Symmetric Key Authentication Services Revisited

Bruno Crispo Bogdan C. Popescu Andrew S. Tanenbaum
crispo@cs.vu.nl bpopescu@cs.vu.nl ast@cs.vu.nl

Department of Computer Science,
Vrije Universiteit, Amsterdam, The Netherlands

Abstract

Most of the symmetric key authentication schemes deployed today are based on principles introduced by Needham and Schroeder [17] more than twenty years ago. However, since then, the computing environment has evolved from a LAN-based client-server world to include new paradigms, including wide area networks, peer-to-peer networks, mobile ad-hoc networks and ubiquitous computing. Also, there are new threats, including viruses, worms and denial of service attacks.

In this paper we review existing symmetric key authentication protocols in the light of these changes, and propose a authentication infrastructure design specifically tailored to address the latest developments in the distributed computing landscape. The key element in our design is placing the authentication server off-line, which greatly strengthens the security of its cryptographic material and shields it from denial of service attacks. Although the authentication server is not accessible on-line, our scheme can handle a dynamic client population, as well as critical issues such as re-issuing of keys and revocation.

1 Introduction

Authentication is the foundation of most security services. The LAN-based, client-server-centric distributed computing environment of the mid 80's and early 90's was the golden age of authentication protocols based on symmetric key cryptography [17, 18, 15]. However, the distributed computing landscape has changed in the past few years: migration to wide area networks (WAN), peer to peer (P2P), mobile ad-hoc networks (MANET), and ubiquitous computing are just the major paradigm shifts. Authentication protocols based on public key cryptography are deemed to be better suited for this new environment, so recently they have been overshadowing the older symmetric key-based designs. Nevertheless, public key cryptography has its limitations: it is slower and requires larger keys than symmetric key cryptography, and involves CPU-intensive computations, which make it unsuitable for small, battery powered devices. Furthermore, developments in quantum computing may bring an end to some public key cryptosystems [20] (however, this is an unlikely scenario at least in the near future).

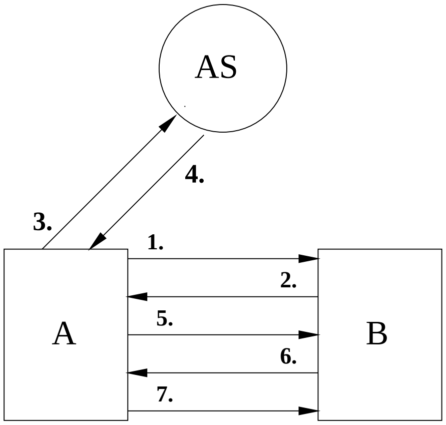
Given the fact that PKIs are by no means the "silver bullet" that solves all the problems related to authentication in distributed systems, it seems worth exploring whether protocols based on symmetric key encryption can be re-engineered to be made more secure and reliable and constitute a viable alternative in all the cases where authentication rather than non-repudiation is the requirement. In this paper we examine symmetric key authentication protocols in this new light, point out the limitations of current designs, and propose an authentication infrastructure which overcomes these limitations, and is better suited for the reshaped distributed computing environment. The pivotal point in our design is placing the trusted authentication authority off-line, which removes the vulnerabilities present in existing protocols, in particular their exposure to hacking and denial of service (DoS) attacks. Although clients can no longer directly access the authentication server, our infrastructure can still handle a dynamic client population, with the condition that the maximum size of this population is known in advance. To the best of our knowledge, this is the first symmetric key authentication infrastructure that is based on an *off-line* trusted third party (TTP) *and* supports a dynamic client population.

The rest of the paper is organized as follows: in Section 2 we give a brief overview of the foundations of symmetric key authentication protocols. Having placed our efforts in this context, in Section 3 we elaborate on the motivation for this paper, and Section 4 we look at related work, focusing on protocols that allow the authentication server to be placed off-line. Following this, in Section 5 we describe the proposed authentication infrastructure, in Section 6 we look at key update and revocation issues, and in Section 7 we briefly describe our prototype implementation and a number of performance measurements we have done on it. We conclude in Section 8.

2 Symmetric Key Authentication Protocols

Symmetric key authentication protocols can be divided in two categories depending upon how the freshness of key distribution messages is determined. One category uses challenge/response and nonces [17], the other one is based on timestamps [8]. Protocols using timestamps need fewer messages than the ones based on nonces [10], the downside being they require loosely synchronized clocks. On the other hand, protocols based on nonces require good random number generators and state storage, in order to prevent certain types of reflection and replay attacks. The protocols we introduce in this paper make use of both nonces and timestamps since we consider loosely synchronized clocks and good random number generators normally present in today's distributed systems.

Most symmetric key authentication protocols derive from the seminal work of Needham and Schroeder [17, 18]. As shown in Figure 1, the Needham-Schroeder protocol consists of the following messages [1]:



(1)	$A \longrightarrow B$:	A
(2)	$B \longrightarrow A$:	$\{A, N_{B_0}\}_{K_{BAS}}$
(3)	$A \longrightarrow AS$:	$A, B, N_A, \{A, N_{B_0}\}_{K_{BAS}}$
(4)	$AS \longrightarrow A$:	$\{A, B, N_A, K_{AB}\}_{K_{AAS}}$, $\{A, B, N_{B_0}, K_{AB}\}_{K_{BAS}}$
(5)	$A \longrightarrow B$:	$\{A, B, N_{B_0}, K_{AB}\}_{K_{BAS}}$
(6)	$B \longrightarrow A$:	$\{N_{B_1}\}_{K_{AB}}$
(7)	$A \longrightarrow B$:	$\{f(N_{B_1})\}_{K_{AB}}$

Figure 1: The Needham-Schroeder protocol

The goal of the protocol is to allow two principals, A and B, to authenticate each other and establish a secure communication channel. A trusted authentication server AS shares

[1] This is the version of the protocol that fixes the flaw described in [8]

a long term symmetric key with each principal and is capable of generating and sending "good" session keys on the request of these principals.

The first two messages are needed so that B can generate a nonce that will be included in the subsequent messages generated by the AS for that particular session. This avoids certain replay attacks using old, compromised session keys [8]. Once it has B's nonce, A requests a session key from the AS in message 3, also including her own nonce. The AS answers with two tickets - one for A and one for B - containing the session key (K_{AB}) and the nonces supplied by both parties (this guarantees freshness). B's ticket is then forwarded by A to B. Finally, messages 6 and 7 are used by B to ensure that A is online and thwart replay attacks. In Message 7, f is a previously agreed-upon secure hash function.

A large number of authentication schemes [15, 4, 12] have been designed based on the original Needham-Schroeder protocol. All of them require the AS to be on-line, since the two principals need to contact it at least for the first secure session they want to establish. This approach leads to a security infrastructure highly dependent on the AS. We will show that by separating the task of authentication from the one of generating and distributing session keys, it is possible to design authentication infrastructures that scale better, and are more efficient and reliable.

3 Motivation for a New Design

There are two main reasons why we believe symmetric key authentication services require an update: first, the state of the art implementations in this area [15, 4, 12] are based upon ten year old designs, with clear limitations. Second, the past few years have brought a number of major technological advances, but to our knowledge, no new symmetric key authentication technique based on these advances has yet been proposed.

3.1 New Developments in Distributed Computing

The distributed computing landscape has been more or less reshaped in the past years by a number of technological advances. Table 1 lists the major paradigm shifts, points out their consequences, and explains why existing symmetric key authentication infrastructures are not well suited to handle them.

New paradigm	**Consequences**	**Limitations of existing symmetric key authentication infrastructures**
Migration to WAN	Network latency and bandwidth display great variability. DoS attacks are much more frequent and harder to prevent.	They were designed assuming "almost" synchronous LAN communication. Possible DoS attacks were not directly addressed in their design.
Personal computing devices (i.e., PDA's, laptops, smart-phones, etc.)	Users possess powerful **personal** computing devices. Such devices can generate good random numbers and symmetric keys. They have enough memory to store millions of keys. Users do not share these devices.	They were designed for a world consisting of shared workstations, where the only piece of information a user could securely carry around was a short password.
Peer-to-peer	User to user interaction becomes a lot more frequent.	Needham-Schroeder schemes mostly deal with the client-server model.
MANET	User devices may be portable and equipped with wireless adapters. Continuous connectivity cannot be assumed in a wireless network environment	They require an on-line authentication server, reachable by all parties.

Table 1: New developments in the distributed computing landscape

3.2 Vulnerabilities in Traditional Symmetric Key Authentication Infrastructures

Existing symmetric key authentication infrastructures require the participation of the TTP not only during the authentication phase, but also for generating the session key.

Thus, the TTP is actively involved every time any two clients need to establish a secure connection. This leads to the following shortcomings:

- The AS is a **single point of failure** because when the AS is out of service users cannot independently establish a new secure session. This makes it a particularly attractive target for DoS attacks.
- The AS is a performance **bottleneck**, since all the users need to contact the server for each new session they want to establish.
- Session keys are generated and distributed by the AS upon request. This means the AS server must be on-line. As a consequence the AS is an **highly sensitive target** since compromising the AS would result in a possible compromise of all the subsequent private communications among all users registered with that particular AS. Furthermore key material is **continuously exposed** since the AS needs to be online.

Our authentication infrastructure overcomes these limitations, and at the same time is better suited for the re-shaped distributed computing landscape. The pivotal point in our design is placing the AS server off-line - this reduces the risk of compromising the AS's cryptographic material, shields it from DoS attacks, and makes the infrastructure more appropriate for environments such as MANET, where continuous network connectivity cannot be assumed.

4 Related Work

As we already mentioned, most of the existing symmetric key authentication infrastructures [15, 4, 12] suffer from the limitations pointed out in Section 3, which stem from the need for the AS to be always on-line. However, the idea of redefining the role of the AS by decoupling the initial authentication of principals from the subsequent use of their session keys is not completely new, and a number of protocols aiming at this have already been proposed. The Neuman-Stubblebine [19] and KSL [14] protocols are two examples. Both these protocols allow session keys generated in an initial exchange involving the AS to be re-used in subsequent sessions, which do not involve the AS. As a result, the load on the AS can be greatly reduced, which overcomes some of the limitations mentioned earlier. However, in both these protocols the emphasis is on the use of nonces versus timestamps for freshness purposes, rather than re-designing the role of the AS. Furthermore, both these protocols are vulnerable to the attack described in [11], due to the way they re-use old session keys in the repeated authentication part of the protocol. Another drawback of both these protocols is that they are asymmetric: although the keys generated for the first session can be reused, only the initiator of the first exchange can start subsequent sessions.

In [13] Kao and Chow propose a protocol allowing re-using of session keys that is resistant to the attack described in [11]. This protocol is also symmetric, and provides a better solution compared to the previous ones. However, it still requires the two clients to contact the AS server for the first secure session they want to establish; although the traffic towards the AS is greatly reduced, the server still needs to be on-line, and thus subject to the security threats mentioned earlier.

A similar solution has been proposed by Boyd [6, 7]. His protocol introduces a novel way to provide freshness by random input generated by users and a long term shared key distributed initially by the server. The protocol relies on the security property of *keyed hash functions* used as a basic primitive to generate fresh session keys. In detail, this protocol is as follows:

(1) $A \longrightarrow AS$: A, B

(2) $AS \longrightarrow A$: $\{A, B, K_S\}_{K_{AAS}}, \{A, B, K_S\}_{K_{BAS}}$

(3) $A \longrightarrow B$: $A, B, \{A, B, K_S\}_{K_{BAS}}, N_A$

(4) $B \longrightarrow A$: $[N_A]_{K_{AB}}, N_B$

(5) $A \longrightarrow B$: $[N_B]_{K_{AB}}$

where $K_{AB} = f(K_S, N_A, N_B)$ and f is an agreed-upon keyed hash function. $[M]_K$ is a transformation that only provides integrity (e.g. MAC). Once two clients run the above protocol, they can subsequently re-authenticate without contacting the server, by producing a new authenticated and fresh session key by completing the following protocol:

(1) A $\longrightarrow$ B: A, B, N'_A

(2) B $\longrightarrow$ A: $[N'_A]_{K'_{AB}}, N'_B$

(3) A $\longrightarrow$ B: $[N'_B]_{K'_{AB}}$

where $K'_{AB} = f(K_S, N'_A, N'_B)$. The fact that K'_{AB} depends on both N'_A and N'_B provides an association between message 2 and 3, thus preventing oracle session attacks [5]. The protocol is symmetric, since either A or B can initiate it. What it is still unsatisfactory here is that no specific expiration date is set for the long term secret K_S, leading to the possibility of cryptanalytic attacks. Despite this, [6] is the first to acknowledge that re-usable session keys do not come for free, since they require revocation mechanisms (to guard against possible key compromise), but does not propose any specific mechanism to address the revocation problem.

5 A Symmetric Key Authentication Framework based on Off-Line TTPs

We propose a symmetric key authentication framework based on an **off-line** TTP. Most importantly, our framework can accommodate a dynamic client population and specifically addresses the problems of key update and revocation. Our system model consists of the following entities:

- A trusted **Authentication Server** (AS). The authentication server is responsible with registering clients - associating a number of attributes (names for example) to a cryptographic identity (in this case a set of symmetric keys). The AS is a key element in our security infrastructure, and its compromise is a catastrophic event. In order to strengthen its security and to shield it from DoS attacks, the AS is not accessible on-line.

- The **clients** - a number of computing devices interacting with each other. These include both human users and a variety of electronic services. Based on the peer-to-peer paradigm, we assume that any random pair of clients may want to interact (and authenticate each other). Clients may use a great variety of computing platforms - ranging from personal digital assistants and smart phones to high end application servers. We assume each client has a reasonably powerful CPU capable of performing symmetric cryptographic operations, a reasonably large amount of memory (both volatile and non volatile) and a network connection. However, our system does not require continuous network connectivity.

- A number of **infrastructure directories**. These are semi-trusted entities, in the sense that their compromise will not lead to a security breach, but may result in denial of service. Their purpose is to guarantee availability and they work more or less like caches.

Before a client can start using the authentication infrastructure, it has to go through a registration phase, which requires a secure physical channel between the client and the AS.

By no means is this registration step specific only to our framework; in any authentication protocol based on TTPs there is an implicit registration phase, when a new client establishes a shared master secret with the AS by out-of-band means.

During the registration phase, the AS presumably checks the client's identity much in the same way as a Certification Authority would do it in a PKI. After verifying its credentials, the AS issues the client an endorsement, in the form of the *client authentication database*, which the client can use to authenticate itself to other clients. This endorsement is only valid for a limited period of time (in the order of months, even years). After this endorsement expires, the client needs to contact the AS again in order to obtain new authentication material - we call this step *key update.* The AS can also render a client's endorsement unusable before its natural expiration - we call is step *key revocation.*

Our design relies on two basic building blocks - the *client introduction certificate* - which is similar to the renewable token suggested in [6, 7], and the *client authentication database* - which enables a client to collect all necessary authentication material at registration time, thus making subsequent communication with the AS un-necessary. In the next subsection we sketch a naive authentication protocol that illustrates the use of these building blocks.

5.1 A Naive Solution

Let us consider a security realm consisting of an authentication authority AS and a number of clients. As discussed in the previous section, clients first have to go through a registration phase, requiring a secure physical channel to the AS, after which they never have to contact the AS again, except for credentials re-issuing. For the sake of simplicity, let us first consider scenario where a client A only needs to communicate with one other client B. $A's$ registration phase works as follows:

(1) A $\longrightarrow$ AS: A, B
(2) AS $\longrightarrow$ A: $K_{AAS}, K_S, \{A, B, K_S\}_{K_{BAS}}$

Since the registration phase is done over a secure physical channel, the key K_S shared by A and B can be sent in clear, and only $A's$ ticket for B needs to be encrypted with $B's$ master key. A also gets K_{AAS} - the *master client key* - which is used by the AS to encrypt the tickets other clients will use to contact A.

Once it completes the registration phase, A does not need to contact the AS ever again, since we assumed it only wants to talk to B. However, we still have a problem here, since both $A's$ and $B's$ registration credentials may expire after a certain time, but the AS has no control over how long the $\{A, B, K_S\}_{K_{BAS}}$ renewable ticket can be used. To address this problem, we add $A's$ credentials issue and expiration date in the ticket, which now becomes: $\{A, B, T_{issue_A}, T_{expire_A}, K_S\}_{K_{BAS}}$; $B's$ credentials issue and expiration date - $(T_{issue_B}, T_{expire_B})$ - are also sent in clear to A as part of the second message. We define the enhanced ticket as *A's introduction certificate to B* - $\mathbf{IC_{AB}}$. We also define the key K_S in the certificate as *A's secure introduction key to B* - $\mathbf{SIK_{AB}}$.

The purpose of introduction certificates is equivalent to the purpose of public-key certificates. Both of them are used to authenticate strangers. The difference is that the SIK in the IC serves only two clients, while a client's public key in a public-key certificate can be used by all the clients in the realm.

A can now use the IC it has received from the AS to initiate secure sessions with B as follows:

(1) A $\longrightarrow$ B: A, B, IC_{AB}, N_A
(2) B $\longrightarrow$ A: $[N_A]_{K_{AB}}, N_B$
(3) A $\longrightarrow$ B: $[N_B]_{K_{AB}}$

where $K_{AB} = f(K_S, N_A, N_B)$ is the new session key. Both A and B are supposed to terminate the protocol if their local time does not fall in the (T_{issue}, T_{expire}) interval specified by the AS for the other party.

The procedure outlined above can easily be extended to accommodate the entire client population: assuming $\mathcal{N}$ to be the set of all N clients, the AS can give A $(N-1)$ introduction certificates, one for every other client; all this information can then be organized into a set of $(Identity_J, T_{issue_J},\ T_{expire_J}, SIK_{AJ}, IC_{AJ})$ tuples, with $J \in \mathcal{N}$ and $J \neq A$, which form *A's authentication database.* Each client's authentication database contains $N-1$ tuples, and in order to provide full connectivity, the AS needs to generate N such databases, one for each client.

By providing each client with an authentication database, we can now place the AS off-line: after registration, a client has an introduction certificate and a SIK for every other client, so it does not need to contact the AS again (at least not until its registration credentials expire). Two registered clients can then authenticate and establish a secure channel using the above described protocol.

The system architecture described so far succeeds in providing symmetric key authentication services to a population of N clients, without requiring the AS to be on-line. However, this is a naive solution; its main drawback is that it assumes a static client population, since each client needs to be given a SIK and an IC for every other client at registration time. Furthermore, key renewal and revocation is likely cause serious trouble, since changing one client's database also require updating all the other clients. In the next section we will describe a more realistic solution, which assumes a dynamic environment where clients may leave and join at any time, and where keys can be revoked before their natural expiration.

5.2 The Proposed Architecture

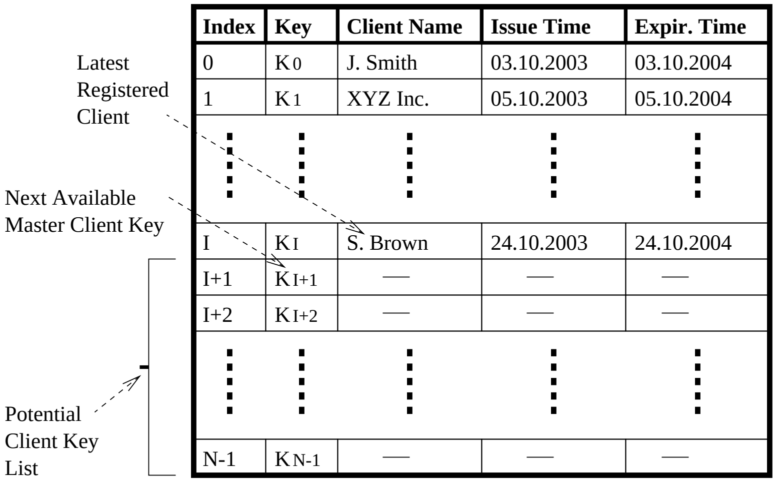
The protocol described in the previous section can only accommodate a static client population; this is a clear drawback. The authentication infrastructure we describe in this section overcomes these limitations; it can accommodate a dynamic client population, with the condition that the maximum size of this population is a-priori known.

The idea is to give every client in the realm a secure *certified* introduction key for every other *potential* client at registration time. Since identity information and issue/expiration times cannot be known in advance for future clients, this information needs to be explicitly exchanged during the authentication phase by the two clients involved, which results in a slightly modified authentication protocol. As we will show, this also requires certain modifications in the format of the introduction certificate and of the client authentication database.

In detail, the registration phase works as follows: assuming the maximum client population size N is known in advance, the AS starts with a *potential client key list* of N symmetric keys. When a new client A wants to register, the AS takes the next unused key K_I in the list and passes it to the client over the secure registration channel (K_I now becomes the *master client key* for A). The AS then updates its client records (shown in Figure 2) by associating $A's$ identity to the corresponding index I in the key list, the client's registration time - T_{issue_I}, and the time after which the client's registration expires - T_{expire_I}. The AS then generates for A an introduction certificates database consisting of $N-1$ (SIK_{IJ}, IC_{IJ}) tuples, for $J \in \{0,..,N-1\}$ and $J \neq I$, with SIK_{IJ} being a random symmetric key, and $IC_{IJ} = \{SIK_{IJ}, I, J,\ \textit{SHA-1}(Name_A), T_{issue_I}, T_{expire_I}\}_{K_J}$. The introduction certificates database can then be passed to the client by means of the same secure channel. Thus, the registration phase for client A is as follows:

(1) A $\longrightarrow$ AS: $Name_A$
(2) AS $\longrightarrow$ A: $I, K_I, T_{issue_I}, T_{expire_I}, (SIK_{IJ}, IC_{IJ})_{J\in\{0,..,N\},J\neq I}$

Alternatively, $A's$ authentication database can be encrypted under K_I and placed on the untrusted directories, so that A can download it when needed. As an optimization, the AS can even encrypt individual rows in the database (that is individual (SIK_{IJ}, IC_{IJ}) pairs); in this way, A can only download the introduction certificates it needs during the authentication phase (however, if the size of the authentication database is too large, this may expose the master client key to cryptanalytic attacks).



Index	Key	Client Name	Issue Time	Expir. Time
0	K_0	J. Smith	03.10.2003	03.10.2004
1	K_1	XYZ Inc.	05.10.2003	05.10.2004
⋮	⋮	⋮	⋮	⋮
I	K_I	S. Brown	24.10.2003	24.10.2004
I+1	K_{I+1}	—	—	—
I+2	K_{I+2}	—	—	—
⋮	⋮	⋮	⋮	⋮
N-1	K_{N-1}	—	—	—

Figure 2: AS client records

For the authentication phase, consider a client A that wants to establish a secure session with a client B. We assume both A and B have registered with the AS, and they have been assigned the client indices I and J respectively. The protocol is as follows:

$$\begin{array}{lll} (1) & \text{A} \longrightarrow \text{B}: & I, N_A \\ (2) & \text{B} \longrightarrow \text{A}: & J, N_B, IC_{JI} \\ (3) & \text{A} \longrightarrow \text{B}: & \{Name_A, N_B\}_{K_{AB}}, IC_{IJ} \\ (4) & \text{B} \longrightarrow \text{A}: & \{Name_B, N_A\}_{K_{AB}} \end{array}$$

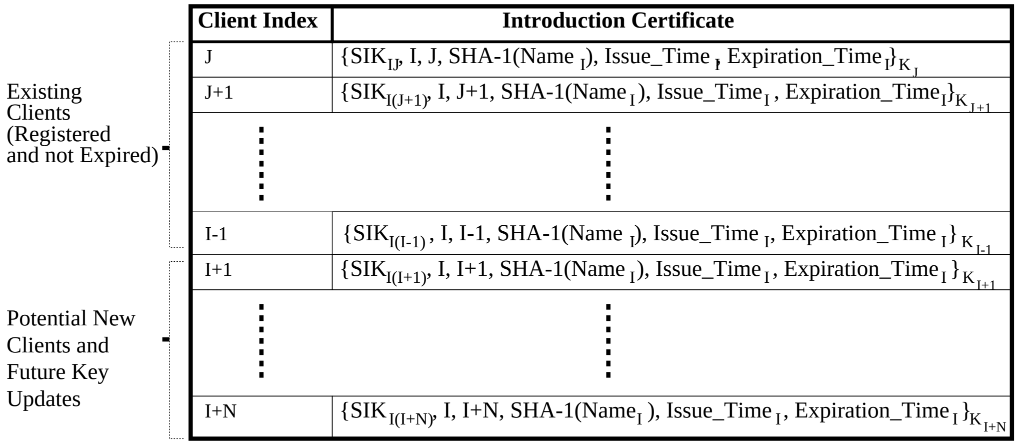
In the above protocol, $K_{AB} = f(SIK_{IJ}, SIK_{JI}, N_A, N_B)$. We assume that initially A and B are complete strangers (they do not know each other's client indices). In steps (3) and (4) of the protocol, A and B exchange their names, protected under the shared secret (so no attacker can infer the identities of the authenticating parties). Before producing any cipher-text, both A and B decrypt IC_{JI} and IC_{IJ} respectively, and check that the expiration time in these certificates has not already passed; this prevents the usage of old (potentially compromised) ICs. Also, both parties must compute the *SHA-1* digest of the other party's name and make sure it matches the digest in the IC.

6 Key Update and Revocation

Our authentication scheme makes clients' transactions independent from the AS, which can now be placed off-line. The price we have to pay for this is lack of freshness: in our protocol SIKs are not freshly generated for each session but instead pre-distributed by means of an authentication database. However, because of possible cryptanalytic attacks, symmetric keys can only be used for a limited time, after which they should be discarded and replaced with fresh cryptographic material; furthermore, when exceptional events occur, keys may also need to be revoked. In this section we show how the scheme we propose can be modified in order to allow efficient key update and revocation.

6.1 Key Update

When a client registers, it receives from the AS a symmetric master client key and an authentication database. To prevent key compromise due to cryptanalytic attacks, the AS also sets a limit on how long the client is allowed to use this key material. This time limit is expressed through the T_{expire} value present in each IC. All the clients in the realm are required to reject an IC for which the T_{expire} has passed (this has the additional benefit of giving the AS certain control over the client). A client whose T_{expire} has passed needs to contact the AS to get new keys (key update).



Existing Clients (Registered and not Expired)
Potential New Clients and Future Key Updates

Client Index	Introduction Certificate
J	$\{SIK_{IJ}, I, J, SHA\text{-}1(Name_I), Issue\_Time_I, Expiration\_Time_I\}_{K_J}$
J+1	$\{SIK_{I(J+1)}, I, J+1, SHA\text{-}1(Name_I), Issue\_Time_I, Expiration\_Time_I\}_{K_{J+1}}$
⋮	⋮
I-1	$\{SIK_{I(I-1)}, I, I-1, SHA\text{-}1(Name_I), Issue\_Time_I, Expiration\_Time_I\}_{K_{I-1}}$
I+1	$\{SIK_{I(I+1)}, I, I+1, SHA\text{-}1(Name_I), Issue\_Time_I, Expiration\_Time_I\}_{K_{I+1}}$
⋮	⋮
I+N	$\{SIK_{I(I+N)}, I, I+N, SHA\text{-}1(Name_I), Issue\_Time_I, Expiration\_Time_I\}_{K_{I+N}}$

Figure 3: The client IC database after a key update. It is assumed the client performing the update is assigned the new client index I, and the index of the earliest registered client not yet expired is J. N is the maximum number of clients in the realm.

One property that needs to be enforced here is *locality*: a key update should only affect the client that performs it, and none of the other clients. This is essential if we want to achieve our goal of keeping the AS off-line.

For the sake of simplicity, let us assume the client master key lifetime is the same for all clients (however, the key update mechanism is more or less the same, even if client key lifetime is not the same for all clients, only the formula for calculating the total memory requirements will change). If N is the maximum number of clients in the realm, the locality property can be achieved by requiring clients to store an authentication database consisting of at most $2 * N$ ICs. Key update works as depicted in Figure 3: when contacted by client A for a key update, the AS creates a new database consisting of ICs for all the other clients not expired at that moment (at most N) **and** N extra ICs, for the next N consecutive keys in the potential clients key list. The AS needs to ensure it always has at least N unused entries in this list, by generating new keys when it drops below this threshold. In this way, client A is guaranteed to have an IC for every other non-expired client in the realm. Furthermore, should new clients register, or existing clients perform the key update, they will be assigned one of the next N unused client master keys, for which A is also given an IC. Since there can be at most N clients, and the master client key lifetime is the same for all of them, there can be at most N new client master keys issued until $A's$ key will expire again, so A is guaranteed to have ICs for every new client master key to be issued. In this way, clients only need to contact the AS for key updates, and the locality property is achieved.

6.2 Key Revocation

A consequence of using long-term keys is the possibility that some of these keys may be compromised before their normal expiry time, so they need to be revoked. We distinguish two cases:

- a client's device is lost, stolen or damaged; the entire client database is compromised; the client needs to contact the AS to acquire new credentials (key update). Furthermore, the client's old credentials need to be rendered unusable, so that no other party can impersonate the client (revocation).

- a client quits or misuses the service; again, the AS needs to render the client's keys unusable (revocation).

The revocation mechanism we propose is based on certificate revocation lists (*CRL*): the AS keeps a list consisting of the indices of all clients whose authentication databases have been revoked, and periodically pushes this list to the infrastructure directories. Because the directories are not trusted not to tamper with this list, clients need a mechanisms to verify its integrity. To facilitate this, the AS computes one *CRL authentication code* for each client (and potential client) in the realm. For a client index I, the CRL authentication code is the *HMAC-SHA-1* [16] of the CRL using the client master key K_I. All the CRL authentication codes are then pushed to the infrastructure directories, together with the actual CRL.

When a client wants to verify the freshness of a given introduction certificate, it first needs to download the revocation list from the closest directory. The client then requests the CRL authenticator code corresponding to its client index, and verifies it using its master key. Once the CRL verification has succeeded, the client can proceed with validating the IC, by verifying that the client index in the IC is not present in the revocation list.

We can see that in this case revocation is more expensive than in a traditional PKI: for each CRL, the AS needs to generate a number of CRL authenticator codes linear to the number of clients; for traditional PKIs the CRL only needs to be signed once. This workload can be reduced if the infrastructure directories are trusted to correctly disseminate revocation information, in which case, the AS does not need to generate any CRL authenticator codes, (but clients need to establish secure channels with the directories when downloading the revocation list).

Key revocation also has implications on size of the client introduction certificates database. In the previous section, we showed that in order to ensure that a client only needs to contact the AS for registration or key update, it needs an IC database with at most $2 * N$ entries. When calculating this, we assumed N to be the maximum number of clients in the realm, so during a key lifetime, at most N key updates could occur. However, if clients can revoke their keys before expiration time, the total number of key issued can be larger than the number of clients (some clients may be issued more than one key during a key lifetime interval). Assuming that P is the probability that a client master key is revoked before expire, the new maximum size for the client IC database becomes $(2 + P) * N$.

It is worth noticing that although revocation significantly increases the AS workload, it does not require it to be on-line (it only needs an unidirectional network connection to the directories in order to periodically push the revocation information); furthermore, the tasks of credential issuing and revocation can be separated, as suggested in [2]. By introducing a separate revocation authority, we can even have the AS disconnected from the network (since it does not even need to push the revocation information to directories), which would greatly increase its security.

7 Performance Evaluation

We are in the advanced stages of building a prototype implementation for our authentication infrastructure, and we plan to experimentally deploy it at the Vrije Universiteit campus in Amsterdam. Our prototype consists of an Authentication Server and a client credential management library.

The Authentication Server is a stand-alone application that manages the master client key list, generates the IC database for client registration and credentials update, and manages credentials revocation. Its C source code consists of about two thousand lines of code. At initialization, the AS administrator needs to specify the maximum expected client population size, as well as the maximum expected revocation rate. Based on these values, the server generates a master client key list. During normal server operation, this list is stored in memory; the server also writes it on disk (as a binary file), protected under a password, so it can survive potential server crashes. The command line interface allows the AS administrator to register new clients, update existing clients' credentials and revoke issued credentials.

In the case of client registration and credentials update, the output of the operation is a file storing the newly generated client authentication database; in the case of credentials revocation, the output is a binary file consisting of the (updated) CRL, together with CRL authentication codes for all the master client keys. The output file then needs to be transferred to the target system (the client's computer/PDA in the case of the authentication database, the revocation directory for the CRL) by some secure out-of-band mechanism (for example stored on a ZIP-drive, CD-ROM, memory stick). Consistent with our goal of keeping the AS server strictly off-line, we do not provide any support for transferring results via regular network connections.

We are currently developing the credentials management library, which will provide application programmers with an interface similar to the BSD Socket Interface. The library is initialized with the name of the authentication database file (possibly password-protected) obtained by the user from the AS server. After this, connecting our secure sockets involves executing the authentication protocol described in Section 5.2, with the shared key obtained at the end of the protocol being used to protect the future data traffic between the two endpoints. In addition to the regular connection information (the other party's network address, transport protocol, etc.), our secure sockets also store the authenticated name of the client at the other end. Two potential applications we have in mind are de-centralized ICQ-like chat services (users can authenticate each other without the need for a trusted-online server as it is in existing such applications), and one-to-one authentication for PDAs.

Finally, we have performed a number of experiments to measure the performance of our implementation; these were performed on a AMD Duron 750MHz 64K cache system, with 196MB RAM, running Linux Mandrake 9.1. Our code was compiled using the GNU C compiler, version 3.2.2. For the remaining of this section, we assume that all credentials are issued with the same lifetime, and the revocation probability (for that lifetime) is 0.1 (this later number is based on the results published in [3]). According to the formula derived in the previous section, this leads to an authentication database size of $2.1 * N$ entries per client, where N is the maximum expected client population size. We used the AES algorithm [1] for encrypting the ICs and for the authentication protocol. In both cases the key size was 128 bits.

The first experiment evaluates the performance of the AS server implementation. We measured the amount of time required to initialize the server (generating the master client key list) and to generate one client authentication database, for various maximum expected client population sizes. Figure 2 summarizes the results.

Number of master keys	**AS initialization time**	**Generating one client database**
1000	0.01 sec	0.04 sec.
10000	0.04 sec	0.32 sec.
100000	0.34 sec	2.97 sec.
1000000	3.36 sec	30.5 sec.
2000000	6.61 sec	59.8 sec.
5000000	16.55 sec	150.1 sec.

Table 2: AS server performance measurements

Not surprisingly, the amount of time needed for initialization and authentication database generation grows linearly with the maximum expected number of clients. The time to initialize the server (generating the client master key list) is by all means negligible. For the client authentication database generation, we can see that even for very large expected client populations (in the order of millions), the the time required is less than two minutes. We assume this is acceptable, considering that client registration and credentials update are rare events (once a year), and they anyway involve some sort of human to human interaction (in order to transfer the authentication database from the off-line AS server to the client system) which is much more time-consuming.

The second experiment compares our authentication protocol with public-key based SSL [9]. Our protocol is implemented in C using the OpenSSL Crypto library; we compare it with the protocol implemented by the OpenSSL SSL library (with the *AES128-SHA:RC4-MD5:DES-CBC-SHA:RC4-SHA* cipher suites enabled); the two authenticating end-points are processes running on the same host (the 750MHZ AMD Duron described earlier), so that network latency does not influence the experiment. For our protocol we have the two parties store their entire authentication database in memory. For SSL we use the "server and client authentication" option, and the public key algorithm is RSA with 1024 bit keys. The results, shown in Table 3, were obtained after running each type of authetication session for 10000 times and taking the average.

Authentication Protocol	**Duration**
our protocol - 1000 entries per client database	0.37 msec.
our protocol - 10000 entries per client database	0.38 msec.
our protocol - 100000 entries per client database	0.38 msec.
our protocol - 1000000 entries per client database	0.38 msec.
SSL - 1024 bit RSA keys (client and server authentication)	11.6 msec.

Table 3: SSV vs. our protocol - performance comparison

Not surprisingly, the size of the authentication database has little influence on the performance of the protocol, since the entire database is stored in memory. We can see that our authentication protocol is an order of magnitude faster than SSL. We expect the relative speedup to be even more significant on PDAs, normally equipped with less powerful CPUs. Besides the speedup, probably the biggest advantage of our protocol is that the symmetric key cryptographic operations it involves are much less CPU-intensive than the public key cryptographic operations needed by SSL, which is a great advantage for battery-powered PDAs.

Finally, another factor that should be taken into account when evaluating our architecture, is size of the client authentication database, which grows linearly with the maximum expected client population size. As shown in Table 4, the size of one *(SIK, IC)* database entry is 96B. Given that the number of entries in a client database is $2.1 * N$, where N is the maximum expected client population size, we can see that this database size grows linearly from 96KB for an authentication realm of thousand clients to 96MB for a million clients realm.

Entry Format:
(SIK, $\{SIK, Index_{src}, Index_{dest}, SHA\text{-}1(Name), T_{issue}, T_{expire}\}_{Key_{client}}$)

Field	**Size**
SIK	16B
$Index_{src}$	4B
$Index_{dest}$	4B
$SHA\text{-}1(Name)$	20B
T_{issue}	4B
T_{expire}	4B
SHA-1 over above fields	20B
Total IC size	72B
Total after encryption in CBC mode	80B (5 blocks)
Grand total (encrypted IC + SIK)	96B

Table 4: Authentication database entry size

For efficiency and security reasons, clients should be able to store their entire authentication database in memory; considering the above numbers we can conclude that, at least for today's PDAs, our authentication infrastructure could scale up to authentication realms of at most hundred thousand clients (so the database size does not exceed 10MB).

8 Conclusion and Future Work

In this paper we have presented a symmetric key authentication infrastructure based on an off-line TTP. Because the TTP is off-line, it is shielded from both hacking and DoS attacks; the fact that authentication does not depend on continuous network connectivity makes our scheme particularly suited for MANET environments. Our authentication architecture can support a dynamic client population, with the condition that the maximum size of this population is known in advance. We are able to achieve this by trading memory for flexibility under the assumption that large storage devices are becoming a commodity in today's computing environment.

The scheme presented in this paper works for a single security domain, under the jurisdiction of a single authentication server. As future work we would like to expand our scheme to the case of multiple domains and we are now working on the details of these extensions.

References

[1] Advanced Encryption Standard. FIPS 197, NIST, US Dept. of Commerce, Washington D. C. November 2001.

[2] Lampson B., Abadi M., Burrows M., and Wobber E. Authentication in Distributed Systems: Theory and Practice. *ACM Trans. on Computer Systems*, 10(4):265–310, Nov. 1992.

[3] S. Berkovits, S. Chokhani, J.A. Furlong, J.A. Geiter, and J.C. Guild. Public Key Infrastructure Study: Final Report. Produced by the MITRE Corporation for NIST, 1994.

[4] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution. *IEEE/ACM Trans. on Networking*, vol. 3(1):31–41, 1995.

[5] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of a Family of Attack-Resistant Authentication Protocols. *IEEE J. on Selected Areas in Communication*, 11(5):679–693, June 1999.

[6] C. Boyd. A Class of Flexible and Efficient Key Management Protocols. In *Proc. 9th IEEE Computer Security Foundation Workshop*, 1996.

[7] C. Boyd. A Framework for Design of Key Establishment Protocols. In *Proc. First Australasian Conf. on Information Security and Privacy*, pages 146–157, June 1996.

[8] D. Denning and G.M. Sacco. Timestamp in Key Distribution Protocols. *Commun. of the ACM*, 24(8):533–536, 1981.

[9] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol Version 3.0. Internet Draft (expired), Nov. 1996.

[10] L. Gong. Lower Bounds on Messages and Rounds for Network Authentication Protocols. In *Proc. of the 1st ACM Conf. on Computer and Commun. Security*, November 1993.

[11] T. Hwang, N.Y. Lee, C.M. Li, M.Y. Ko, and Y.H. Chen. Two attacks on Neuman-Stubblebine Authentication Protocols. *Information Processing Letters*, 55:103–107, 1995.

[12] ISO/IEC. *ISO/IEC 9798-2 - Information Technology - Security techniques - Entity Authentication - Part2: Mechanisms using symmetric encipherment algorithms*, 1999.

[13] I.L. Kao and R. Chow. An Efficient and Secure Authentication Protocol Using Uncertified Keys. *ACM Operating System Review*, 29(3):14–21, 1995.

[14] A. Kehne, J. Schonwalder, and H. Langendolfer. A Nonce-Based Protocol for Multiple Authentication. *ACM Operating System Review*, 26(4):84–89, 1992.

[15] J.T. Kohl and B.C. Neuman. The Kerberos Network Authentication Service (Version 5). Technical report, IETF Network Working Group, 1993. Internet Request for Comments RFC-1510.

[16] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, Feb. 1997.

[17] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. of the ACM*, 21(12):993–999, 1978.

[18] R.M. Needham and M.D. Schroeder. Authentication Revisited. *ACM Operating System Review*, 21(7):7–7, 1987.

[19] B. Clifford Neuman and S.G. Stubblebine. A Note on the Use of Timestamps as Nonces. *ACM Operating System Review*, 27(2), 1993.

[20] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. of the 35th Annual IEEE Symp. on the Foundations of Computer Science*, pages 124–134, 1994.