$$

We have a contradiction and hence our assumption that $c = c'$ is incorrect. Thus,

$c \neq c'$ with probability $1/q$ and hence the signer can fool the receiver into thinking a valid signature is a forgery with probability $1/q$.

Finally, this scheme is non-transferable because of the randomness involved in the signature verification stage. Recall that the verifier randomly selects $x_1, x_2 \in \{1, 2, ..., q-1\}$. The verifier is unable to take on the role of the signer with a third party because they will not be able to respond to a new user's random requests x_1', x_2' unless they happen to be the exact requests x_1, x_2 of the verifier. Since a is unknown to the verifier who must reply with $w = (S^{x_1} y^{x_2})^{a^{-1} \bmod q} \bmod p$, the verifier is unable to convince a third party that a given signature is valid.

5.3 Convertible Undeniable

Convertible undeniable signatures were introduced as a modification of undeniable signatures by Boyar et al. [9] in 1991. A convertible undeniable signature scheme has the additional property that the signer can release a piece of secret information allowing all of her undeniable signatures to become ordinary digital signatures. These ordinary digital signatures are universally verifiable. That is, the signer is no longer required to aid in the verification process. The revealed secret information must be selected carefully so that it does not allow others to create valid signatures on behalf of the signer.

As an application of convertible undeniable signature schemes, suppose the signer signs all documents throughout her lifetime with a convertible undeniable signature. The secret piece of information is protected by a lawyer or family member. After the signer's death, the secret information is released to the public, so that all her signatures can be verified and no one else can create or alter signatures on behalf of the signer. Another common use of convertible signature schemes occurs in business. In case a business becomes bankrupt, the company should use a convertible signature scheme to sign their product so that customers are still able to verify that they have received an authentic copy of the product without cooperation of the company.

In the example to be presented due to Michels and Stadler [44], there is a protocol which allows the signer to convince a verifier about the validity or invalidity of a signature without leaking any information which would give the verifier the ability to sign. In this particular example, the protocol is a zero-knowledge proof of the equality or inequality of two discrete logarithms. We will begin our description of this convertible undeniable signature scheme with this interactive proof which will be employed in the signature verification stage.

In the following example, G is a cyclic group of prime order q generated by α in which computing discrete logarithms is infeasible, and H_G and H_l are collision resistant hash functions such that $H_G : \{0,1\}^* \rightarrow G$ and $H_l : \{0,1\}^* \times G \rightarrow \{0,1\}^l$, where $l \approx 160$ in practice.

Zero-Knowledge Proof of Equality of Discrete Logarithms

We will assume that the signer or prover knows the discrete logarithm x of $y = \alpha^x$ and wants to assist the verifier in deciding whether $\log_\beta z = \log_\alpha y$ for $\beta, z \in G$. Note that this protocol does not reveal the secret value x. The protocol between the signer and verifier is outlined as follows:

1. The verifier randomly chooses $u, v \in \mathbb{Z}_q$, computes $a = \alpha^u y^v$ and sends a to the signer.

2. The signer randomly chooses $k, \tilde{k}, w \in \mathbb{Z}_q$, computes $r_\alpha = \alpha^k, \tilde{r}_\alpha = \alpha^{\tilde{k}}$, $r_\beta = \beta^k$ and $\tilde{r}_\beta = \beta^{\tilde{k}}$, and sends $r_\alpha, r_\beta, \tilde{r}_\alpha, \tilde{r}_\beta$, and w to the verifier.

3. The verifier sends u and v to the signer.

4. The signer halts if $a \neq \alpha^u y^v$. Otherwise she computes $s = k - (v+w)x \bmod q, \tilde{s} = \tilde{k} - (v+w)k \bmod q$ and sends s and $\tilde{s}$ to the verifier.

5. The verifier checks that $\alpha^s y^{v+w} = r_\alpha$, $\alpha^{\tilde{s}} r_\alpha^{v+w} = \tilde{r}_\alpha$, and $\beta^{\tilde{s}} r_\beta^{v+w} = \tilde{r}_\beta$ and then concludes that $\log_\beta z = \log_\alpha y$ iff $\beta^s z^{v+w} = r_\beta$.

Key Generation

The signer randomly chooses two secret integers $x_1, x_2 \in \mathbb{Z}_q$ and computes $y_1 = \alpha^{x_1}$ and $y_2 = \alpha^{x_2}$. Then her secret key is (x_1, x_2) and her public key is (y_1, y_2).

Signature Generation

In order to sign the message m, the signer does the following:

1. Selects a random integer $k \in \mathbb{Z}_q$ and computes $R = \alpha^k$.

2. Computes $\tilde{R} = H_G(R)^{x_2}$, $c = H_l(m, \tilde{R})$, and $S = k - cx_1 \pmod q$.

Then, the resulting signature on the message m is $(\tilde{R}, S)$.

Signature Verification

The verifier accepts the signature $(\tilde{R}, S)$ if $\log_{H_G(\alpha^S y_1^{H_l(m,\tilde{R})})} \tilde{R} = \log_\alpha y_2$ and rejects if the equality does not hold. The equality or inequality of the discrete logarithms is proven by executing the interactive protocol described above. Note that the discrete logarithm in this case is the signer's private key x_2 and is not revealed during this protocol. Verification works since

$$\begin{aligned}
\log_{H_G(\alpha^S y_1^{H_l(m,\tilde{R})})} \tilde{R} &= \log_{H_G(\alpha^{k-cx_1} y_1^c)} \tilde{R} \\
&= \log_{H_G(\alpha^k \alpha^{-cx_1} \alpha^{x_1 c})} \tilde{R} \\
&= \log_{H_G(\alpha^k)} \tilde{R} \\
&= \log_{H_G(R)} \tilde{R} \\
&= x_2 \\
&= \log_\alpha y_2.
\end{aligned}$$

To convert this signature scheme into an ordinary digital signature scheme, the signer publishes her secret key x_2. Then anyone can verify her signatures by checking if

$$H_G(\alpha^S y_1^{H_l(m,\tilde{R})})^{x_2} = \tilde{R}.$$

To see that this universal verification works, note that

$$H_G(\alpha^S y_1^{H_l(m,\tilde{R})})^{x_2} = H_G(\alpha^{k-cx_1} \alpha^{x_1 c})^{x_2} = H_G(\alpha^k)^{x_2} = H_G(R)^{x_2} = \tilde{R}.$$

Security

Like the ordinary undeniable signature scheme, the security of a convertible undeniable signature scheme depends on four factors: *unforgeability*, *invisibility*, *completeness and soundness of verification*, and *non-transferability.*

In the scheme described above, suppose an attacker attempting to forge a signature is given the secret integer x_2. In order to forge a signature, the attacker needs to compute a pair $(\tilde{R}, S)$ that satisfies the verification condition $H_G(\alpha^S y_1^{H_l(m,\tilde{R})})^{x_2} = R$. In order to compute S, the attacker must compute $S = k - cx_1 \bmod q$. But the attacker has no way of knowing the signer's secret key x_1. Therefore, the scheme is unforgeable.

The invisibility requirement maintains that there is no efficient algorithm which, on input the public key y_1, a message m, and a possible signature S, can decide if S is a valid signature. In this scheme, validity is decided based on the equality or inequality of two discrete logarithms. Since even the value of these two discrete logarithms is unknown, there is no way one could decide if they are equal without performing the interactive protocol as part of the scheme. Thus, our scheme is invisible.

Our scheme is non-transferable because of the zero-knowledge property of the interactive protocol. During this protocol, there is no information leaked about the value of the discrete logarithm. Thus, the verifier is unable to act as the prover and carry out this protocol with a third party. Because of the randomly chosen challenges, namely u and v, the verifier is unable to compute S in step 4 of the protocol since it involves the secret x and the randomly selected v, which with high probability will be different from the v chosen by the verifier in the original proof.

It follows that our scheme is complete and sound from Michels and Stadler's proof that the interactive protocol for proving two discrete logarithms are equal, is complete and sound. [44]

5.4 Designated Confirmer

A designated-confirmer signature is an undeniable signature that can be proven valid by a third party appointed by the signer. Recall that an undeniable signature can only be verified with the cooperation of the signer. We can see that this poses a problem if the signer becomes unavailable or refuses to cooperate. Designated-confirmer signatures were introduced by Chaum in 1994 [20] as a way to overcome this problem. By appointing a third party, or *designated confirmer*, the signer is not needed for the verification process since the confirmer can act on the signer's behalf.

An interesting application of designated-confirmer signatures is private contract signing. Garay, Jakobsson and MacKenzie [28] introduced the notion of abuse-free contract signing between two users which combines designated-confirmer and convertible undeniable schemes. Suppose A and B are signing a contract. If the contract is not abuse-free, then, if B signed the contract first, A could prove to a third party that B is committed to the contract while A is not. Then A could use B's willingness to sign to her advantage in her attempts to secure a better contract. With abuse-free contract signing, both parties have to sign the contract, after which the confirmer proves the validity of the signatures and converts them into universally verifiable signatures.

There are four parties involved in a designated-confirmer signature scheme: the signer, recipient, confirmer and verifier. The signer and the confirmer have public key and private key pairs corresponding to distinct signature schemes. The recipient of the signature has no keys at all. The verifier is a party whom the recipient wishes to prove validity of the signature. Designated-confirmer schemes are made up of two stages. The Signing Protocol is between the signer and the recipient. The signer signs the message with her private key and the confirmer's public key in such a way that the recipient is convinced that the signature can be verified by the confirmer. The Confirmation Protocol between the confirmer and the verifier leaves the verifier convinced that the supposed signature is valid. The recipient of the signature is unable to prove this to the verifier without the confirmer. As an optional stage, the designated confirmer signature may be convertible into an ordinary signature with a Conversion Protocol.

We will now describe Chaum's original designated-confirmer scheme [20]. In this example, let p be a large prime so that the discrete logarithm problem in $\mathbb{Z}_p^*$ is infeasible, and let $g \in \mathbb{Z}_p^*$ be a generator. Let (N, e) be the signer's RSA public key, where d is the corresponding private key. The confirmer's private key is $z \in \mathbb{Z}_{p-1}^*$ and her corresponding public key is $h = g^z \bmod p$.

Signing Protocol

In this protocol, the signer computes a designated-confirmer signature on the message $m \in \{0,1\}^*$ and convinces the receiver that the signature is valid in such a way that the receiver is unable to prove the signature is valid to a third party. The procedure between the signer and receiver is as follows:

1. The signer selects a random value $x \in \mathbb{Z}_{p-1}^*$ and computes $A = g^x \bmod p$, $B = h^x \bmod p$ and $\alpha = (F(A, B) \oplus H(m))^d \bmod N$, where F is a combining function and H is a hash function. (Recall that a combining function unites two or more functions through concatenation, composition, multiplication, etc.)

2. Then the signer sends the signature (A, B, α) to the receiver.

3. The receiver selects random $s, t \in \mathbb{Z}_{p-1}^*$, computes $c = g^s h^t \bmod p$ and sends c to the signer.

4. The signer selects a random $q \in \mathbb{Z}_{p-1}^*$, computes $a = g^q \bmod p$ and $b = (ca)^x \bmod p$ and sends a, b to the receiver.

5. The receiver reveals her random integers s and t to the signer.

6. The signer checks that the receiver formed c correctly by verifying that $c = g^s h^t \bmod p$. If this equation holds, the signer reveals her random q to the receiver.

7. The receiver checks that $a = g^q \bmod p$, $b/A^q \equiv A^s B^t \pmod p$ and $F(A, B) \oplus H(m) \equiv \alpha^e \pmod N$. If these conditions hold, the receiver is convinced that $B = A^z \bmod p$ but has no way to prove this to anyone else.

To see that these conditions convince the receiver, note that

$$b/A^q \equiv A^s B^t \pmod p$$

$$\begin{aligned}
&\Leftrightarrow\ (ca)^x/g^{xq} \equiv g^{xs}B^t \pmod p\\
&\Leftrightarrow\ (g^s h^t)^x \equiv g^{xs}B^t \pmod p\\
&\Leftrightarrow\ h^{tx} \equiv B^t \pmod p\\
&\Leftrightarrow\ g^{zx} \equiv B \pmod p\\
&\Leftrightarrow\ A^z \equiv B \pmod p.
\end{aligned}$$

Confirmation Protocol

In this protocol, a confirmer designated by the signer can help the receiver prove to a third party that the designated-confirmer signature (A, B, α) is valid. This protocol is between the confirmer and a third party verifier whom the receiver wishes to convince.

1. The verifier checks that $F(A, B) \oplus H(m) \equiv \alpha^e \pmod N$
2. The verifier selects random integers $u, v \in \mathbb{Z}^*_{p-1}$, computes $k = g^u A^v \bmod p$ and sends k to the confirmer.
3. The confirmer selects a random $w \in \mathbb{Z}^*_{p-1}$, computes $l = g^w \bmod p$ and $n = (kl)^z \bmod p$, and sends l and n to the verifier.
4. The verifier reveals her secret u, v to the confirmer.
5. The confirmer checks that k was formed correctly by verifying that $k = g^u A^v \bmod p$. If this equation holds, the confirmer reveals w to the verifier.
6. The verifier checks that $l = g^w \bmod p$ and $n/h^w \equiv h^u B^v \pmod p$. If these conditions hold, the verifier is convinced that the original signer's signature

is correct but has no way to prove this to anyone else.

To see how the verifier is convinced that the signer properly computed $B = A^z \bmod p$, where z is the confirmer's private key, note that

$$\begin{aligned} & n/h^w \equiv h^u B^v \pmod p \\ \Leftrightarrow\ & (kl)^z/h^w \equiv g^{zu} B^v \pmod p \\ \Leftrightarrow\ & (g^u A^v g^w)^z/g^{zw} \equiv g^{zu} B^v \pmod p \\ \Leftrightarrow\ & g^{uz} A^{vz} \equiv g^{zu} B^v \pmod p \\ \Leftrightarrow\ & A^z \equiv B \pmod p. \end{aligned}$$

Conversion Protocol

The confirmer can convert the designated-confirmer signature into an ordinary digital signature which can be universally verified as follows:

1. The confirmer selects a random $w \in \mathbb{Z}^*_{p-1}$ and computes $r = A^w \bmod p$ and $y = w + zH(A, r) \bmod (p-1)$ and sends (r, y) to the verifier.
2. The verifier checks that $A^y \equiv rB^{H(A,r)} \pmod p$ and accepts the designated confirmer signature as valid.

The verifier can now convince anyone else that the signature is valid by publishing the pair (r, y). Verification works since

$$\begin{aligned} & A^y \equiv rB^{H(A,r)} \pmod p \\ \Leftrightarrow\ & A^y/r \equiv B^{H(A,r)} \pmod p \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \quad A^{y-w} \equiv B^{H(A,r)} \pmod p \\
&\Leftrightarrow \quad A^{(y-w)/H(A,r)} \equiv B \pmod p \\
&\Leftrightarrow \quad A^{z} \equiv B \pmod p.
\end{aligned}$$

Given the pair (r, y), any user can be convinced that the signature is valid since only the confirmer who knows the secret z such that $B = A^z \bmod p$ can compute $y = w + zH(A, r) \bmod (p-1)$. By the assumption of the intractability of the discrete logarithm problem, no other user can find such an exponent z to generate y.

Security

There is no formal proof of security for the scheme outlined above. We will only note that the security consideration of non-transferability associated with undeniable signatures is maintained in designated-confirmer signatures. That is, once a verifier is convinced that a signature is valid, she has no way to convince any other party of this. This property is fulfilled in this scheme by the intractability of the discrete logarithm problem.

Bibliography

[1] M. Abe and E. Fujisaki. How to date blind signatures. In *Advances in Cryptology - ASIACRYPT '96*, pages 224–251, 1996.

[2] M. Abe and T. Okamoto. Provably secure partially blind signatures. In *Advances in Cryptology - CRYPTO 2000*, pages 271–286, 2000.

[3] R. Anderson. In *Proceedings of 4th ACM Conference on Computer and Communications Security*, 1997. Invited lecture.

[4] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudick. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology - CRYPTO 2000*, pages 255–270, 2002.

[5] P. Béguin and J. Quisquater. Fast server-aided RSA signatures secure against active attacks. In *Advances in Cryptology - CRYPTO '95*, pages 57–69, 1995.

[6] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology - EUROCRYPT '98*, pages 236–250, 1998.

[7] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology - CRYPTO '94*, pages 216–233, 1994.

[8] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Advances in Cryptology - EUROCRYPT '97*, pages 163–192, 1997.

[9] J. Boyar, D. Chaum, I. Damgard, and T. Pederson. Convertible undeniable signatures. In *Advances in Cryptology - CRYPTO '90*, pages 189–205, 1991.

[10] C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology - ASIACRYPT 2000*, pages 58–71, 2000.

[11] E. Bresson and J. Stern. Efficient revocation in group signatures. In *Proceedings of PKC 2001*, pages 190–206, 2001.

[12] E. Brickell, D. Gordon, K. McCurley, and D. Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology - EUROCRYPT '92*, pages 200–207, 1992.

[13] J Camenisch. Efficient and generalized group signatures. In *Advances in Cryptology - EUROCRYPT '97*, pages 465–479, 1997.

[14] J. Camenisch, J. Piveteau, and M. Stadler. Fair blind signatures. In *Advances in Cryptology - EUROCRYPT'95*, pages 209–219, 1995.

[15] J. Camenisch, J. Piveteau, and M. Stadler. An efficient fair payment system.

In *ACM Conference on Computer and Communications Security*, pages 88–94, 1996.

[16] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology - CRYPTO '97*, pages 410–424, 1997.

[17] J.L. Camenisch, J. Piveteau, and M.A. Stadler. Blind signatures based on the discrete logarithm problem. In *Advances in Cryptology - EUROCRYPT '94*, pages 428–432, 1995.

[18] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology - CRYPTO '82*, pages 199–203, 1982.

[19] D. Chaum. Security without identification: transaction systems to make big brother obselete. *Communications of the ACM*, 28:1030–1044, 1985.

[20] D. Chaum. Designated confirmer signatures. In *Advances in Cryptology - EUROCRYPT '94*, pages 86–91, 1994.

[21] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Advances in Cryptology - CRYPTO '89*, pages 212–216, 1990.

[22] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology - EUROCRYPT '91*, pages 644–654, 1991.

[23] H. Chien, J. Jan, and Y. Tseng. RSA-based partially blind signature with low computation. In *Proceedings of the 8th International Conference on Parallel and Distributed Systems*, pages 385–389, 2001.

[24] D.W. Davies and W.L. Price. *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*. John Wiley & Sons, 1989.

[25] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of 28th Annual FOCS*, pages 427–437, 1987.

[26] A. Fiat. Batch RSA. In *Advances in Cryptology - CRYPTO '89*, pages 175–185, 1990.

[27] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, pages 186–194, 1987.

[28] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology - CRYPTO '99*, pages 449–466, 1999.

[29] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Advances in Cryptology - CRYPTO '96*, pages 157–172, 1996.

[30] H. Ghodosi and J. Pieprzyk. Repudiation of cheating and non-repudiation of Zhang's proxy signature schemes. In *Proceedings of ACISP'99*, pages 129–134, 1999.

[31] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.

[32] L. Guillou and J. Quisquater. A “paradoxical” identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology - CRYPTO '88*, pages 216–231, 1990.

[33] L. Harn. Batch verifying multiple DSA-type digital signatures. In *Electronic Letters*, volume 34, pages 870–871, 1998.

[34] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *Advances in Cryptology - CRYPTO 2001*, pages 332–354, 2001.

[35] M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive RSA for constant-size thresholds. Unpublished manuscript, 1995.

[36] C. Laih, H. Sun, and W. Yang. On the design of RSA with short secret exponent. In *Advances in Cryptology - ASIACRYPT '99*, pages 150–164, 1999.

[37] C. Laih and S. Yen. Improved digital signature suitable for batch verification. In *Proceedings of the IEEE - Transactions on Computers*, volume 44, pages 957–959, 1995.

[38] N. Lee, T. Hwang, and C. Wang. On Zhang's nonrepudiable proxy signature scheme. In *Proceedings of ACISP '98*, volume 1438, pages 415–422, 1998.

[39] M. Mambo, E. Okamoto, and K. Uwuda. Proxy signatures for delegating signing operation. In *Proceedings of 3rd ACM Conference on Computer and Communications Security*, pages 48–57, 1996.

[40] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, chapter 2.4.2. CRC Press, 1996.

[41] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, chapter 11.6.3, pages 466–468. CRC Press, 1996.

[42] R. Merkle. One way hash functions and DES. In *Advances in Cryptology - CRYPTO '89*, pages 428–446, 1989.

[43] R. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89*, pages 218–238, 1990.

[44] M. Michels and M. Stadler. Efficient convertible undeniable signature schemes. In *Proceedings of SAC '97*, pages 231–244, 1997.

[45] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the 10th PODC*, pages 51–59, 1991.

[46] B. Pfitzmann. Sorting out signature schemes. In *Proceedings of 1st ACM Conference on Computer and Communications Security*, pages 74–85, 1993.

[47] B. Pfitzmann. *Digital Signature Schemes: General Framework and Fail-Stop Signatures*, chapter 3, pages 33–36. Springer-Verlag, 1996.

[48] B. Pfitzmann and M. Waidner. Attacks on protocols for server-aided RSA computation. In *Advances in Cryptology - EUROCRYPT '92*, pages 153–162, 1993.

[49] T. Rabin. A simplified approach to threshold and proactive RSA. In *Advances in Cryptology - CRYPTO '98*, pages 89–104, 1998.

[50] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology - ASIACRYPT 2001*, pages 552–565, 2001.

[51] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proceedings of 26th ACM Symposium on Theory of Computing*, pages 522–533, 1994.

[52] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology - CRYPTO '84*, pages 47–53, 1985.

[53] Douglas R. Stinson. *Cryptography: Theory and Practice*, chapter 6, pages 219–223. CRC Press, 1995.

[54] E. van Heyst and T. Pedersen. How to make efficient fail-stop signatures. In *Advances in Cryptology - EUROCRYPT '92*, pages 366–377, 1993.