The Cramer-Shoup Strong-RSA Signature Scheme Revisited

Marc Fischlin

Fraunhofer-Institute Secure Telecooperation (SIT),
Security and Smart Card Technologies (SICA)

marc.fischlin@sit.fraunhofer.de
http://www.sit.fraunhofer.de/~fischlin/

Abstract. We discuss a modification of the Cramer-Shoup strong-RSA signature scheme. Our proposal also presumes the strong RSA assumption, but allows faster signing and verification and produces signatures of roughly half the size. Then we present a stateful version of our scheme where signing (but not verifying) becomes almost as efficient as with RSA-PSS. We also show how to turn our signature schemes into "lightweight" anonymous yet linkable group identification protocols without random oracles.

1 Introduction

Existential unforgeability under adaptive chosen-message attacks has become the salient security criterion for signature schemes. For instance, the well-known RSA-PSS scheme [4, 17] meets this requirement under the RSA assumption in the random oracle model. But only very few efficient schemes are known to achieve this security level without relying on random oracles. One of these schemes is the Cramer-Shoup signature scheme [10] which is provably secure under the strong RSA (aka. flexible RSA) assumption.

Here, we present an improvement of the Cramer-Shoup scheme which also forgos random oracles and is merely based on the strong RSA assumption (and a collision-intractable hash function for long messages). In the original Cramer-Shoup scheme each signature requires the signer to generate a prime and to compute two exponentations, one exponentiation with a full-fledged exponent and the other one with a smaller exponent. Our solution eliminates the "small" exponentiation which, according to the implementation figures of the Cramer-Shoup scheme [10, 18], saves almost one third of the time for signature generation (when standard speed-up methods like preprocessing and Chinese remainder are used).

Additionally, our proposal almost halves the size of a signature, e.g., for a 1024-bit RSA modulus a signature now has 1350 bits instead of 2200 bits as in the original scheme. The size of the public key in our case marginally grows, but verification too becomes slightly faster and the key generation times are essentially identical.

We then present a stateful variation of our signature system. The state information consists of a short prime, typically less than 60 bits long. But this state information buys us another efficiency improvement for the signer while keeping the effort for the verifier unchanged. Namely, the expensive prime generation for each signature almost vanishes, such that the signer mainly has to compute a full-fledged exponentiation. Therefore, signing becomes almost as efficient as for RSA-PSS (yet, in our case, additional preprocessing techniques apply). Moreover, the signature size in our stateful variant is decreased even further, by approximately 100 bits, making it comparable to the size of RSA-PSS signatures. Still, RSA-PSS is significantly superior with respect to key generation and verification, and is of course stateless.

At the end of this paper, we touch anonymous group identification protocols in which users can prove membership in a group without dicslosing their identity. We discuss how to construct a "lightweight" anonymous (yet linkable) group identification scheme from our signature schemes. Our solution does not need random oracles, and the group's common public key as well as the performance of a single identification is independent of the number of users.

2 A Modification of the Cramer-Shoup Protocol

In this section we recall the original Cramer-Shoup scheme, introduce our modification and prove it to be secure, and compare our proposal to the original protocol.

We adhere to the notation in [10]; still, the protocol description should be intelligible without [10]. We remark that the strong RSA assumption (introduced by Barić and Pfitzmann [2] as well as Fujisaki and Okamoto [13]) says that for a random RSA modulus n and a random element $z \in \mathbb{Z}_n^*$ it is infeasible to find an integer $e \geq 2$ and the e-th root of z in $\mathbb{Z}_n^*$. Hence, compared to the ordinary RSA assumption where the exponent is given, a solution for the strong RSA problem allows to come up with a self-determined exponent.

Recently, Damgård and Koprowski [11] have generalized the Cramer-Shoup signature scheme to generic groups for which the strong root assumption, the counterpart to the strong RSA assumption in $\mathbb{Z}_n^*$, holds. We note that our improvements here also apply to the model of Damgård and Koprowski.

2.1 Original Cramer-Shoup Signature Scheme

The original Cramer-Shoup scheme works as follows:

Key Generation: Generate $n = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ for primes p, q, p', q'. Also pick two quadratic residues $h, x \in \mathrm{QR}_n$ and a random $(l+1)$-bit prime e'. The public verfication key is (n, h, x, e') and the private key is (p, q).

Signing: To sign a message m compute the l-bit hash value $H(m)$ with a collision-intractable hash function $H(\cdot)$. Pick a random $(l+1)$-bit prime $e \neq e'$ and a random $y' \in \mathrm{QR}_n$, compute x' where

$$(y')^{e'} = x' h^{H(m)} \bmod n$$

as well as y with

$$y^e = x h^{H(x')} \bmod n.$$

Computing this e-th root is easy given the factorization of n. The signature equals (e, y, y').

Verification: First check that e is an odd $(l+1)$-bit integer different from e', then compute $x' = (y')^{e'} h^{-H(m)}$ and verify that $x = y^e h^{-H(m)}$.

Note that computing the e-th root of $x h^{H(x')}$ corresponds to an exponentiation with a full-fledged exponent $e^{-1} \bmod \varphi(n)$, while the computation of y' solely involves "small" l-bit exponents. If these exponentiations are performed in $\mathbb{Z}_n^*$ then the running times differ significantly. However, when using the Chinese remainder and appropriate preprocessing methods, the implementation results in [10, 18] show that both exponentiations roughly need the same time. Specifically, according to [10, 18] the prime generation and the exponentiations then each take approximately one third of the total signing time.

2.2 Modified Cramer-Shoup Signature Scheme

One can view the value $H(x')$ as a trapdoor commitment of the message m, using the RSA trapdoor commitment scheme. Therefore, as pointed out in [10], one may replace this part with any other appropriate trapdoor commitment. Indeed, [10, Sec. 5] suggest as an example a trapdoor commitment based on the discrete-log assumption. By this, the signature length shrinks to almost half of the original size. Unfortunately, this advantage disappears again if one switches to other trapdoor commitments based on the RSA or factoring assumption, or even general one-way functions.

The second part of the signature generation can be thought of as a representation problem. That is, a representation of x with respect to h, e, n is a pair (α, y) such that $h^\alpha y^e = x \bmod n$. In this sense, a signature in the original protocol requires that one finds a representation of x involving the hash value $-H(x')$ and a self-determined exponent e. In the modified signature scheme here, we assimilate the trapdoor commitment to the representation problem:

Key Generation: Generate $n = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ for primes p, q, p', q'. Also pick three quadratic residues $h_1, h_2, x \in \mathrm{QR}_n$. The public verfication key is (n, h_1, h_2, x) and the private key is (p, q).

Signing: To sign a message m calculate the l-bit hash value $H(m)$ with a collision-intractable hash function $H(\cdot)$. Pick a random $(l+1)$-bit prime e, a random l-bit string α and compute a representation $(-\alpha, -(\alpha \oplus H(m)), y)$ of x with respect to h_1, h_2, e, n, i.e.,

$$y^e = x h_1^{\alpha} h_2^{\alpha \oplus H(m)} \bmod n.$$

Computing this e-th root y from $x h_1^{\alpha} h_2^{\alpha \oplus H(m)}$ is easy given the factorization of n. The signature is given by (e, α, y).

Verification: Check that e is an odd $(l+1)$-bit integer, that α is l bits long, and that $y^e = x h_1^{\alpha} h_2^{\alpha \oplus H(m)} \bmod n$.

The idea of splitting $H(m)$ into random (but dependent) parts α and $\alpha \oplus H(m)$ is not new. It has already been applied for the well-known protocol for proving knowledge of one out of two discrete logarithms [9] and for security amplification reducing chosen-message attacks to random-message attacks [8]. As we will discuss below, it also gives the desired trapdoor information for proving security here.

We remark that we may instead select α at random in $\mathbb{Z}_e$ and split the message into α and $\alpha + H(m) \bmod e$. Moreover, we may alternatively define y in the signature generation as the unique value such that $x = h_1^{\alpha} h_2^{\alpha \oplus H(m)} y^e \bmod n$, i.e., rearrange the equation to derive a "well-formed" representation problem. Our security proof also works for these variations, even when combined.

2.3 Performance Comparison

Compared to the original scheme with signature size $2|n| + l + 1$, both the modification here as well as the one using the discrete-log trapdoor commitment produce signatures of size $|n| + 2l + 1$. Disadvantegeously, both modifications slightly increase the size of the public key, e.g., adding $|n| - l$ bits in our case. However, this is outweighed by the shorter signatures.

The same speedup techniques as in [10, Sec. 3, 6 and 7] apply here (e.g., computation via Chinese remainder, faster prime number generation, taking e-th roots efficiently, precomputation techniques, etc.). In particular, selecting $x = h_1^a$ and $h_2 = h_1^{a'}$ for appropriate a, a' and storing a, a' in the secret key, the effort to compute the e-th root of $x h_1^{\alpha} h_2^{\alpha \oplus H(m)} = h_1^{\alpha + a + a'(\alpha \oplus H(m))}$ is (almost) the same as in the original scheme for $x h^{H(x')} = h^{a + H(x')}$. That is, the signer first computes $f = e^{-1}(a + H(x')) \bmod \varphi(n)$ in the original scheme or, in our case, $f = e^{-1}(\alpha + a + a'(\alpha \oplus H(m))) \bmod \varphi(n)$ with a few more operations. Then the signer calculates $h^f \bmod n$ resp. $h_1^f \bmod n$, possibly using the Chinese remainder method and preprocessing techniques. In summary, since our proposal does not require the computation of the separate trapdoor commitment we eliminate the "small" exponentiation almost for free.

For signature verification, the cost for the verifier for checking the single equation in our scheme with two generators h_1, h_2 is only marginally higher than the cost of checking the equation $y^e = xh^{H(x')}$ with a single generator h in the original scheme. The reason is that, using standard methods, such exponentiations can be carried out with roughly the same effort as in the single generator case. Yet, in our case the additional verification of the trapdoor commitment disappears.

Unfortunately, all solutions share the expensive prime generation of e. A profound algorithm to generate e has been presented in [10, Sec. 6] (see also the corresponding implementation results in [18]). Another possible improvement is to decrease the length of e at the cost of a larger public key. Namely, if we put, say, three values h_1, h_2, h_3 into the public key, then we can divide the hash value $H(m)$ into halves $H_1(m), H_2(m)$ of 80 bits each, and choose α and e to be 80 and 81 bits, respectively. A signature is then described by the equation $xh_1^{\alpha} h_2^{\alpha \oplus H_1(m)} h_3^{\alpha \oplus H_2(m)} = y^e$, and the signature length is about 80 bits shorter. The security proof in the next section straightforwardly extends to this case.

If we choose three generators h_1, h_2, h_3, then the effort for the signer to compute the e-th root y given stored values a, a', a'' does not change significantly in comparison to the case of two generators. But an 81-bit prime e is much easier to find than a 161-bit one. The verifier now has to perform a faster to compute "quadruple" exponentiation $h_1^{\alpha} h_2^{\alpha \oplus H_1(m)} h_3^{\alpha \oplus H_2(m)} y^e$ with 81-bit exponents instead of a "triple" exponentiation $h_1^{\alpha} h_2^{\alpha \oplus H(m)} y^e$ with 161-bit exponents.

Our signature scheme also has the feature that for short messages, e.g., of 80 bits, a collision-intractable hash function becomes obsolete and the signer may choose e also as a shorter prime, e.g., 81 bits or even 41 bits with the trick above. Moreover, signing and verifying become slightly faster. This may be interesting for identification protocols, where users identify by signing short random messages.

2.4 Security Proof

We discuss that the modified signature scheme is secure against adaptive chosen-message attacks. Basically, the proof follows the one in [10].

Note that in an adaptive chosen-message attack the adversary is given the public key of the signer and can ask the signer to sign arbitrary messages. The choice of the next message submitted to this signature oracle is adaptively determined by the data gathered before. Finally, the adversary outputs a message that has not been signed by the oracle, together with a putative signature for this message.

Let m_i be the i-th query to the signer and (e_i, α_i, y_i) denote the answer. Let m and (e, α, y) be the putative forgery of the adversary. We assume that all e_i chosen by the signer during an attack are distinct (yet, the adversary's choice e may equal some e_j), and that $H(m) \neq H(m_i)$ for all m_i (otherwise we have found a collision $m \neq m_i$).

There are two types of forgers (dubbed according to [10]):

Type II: The adversary outputs $e = e_j$ for some j.
Type III: The adversary outputs a new e, different from all e_i.

Type I forgers as in [10] disappear due to our modification. We show that type II forgers contradict the (ordinary) RSA assumption, whereas type III forgers refute the strong RSA assumption.

Type II Forger

We assume that we know j, otherwise we can guess it. Since $H(m_j) \neq H(m)$ we have $\alpha_j \neq \alpha$ or $\alpha_j \oplus H(m_j) \neq \alpha \oplus H(m)$. With probability 1/2 we can guess in advance which case will happen, and we assume for simplicity that $\alpha_j \neq \alpha$ here. The other case is treated analogously.

We are given n, $z \in \mathbb{Z}_n^*$ and an odd prime r and are supposed to output $z^{1/r}$. To do so, we invoke the type II forger on the following public key and signature oracle: Set $e_j = r$ and for all $i \neq j$ choose a random $(l+1)$-bit prime e_i (where i is bounded by the number of queries to the signature oracle in the attack). Let

$$h_1 = z^{2 \cdot \prod_{i \neq j} e_i}, \quad h_2 = v^{2 \cdot \prod_i e_i}, \quad x = h_1^{-\beta} \cdot w^{2 \cdot \prod_i e_i}$$

for random $v, w \in \mathbb{Z}_n^*$ and a random l-bit string β. The "prepared" public key is (n, h_1, h_2, x).

To sign the i-the message on behalf of the signer, $i \neq j$, select an l-bit string α_i and compute

$$\begin{aligned} y_i &= w^{2 \cdot \prod_{k \neq i} e_k} \cdot \left(z^{2 \cdot \prod_{k \neq j, k \neq i} e_k}\right)^{\alpha_i - \beta} \cdot \left(v^{2 \cdot \prod_{k \neq i} e_k}\right)^{\alpha_i \oplus H(m_i)} \\ &= \left(x h_1^{\alpha_i} h_2^{\alpha_i \oplus H(m_i)}\right)^{1/e_i} \end{aligned}$$

For the j-th signature query set $\alpha_j = \beta$ and compute y_j as[1]

$$y_j = w^{2 \cdot \prod_{k \neq j} e_k} \cdot \left(v^{2 \cdot \prod_{k \neq j} e_k}\right)^{\alpha_j \oplus H(m_j)} = \left(x h_1^{\alpha_j} h_2^{\alpha_j \oplus H(m_j)}\right)^{1/e_j}$$

It is not hard to see that the data in this simulation is identically distributed to the one in a real attack. In particular, x and the signatures for $i \neq j$ are distributed independently of β, and therefore α_j in this simulation has the same distribution as in an actual attack.

The adversary's output yields another representation of x with respect to n, h_1, h_2 and $e_j = r$. More precisely,

$$h_1^{-\alpha_j} h_2^{-(\alpha_j \oplus H(m_j))} y_j^r = x = h_1^{-\alpha} h_2^{-(\alpha \oplus H(m))} y^r \bmod n.$$

And, plugging in the preselected values,

$$\begin{aligned} h_1^{\alpha - \alpha_j} &= h_2^{(\alpha_j \oplus H(m_j)) - (\alpha \oplus H(m))} \cdot (y y_j^{-1})^r \\ z^{2 \cdot \prod_{i \neq j} e_i \cdot (\alpha - \alpha_j)} &= \left(v^{2 \cdot \prod_{i \neq j} e_i \cdot ((\alpha_j \oplus H(m_j)) - (\alpha \oplus H(m))} \cdot y y_j^{-1}\right)^r \end{aligned}$$

[1] If we had bet on $\alpha_j \oplus H(m_j) \neq \alpha \oplus H(m)$ then we would have basically swapped the roles of h_1 and h_2 and would now set $\alpha_j = \beta \oplus H(m_j)$.

Since $|\alpha - \alpha_j| \in \mathbb{Z}_r - \{0\}$ and all e_k are relatively prime, we can compute an r-th root of z by standard procedures (see, for instance, [10]).

Type III Forger

This case is almost identical to the one discussed in [10]. Namely, given n, z preselect all e_i and set

$$h_1 = z^{2 \cdot \prod_i e_i}, \quad x = h_1^a, \quad h_2 = h_1^{a'}$$

for random $a, a' \in \{1, \ldots, n^2\}$. As h_1 is a generator of QR_n with high probability and since $a, a' \bmod p'q'$ are statistically close to the uniform distribution on $\mathbb{Z}_{p'q'}$, the values x, h_2 are almost uniformly distributed quadratic residues. Also, we can sign any query m_i since we know the e_i-th roots of $x h_1^{\alpha_i} h_2^{\alpha_i \oplus H(m_i)}$ for any α_i. On the other side, the forgery yields the equation

$$y^e = x h_1^{\alpha} h_2^{\alpha \oplus H(m)} = z^m$$

where

$$m = 2 \cdot \prod_i e_i \cdot (a + \alpha + a'(\alpha \oplus H(m))).$$

The fact that $e \not| m$ with non-negligible probability and that we can compute a non-trivial $e/\gcd(e, m)$-th root of z now follows as in [10]. Specifically, if r is a prime dividing e, then r clearly does not divide $2 \cdot \prod_i e_i$. Write a as $a = bp'q' + c$ for $0 \leq c < p'q'$ and note that the adverary's view is essentially independent of b, even if given c. Hence, $b \bmod r$ is almost uniform on $\mathbb{Z}_r$ and the probability that $r | (a + \alpha + a'(\alpha \oplus H(m)))$ or, equivalently, that $a + \alpha + a'(\alpha \oplus H(m)) = 0 \bmod r$ is negligibly close to $1/r$.

We conclude that with probability close to $1 - 1/r$ for the smallest prime factor r of e we have $e \not| m$. Once more, in this case it is easy to compute a non-trivial $e/\gcd(e, m)$-th root of z by standard techniques.

3 Efficient Stateful Signatures

In this section we present a stateful variation of our signature scheme above. This modification here removes the necessity to generate a 161-bit prime for each signature. Instead, the signer uses a smaller prime number, e.g., up to 60 bits, which he must update after each signing.

3.1 Description

Our stateful scheme can be outlined as follows. Instead of using random $(l+1)$-bit primes e (such that $e \geq 2^l - 1$) as before, the signer here uses shorter primes e and the smallest power e^t such that $e^t \geq 2^l - 1$. The signer starts with $e_1 = 3$ or, for security reasons [6], rather with $e_1 = 2^{16} + 1$. After signing with this prime,

he proceeds to the next prime e_2 after e_1. Having used e_2 the signer then picks e_3 for the next signature and so on. More generally, for the j-th signature the signer uses the j-th prime e_j (after the offset $2^{16}+1$). For this, the signer stores the current prime e and, after signing, updates it with algorithm $\mathsf{nextprime}(e)$ which generates the closest prime after e.

Key Generation: Generate $n = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ for primes p, q, p', q'. Also pick three quadratic residues $h_1, h_2, x \in \mathrm{QR}_n$ and initialize $e = 2^{16} + 1$. The public verfication key is (n, h_1, h_2, x) and the private key is (p, q) and the state information is given by e.

Signing: To sign a message m calculate the l-bit hash value $H(m)$ with a collision-intractable hash function $H(\cdot)$. Find the smallest integer t such that $e^t \geq 2^l - 1$. Then pick a random l-bit string α and compute a representation $(-\alpha, -(\alpha \oplus H(m)), y)$ of x with respect to h_1, h_2, e^t, n, i.e.,

$$y^{e^t} = x h_1^{\alpha} h_2^{\alpha \oplus H(m)} \bmod n.$$

Computing this e^t-th root y from $x h_1^{\alpha} h_2^{\alpha \oplus H(m)}$ is easy given the factorization of n. The signature is given by (e, α, y). Finally, update e as $\mathsf{nextprime}(e)$.

Verification: Given a putative signature (e, α, y) check that α is l bits long, that $e \geq 2^{16} + 1$ is odd, and that $y^{e^t} = x h_1^{\alpha} h_2^{\alpha \oplus H(m)} \bmod n$, where t is the smallest integer such that $e^t \geq 2^l - 1$.

3.2 Performance Comparison

In comparison to the modified Cramer-Shoup scheme in the previous section, in this stateful scheme here the signer needs to generate a much shorter prime. In fact, if the signer signs at most 2^{50} messages, then less than 60 bits are enough to store the current prime.

We remark that short primes are easier to generate. For example, using an observation by Bleichenbacher [5], reported in [15] and also pointed out in [10], there is a determinsitic primality test for short numbers. Bleichenbacher has shown that it suffices to run the Miller-Rabin primality test with fixed bases $\mathcal{B} = \{2, 3, 5, 7, 11, 13, 23\}$ in order to identify primes up to 53 bits. Additionally, for shorter primes, e.g., of 30 bits, an even smaller base suffices (see [5, Chap. 3]). Together with standard trial division, this gives a very efficient way to generate the primes (up to the approximately first 2^{45} signatures).

In summary, the signing time is now dominated by the computation of the e^t-th root y. Thus, the overall effort for signing is pretty close to the one for RSA signatures where one also computes an e-th root. Yet, in contrast to RSA where the exponent and its inverse are fixed, here we must first compute the inverse of e^t when signing. But in favor of our scheme we observe that preprocessing for the computation of the root y is applicable.

Surprisingly, the gain in efficiency compared to the stateless version also comes with a decrease in signature size. Namely, instead of 161-bit primes, a signature now contains the j-th smallest prime after $2^{16}+1$. As noted above, 60 bits are usually sufficient to append this prime to the signature, thus saving another 100 bits.

3.3 Security Proof

Once more, assume that an adversary successfully runs an adaptive chosen-message attack. Denote the i-th submission to the signer by m_i and let (e_i, α_i, y_i) be the answer. We also denote by t_i the corresponding exponent such that $e_i^{t_i} \geq 2^l - 1$. The putative forgery is given by m and (e, α, y). For simplicity, assume again that $H(m) \neq H(m_i)$ for all m_i.

The two types of forgers are:

Type II: The adversary outputs e such that $e_j | e$ for some j.
Type III: The adversary outputs e such that no prime e_j divides e.

In a sense, these types generalize the ones of the proof in the previous section. It is therefore not surprising that the proof carries over immediately: Type II forgers will contradict the RSA assumption (for a small given exponent $r = e_j$, that is, the j-th prime e_j after $2^{16}+1$), whereas type III forgers refute the strong RSA assumption.

Type II Forger

This part of the proof is similar to the case of type II forgers before. Suppose that we know j with $e_j | e$ in advance. Again, from $H(m_j) \neq H(m)$ it follows $\alpha_j \neq \alpha$ or $\alpha_j \oplus H(m_j) \neq \alpha \oplus H(m)$, and, to simplify, we only treat the case $\alpha_j \neq \alpha$ here.

Given n, $z \in \mathbb{Z}_n^*$ and the j-th prime e_j after $2^{16}+1$, we feed the adversary the following data: For all $i \neq j$ compute the i-th prime e_i and the exponent t_i and set

$$h_1 = z^{2 \cdot \prod_{i \neq j} e_i^{t_i}}, \quad h_2 = v^{2 \cdot \prod_i e_i^{t_i}}, \quad x = h_1^{-\beta} \cdot w^{2 \cdot \prod_i e_i^{t_i}}$$

for random $v, w \in \mathbb{Z}_n^*$ and a random l-bit string β. These values make up the public key (n, h_1, h_2, x).

To simulate the signing process for the i-the message, $i \neq j$, choose a random l-bit value α_i and compute

$$\begin{aligned} y_i &= w^{2 \cdot \prod_{k \neq i} e_k^{t_k}} \cdot \left(z^{2 \cdot \prod_{k \neq j, k \neq i} e_k^{t_k}} \right)^{\alpha_i - \beta} \cdot \left(v^{2 \cdot \prod_{k \neq i} e_k^{t_k}} \right)^{\alpha_i \oplus H(m_i)} \\ &= \left(x h_1^{\alpha_i} h_2^{\alpha_i \oplus H(m_i)} \right)^{1/e_i^{t_i}} \end{aligned}$$

For the j-th signature query set $\alpha_j = \beta$ and compute y_j as

$$y_j = w^{2 \cdot \prod_{k \neq j} e_k^{t_k}} \cdot \left(v^{2 \cdot \prod_{k \neq j} e_k^{t_k}} \right)^{\alpha_j \oplus H(m_j)} = \left(x h_1^{\alpha_j} h_2^{\alpha_j \oplus H(m_j)} \right)^{1/e_j^{t_j}}$$

A successful forgery of the adversary yields another representation of x with respect to n, h_1, h_2 and $e_j^{t_j}$. Specifically,

$$h_1^{-\alpha_j} h_2^{-(\alpha_j \oplus H(m_j))} y_j^{e_j^{t_j}} = x = h_1^{-\alpha} h_2^{-(\alpha \oplus H(m))} y^{e_j^{t_j}} \bmod n.$$

Therefore,

$$h_1^{\alpha - \alpha_j} = h_2^{(\alpha_j \oplus H(m_j)) - (\alpha \oplus H(m))} \cdot (y y_j^{-1})^{e_j^{t_j}}$$
$$z^{2 \cdot \prod_{i \neq j} e_i^{t_i} \cdot (\alpha - \alpha_j)} = \left(v^{2 \cdot \prod_{i \neq j} e_i^{t_i} \cdot ((\alpha_j \oplus H(m_j)) - (\alpha \oplus H(m))} \cdot y y_j^{-1}\right)^{e_j^{t_j}}$$

Because $\alpha \neq \alpha_j$ and $e_j^{t_j} \geq 2^l - 1$ there is some integer $k < t_j$ such that

$$\alpha = \alpha_j \bmod e_j^k \quad \text{and} \quad \alpha \neq \alpha_j \bmod e_j^{k+1}$$

Applying well-known techniques (see again [10], for example) we can compute an $e_j^{t_j}$-th root a of $h_1^{e_j^k}$. Since raising elements to the e_j-th power is a permutation, it follows that $a^{e_j^{t_j - k - 1}}$ is an e_j-th root of $h_1 = z^{2 \cdot \prod_{i \neq j} e_i^{t_i}}$. Once more, this gives us straightforwardly an e_j-th root of the given value z.

Type III Forger

Observing that the smallest prime factor r of the adversary's choice e does not divide any $e_i^{t_i}$ (by assumption), this part of the proof is identical to the previously given proof. We thus omit this part and note that this completes the proof.

Remark

The devil's advocate may object that, in this scheme here, type II adversaries need to break RSA for a small exponent $e \geq 2^{16} + 1$ only, whereas in the stateless version the adversary must use a large $(l+1)$-bit prime. However, we remark that for low public exponents $e \geq 2^{16} + 1$ no better attacks than the ones for large exponents are known [6]. Additionally, our scheme supports a flexible offset and one may start with larger primes, for example, $e_1 \geq 2^{40}$. But this also means a slight loss in efficiency.

4 "Lightweight" Anonymous Group Identification

With an anonymous group identification scheme each user of a group is able to prove membership in the group while hiding his identity among the group members. Below, we present an anonymous group identification scheme which does not rely on random oracles, and where both the size of the group's public key as well as the computational effort for an identification are independent of the number of users in the group. For convenience we only present the protocol

based on our stateless signature scheme; it is easy to adapt it to the case of stateful signatures.

Unfortunately, our protocol is linkable in the sense that a verifier is able to decide if two identifications have been carried out by the same user (although the verifier is unable to specify the user among the group members). An obvious countermeasure against this problem is to frequently refresh one's membership and receive new keys (as in the case of pseudonyms).

Also note that the group manager is able to identify on behalf of any user (besides the fact that the manager can issue keys for fake users). Still, our protocol enjoys other strong security characteristics: it is for instance secure against any number of users that actively coorperate to intrude as another honest user; details follow.

Several anonymous group identification schemes (which can be derived for example from group signature schemes) have been constructed in the past, e.g., [12, 7, 1, 14], each with different security and performance features. Our solution seems to exceed all these protocols in performance, but at the cost of unlinkability.

4.1 Description

The group manager in our anonymous identification scheme picks an RSA modulus $n = pq$ of strong primes $p = 2p' + 1, q = 2q' + 1$, and a random element $x \in \mathrm{QR}_n$ together with a generator h_1 of QR_n. The values (n, x, h_1) make up the group's public key. If a user u wants to join, then the manager picks a random $(l+1)$-bit prime e_u and a random l-bit value α_u, and computes y_u such that $h_1^{\alpha_u} y_u^{e_u} = x \bmod n$. The manager hands the pair (α_u, y_u) and e_u to the user.[2]

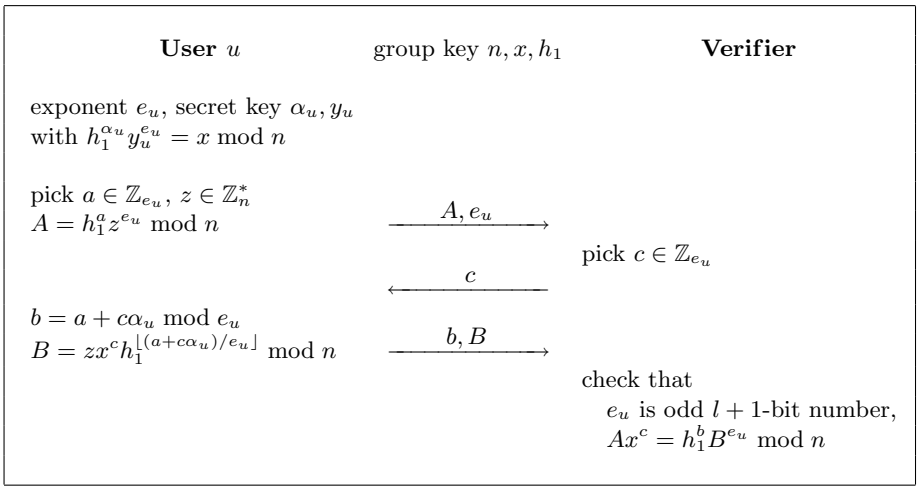
Next, we describe the identification protocol; it is also depicted in Figure 1. If a user u wants to identify as a group member to some verifier, both parties run Okamoto's RSA identification protocol [16] on the user's key and the group's public key. That is, the user picks $a \in \mathbb{Z}_{e_u}$, $z \in \mathbb{Z}_n^*$ in order to calculate $A = h_1^a z^{e_u} \bmod n$ and sends this value A with e_u to the verifier.[3] The verifier answers with a random challenge $c \in \mathbb{Z}_{e_u}$ and the user conclusively transmits b, B where $b = a + c\alpha_u \bmod e_u$ and $B = z x^c h_1^{\lfloor (a+c\alpha_u)/e_u \rfloor} \bmod n$. The verifier checks that e_u is an odd $l+1$-bit number and the correctness condition $Ax^c = h_1^b B^{e_u} \bmod n$ of the identification protocol.

Note that we can add "threshold admittance levels" to our identification protocol almost for free. That is, each user u is assigned a privilege number ℓ_u

[2] For ease of notation we switch to a "well-formed" representation problem as explained at the end of Section 2.2. Also for simplicity we have chosen the version with "large" $(l+1)$-bit primes e_u. The protocol can be easily adapted to work with shorter primes e_u and powers $e_u^{t_u} \geq 2^l - 1$ instead.

[3] Okamoto's protocol does not require to send the exponent e_u as the exponent is already part of the public key. Here, the group's public key does not contain the users' exponents, so we let the user append it to the protocol data. Indeed, this is what makes our protocol linkable.

Fig. 1. Anonymous Identification Protocol



and this user is only allowed to enter (by means of identification) level ℓ areas for $\ell_u \geq \ell$. This feature is easy to accomplish in our scheme by demanding that, in order to enter level ℓ, the user u must identify with respect to an $(l+1+\ell)$-bit (or larger) number e_u, and by letting the group manager distribute corresponding exponents to the users when joining.

4.2 Security

Basically, our identification protocol inherits security from our signature scheme. Think of the group manager giving each new user u a signature for random message α_u. Note that this message α_u is chosen by the group manager, i.e., this setting corresponds to a random-message attack. Therefore, we do not need a trapdoor commitment nor a random splitting.

If some malicious user u^*, either a member or not, successfully identifies as another member using an exponent e_u of an honest user u, then, by the proof-of-knowledge property of Okamoto's scheme, we can extract a representation (α^*, y^*) of x with respect to e_u from this identification attempt. As Okamoto's identification is witness-independent, we have $\alpha_u \neq \alpha^*$ with probability $1 - 2^{-l}$ for the user's secret key (α_u, y_u). In this case, party u^* thus forges a signature of a new message α^* which is infeasible under the RSA assumption. Similarly, if u^* chooses a new e_{u^*} and successfully proves membership, we obtain a successful signature forgery for message α^* for this e_{u^*}, contradicting the strong RSA assumption.

We remark that security even holds with respect to any adversary that controls all corrupted users and who may adaptively decide to join further malicious

users, to corrupt existing parties, and to run protocols with the honest users before trying to intrude. Using techniques developed in [3], one can even extend it to the case that some dishonest user u^* tries to intrude in the name of a user u while executing the identification protocol with that user u (in the presence of so-called session IDs).

Acknowledgments

We thank Ronald Cramer and Victor Shoup for comments.

References

1. G. Ateniese, J. Camenisch, M. Joye, G. Tsudik: A Practical and Provably Secure Coalition-Resistant Group Signature Scheme, *Advances in Cryptology—Crypto 2000, Lecture Notes in Computer Science, Vol. 1880, pp. 255–270, Springer-Verlag*, 2000.
2. N. Barić, B. Pfitzmann: Collision-free Accumulators and Fail-Stop Signature Schemes Without Trees, *Advances in Cryptology—Eurocrypt '97, Lecture Notes in Computer Science, Vol. 1233, pp. 480–495, Springer-Verlag*, 1997.
3. M. Bellare, M. Fischlin, S. Goldwasser, S. Micali: Identification Protocols Secure Against Reset Attacks, *Advances in Cryptology—Eurocrypt 2001, Lecture Notes in Computer Science, Vol. 2045, pp. 495–511, Springer-Verlag*, 2001.
4. M. Bellare, P. Rogaway: The Exact Security of Digital Signatures — How to Sign with RSA and Rabin, *Advances in Cryptology—Eurocrypt '96, Lecture Notes in Computer Science, Vol. 1070, pp. 399–416, Springer-Verlag*, 1996.
5. D. Bleichenbacher: Efficiency and Security of Cryptosystems Based on Number Theory, *Ph.D. thesis, Swiss Federal Institute of Technology, Zürich*, 1996.
6. D. Boneh: Twenty Years of Attacks on the RSA Cryptosystem, *Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203–213*, 1999.
7. D. Boneh, M. Franklin: Anonymous Authentication with Subset Queries, *Proceedings of the 6th ACM Conference on Computer and Communication Security, pp. 113-119*, 1999.
8. R. Cramer, I. Damgård, T. Pedersen: Efficient and Provable Security Amplification, *CWI Reports, Computer Science, CS-R9529*, 1995.
9. R. Cramer, I. Damgård, B. Schoenmakers: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols, *Advances in Cryptology—Crypto'94, Lecture Notes in Computer Science, Vol. 839, pp. 174–187, Springer-Verlag*, 1994.
10. R. Cramer, V. Shoup: Signature Schemes Based on the Strong RSA Assumption, *ACM Transactions on Information and System Security (ACM TISSEC), 3(3), pp. 161–185*, 2000.
11. I. Damgård, M. Koprowski: Generic Lower Bounds for Root Extraction and Signature Schemes in General Groups, *Advances in Cryptology—Eurocrypt 2002, Lecture Notes in Computer Science, Springer-Verlag*, 2002.
12. A. De Santis, G. Di Crescenzo, G. Persiano: Communication-Efficient Anonymous Group Identification, *Proceedings of the 5th ACM Conference on Computer and Communication Security, pp. 73-82*, 1998.

13. E. Fujisaki, T. Okamoto: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations, *Advances in Cryptology—Crypto '97, Lecture Notes in Computer Science, vol. 1294, pp. 16–30, Springer Verlag*, 1997.
14. C. Lee, X. Deng, H. Zhu: Desing and Security Analysis of Anonymous Group Identifcation Protocols, *Public Key Cryptography (PKC) 2002, Lecture Notes in Computer Science, Springer-Verlag*, 2002.
15. U. Maurer: Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters, *Journal of Cryptology, vol. 8, pp. 123–155, Springer-Verlag*, 1995.
16. T. Okamoto: Provable Secure and Practical Identification Schemes and Corresponding Signature Schemes, *Advances in Cryptology—Crypto '92, Lecture Notes in Computer Science, vol. 740, pp. 31–53, Springer Verlag*, 1993.
17. RSA Cryptography Standard: PKCS #1 v2.1, *available at* `www.rsa.security.com/rsalabs/pkcs`, June 2002.
18. T. Schweinberger, V. Shoup: ACE — The Advanced Cryptographic Engine, *available at* `www.shoup.net`, August 2002.