ROUGH DRAFT
Protocols for Use in the New Identification System

Chris Laas

June 42, 2000

`$Id: protocols.tex,v 1.7 2000/08/16 13:49:23 golem Exp $`

1 Introduction

This paper attempts to document the protocol design options available to the project. In particular, the three primary mechanisms — individual identification, group authentication, and digital cash — are addressed, along with key management for the CA hierarchy. For individual identification and digital cash, the likely most efficient and secure protocols can be taken directly from the literature (and examples are listed); in these cases, the protocols presented are to be taken as examples or placeholders. Because they are constructed from general cryptographic primitives rather than more low-level primitives, they are simpler to understand and initially implement; however, as the system matures, it will probably be necessary to replace these systems with specialized systems. On the other hand, the group authentication mechanism presented here is constructed from low-level primitives such as modular exponentiation; it is based heavily on the system presented in [OOK90], and derives its security from the RSA assumption. The extensions to the system are based on simple, well-known cryptographic techniques, and are provably secure (although care must be taken to audit the system for flawed assumptions, as always). The description of the CA hierarchy is straightforward: a few points are clarified, and some contingency plans laid out.

In general, the system as presented attempts to preserve hardware agnosticism, because this allows greater flexibility to users, greater adaptibility to change, and in general is a good design principle. However, it is usually assumed that hardware offers reasonably decent cryptographic accelerators (providing modular multiplication/exponentiation and block cipher functionality) but limited memory. The computation and memory requirements of each protocol are set out, along with parameters which can be tweaked to trade off security and efficiency. Terminals are sometimes assumed to have fast and reliable network connections, although allowances are made to terminals without such connections (with caveats, such as limited revocation functionality).

This document is by no means to be taken as a "hard" committment to a particular design; rather, it exists to lay out various paths that may be taken by the implementors. Boxed material, in particular, is parenthetical text intended to describe an alternative way of implementing a particular aspect of the system.

2 Protocols for Individual Identification

2.1 Overview

The individual identification mechanism is the simplest to implement: there are many systems in the literature which can perform this function in a plug-and-play manner. The requirements for this subsystem are as follows: an individual can acquire any number of identity certificates from any number of CAs. If the individual remains anonymous during the issuing process (or the issuing process is blinded), the certificate serves as a pseudonym; on the other hand, if the CA stipulates within the signed certificate that it is willing to vouch for the holder's identity, the certificate serves as a "hard" identity (i.e. is associated with a real person or entity). In most respects, these two types can be treated similarly, although different applications will wish to use different levels of pseudonymity. An identity must support the following operations:

- Signature on a message (and, of course, verification of such signatures).
- Proof of knowledge of the secret key of the certificate.
- Optionally, a trusted-third-party mechanism for keeping links between identities (i.e. pseudonyms and real names) in escrow. While this could be useful in future applications (such as a secure library system), it is not currently vital.

These primitive operations will be used to build applications.

Proof Modes

A proof of identity is a protocol in which a prover demonstrates knowledge of the secret key corresponding to a given public key; the public key is usually contained in a certificate signed by a CA. There exists a distinction between *zero-knowledge proofs (ZKPs) of identity* and *signatures of identity*: in a ZKP, the verifier learns no information which he could not have computed himself, whereas the opposite is true in a signature scheme. When a proof of knowledge is used as a building block of another protocol, often one or the other property is vital; in the proofs of identity we will use, either mode could be used.

An advantage of the ZKP mode is that it leaves no transcript which could serve as an unforgeable proof that a transaction occurred; on the other hand, the transcript from a signature of identity could be used to convince an outside third party that the transaction occurred. This property does not seem vital for the applications currently under consideration, but could become so in the future.

In some situations a ZKP can be considered more secure than a signed proof, because a collection of transcripts cannot help an attacker break the system; however, this is merely one factor out of many. The signed proof has the advantage that one can be contructed from any signature scheme, and the existence of a signature scheme is required by this setup anyway. On the other hand, several schemes which support both signed and zero-knowledge modes exist.

As mentioned above, there exist several systems which fulfill the above properties in the literature; many proofs of knowledge can be used to build such systems. For example, the system of Fiat and Shamir [FS86] is elegant and efficient and has withstood the tests of time. It may be advantageous to include more than one implementation: this may allow further tradeoffs between complexity, efficiency, and functionality.

Fiat-Shamir and related protocols
The problem of identification is an old one in cryptography, and many systems have been designed to solve it, some more successfully than others. A system based on cryptographic primitives, like the one described below, is a conservative option; there also exists the option of basing an identification system on almost any proof of knowledge (zero-knowledge or otherwise). For example, the basic Fiat-Shamir system [FS86] is generally much more computationally efficient than the system presented below, can be constructed with similar memory efficiency, and allows both zero-knowledge and signed proof modes. (However, the original scheme also allows the common mafia fraud and other frauds; see [DGB87], [BD90].) There exist innumerable extensions of the basic Fiat-Shamir scheme, with varied success (see [OO88], [MS88], [GQ88], [BDPW89], [OS90], [Nac92], [Oka92], and these don't even touch upon discrete log or NP-complete problem based schemes); all in all, though, since the requirements of the identification subsystem are so basic, many systems will be acceptable.

For purposes of illustration, a simple subsystem fulfilling the basic objectives, using nothing more than basic cryptographic primitives, is presented below.

2.2 System initialization

Before system initialization time, the following algorithms are agreed upon:

- A public key cryptosystem A (for individuals) which provides signature functions (e.g. RSA, ElGamal, DSA, Fiat-Shamir, etc.). The functions $Sign_{sk_A}$ and $Verify_{pk_A}$ are supported.
- A public key cryptosystem C (for the CA) which provides signature and encryption functions (e.g. RSA, ElGamal, DSA+RSA, etc.). The functions $Sign_{sk_C}$, $Verifypk_C$, $Encrypt_{pk_C}$, and $Decrypt_{sk_C}$ are supported. (This cryptosystem need not be the same as the one chosen above, but RSA would be a natural choice for both.)
- A collision-intractable hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k$ (e.g. SHA).

At system initialization time, the CA generates a key pair (sk_c, pk_c) for the cryptosystem C. In addition, each CA is given a unique identifier CA;

this may be simply a descriptive string (e.g. "ATHENA.MIT.EDU") or a hash thereof.[1]

Programmed into each (non-CA) device at initialization are the following parameters:

- $Sign_{sk_A}$, $Verify_{pk_A}$
- $Verify_{pk_C}$, $Encrypt_{pk_C}$
- $\mathcal{H}$
- pk_c

pk_c may be updated (in the long term) by a key management protocol external to this section.

2.3 Registration with a CA

When an individual registers with a CA, he is assigned a unique identifier I. If the CA vouches for that individual's identity, the identifier may be a hash of that person's name and/or MIT ID, SSN, Athena username, etc. (or if the entity is not a person, a host name, organization name, etc.); otherwise, it may be a serial number or random number. The only system requirement is that the identifier be unique (among the identifiers issued by that CA) for all time, with overwhelming probability.

The identifier I is stored in the CA's database, along with name, any auxiliary information, and starting and expiration dates.

2.4 Certificate issuing protocol

A new identity is immediately issued a certificate; and, an individual in good standing with a CA may apply for a certificate renewal. In either case, the protocol is the same: the individual Alice with identity I contacts the CA Carol, and perform the following steps.

1. Carol verifies that Alice is indeed the individual associated with the identity I. This may be by a proof of ownership of the previous certificate issued to I[2], a proof of identity associated with another CA which this CA trusts (possibly useful for secondary CAs), or an out-of-band

[1] Although the authenticity of the CA's public key is treated as axiomatic here, this requirement will be relaxed in a later section on the CA authentication tree.

[2] This is a dangerous mechanism, since it does not limit the damage due to compromise, but possibly useful in some cases.

mechanism such as physical presentation of picture ID (the most likely for the primary identity CAs). Carol also may verify that a minimum time has passed since the last certificate issued to I; this might help prevent attacks requiring large numbers of signatures generated by the CA.

2. Alice generates a key pair (sk_a, pk_a). If the CA demands revocation ability (a policy decision, but in most cases it should), Alice generates a message

$$m_b = (\text{“bill of bad health”}; CA; I; pk_a)$$

signs it with sk_a, encrypts it with pk_c, and sends the signed, encrypted message $E_b = Encrypt_{pk_c}(Sign_{sk_a}(m_b))$ to Carol. [3] Otherwise, Alice merely sends $Sign_{sk_a}(CA; I; pk_a)$ to Carol.

3. Carol verifies the signature on the bill of bad health $b = Decrypt_{sk_c}(E_b)$, if it demanded one; else, it verifies the signature on the public key. It also verifies that the identity I has never used the public key pk_a before.[4] If it accepts, it generates a certificate

$$c = Sign_{sk_c}(I; \textit{start-time}; \textit{renew-time}; \textit{expire-time}; \textit{aux-data}; pk_a)$$

and sends it to Alice. The distinction between *start-time*, *renew-time*, and *expire-time* are clarified in [Riv98]; it is fairly self-evident. The auxiliary data may contain restrictions on use of the certificate; or, it may contain additional identifying information such as a “real name”, MIT ID, SSN, etc.

Carol also files the item (c, b) in its database entry for the identity I, retaining all the previous items, of course. [5]

4. Alice verifies the CA's signature on the certificate. If Alice complains, Carol should be willing to re-send the exact same certificate as before, or Carol may be willing to begin the process again from step one. In the latter case, Carol should enforce a set of timeouts and retry limits

[3] The “bill of bad health” is an idea taken from Rivest[Riv98].

[4] Much preferably, it would verify that *no* identity had ever used that public key before.

[5] If the database is stored less securely than sk_c, it may make sense to store $(c, Encrypt_{pk_c}(b)) = (c, E_b)$ instead of (c, b), since this would prevent an adversary with access to the database from issuing fraudulent bills of bad health. Of course, public-key encrypting b is not necessary; a symmetric cipher may be used if the key is kept as securely as sk_c.

so that Alice cannot obtain many signatures. (Of course, when face-to-face verification is used, these restrictions are mostly irrelevant.)

2.5 Certificate showing protocol

To prove her identity I registered with a CA CA to a verifier Victor, Alice engages in the following protocol with Victor:

1. Victor generates a nonce r and sends it to Alice. The nonce may be random, pseudorandom, or may merely be a serial transaction number.[6]

2. Alice hashes the nonce, computing $\mathcal{H}(r)$, and generates a nonce of her own, s, using any of the above techniques. Alice then generates the signed message

$$Sign_{sk_a}(\text{“showing reply”}; \mathcal{H}(r); s; \textit{Victor-ID}; c)$$

   and sends it to Victor.[7]

3. Victor verifies the CA's signature in the certificate c (using the public key pk_c stored locally by Victor), verifies that the certificate is not past its expiration date (if Victor is paranoid, he may refuse to accept a certificate past its suggested renewal date), and then verifies Alice's signature on the message (using the public key pk_a found in the certificate). Additionally, if Victor has a network connection, he may choose to consult the "suicide bureau" to check for a revocation message (bill of bad health, or suicide note) for the public key pk_a. (If Victor is cooperating to keep Alice's identity private, he may need to use a different mechanism for this; for example, he could set himself up as a branch of the suicide bureau and consult himself in this step.) If all of these tests pass, Victor accepts.

[6] If a serial number is used, *Victor-ID* may not be omitted in the next step; for Victor's security, it is necessary that *Victor-ID* be unique to Victor.

[7] Victor-ID is omitted if, for some reason, Alice does not know Victor's identifier. Victor often will have identified himself in a previous stage of the protocol. Also note that, if the signing procedure does not incorporate a hash on the input message, the message body here should be explicitly hashed. In general, this protocol should follow the principle "hash along any value which may be relevant, just in case".

2.6 Generation of signed messages

This system trivially provides signature functionality. To sign a message m with a signature "by identity I", Alice generates the message $Sign_{sk_a}(m; c)$ and distributes that; anyone can verify the CA's signature in c, and then verify Alice's signature on the message.[8]

2.7 Revocation of certificates

Two-level expiration dates (and possibly daily-renewed secondary certificates) should suffice for most revocation purposes; however, if speedy "short-circuit" revocation is needed, and the relevant verifiers are all connected via a high-speed, reliable network, then a CRL-like revocation mechanism can be employed. As in [Riv98], a "suicide bureau" (or possibly several such) is established. In the case of key compromise, Alice may send a self-signed "suicide note" of the form

$$Sign_{pk_a}(\text{“suicide note”}; pk_a)$$

to a branch of the suicide bureau, which promises to deliver the note to any party which queries about pk_a, and also to distribute the note to all other branches of the suicide bureau. Any party which encounters a suicide note may forward it to the suicide bureau; and any party which sees a suicide note (and verifies it) may consider the associated key to be compromised (and hence revoked). (Alice may wish to keep a copy of a suicide note in a safe place such as a floppy diskette, in case she loses the secret key to e.g. a thief.)

Additionally, the "bill of bad health" escrowed by the CA during the issuing process can be used by the CA to revoke the certificate. It should function (with respect to the suicide bureau and any verifiers) identically to a suicide note; the distinction is necessary, however, because identifying the source of and reason for the revocation is important for security purposes.

[8] Of course, the signature cannot be considered valid if the certificate has been revoked; and the signature is certainly suspect if the certificate is past its expiration date, though how to deal with this depends on the application.

CRLs
The "bill of bad health" is similar to a CRL, and functions much the same way in this system. However, note that as described above, the CA never signs the bill, and hence if anyone receives a bill of bad health, he knows that *either* the user of that key pair *or* that CA has released it to the suicide bureau. In other words, one must not assume that the CA is the broadcaster of a bill of bad health. If the CA must sign the bill of bad health for it to be considered valid, this property is no longer true. This has dubious advantage, and is somewhat less efficient; in addition, it requires the suicide bureau to know something about the CA structure, whereas in the above system the suicide bureau simply (and elegantly) accepts any self-signed suicide message (with a small amount of auxiliary data such as CA identifier for a bill of bad health). On the other hand, it seems possible that a simple suicide bureau as described above could be susceptible to a denial of service attack, if its mass storage for suicide notes could be overflowed.
It's also worth noting that a system using CA-signed bills of bad health is extremely similar to a CRL system; why not simply use CRLs? It would be a viable option; the primary difference would seem to be that CRLs lack the simplicity and generality of self-signed suicide notes. A suicide bureau needs only to accept self-signed messages with small payloads, whereas a CRL forwarder needs to know about the CA hierarchy.

2.8 Efficiency analysis

Since no protocol used is unusual or expensive, there are no anticipated problems. The issuing process costs Alice and Carol one signature and one verification each, plus one encryption for Alice and one decryption for Carol, plus any overhead associated with establishing an encrypted channel, and with initially authenticating Alice. The showing protocol costs Alice one signing operation and Victor two verifications; it may be desirable to encrypt the exchange to prevent eavesdroppers from learning plaintext-signature pairs in order to mount a cryptographic attack, but encryption is not necessary to maintain message integrity. If Victor contacts the suicide bureau and retrieves a suicide note, he will have to perform an additional verification. Revocation costs no cryptographic operations initially, although it requires a high-speed reliable network.

The storage requirements of a single certificate are fairly minimal, and are dominated by the public and secret key pair. The CA must store all certificates "in perpetuity," so as not to reissue a new certificate on a previously-used public key; however, these memory requirements can be minimized, and the database can be optimized for memory usage if the cost ever becomes high.

2.9 Possible improvements

This simple system is meant more for demonstration than practical purposes, but would suffice in a real situation. One particularly dangerous case in this system is that of private-key compromise (or of intentional private key "sharing"), since an individually-held certificate is the ultimate source of an individual's authorizations. Some ideas to help combat this may be found in [GPR97]. In particular, the simple idea of using short-lived certificates for services and long-lived certificates only for obtaining short-lived certificates may well find application here; the long-lived certificates need not even reside in the token, if a more secure storage area can be found. This approach is all the more attractive given the "daily-renewed" certificates employed in the group mechanism described below.

3 Key Management Protocols for the CA Tree

3.1 Overview

Since all authority in this system flows from the CAs, some special care must be taken to ensure that their public keys are authentic and can be updated when necessary.[9]

In outline, the structure of the CA hierarchy is as follows. Each large organization (for example, MIT) maintains a largely independent CA tree. At the root of the tree is that organization's Root CA, which is largely embodied as a rarely-used, high-security, infinitely-lived key pair. This key pair is used to certify that organization's Master CA public key; the Master CA's key pair is renewed infrequently but regularly, on the order of every year to every decade. The Master CA behaves almost exactly as an "individual-certifying" CA (see above section), but the entities registered to

[9] A system with static CA keys would not be able to update the keys (and possibly key sizes) periodically to protect against cryptographic attacks which require many ciphertexts, and would also be lain prostrate before a CA key compromise. Even when security parameters are chosen so that cryptographic key compromise is infeasible, there is always the possibility of machine or network security compromise or insider fraud.

the Master CA are only the various administrative CAs: human-identifying CAs, verifier-identifying CAs, workstation-identifying CAs, group membership CAs, and bank CAs. The interactive showing protocol defined above will be largely irrelevant here, but the signature protocols will be vital. Also, the public key cryptosystems used by CAs may differ from that used by e.g. humans or verifiers; a human-identifying CA's key pair must be usable for both signature and encryption functions, for example, and the group membership and bank CAs' public and secret keys will have highly specialized functions.[10]

Organizations may cross-certify in two ways. If one organization decides to trust in all of another organization's CAs, it may certify that organization's Master CA (not Root CA) using its Master CA key pair (not its Root CA key pair); this will create certification paths from one organization to the other. Alternatively, an organization may simply choose to trust a subset of another organization's CAs by certifying those CAs with its Master CA key pair, just as it would its own CAs. (The auxiliary data should contain indiciation of "ownership" or "affiliation", however.)

In summary, the Root CAs are the ultimate source of authority for the members of the organizations; the Master CAs are the active, effective sources of authority, and organizations cross-certify their Master CAs. Master CAs certify regular CAs, which perform the real work of the system.

4 Protocols for Group Identification: Base System

4.1 Overview

This section describes the "base system" for the group authentication subsystem. It lacks the complex mechanism employed by the full group authentication subsystem to provide "short-circuit" revocation of group membership, and so is significantly easier to understand, but its security is based on the same principles as the full mechanism; the subsystem is heavily based on [CT89] and [OOK90]. Despite its simplicity compared to the full system (described in the next section), the base system could serve in a production system: the primary disadvantage is that all revocations would have to wait

[10] The astute reader will note that, as described above, the identification protocols do not provide for such divergent key-pair purposes. Although the sentiment is a healthy one, and care must be taken to examine each aspect of this subsystem, ultimately the Master CA can sign pretty much any arbitrary piece of data as a public key of a sub-CA. In fact, even the system as described above could be used, since the auxiliary information's form is not strictly defined, and hence could be used to certify CA public key material.

until the next renewal cycle to take effect, and thus renewal cycles would likely have to be on the order of a day, and almost certainly no more than a week.

There may be multiple CAs for the group subsystem, each administrating an independent group structure. Verifier terminals (such as door locks) may be fairly simple devices with network connections, simple crypto processors, and a small amount of local NVRAM. For each CA, each user token need at minimum store only two large values, totaling 256 bytes (assuming 1024 bit moduli), plus a certain (variable) amount of group specification information, which will likely be of the same order of magnitude of size.

4.2 System initialization

Before system initialization time, the following algorithms are agreed upon:

- A collision-intractable hash function $\mathcal{H} : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ (e.g. SHA) [11]

At system initialization time, the CA generates the following values:

- Large secret primes p, q
- The CA public key $n = pq$
- A random initial public key $y_0 \in_{\mathcal{R}} \mathbb{Z}_n^*$

The CA also decides to trust at least one identification-vouching CA (see previous section above), and obtains that CA's public key pk_i (via some key management protocol, or simply via an out-of-band mechanism).

Programmed into each device at initialization are the following parameters:

- $\mathcal{H}$
- n
- y_0

n and y_0 may be updated (in the long term) by a key management protocol external to this section.

Registration of individuals with this CA is considered to be implicit in registration with any identification-vouching CA trusted by this CA. It is assumed that individuals can prove their identity and that each has a unique identifier I, with respect to some identification-vouching CA.

[11] Technically, since this depends on n, $\mathcal{H}$ is really chosen at system initialization time after n is chosen. However, this is a minor detail.

4.3 Group creation and destruction

To create a group, a party submits a signed request to the CA containing the following information:

- Proof of authorization to create a group. There are two possibilities: either he is a member of a privileged group-creators group, in which case the party signs with his individual key, or he is an external party such as a Kerberos principal, in which case he authenticates in a CA-specific way external to this specification.
- The group's owner (which may be an individual, another group, or an external party)
- A list of initial members (individuals or groups)
- A list of verifiers ("interested parties", such as door locks)
- Any auxiliary information

The CA checks the signature and the authorization status of the signator. If it accepts, the CA generates a unique group identifier G and a unique small prime p_G, which it stores along with the creation request.[12]

Any verifiers interested in checking membership in G should be given the values G and p_G (as well as the public values n and y_0). In this system, it is possible for verifiers to operate offline, so long as they periodically check with the CA to verify that the group still exists, the group prime has not changed, and the system parameters have not changed. These checks could be performed out-of-band, by a system administrator.

To destroy a group, the owner or a privileged individual submits an individually signed destruction request, and the CA marks the group as destroyed. The CA may remove the group from the active database and put the records in cold storage, but the group identifier G should never be reused. The group prime p_G may only be reused after all verifiers previously relying on it have been updated.

[12] As [CT89] points out, it is possible to get away with using fewer primes when groups often have subgroups, since one of the subgroups may use the same prime as the supergroup. This technique may be implemented with minor changes to the protocols. However, using it may cause the subgroup structure to become rigid, or could increase the communications overhead of showing protocols, and so care must be taken to weigh the pros and cons.

4.4 Addition and removal of members

To add or remove a member, a group owner simply submits an individually signed request to the CA, which updates the database.[13] The affected individual may be notified by the group owner, the CA, or by some other out-of-band means. No further action need be taken by the CA until the individual requests certificates.

There is no CRL mechanism to allow revocations to take effect before the current batch of "day" certificates expires.[14]

4.5 Daily key renewal

The CA public key y_k is renewed daily (or weekly, or at some other convenient period). On the morning of day k, a CA computes $T_k = \prod_G p_G \mod \phi(n)$ (using all groups G which remain active on day k). All devices independently compute the value $y_k = \mathcal{H}(y_{k-1})$. The CA, which knows the factorization of n, additionally computes $x_k = y_k^{1/T} \mod n$; x_k is the master secret key for the day.

4.6 Group membership certificate issuing protocol

To obtain group certificates for day k from a given CA Carol, an individual Alice (with identity I with respect to the identification-vouching CA whose public key is pk_i) establishes an encrypted, authenticated channel with Carol, and requests certificates. Carol looks up the list of groups Γ of which Alice is a member, and computes the value

$$v_{Ik} = \prod_{G \in \Gamma} p_G$$

Alice's key for the day is defined as

$$z_{Ik} = y_k^{1/v_{Ik}} \mod n$$

Carol sends Alice the values (v_{Ik}, z_{Ik}); Alice accepts if the relation

$$y_k = z_{Ik}^{v_{Ik}} \mod n$$

[13] Although it is a matter of policy, there seems to be no reason why group owners should be permitted to retain anonymity in this process; and for auditing purposes, it may be desirable for these requests to be individually signed.

[14] This deficiency is addressed in the next section, although there remain some thorny issues with respect to the interaction of privacy and revocation.

holds. (It is important that Alice use her internally-computed value of y_k here, because this prevents Carol from cheating and attempting to distinguish members of a group by giving them different values of y_k.) If Alice's token explicitly knows Γ and $\{p_G | G \in \Gamma\}$ (which would be advisable if space allows), it should check the value for v_{Ik}, as well.

It is worth noting that if Alice trusts any piece of hardware which is usually or always on the net, she can use it as a proxy to retrieve her certificate and hold it until she requests it. This benefits Alice because the CA cannot even track her by her initial logon to the network to retrieve certificates, enhancing her privacy; this benefits the CA because it allows the CA to smooth out peak demand periods, keeping response times short. The primary downside is the requirement that Alice trust a piece of hardware on the network, which would likely be a workstation of some sort; workstations which are continuously on the net are not very secure.

4.7 Interactive proof of membership

To prove membership in the group G, Alice computes the group secret key $w_{Gk} = z_{Ik}^{v_{Ik}/p_G}$ (p_G may be stored by Alice's token, or may be provided as a challenge). Alice then uses w_{Gk} to perform a zero-knowledge proof of knowledge of a p_Gth root of y_k, modulo n.[15]

4.8 Efficiency analysis

4.8.1 Storage requirements

A CA must store a constant amount of key values (adding up to at most a few kilobytes), a small unique prime for each group, and a database of individuals and groups, which will take up by far the most space. Nevertheless, the storage requirements on a CA are very mild.

A verifier may have very little memory: it need only store the public modulus n, the day public key y_k, and its group prime p_G, along with any auxiliary information. (A verifier may be keyed to respond to more than

[15] There are several such in the literature. A basic such proof repeats the following three moves t times:

1. Prover chooses $r \in_{\mathcal{R}} \mathbb{Z}_n^*$, and sends $s = r^{p_G} \mod n$ to Verifier.
2. Verifier sends one bit $c \in_{\mathcal{R}} \{0,1\}$ to Prover.
3. Prover sends $a = rw_{Gk}^c \mod n$ to Verifier. Verifier accepts this round if $a^{p_G} = sy_k^c \pmod{n}$.

one group, but it should be encouraged to create a new group in the CA for this purpose.)

A user token need only store (at minimum) the public modulus n and the day public key y_k; it should also store the list of groups it holds membership in, as well as those primes. (A rough estimate indicates that a list of 50 groups could fit in one kilobit, if packed tightly.) In addition, ample extra temporary storage in the user token would permit preprocessing to speed up proofs of membership by an order of magnitude.

4.8.2 Computation requirements

A CA's primary load is the generation of new day certificates for users; however, with some simple optimizations, this can be a very light load. The CA can work on creating tomorrow's day certificates all day today and cache the results; hence, the computational burden on the CA will be very light in that respect. Establishing an authentic and encrypted channel with each token every morning will be the greatest burden, but it should not be difficult to handle a load of 30,000 users, especially if trusted proxies are used to spread out the peak-demand periods.

A verifier has low computation requirements; it need only play as the verifier in a proof of knowledge of an eth root modulo n. In the simple scheme described above, this requires generation of a small quantity of random bits (equal to the security parameter, e.g. 40 bits), and the ability to perform modular multiplication and exponentiation by a small exponent (e.g. 20 bits). Using the above parameters, the verifier would need to perform roughly $40 * 15 * 2 = 1200$ modular multiplications per verification; assuming negligible communication latency between token and verifier and the above parameters, this requires a coprocessor which can perform a modular multiplication in 833 μs, if the transaction is to be completed within one second. Assuming conservative parameters such as 1024-bit moduli and 40 trials, this may well be reasonable for a cheap compact device[16]; toning the security parameters (especially the number of trials) down slightly to compensate seems possible, though, and it may be possible to modify the proof algorithm to gain efficiency.

The token's primary computational requirement is acting as the prover in the proof of knowledge of an eth root modulo n. Although, in the given algorithm, the token must perform the same quantity of modular multiplications as the verifier, it may perform the great bulk of that work (computing

[16] A brief foray into chip card accelerators and microprocessor-based timings seems to indicate that 1 ms is a reasonable expectation for 1024-bit modular multiplication.

the commitments) as precomputation. Hence, assuming that all the possible precomputation has been performed, the online work load of the token will only be approximately 20 modular multiplications. This gives the token over 48 ms to compute each modular multiplication, which is no problem.

5 Protocols for Group Identification: Full System

5.1 Overview

By adding a few additional primitives and tightening the constraints on the system somewhat, it is possible to implement a "short-circuit" revocation mechanism. This will permit group owners to instantly revoke membership in a group (effective immediately at verifiers which are connected to the CA via a high-speed reliable network); the primary costs are a general increase in complexity and memory requirements, and an $O(r \log n)$ (where n is the size of the group and r is the number of members revoked) performance penalty during verifications due to short-circuit revocations.[17]

5.2 Interactive proof of membership in the union of several groups

The base system easily admits of a zero-knowledge proof of membership in the union of a set of group; this proof will form the basis of our revocation mechanism. Any challenge-response zero-knowledge proof of knowledge of a root would suffice as a primitive in this protocol. However, for concreteness, we will focus on the extended Fiat-Shamir protocol as the primitive proof of knowledge.

Let us say that Victor wishes to verify that Alice is in the set

$$\bigcup_{i=1}^{m} G_i$$

and let us assume that Alice is actually in the group G_j for some $1 \leq j \leq m$. (If Alice is in more than one of the candidate groups, she may pick one at random; ultimately, the protocol reveals no knowledge about which group Alice is a member of.) Alice computes the group secret key

$$w_{G_j k} = z_{Ik}^{v_{Ik}/p_{G_j}}$$

[17] "Obvious" and "more efficient" means to achieve this goal would initially seem to exist; however, all of these that we have considered suffer from a failure to protect against coalitions of more than one or two revoked members.

At this point, the interactive protocol begins. m independent parallel executions of a membership-proving protocol begin, one for each of the candidate groups; in the extended Fiat-Shamir scheme, Alice computes the commitments (each using the respective primes p_{G_j}). However, for each of the groups which is *not* G_j, Alice prepares to *cheat*: that is, she picks a random challenge bit $c_i \in \{0, 1\}$, $i \neq j$, and computes a "commitment" with which she can convince Victor assuming that his challenge is c_i in that instance of the protocol. For the group G_j, Alice computes the commitment in the usual way. Alice then sends the commitments to Victor (keeping j and all the c_i secret, of course).

Victor then returns a single challenge bit c. Alice computes her "real" challenge c_j such that

$$c = \bigoplus_{i=1}^{m} c_i$$

and generates the "real" response in instance j of the protocol. (She can do this because she knows the group secret key $w_{G_j k}$.) In every other instance of the protocol, she "fakes" a valid response; she can do this because she knew c_i already for $i \neq j$. Alice then sends all the c_i and all the responses to Victor. Victor verifies that

$$c = \bigoplus_{i=1}^{m} c_i$$

and verifies the validity of all of Alice's responses (given the challenges c_i). If all these tests pass, Victor accepts.

In more concrete terms, the protocol (using extended Fiat-Shamir) goes as follows. The following steps are repeated t times (where t is a security parameter):

1. For each $i \neq j$, Alice generates $c_i \in_{\mathcal{R}} \{0, 1\}$, and lets $c' = \bigoplus_i c_i$ (over $i \neq j$). Alice then generates $a_i \in_{\mathcal{R}} \mathbb{Z}_n^*$ for all $i \neq j$, and computes

   $$s_i = a_i^{p_{G_i}} y_k^{-c_i} \mod n$$

   as her "cheating" commitments. Alice also generates $r \in_{\mathcal{R}} \mathbb{Z}_n^*$ and computes $s_j = r^{p_{G_j}} \mod n$ as her "real" commitment. Alice sends all of the s_i to Victor.

2. Victor generates a random $c \in_{\mathcal{R}} \{0, 1\}$, and sends it to Alice.

3. Alice computes $c_j = c \oplus c'$ and generates

$$a_j = r w_{G_j k}^{c_j} \mod n$$

   Alice sends all of the c_i and a_i (for $1 \leq i \leq m$, including $i = j$) to Victor.

4. Victor checks that $c = \bigoplus_{i=1}^{m} c_i$. Victor then checks that $a_i^{p_{G_i}} = s_j y_k^{c_i} \pmod{n}$, for all i. If all these tests pass, Victor accepts.

It is easy to see that this protocol is zero-knowledge, including releasing no information as to which group Alice is actually a member of. The security of the protocol for Victor follows from the fact that if Alice can break this protocol (i.e. if she can convince Victor despite not knowing any of the specified roots of y_k), then Alice can break one of the constituent extended Fiat-Shamir protocols.

Finally, it's easy to see how this technique can be extended to use any challenge-response zero-knowledge proof as a primitive; hence, using a more efficient proof than extended Fiat-Shamir will not break this construction.

5.3 Converting the base system into the full system

Armed with this new protocol, it is at once clear how we can allow revocation within groups. We must first make a distinction between the "real" groups, externally visible to users, and the "virtual" groups, which will serve as the groups used by the base system. The virtual groups will be subsets of the real groups, and there will be many more virtual groups than real groups: there will be approximately $2n$ virtual groups to every real group, where n is the number of members in the virtual group. Hence, the number of virtual groups will be bounded above by two times the number of users in the system times the number of groups in the system, and simply specifying the forms of the groups will take up a good deal of memory. However, careful use of redundancy in the specifications of the group structure, and using reductions in the number of virtual groups (and unique primes for virtual groups) wherever possible, will allow these specifications to fit quite reasonably into small memories; experimental results seem to hold up this hypothesis.

Even if the number of users and groups spirals well beyond the anticipated levels, it will not be difficult to compact the group specifications to manageable levels. For example, if a group specification S_G grows large, user tokens will no longer be able to store S_G in their memories at all times.

However, the tokens may store a hash $\mathcal{H}(S_G)$ of this specification and "swap out" the specification to other servers. Then, when the specification is again needed, it can be retrieved, and the hash value checked. Such a system can fairly easily be constructed to preserve anonymity in this context, although in certain other contexts this is not possible.

So let us assume that the group structure can be efficiently specified and essentially stored in the user tokens. We will begin by considering each real group in turn. For each real group, we will assign a unique binary identification string to each member of the group, and construct a binary tree with the members as leaves. Now, each node in the binary tree represents a virtual group, consisting of all of the members who are descendents of this node.[18]

During normal operation, a user will prove membership in a group by proving knowledge of the root of y_k corresponding to the node at the root of that group's tree.[19] This is naturally equivalent to the base system. However, when it becomes necessary to short-circuit-revoke the membership of some users, we will need to express the "privileged" group P (the initial members of the group minus the revoked members) as a union of virtual groups. To do so, the following algorithm is used:

1. Begin by setting the set of priviledged nodes A to the set of leaves corresponding to users in P.

2. While any two nodes u, v in A share a common parent w, remove u and v from A and insert w into A.

At the end of this algorithm, A contains an optimal set of virtual group nodes; and it is easily shown that $|A| = O(\min(k, r \log n))$ where $k = |P|$, n is the size of the original group, and $r = n - k$.

Some additional restrictions will be necessary to attempt to ensure that the CA cannot violate group members' privacy by issuing bogus revocation requests; however, these will be fairly straightforward, relying on the group owner's trustworthiness rather than the central clearinghouse CA.

[18] A savings of almost 50% on the worst-case number of virtual groups can be made by noting that the lowest level of nodes is, of course, simply individual members; these may be given a single prime each, reducing the number of redundant virtual groups.

[19] Two different meanings of the word "root" are used in this sentence.

6 Protocols for Financial Transactions

This section is expected to contain two major subsections: a description of a "placeholder" transaction system (based on unforgeable online transfer orders), without privacy; and a careful weighing of the pros and cons of various privacy-protecting systems discussed in the literature. This will largely involve transcribing from notes and from other "satellite" papers; the strong-of-stomach way wish to read through /afs/sipb/user/golem/mitcard/cash/online-offline.tex, which consists of some considerations I've already transcribed to electronic form, although they are still in rough outline form.

References

[BD90] Thomas Beth and Yvo Desmedt. Identification tokens—or: Solving the chess grandmaster problem. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 169–176. Springer-Verlag, 1991, 11–15 August 1990.

[BDPW89] Mike V. D. Burmester, Yvo Desmedt, Fred Piper, and Michael Walker. A general zero-knowledge scheme. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 122–133. Springer-Verlag, 1990, 10–13 April 1989.

[CT89] Gerald C. Chick and Stafford E. Tavares. Flexible access control with master keys. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 316–322. Springer-Verlag, 1990, 20–24 August 1989.

[Dam90] I. B. Damgård, editor. *Advances in Cryptology—EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991, 21–24 May 1990.

[DGB87] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol (extended abstract). In Carl Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 21–39. Springer-Verlag, 1988, 16–20 August 1987.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[Gol88] S. Goldwasser, editor. *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 21–25 August 1988.

[GPR97] Oded Goldreich, Birgit Pfitzmann, and Ronald L. Rivest. Self-delegation with controlled propagation — or — what if you lose your laptop. *Theory of Cryptography Library*, September 1997.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In Christoph G. Günther, editor, *Advances in Cryptology—EUROCRYPT 88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer-Verlag, 25–27 May 1988.

[MS88] Silvio Micali and Adi Shamir. An improvement of the Fiat-Shamir identification and signature scheme. In Goldwasser [Gol88], pages 244–247.

[Nac92] David Naccache. A Montgomery-suitable Fiat-Shamir-like authentication scheme. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 488–491. Springer-Verlag, 24–28 May 1992.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1993, 16–20 August 1992.

[OO88] Kazuo Ohta and Tatsuaki Okamoto. A modification of the Fiat-Shamir scheme. In Goldwasser [Gol88], pages 232–243.

[OOK90] Kazuo Ohta, Tatsuaki Okamoto, and Kenji Koyama. Membership authentication for hierarchical multigroups using the extended Fiat-Shamir scheme. In Damgård [Dam90], pages 446–457.

[OS90] H. Ong and C. P. Schnorr. Fast signature generation with a Fiat Shamir–like scheme. In Damgård [Dam90], pages 432–440.

[Riv98] Ronald L. Rivest. Can we eliminate certificate revocation lists? In Rafael Hirschfeld, editor, *Financial Cryptography '98*, volume 1465 of *Lecture Notes in Computer Science*, pages 178–183. Springer-Verlag, February 1998.