Symmetric and Asymmetric Authentication: A Study of Symmetric and Complementary Properties and Their Effect on Interoperability and Scalability in Distributed Systems

Olga Kornievskaia
aglo@umich.edu

A thesis proposal submitted in partial fulfillment
of the requirements for the degree of Doctor of Philosophy
(Computer Science & Engineering)
at the University of Michigan
June, 2002

Advisor:
Peter Honeyman

Committee Members:
Brian Noble
Atul Prakash
Paul Resnick
Carl Ellison

Contents

Chapter 1

Introduction

1.1 Motivation

"There ain't no such thing as a free lunch"
– Robert Heinlein

Authentication and authorization, the essential mechanisms for securing access to resources, have been studied for years and many solutions have been proposed, implemented, and widely used. While developing highly distributed authentication protocols and authorization mechanisms, questions regarding user autonomy, the growing size of systems, assumptions about authority and trust, and accomodating unreliable networks have been considered. The uncoordinated and unconstrained complexity as well as the spectrum of requirements of distributed systems has led to the development of vast variety authentication and authorization mechanisms: local login (OS-specific (e.g., UNIX), distributed authentication (e.g., Kerberos)), secure remote login (SSH, Telnet, VPN), secure data transfer (scp, https, kftp), secure email protocols (IMAP, POP, PGP, S/MIME), secure file systems (AFS, NFS), secure directories (LDAP) to name a few.

Interoperability and integration of individual, but potentially similar, components of a distributed system is important because (i) it reduces and hides complexity from the user, and (ii) it introduces a scalable form of reusability. Applications, protected by the same distributed authentication protocol, are often well integrated. For example, as a part of initial login, users get access to a variety of Kerberos protected service: AFS file servers, IMAP or KPOP email servers, PTS servers, LDAP directory servers; this single signon feature is vital to building scalable and secure systems. However, many applications – other password-based or public key base authentication services – are not well integrated. For example, mutual authentication on the Web employs different authentication methods: SSL for server authentication and Kerberos passwords for user authentication. As the result, a user must deal with multiple credentials and becomes painfully aware of the system's complexity.

We outline the problems with the current, non-integrated approach.

- **Credential management is hard**.
  - Users and security administrators have to manage multiple accounts for each user. For example, a student at the University of Michigan can easily have five different accounts: (i) a general university account (*umich*), (ii) a particular school account (*engin*), (iii) a departmental account (*eecs*), (iv) a specific lab account (*citi*), and (v) an account on a specialized machine (e.g., with multimedia or parallel processing resources).
  - Multiple places store credentials: (i) Kerberos ticket cache, (ii) protected file system stores private keys for SSH and PGP, and (iii) Web browser's certificate cache stores SSL and S/MIME certificates.
  - Users deal with multiple public/private key pairs: (i) email keys (PGP), (ii) Web certificates (SSL), (iii) keys for secure remote login (SSH).
  - To provide user mobility and easy access to public key credentials, a distributed file system or secure hardware (e.g., a smart card that stores both public and private keys) is needed.
- **No user certificates and no PKI** present the following problems.
  - Multiple, non interoperable authentication mechanisms on the Web.
  - Protocols that require end entity certificates prove difficult to deploy (e.g., IPSec).
- **Proxying** is an emerging solution to the complex infrastructure and the result of developing and changing face of a distributed system.
  - **Architecture shifts**. A shift from the client-server model to client-proxy-server(s) introduces new challenges. Client requests are becoming more complex and require contact with multiple servers. A proxy in front of multiple servers establishes a combined front. It removes and hides complexity from the client. Lastly, a proxy is a solution that integrates legacy applications – the ones that cannot be modified to better fit the situation – into the ever-changing system.

    Furthermore, we distinguish a shift from a simple proxy architecture to client-server-server(s)...server(s) architecture that involves a chain of servers, each of which can itself be a root of a subtree. A job in the Grid environment serves as an example of such complex interaction.
  - **Proxied authentication**. Web access to backend services such as file servers, mail servers, and directories awoke considerable interest. In this case, a Web server serves as a proxy server between the client and the backend service. Backend services have predefined authentication protocols (e.g., Kerberos) that, in practice, are incompatible with the client-server authentication protocol used by the proxy (e.g., SSL with certificates). The challenge then is to resolve the mismatch in authentication.
  - **Authorization**. Proxied authorization complicates access control because applications often use ACL-based authorization mechanisms

and rarely support delegation. The two common ways of dealing with the problem is to either modify the existing authorization mechanism to handle delegated credentials (capabilities) or transfer identity credentials to the proxy and empowering it to act as the user.

We briefly outline some questions and concerns that arise from integration of security mechanisms. Each one has a solution within a given system but in an integrated environment it might require a different approach.

- **Namespace**. Each authentication mechanism is tied to a namespace convention for users and resources. In an integrated environment with multiple authentication systems and corresponding namespaces, the integration of namespaces presents a challenge. For example, to integrate just two namespace, should a single, combined namespace be created or should a mapping between the namespaces is created? First approach has scalability issues and second might not be possible to achieve.
- **Access control**. For a complex (proxied) environment described above, the control of server's actions is important. In an integrated system, for each available system resource access control policies have to be enforced; whether or not policies have to change due to integration of other components. Authorization is tightly connected with the namespace issue and authentication credentials. In an integrated system where both authorization credentials and identify certificates are used, access control would require some sort of modification because the namespace of those systems cannot be merged.
- **Delegation**. For a given authentication protocol, delegation of identity credentials is a debated topic. How can we delegate across authentication protocols?
- **Transparency vs. control**. How much should a client be involved? If a client desires to maintain control of his actions, e.g., specifying a customized security policy, allowing or disallowing delegation of credentials, transparency has to be sacrified. The issue of control boils down to trust. If all services are (some what) trusted, then does the client still need to specify a security policy stating who is allowed to do what?
- **Security policy**. In an integrated environment, in addition to identifying, specifying, and enforcing a security policy for a service, security policies of the other services have to be accounted for. Furthermore, user-specified security policy has to account for inability to know all the services for any given request and must prevent the user from specifying a policy that would compromise the system.
- **Cost of integration**. How much modification to already existing software is allowed or needed? Would the administration of the integrated services be harder, more complex, overall leading to a less secure system and how can it be assessed? Would integration of the system lead to getting the worst of each component? Integration could lead to either a simplified solution or, if things go bad, to a more complex one.

1.2 Thesis Statement

The goal of this thesis is to identify structural symmetries in symmetric and asymmetric authentication and show that interoperability between such systems is sound. We design, implement, and analyze the interoperability tools leveraging off the similarities in properties such as time, network availability, and granularity of access.

The rest of the thesis is organized as follows. In Chapter 2 we discuss the background of authentication. In Chapter 3 we compare symmetric and asymmetric authentication, develop taxonomy of authentication, and finally discuss an interoperability framework based on the exposed symmetry. The following chapters concentrate on specific interoperability tools. In Chapter 4 we introduce a new public key infrastructure that is built on a basis of symmetric key infrastructure. In Chapter 5 we introduce a mechanism for using public key credentials to get Kerberos credentials in a proxied environment. In Chapter 6 we present an application of the Kerberized Public Key Infrastructure and the Kerberized Credential Translation in the context of securing a reservation request in the Grid environment.

Chapter 2

Background

This section is devoted to the discussion of previous work on authentication, including an overview of basic concepts and a survey of related work.

2.1 Authentication: Protocols and Systems

In this section, we present background on authentication in distributed systems. We start by providing a definition of an authentication protocol. We outline the objectives and common properties of authentication protocols. We also give a definition of an authentication system. Section 2.1.2 provides an overview of a symmetric key authentication system. Section 2.1.3 gives an overview of an asymmetric key authentication system.

An authentication protocol allows one or more participants to verify (i) the identity of the other parties on the basis of at least one of the following: something known (e.g., a shared key), something possessed (e.g., smartcard), or something inherent (e.g., biometrics) and (ii) the active presence of the other during the process. The verification process should not allow the verifier to reuse an authentication exchange with the goal of impersonating the entity. At the same time, the process must provide the verifier with enough confidence that an attacker is not trying to impersonate a legitimate entity. Different authentication protocols might be categorized based on the following properties: type of cryptography (symmetric vs. asymmetric), reciprocity of authentication (mutual vs. one-way), key exchange, computational and communication efficiency, real-time involvement of a third party (on-line vs. off-line), nature of trust required from a third party, nature of security guarantees, and storage of secrets.

Authentication system is defined by (i) an authentication protocol, (ii) naming, (iii) security policy, and (iv) model.

A model refers to the architectural type that ties the system components together. Two architectural models – centralized and distributed – have been previously identified and currently widely used in existing systems. In a cen-

tralized model, a single (possibly replicated) server is responsible for providing certain services to predefined set of of principals. A principal is either a user or a resource. The server is responsible for naming the principals. The authentication protocol used defines the level of trust given to the server ranging from fully trusted server in the symmetric authentication protocol to partially trusted server in the asymmetric authentication protocol. In a distributed model, the space of participants is broken down into groups or domains. Each of the components defines an internal architecture that could be centralized in the simplest case or distributed. All the low-level components are then arranged in some way and relationships among componentsare described. There are three most commonly used distributed architectures: hierarchical, peer-to-peer (mesh), and a hybrid of both. While a hierarchical model requires a single trusted root and the complexity of establishing relationships in the peer-to-peer model is $\mathcal{O}(\mathcal{N}^2)$, a hybrid model combines the two model and tries to inherit the better characteristics of each one: the scalability of the hierarchical approach and the flexibility of the peer-to-peer model.

2.1.1 Terminology and Notations

This section describes the basic terminology, used in the literature and throughout the thesis.

Encryption (decryption) is a function, $\mathcal{E}$ ($\mathcal{D}$), that takes as an input a key, $\mathcal{K}_e$ ($\mathcal{K}_d$), and a message, $\mathcal{M}$. The output of the encryption operation is called a cipher text, $\mathcal{C} = \mathcal{E}(\mathcal{K}_e, \mathcal{M})$. The output of the decryption function is such that $\mathcal{M} = \mathcal{D}(\mathcal{K}_d, \mathcal{E}(\mathcal{K}_e, \mathcal{M}))$.

Symmetric key encryption, also referred to as conventional or classical encryption in the literature, is an encryption algorithm such that $\mathcal{K}_e = \mathcal{K}_d$ and written as $\mathcal{K}$.

Asymmetric key encryption, also referred to as a public key encryption, is an encryption algorithm where $\mathcal{K}_e \neq \mathcal{K}_d$. The relationship between the keys is such that $\mathcal{M} = \mathcal{D}(\mathcal{K}_d, \mathcal{E}(\mathcal{K}_e, \mathcal{M})) = \mathcal{E}(\mathcal{K}_d, \mathcal{D}(\mathcal{K}_e, \mathcal{M}))$. To emphasize the inverse relationship between the keys, $\mathcal{K}_e$ and $\mathcal{K}_d$ are written as $\mathcal{K}$ and $\mathcal{K}^{-1}$.

$\{\mathbf{M}\}_K$ represents a message, $\mathcal{M}$, encrypted under a key, $\mathcal{K}$.

Trusted Third Party (T3P) is an entity involved in establishing a secure channel between two participants. The nature of its involvement depends on the type of cryptography used.

Authentication Server (AS) also referred to as a key distribution server (KDC), is a trusted third party involved in a symmetric key authentication protocol.

Certificate Authority (CA) is a trusted third party involved in a public key authentication protocol.

2.1.2 Symmetric Key Authentication

In this section, we focus on authentication protocols that verify identity based on a secret key shared between the participants. While many symmetric key authentication protocols exist, we present the Needham and Schroeder protocol [91] because it serves as a foundation for the rest of the authentication protocols, such as Kerberos [94, 117], the most commonly used symmetric key authentication system. We choose Kerberos as a representative of symmetric key authentication systems and describe its terminology and protocol.

Assumptions, Notations, and Objectives:

- Participants: *Alice*, ($\mathcal{A}$), *Bob*, ($\mathcal{B}$), and Authentication Server, ($\mathcal{AS}$).
- *Alice* and $\mathcal{AS}$ share a secret, $\mathcal{K}_a$.
- *Bob* and $\mathcal{AS}$ share a secret, $\mathcal{K}_b$.
- *Alice* and *Bob* have no shared secrets.
- *Bob* wants to authenticate *Alice*. For simplicity, we consider one-way authentication. Mutual authentication can be easily achieved with an extra round of messages between *Alice* and *Bob*.

Needham and Schroeder protocol (with Denning and Sacco's [26] correction):

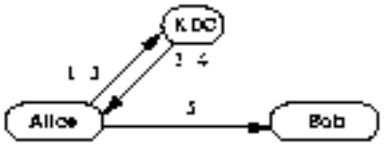
1. $\mathcal{A} \rightarrow \mathcal{AS}$: $\mathcal{A}$, $\mathcal{B}$, $\mathcal{N}_a$
2. $\mathcal{AS} \rightarrow \mathcal{A}$: $\{T, \mathcal{N}_a, \mathcal{B}, \mathcal{K}_{ab}, \{T, \mathcal{K}_{ab}, \mathcal{A}\}_{K_b}\}_{K_a}$
3. $\mathcal{A} \rightarrow \mathcal{B}$: $\{T, \mathcal{K}_{ab}, \mathcal{A}\}_{K_b}$

Kerberos, developed at the Massachusetts Institute of Technology, is a network authentication system, and it is illustrated in Figure 2.1. Authentication is achieved when one party proves to another knowledge of a shared secret. To avoid quadratic explosion of key agreement requirements, Kerberos relies on a trusted third party, referred to as a Key Distribution Center (KDC). *Alice*, a Kerberos principal, and *Bob*, a Kerberized service, each establish a shared secret with the KDC.

At login, *Alice* receives a Ticket Granting Ticket (TGT) from the KDC. She uses her password to retrieve a session key encrypted in the reply. The TGT allows *Alice* to obtain tickets from a Ticket Granting Service (TGS) for other Kerberized services thus providing a single signon mechanism for such services. To access a Kerberized service, *Alice* presents her TGT and receives a service ticket, $\{Alice, Bob, \mathrm{K}_{A,B}\}_{K_B}$. To authenticate to *Bob*, *Alice* constructs a timestamp-based authenticator, $\{\mathrm{T}\}_{K_{A,B}}$, proving to *Bob* that she knows the session key inside of the service ticket. If mutual authentication is desired, *Bob* would reply back with an authenticator encrypted under the session key, in turn proving to *Alice Bob's* identity.

2.1.3 Asymmetric Key Authentication

This section looks at asymmetric, public key, based authentication protocols. As in the previous section, to show the foundation of the asymmetric authentication, we start by describing the Needham and Schroeder protocol, but this time, we outline the public key version of it. Then, we present a widely used



LOGIN PHASE:	ONCE PER SESSION
1. *Alice* → *KDC*:	"Hi, I'm *Alice*"
2. *KDC* → *Alice*:	TGT = $\{$*Alice*, TGS, $K_{A,TGS}\}_{K_{TGS}}$, $\{K_{A,TGS}, T\}_{K_A}$
ACCESSING SERVICES:	EVERY TIME BEFORE ACCESSING A SERVICE
3. *Alice* → *TGS*:	*Alice*, *Bob*, TGT, $\{T\}_{K_{A,TGS}}$
4. *TGS* → *Alice*:	TKT = $\{$*Alice*, *Bob*, $K_{A,B}\}_{K_B}$, $\{K_{A,B}, T\}_{K_{A,TGS}}$
5. *Alice* → *Bob*:	"Hi, I'm *Alice*", TKT, $\{T\}_{K_{A,B}}$

Figure 2.1: **Kerberos authentication.** Two phases are shown: initial authentication and service ticket acquisition. KDC is the Kerberos Key Distribution Center. TGS is the Ticket Granting Service. Most implementations combine these services. K_{TGS} is a key shared between the TGS and KDC. K_A is a key shared between *Alice* and the KDC, derived from *Alice's* password. $K_{A,TGS}$ is a session key for *Alice* and TGS. $K_{A,B}$ is a session key for *Alice* and *Bob*. T is a timestamp used to prevent replay attacks.

security mechanism called a Secure Socket Layer, SSL [1] [43, 44], that establishes a secure channel using authenticated key exchange mechanisms. While the protocol supports different public key authentication mechanisms, we focus on the certificate-based authentication mechanism and use it as a representative of the asymmetric key authentication protocols.

Assumptions and Notations:

- Participants: *Alice*, ($\mathcal{A}$), *Bob*, ($\mathcal{B}$), and Certification Authority, ($\mathcal{CA}$).
- *Alice* and *Bob* have each other's public keys.

Needham and Schroeder protocol (with Lowe's [71] correction):

1. $\mathcal{A}$→$\mathcal{B}$: $\{\mathcal{N}_a\}_{K_b}$
2. $\mathcal{B}$→$\mathcal{A}$: $\{\mathcal{B}, \mathcal{N}_b, \mathcal{N}_a\}_{K_a}$
3. $\mathcal{A}$→$\mathcal{B}$: $\{\mathcal{N}_a\}_{K_b}$

Secure Socket Layer is a protocol that provides secure network connections, addressing the need for entity authentication, confidentiality, and integrity of messages in the Internet. SSL uses public key cryptography, in particular certificates, to accomplish authentication and secret key cryptography to provide confidentiality and integrity of the communication channel. What makes this protocol extremely attractive is that support for SSL is universal among Web browsers and servers.

[1] SSL is renamed by IETF as Transport Layer Security, TLS [27]

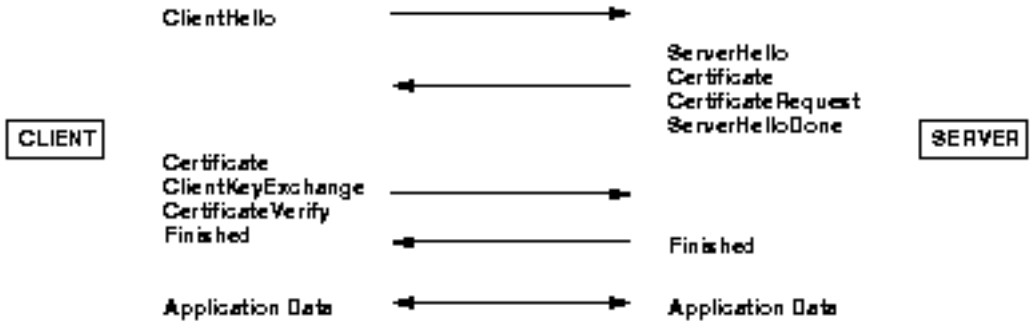


Figure 2.2: **SSL handshake protocol.** The SSL protocol consists of a set of messages and rules about when to send each one. The client initiates a communication and proposes a set of SSL options to use for the exchange. The final decision lies with the server, who selects from the proposed options. To improve performance some messages are sent together, e.g., 2-5. The protocol happens in two stages. The initial negotiation and selection of security context for the session ends with SERVERHELLODONE. The FINISHED messages signify the end of the negotiation and is subject to the negotiated cipher suite; messages are encrypted and authenticated according to that suite.

SSL consists of two sub-protocols: the SSL *record* protocol and the SSL *handshake* protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol uses the record protocol to negotiate a security context for a session. SSL supports numerous encryption and digest mechanisms that the client and the server negotiate during the SSL handshake. Figure 2.2 shows the exchange of messages in the handshake. Authentication in SSL is based on a public key challenge-response protocol [28, 102] and X.509 [39] identity certificates. Certificates were introduced by Kohnfelder [65] as a practical way of securely distributing public keys.

1. $\mathcal{A} \rightarrow \mathcal{B}$: $\mathcal{N}_a$
2. $\mathcal{B} \rightarrow \mathcal{A}$: $\mathcal{N}_b$, $\{T, \mathcal{B}, \mathcal{K}_b\}_{K_{ca}^{-1}}$
3. $\mathcal{A} \rightarrow \mathcal{B}$: $\{\mathcal{MAC}_K(1,2)\}_{K_a^{-1}}$, $\{T, \mathcal{A}, \mathcal{K}_a\}_{K_{ca}^{-1}}$, $\{\mathcal{K}\}_{K_b}$,

SSL supports mutual authentication. First, a user authenticates the server. The user has the responsibility to assure that he can trust the certificate received in the CERTIFICATE message from the server, the same applies to the server during the client authentication step. That responsibility includes verifying the certificate signatures, validity times, and revocation status. The user then sends his public key certificate and the proof – a digitally signed cryptographic hash of information available to patries – that he possesses the corresponding private key.

2.2 Basis of Authentication

In this section, we explore identification in more detail. One of the objectives of authentication is to *identify* the entity through one of the following: something known (e.g., a cryptographic key), something possessed (e.g., a smartcard), or something inherent (e.g., biometrics). We focus on the first type of authentication.

Ways to represent a user range from a human readable name to a cryptographic key. While at the protocol level identification is always key-based, the system we call *identity-based* system represents identities in a more human-readable form such as a name. Authentication and authorization in such systems are based on the entity's name. Alternatively, in a *key-based* system, a cryptographic key itself is used to represent the entity, thus requiring an authorization scheme to be based on a cryptographic key also. A mapping between the cryptographic key and the name might still exist, but it would hold a different meaning.

2.3 Related Work

In this section, we discuss the related work in the area of authentication. Authentication is hard problem. Many protocols have been published and at a later time weaknesses and flaws were discovered in them, leading the research community to establish better guidelines for protocol designs and also turning to formal reasoning and protocol analysis.

2.3.1 Formalizing Authentication

Meadows [80] identifies four major categories of protocol verification methodologies: (i) use of specification and verification tools, (ii) use of tools based on experts systems to simulate different scenarios, (iii) model the protocol requirements using logic, and (iv) use algebraic term-rewriting properties of cryptographic protocols to model protocols. Many publications [61, 81, 105, 137] survey the research literature on each of the methodologies.

1. **Specification and Verification Method**. Authentication protocols and cryptographic protocols in general have been analyzed by using specification languages and verifications tools. This approach aims at proving the correctness of a cryptographic protocol by treating it like any other program. One technique, suggested in [110, 132], is to represent a protocol as a directed graph. Another technique [61] uses machine-aided verification techniques to generate a proof that verifies the protocol expressed by state invariants. Overall, the drawback in this type of analysis is that it proves correctness but says nothing about security.
2. **Using Experts Systems**. Experts systems have been used to develop and investigate different scenarios [70, 84]. This approach creates a model of the authentication system described in terms of state and analyzes if

starting from an initial state, any of the undesirable states can be reached. While this approach is shown effective to find given flaws in the specified protocol, it cannot detect flaws of unknown types.

3. **Logics of Authentication**. The most popular approach to analyze authentication protocols is through the use of logic. Formal logic enables protocol designers to reason about authentication process by describing how participants' beliefs or knowledge changes during the execution process. The first such logic is due to Burrows *et al.* [15, 16], frequently called BAN logic. It uses predicate logic to analyze authentication protocols. BAN logic has been extended in variety of ways by [5, 46, 49, 59, 75, 113, 114, 120, 122]. The major criticism of the BAN logic can be found in [75, 92, 113, 120]. Nessett [92] showed a weakness in the logic by using it to prove correctness of a flawed protocol. Similarly, Mao *et al.* [75] have shown that BAN logic proved Otway-Rees protocol [95] to be correct while in reality it is flawed. Gaarder and Snekkenes [46] extended the BAN logic to deal with public key crypto systems.
   To evaluate a logic, it is necessary to study the its semantics and show its soundness and completeness as its been done in [5, 121]. However, many logics are either known to be incomplete or no proof has been provided on the completeness of the logic. Thus, it is not known if a particular logic is complete or not. Without the completeness property no formal proof of correctness can be done.
   Logics struggle with the subtle properties of protocols and formalization of the protocol requirements.
4. **Algebraic Modeling**. Another approach, pioneered by Dolev and Yao [33] and later studied by varies researches [82, 77, 78, 79, 80, 125, 135], analyzes authentication protocols through formal models that are based on the algebraic term-rewriting properties. Experts systems approach is very similar to the algebraic modeling. The difference is that, while formal modeling based on experts systems starts with an insecure state and attempts to show that no path could have originated from the initial state, the algebraic modeling tries to show that an insecure state cannot be reached.

Some methodologies are good for pointing out flaws in the protocols and some deal with formally proving either correctness of the protocol or that it meets protocol specification. However, none of the methodologies can prove that an arbitrary authentication protocol is secure. Meadows notes in [81] that "it is unlikely that any formal method will be able to model all aspects of a cryptographic protocol, and thus it is unlikely that any formal method will be able to detect or prevent all types of protocol flaws."

2.3.2 Semantics of Authentication

The meaning of authentication process has been explored in [3, 5, 48, 50, 73, 121, 135].

2.3.3 Design Principles

By drawing on years of experience of building security protocols, several researchers proposed an alternative approach to the formal analysis of authentication protocols and outlined design principles for building them [4, 7, 124, 134, 136].

Syverson [124] describes limitations of the principles proposed by Abadi and Needham [4]. He shows that every rule always has an exception.

2.3.4 Attacks on Authentication Protocols

There is an extensive body of literature on attacking authentication protocols [4, 7, 16, 19, 29, 56, 72, 123] starting from an attack on Needham and Schroeder protocol by Lowe [71] targeting asymmetric key protocol and Dennings and Sacco [26] attacking the symmetric key version.

2.3.5 Timeliness

Time plays a major role in authentication protocols. An authentication protocol is required to guarantee that the parties involved in the authentication process are present during the execution of the protocol. Timestamps, sequence numbers, and other types of nonces are generally used to assess freshness of messages. BAN logic [15] as well as other logics [119] has a special construct to reason about freshness.

As a solution to the attack on the Needham-Schroeder protocol, Denning and Sacco [26] suggest to use timestamps. The version of the protocol with timestamps is later adapted by Kerberos, creating a requirement for loosely synchronized clocks. However, Davis [22, 23] relaxes the requirement for synchronized clocks by slightly modifying the protocol.

In their design principles for authentication protocols, Abadi and Needham [4] talk about timeliness and note that many protocols have failed because of incorrect assumptions about the use nonces. Denning and Sacco [26] demonstrated a flaw in the Needham and Schroeder protocol caused by the lack of time constraints that allowed for an old session key to be reused after being compromised.

2.3.6 Naming

As the binding between a session key and time is important, Abadi and Needham [4] note that binding the identity to the message is also crucial, as demonstrated by several different attacks [71, 4].

Naming is one of the components of authentication systems. Naming principals in a scalable and secure manner has been attempted by many researchers [2, 11, 18, 35, 53, 68, 126, 39]. The designs range from global namespaces (X.500 global namespace) to decentralized local namespaces (SDSI/SPKI local namespaces).

Chapter 3

Symmetric and Asymmetric Authentication

Over two decades ago, Needham and Schroeder noted "that protocols using public-key cryptosystems and using conventional encryption algorithms are strikingly similar" [91]. The goal of this chapter is to outline the similarities between establishing identity through symmetric and asymmetric protocols and use it to build an interoperability framework.

3.1 Semantics of Authentication

In this section, we discuss the semantics of authentication. The meaning of authentication does not depend on the cryptography used to achieve it. In both cases, symmetric key and asymmetric key protocols have the same objectives. However, to formally reason about the two types of authentications we require slightly different aspects of formal logics. Also, at the end of an authentication process, the beliefs and knowledge of the participants in symmetric and asymmetric authentication differ.

3.2 Dimensions of Comparison

In this section, we present taxonomy of authentication protocols and identify symmetric and complementary properties of symmetric and asymmetric identification.

3.2.1 Naming: Identity Binding and Verification

A *credential*, $\mathcal{C}$, binds a name, $\mathcal{A}$, to the key, $\mathcal{K}$, and a lifetime, T, and can be represented as a triple, $\mathcal{C} = \{\mathcal{A}, \mathcal{K}, T\}$. From then on, whoever can prove the knowledge of the key (or the corresponding inverse) is assumed to correspond

	Symmetric	*Asymmetric*
Credential	$\{\mathcal{T}, \mathcal{A}, \mathcal{K}\}_{K'}$	$\{\mathcal{T}, \mathcal{A}, \mathcal{K}_a\}_{K_{ca}^{-1}}$
Proof	$\{\mathcal{T}\}_K$	$\{\mathcal{T}\}_{K_a}$

Table 3.1: **Representation of authentication credentials**. Symmetric and asymmetric credentials are cryptographic bindings of an identity to a key. A credential without the corresponding proof is useless.

to the name stated inside of the credential. Table 3.1 demonstrates structural similarities between credentials.

In the triple, the identifier can either be the name, $\mathcal{A}$, or the key, $\mathcal{K}$. If the key is used as an identifier, then it automatically provides a global namespace, assuming cryptographic keys are unique. In asymmetric authentication, a challenge for the public key is presented; upon success, a binding between the key and the name is considered. Given a public key, a name is looked up.

In identity-based systems, names serve as identifiers. In general, names lack uniqueness and, thus, require additional mechanisms – a naming convention – to establish this property. In symmetric authentication, given a name, a key is looked up. However, as in asymmetric authentication, a session key is verified first, then the name included inside the credential is retrieved.

Semantics of binding are the same. We consider authentication between distrusting parties that, first, are not assumed to prior known each other's identities, and second, require a trusted third party to provide identity proof. Thus, in symmetric and asymmetric authentication alike, credentials have to be vouched by an outsider.

Semantics of verification are the same. A credential can be verified by any entity that possesses the key (or the corresponding inverse key, in the asymmetric case) used to encrypt the credential. However, an asymmetric key-based credential is verifiable by anybody who has and trust the public key that signed the credential; a symmetric key-based credential is verifiable by a single entity.

3.2.2 Time: Lifetime and Freshness

Regardless of the cryptographic type, a credential has a validity period, or *lifetime*, associated with it. While short-lived credentials have to be periodically refreshed, long-lived credentials have to be checked whether or not they are still valid or have been revoked. This shows how the dimension of time is connected to network availability that is discussed later.

In the password-based (symmetric) authentication systems, while a password has a comparatively long lifetime, traditionally, each instance of an authentication protocol generates and uses short-lived credentials after verifying user's password in order to reduce the exposure of the long-lived key generated from the password. When a trusted third party generates a credential for the entity, it associates a short lifetime with the credential. Since credentials have

a relatively short lifetime the risk of compromise is small and thus credential revocation is not considered to be a problem.

In the public key (asymmetric) authentication systems, traditionally, the entity's credentials have long lifetimes and, thus, revocation mechanisms are required. The credential validity period is not (cryptography) type specific. A symmetric key authentication would have to face the same problem – dealing with revoked credentials – if it used long-term credentials.

3.2.3 Network Availability

A protocol is considered to be an *on-line* protocol if the protocol completion requires contact with a third party. Symmetric key authentication protocols are traditionally considered to be on-line and asymmetric key authentication is viewed to be off-line. First, we argue that asymmetric key authentication cannot be classified as off-line. When long-term credentials are used because, the validity status of those credentials has to be verified by revocation mechanisms requiring an interaction with a trusted third party. Second, the on-lineness property depends more on the timeliness property than cryptography type.

3.2.4 Granularity of Access

The *granularity of access* of the authentication protocol depends on whether or not a credential binds the key to two entities (sender and receiver) or just one (sender). We say granularity is fine-grained when a key uniquely defines the participating entities. We say granularity is coarse-grained when a key only identifies one party. Dynamic fine-grained granularity of access requires on-line system.

While in the symmetric authentication system a cryptographic key inside of the credential binds it to both of the parties involved in the authentication process, the credential for the asymmetric authentication binds the key to only one entity. However, the session key that is established during the authentication process uniquely represents the two participants. So the transcript of the asymmetric authentication is equivalent to the fine-grained symmetric credential.

3.2.5 Discussion

Figure 3.1 shows the three dimensions we discussed above. We describe the properties of each of the regions and demonstrate where the particular authentication methods fit.

None of the protocols can fall into the regions 1 and 4 with short-lived credentials and off-line server. To refresh short-lived credentials the trusted third party server must be available on-line. Also since fine-grained access requires involvement of an on-line server, the region 5 (off-line server and fine granularity) is not possible.

Traditionally, symmetric key authentication protocols have short-lived credentials while asymmetric key authentication protocols have credentials with

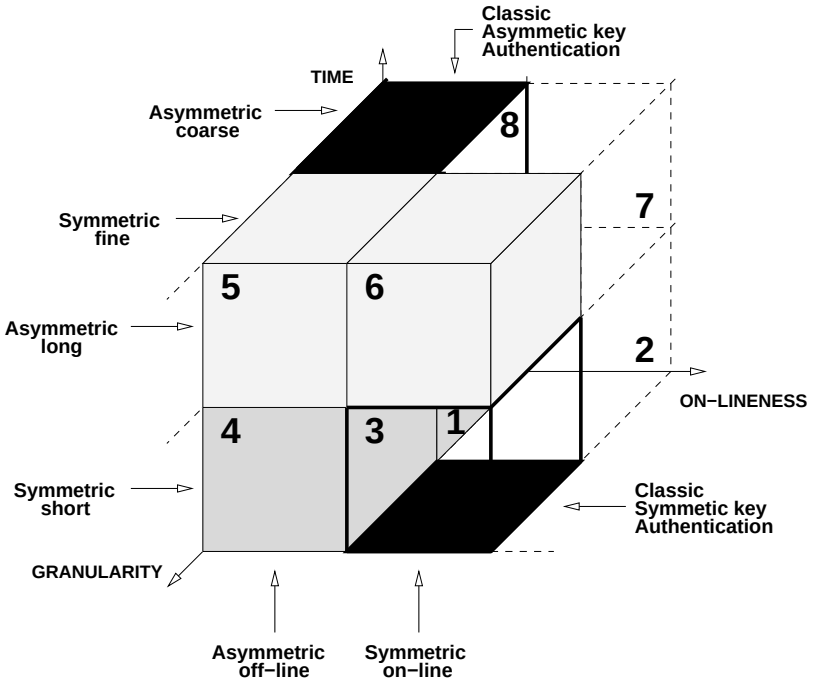


Figure 3.1: **Dimensions: Time, On-lineness, Granularity.** We show how authentication protocols can be categorized based on the three dimensions. We provide two valued ranges for each dimension. The time axis represents the credential lifetime. We consider either short-lived or long-lived credentials. The on-lineness axis represents the on-line property of the trusted third party. We consider off-line vs on-line. The values for granularity of access range from coarse to fine. The shaded area signifies that regardless of the granularity with the off-line scheme we cannot have short-term credentials. We show how traditionally symmetric and asymmetric authentication protocols fit into the proposed dimensions.

long lifetimes. The time property is more connected to the on-lineness than to the type of the cryptography used. By that we mean, regardless of the cryptographic type, time and on-lineness are tightly connected. Short-lived credentials require the on-line server to periodically issuing new credentials. Long-lived credentials could be viewed as having off-line properties. However, if we were to consider a reasonable threat model, where the potential for the credentials to be compromised increases after a certain time interval and additional credential validity checks or revocation checks needs to be enforced, some sort of an on-line server needs to be present in the system.

Inherently, symmetric key authentication protocols have fine granularity of access while asymmetric key have coarse-grained access. However, symmetric key protocols could be made less fine-grained and asymmetric ones less coarse-grained. If the same session key is used for all authentication (i.e., $\mathcal{K}_{ab} = \mathcal{K}_{ac} =$

$\mathcal{K}$), it makes the symmetric key protocols equivalent to the asymmetric protocol on the dimension of granularity of access. Additionally, a new session key for the following messages should be generated by one of the parties; the session key included in the credential should not be used for securing the channel.

One the other hand, if a public/private key pair is generated for each of the two participants and the binding between the public key and an identity also specifies the destination entity, the resulting scheme is equivalent to the symmetric protocol scheme on the dimension of granularity of access.

Needham and Schroeder (based) symmetric key authentication protocol(s) has the following characteristics: short-lived credentials, on-line server, fine granularity of access and fits into the region 3.

Needham and Schroeder (based) asymmetric key authentication protocol(s) has the following characteristics: short-lived credentials, on-line server, coarse granularity of access and fits into the region 8.

3.3 Future Work

The work of this chapter is the main focus of the future work. We need to provide a solid foundation for the interoperability between symmetric and asymmetric authentication systems. We outlined the major dimensions that we are going to compare the systems against. We also liked to consider properties such as trust and threat model. For the completion of the chapter we need to outline an interoperability framework and discuss mechanisms for interoperability.

Chapter 4

Kerberized Public Key Infrastructure

4.1 Introduction

Computer security is the practical application of cryptography. The two widely used types – symmetric key and (asymmetric) public key cryptography – are of the most interest and differ from each other, as the names suggest, on the key and method used for encryption operations. Symmetric cryptosystems use the same key (the *secret* key) to encrypt and decrypt a message. Asymmetric cryptosystems use one key (the *public* key) to encrypt a message and a different key (the *private* key) to decrypt it. In both cases, the fundamental tenet of cryptography is the secrecy of the private key.

Symmetric key cryptosystems suffer from a key distribution problem. How to securely communicate the secret key from the sender to the recipient in a tamperproof fashion? If the secret key can be sent securely, then, in theory, there is no need for the symmetric cryptosystem in the first place – because that secure channel can simply be used to send the message. Frequently, trusted entities are used to solve this problem. Public key cryptosystems provide a different kind of a solution. To communicate securely in a public key world, if message confidentiality is desired, a sender needs the receiver's public key and, if message authenticity is questioned, the receiver needs the sender's public key. The problem of key distribution is then reduced to managing public (vs. secret) keys.

A Public Key Infrastructure (PKI) is a key management system defined by the way it handles the following operations: certification, distribution, secure storage, and revocation of public key values. A process of certification produces a *certificate* that binds a public key to an entity, attribute, or permission. After the public-private key pair is created, the private key is securely stored with proper access control enforced. A distribution mechanism concerns with the availability of the public key to the world. It could be as elaborate as a key

management service that includes registering and querying of public keys or as simple as just relying on certificates for distribution. The last and the most difficult operation is the key revocation, frequently considered to a part of the key distribution mechanism. Revocation is concerned with up to date validity of public keys.

In theory, PKI offers great properties such as easy administration, better security, better geographical reach, and does not need a trusted key management infrastructure. On the other hand, end users must be disciplined in the art of key management. In reality, they are unwilling or unable to perform such a task since key management includes rigorous validation of every public key and stringent security of the long-lived private key.

Most widely known applications that use public key credentials are Web browsers and Web servers. In particular, secure connections on the Web are established with the help of public key certificates, but only server-side authentication is performed. In practice, client authentication on the Web is done with passwords despite the fact that strong authentication mechanisms are available but not used due to the difficulties associated with client certificates. Even in a closed environment, where there is a well defined set of clients and servers but where the number of clients is significantly larger than the number of services, the feasibility of managing client side public key certificates with existing PKIs is questionable.

In a traditional PKI, creation of a certificate requires an out-of-band communication of the public key that is being certified and an out-of-band authentication process such as a face-to-face meeting with an administrator issuing the certificate. The organization does not have to build a PKI from scratch but leverage of already existing infrastructure. We propose to extend Kerberos single signon mechanism to PKI through a service that creates a short-lived public key credentials based on Kerberos credentials and use these certificates for client authentication on the Web. By bootstrapping from an already existing authentication process, we eliminate the need for out-of-band communication. The lifetime of certificates reduces the certificate revocation problem which can further be handled by enforcing negative access controls.

The rest of this chapter is organized as follows. Section 4.2 discusses related work. Section 4.3 presents the design criteria of the system, followed by the protocol description and security analysis. Section 4.4 gives implementation details. Section 4.5 describes future research.

4.2 Related Work

In this section, we review the literature on the problem of certificate revocation, look at other single signon systems and general guidelines about public key infrastructures. The review of the research literature on the certificate revocation is presented to demonstrate the depth and complexity of the key revocation and to, indirectly, promote the use of short-term credentials as a solution to the problem.

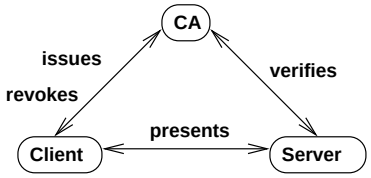


Figure 4.1: **Simple Certificate Revocation Model.** Three entities are shown: certificate authority (CA), client, and server. The CA's responsibility involves issuing and revoking certificates. A client presents a certificate to a server. The server's responsibility is to make sure the certificate is valid by verifying the revocation status of the certificate with the appropriate CA, or an entity – acting on behalf of the CA – responsible for managing information regarding the revoked certificated.

4.2.1 Certificate Revocation

Once the certificate is issued it is assumed to be valid until the stated expiration date. However, there are circumstances such as a loss or compromise of the private key that require the invalidation of the certificate before its expiration time. Revocation mechanisms specify who, how and when performs the process. Figure 4.1 shows a simple model of the revocation system. It contains three entities: (i) a certification authority that creates and revokes certificates by generating revocation lists, (ii) a client that uses certificates, and (iii) a server that verifies certificates. The location of revocation lists and the retrieval method characterize the type of revocation performed (*on-line* vs. *off-line*). Certification authority hides possibly multi-layered and distributed storage architecture of revocation information.

In [41] Fox and LaMacchia make a distinction between the *meaning* and the *mechanics* of revocation. The problem of what it means to revoke a certificate is shown by an example of how path validation during the certificate revocation process can be confusing. Authors show two different certificate chains for the same certificate where one of the certificates in the chain is revoked but the validation process leads to the conflicting results. Authors speculate that it is unclear what might the proper client's behavior be in case of revoked certificates. Answers reside in the area of risk management and policy framework, the discussion of which is beyond the scope of this work. In this section we discuss the *mechanics* of revocation and review different certificate revocation schemes.

A comprehensive background about certificate revocation is presented in [10], where a framework for comparison of revocation schemes is developed, schemes are analyzed, and guidelines for selecting a revocation solution is presented. The schemes were compared against the following properties: performance, timeliness, scalability, security, standards compliance, expressiveness, scheme management, and on-line vs. off-line.

Certificate Revocation Lists (CRLs) scheme [52] is viewed as the traditional certificate revocation scheme and it is based on the idea of *black-listing* credit cards. In this scheme, CRL is a certificate, issued and signed by the CA. It

contains a list of serial numbers of revoked certificates along with a timestamp and some other information. CRLs are issued periodically and distributed to interested parties that cache the information. As more and more certificates are revoked, the size of the CRL grows and becomes expensive to manage and distribute. To improve on the size of the CRLs other mechanisms were proposed:

- CRL Distribution Points: the CRL is divided into segments and distributed between *distribution points*. Now, only a relevant part of the whole CRL is downloaded.
- Delta CRL [51]: only the incremental changes are digitally signed and posted.

In order to reduce the cost of CRL processing, our researchers proposed that not all CRL is retrieved. Instead, a proof of validity is issued about the individual certificate on per request basis.

- Micali's Certificate Revocation System (CRS) [83]: each certificates is signed, stating its validity using the on-line/off-line signatures. This token can either be used directly by clients as a proof of validity or retrieved by the verifier.
- Kocher's Certificate Revocation Trees (CRTs) [64]: instead of singing each individual certificate, a tree structure is created. Each leaf in the tree is a statement regarding the validity of a particular certificate. Parent nodes are computed as a hash of child nodes. A path from the root to the leaf is used as a short proof.
- Naor's and Nissim's authenticated dictionaries [90]: extends CRTs scheme by using 2-3 trees instead of binary trees so that all pathsfrom the root to the leaves have the same length and node operations are done in logarithmic time.

The main problem with all of CRL-based techniques is low timeliness. Some researchers argue that in face of high-valued financial transactions an on-line verification is needed [42, 88]. In general, systems cannot tolerate the revocation delay inherent in CRL schemes and require real-time revocation checking. Alternative solutions to CRLs include On-Line Certificate Status Protocol (OCSP) [89] and Simple Certificate Verification Protocol (SCVP) [74]. In on-line approach, a client queries the server for a validity status for each certificate it needs to verify.

McDaniel and Rubin [76] investigate CRL usability in multiple environments and show where CRLs are appropriate and where other revocation mechanisms are needed. They state that revocation will remain a necessary part of any PKI.

Previous work acknowledge that use of short term certificates reduce the problem of revocation but argue that it is too expensive to re-issue certificates, thus they continue to work on revocation mechanisms. We show that in our system creation of short term certificates is easy.

4.2.2 Single Signon Systems

The growing popularity of public key cryptography has produced the desire for its integration with Kerberos. As a result, a protocol PKINIT [131] allows a user

to use public key credentials in the initial Kerberos authentication thus bootstrapping Kerberos from PKI. Public key distributed authentication (PKDA) [112] goes a step further and proposes for Kerberized services to support PK authentication mechanisms. Such integration requires modification of already existing software and seems unlikely to be adapted.

Schiller and Atkins [108] proposed to use Kerberos to sign PGP keys and to help build the *Web of Trust*. Individual PGP keys are signed by the Kerberized Key Signer Service. All users trust the public key of the Key Signer. It is used as the *introducer* between untrusted users. It provides and vouches for the binding between the name and the PGP public key. By using the Key Signer, one-level hierarchy is introduced as opposed to the original mesh structure of trust relations.

The most relevant to our work is a project at MIT and, then following their example, other institutions have a scheme to create long term client certificates by interfacing with Kerberos as the registration authority to issue the initial client certificate. As we mentioned before, the biggest problem with long term certificates is revocation. If certificates live long enough to be comprised, due to a lost or compromised private key, currently no efficient and satisfactory way is known to revoke such certificate.

A different type of single signon is used for Web authentication and is based on the idea of using a single user id and password to login to multiple Web site. Passport [96] is a protocol developed and deployed by Microsoft. It is an example of a password-based single signon scheme that leverages of existing Web technologies such as HTTP redirects, Javascript, cookies, and SSL. Kormann *et al.*[66] discovered several security flaws and attacks against the service such as incorrect logout procedure that failed to remove user's credentials, bogus merchant and login servers that trick users giving away their authentication information, and misuse of cookies.

Netscape also provides a single signon scheme described in [111]. In this case, single signon means entering user id and password to unlock the database where the private key is store and then use the key and the corresponding certificate during authentication on the Web. A few other products (e.g., eTrust[1], Novell[2], Okiok[3], i3sp[4]) exist that with a single entry of a password allow users access multiple services without re-authentication for each one. In general, the main problem with password-based schemes is that users choose weak passwords.

A single signon environment for NetWare and Kerberos users is created by [6]. The goal of that work is to join the two realms such that administrators are relieved from managing multiple user sets. Both administrative domains require a user to present a password in order to receive services. Relationship between the two password databases has been reviewed. Two schemes were presented to provide a single signon. The first scheme is based on keeping both passwords the same. The other scheme used Kerberos password to derive NetWare password

[1] http://www.etrust.com
[2] http://www.novell.com
[3] http://www.okiok.com
[4] http://www.i3sp.com

so that, at the login time, the user is signed in two both domains.

Recommendations and general properties of certification infrastructure has been studied in [39, 62, 103, 104, 129]. It's been shown that setting up a CA infrastructure is a complex task due to numerous technical and policy issues.

4.3 Design

4.3.1 Design Criteria

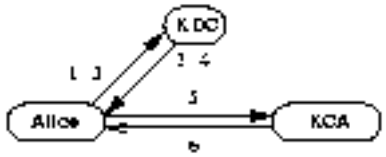
In our design we follow two of the three objectives for the certification authority services that were provided in [103]. We agree that (i) it is important to ensure that a certificate will only be issued if the requestor proves that he knows the corresponding private key. We question but apply the same solution to the other objective that is (ii) to limit the adverse consequences of malicious action by a CA. The last objective is to ensure that CA can be held accountable for malicious behavior and is beyond the scope of our work.

We state the design goals, general security principals, and design choices for our certification infrastructure.

- **Goal(s)**: The main goal can be stated as to create a public key infrastructure that avoids the inherent naming and key management issues of previously proposed architectures. We strive to provide a lightweight solution that enables a single signon to two authentication domains: Kerberos and Public key based systems (e.g., SSL).
- **Principle(s)**: To achieve previously stated objective (i) public keys must be transmitted to the CA in such a way that the CA can be sure that the key has not been modified in transit, and that the user requesting the certificate is the same person who caused the key to be generated (corresponding private key is known to the identified user); and (ii) private key should be stored locally and should never be divulged to any other entity.
- **Choice(s)**: To achieve the goals we chose to (i) build the infrastructure from already existing system – Kerberos – and, thus, avoid management of a new global namespace; and (ii) issue short-term certificates, thus reducing the problem of certificate management to: secure creation and store of certificates.

One of the concerns noted in the literature is the creation and management of the global namespace [18, 20, 68, 101]. In particular, CA must ensure uniqueness of distinguished names used in the certificates. In our design, by building the infrastructure from Kerberos, we also off load the naming to Kerberos. Since within Kerberos domain client ids are assumed to be unique, then by using the same identities in the certificates we also guarantee the uniqueness of distinguished names.

As in case of a regular certification authority, a care must be taken to keep the CA's private key secure.



1-4	Kerberos login
5. *Alice* → *KCA*:	TKT, Auth, K_{pub}, $\text{MAC}_{K_{A,KCA}}(K_{pub})$, $(\text{Auth})_{K_{priv}}$
6. *KCA* → *Alice*:	X.509 certificate, $\text{MAC}_{K_{A,KCA}}$(X.509 certificate)

Figure 4.2: **KX.509 protocol.** Steps 1-4 from Kerberos are not shown. Steps 5 and 6 give the details of messages in KX.509. *Alice* sends a service ticket (TKT), an authenticator (Auth), a public key (K_{pub}), and its MAC. To prove that she knows the corresponding private key (K_{priv}), *Alice* uses it to sign the authenticator included in the Kerberos part of the message. A keyed digest is based on the session key, $\text{K}_{A,KCA}$ and prevents modification of the data.

4.3.2 Protocol Description

We propose a Kerberized service that creates a short-lived X.509 certificate [34, 67]. The exchange of messages and other details of the protocol are shown in Figure 4.2. As in Kerberos, *Alice* gets a TGT from the KDC. To acquire an X.509 certificate, she first requests a service ticket for a Kerberized Certification Authority, KCA. At the same time, *Alice* generates a public/private key pair and prepares a message for the KCA. Along with the public key, she sends the KCA service ticket, $\{Alice, \text{KCA}, \text{K}_{A,KCA}\}_{K_{KCA}}$, and an authenticator, $\{\text{T}\}_{K_{A,KCA}}$. To ensure that the public key has not been tampered with, the MAC of the key is sent in the same message. The session key, $K_{A,KCA}$, is used to compute the MAC of the key. To insure that *Alice* has the corresponding private to the public key used in the message *Alice* signs the timestamp already present as the Kerberos authenticator, $\{\text{T}\}_{K_{priv}}$.

The KCA authenticates *Alice* by checking the validity of the ticket and the authenticator. It verifies that the public key has not been modified. The KCA then generates an X.509 certificate and sends it back to *Alice*. The certificate is sent in the clear; to prevent tampering, the MAC of the reply is attached. The lifetime of the certificate is set to the lifetime of the user's Kerberos credentials. The user's Kerberos identity is included inside the certificate, creating the necessary binding.

4.3.3 Security Analysis

The security of this protocol is based on the security of the Kerberos authentication protocol, the security properties of one-way hash functions, and the

signature scheme used. In our design, we decided to address the following potential threats (presented in [103]):

- **Masquerade**: occurs when a user requests a certificate or a CA issues a certificate and pretends to be a different entity than it really is. We rely on Kerberos authentication of both parties to detect and prevent impersonation.
- **Modification of data**: prevent unauthorized modification of certificate contents.
  - **problem**: modification of certificate contents during transmission.
    solution: use an integrity protection mechanism, e.g., cryptographically secure hash functions.
  - **problem**: modification of stored certificates.
    solution: employ the protection mechanisms of the underlying storage, e.g., file system protection (ACL).
  - **problem**: modification of security attributes prior to being packaged in a certificate, e.g., LDAP identity lookup.
    solution: use secure mechanisms (e.g., Kerberos protected LDAP service).
- **Loss of confidentiality**: anyone in possession of a user's private key can authenticate as that user. Thus, it is important to generate the private key on the client machine and not at the CA site. Also, it is vital to preserve the confidentiality of private keys. This is related to the previously stated threat of modification of stored certificates and same protection mechanism applies. Loss can occur if the key generation process is compromised. To protect against this threat a good source of randomness has to be present on the client machine.

Threats considered in [103] but not discussed here include false repudiation and exceeding authority. The issue of repudiation comes up when a private key is used to sign some information. Since in our system the lifetime of the keys is short, the notion of reputable actions is not well defined. The case of exceeding authority refers to when the CA either issues or revokes a certificate it has not been authorized to do so. While the issue of revocation is not relevant in the proposed system, an unauthorized issue of certificates is not addressed by our design.

The man-in-the-middle attack is prevented by the use of Kerberos and MAC. The denial of service attack is a serious threat and an extensive research has been done in this area. We do not attempt to guard against such attacks.

4.4 Implementation

We implemented the KX.509 protocol to work for both Netscape Navigator (on UNIX, Windows, and Mac OS) and Internet Explorer (on Windows). The `kx509` client and the KCA server are the two basic components involved in issuing user certificates. Navigator maintains a private cache of certificates, but the implementation is platform dependent, undocumented, and version dependent.

```
$:  klist
Ticket cache:  FILE:/tmp/krb5cc_500
Default principal:  aglo@CITI.UMICH.EDU

Valid starting      Expires             Service principal
02/01/02 09:51:43   02/01/02 19:51:44   krbtgt/CITI.UMICH.EDU@CITI.UMICH.
                                        EDU
02/01/02 09:51:47   02/01/02 19:51:44   kca_service/zingara.citi.umich.
                                        edu@CITI.UMICH.EDU
01/31/02 10:51:50   02/01/02 20:51:44   kx509/certificate@CITI.UMICH.EDU
```

Figure 4.3: **Output of klist.** KX.509 certificate and the private key are stored in the Kerberos V ticket cache under the service names of `kx509/certificate`. `kca_service` is the service ticket for the KCA. The other entry is the service ticket for the TGS.

Thus, we elect to save certificates in user's Kerberos ticket cache, which requires the user to add a cryptographic module to the browser. No such modification is required for Explorer. Instead, in order to support both browsers on Windows environment, user certificates are stored in the registry.

Typically, a ticket cache stores a user's TGT and service tickets. MIT's implementation of Kerberos on UNIX allows for variable size tickets, allowing us to store any data of size up to 1250 bytes, which is sufficient to store a certificate and a private key. Figure 4.3 shows the output of the `klist` command, which displays the current contents of a ticket cache. The entry `kca_service/zingara.citi.umich.edu` is the service ticket for the KCA. `kx509/certificate` contains the user's certificate and private key. Storing certificates in a ticket cache has the advantage that when a user logs out from the computer the data stored in the ticket cache is destroyed.

As we mentioned, Navigator needs help to find our certificates. To this end, we use the browser's standard interface to add a cryptographic module that we call `kpkcs11`. When client authentication is required, `kpkcs11` looks up a certificate in the ticket cache and gives it to Navigator.

In our implementation, the user identity information that KCA includes in the certificate is retrieved from a naming service (an X.500 directory). Given a Kerberos principal, KCA looks up the user's first and last name. Additionally, at the end of the distinguished name we attach an email field with the principal name in the local part and the realm in the remote part, for example, `aglo@CITI.UMICH.EDU`.

4.5 Future Work

In this section we outline the scope of future research. For completeness a performance study of the KCA is needed, discussed in Section 4.5.1. Section 4.5.2 proposes to address the problem of namespace management. Section 4.5.3 proposes to extended current functionality of the KCA to issue attribute certificates.

Total num	IMAP reqs	Dialin reqs	Web reqs
605,477	452,773 (74.78%)	63,529 (10.49%)	8,786 (1.45%)

Table 4.1: **Kerberos logs**. For the day 10/13/99 we identify how many AS_REQs were made. Out of the total number of request, we show how many requests came from the following servers: IMAP server, Dialin server, and Web login server.

Finally, Section 4.5.4 proposes to create a comprehensive key management service that includes registration and revocation options.

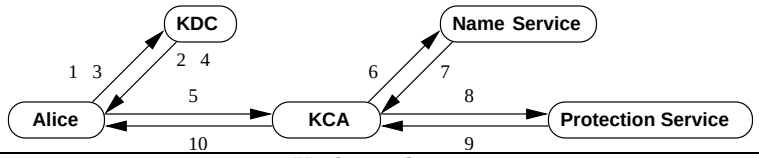
4.5.1 Performance

Problem statement: We propose to study the performance of the KCA service.

We intend to evaluate the usefulness, to test and point out possible problems with the current design and implementation of the system. Two questions to be answered are: (i) what is the throughput of the server, and (ii) what is the client response time. The intended use of the KCA is one request per day per user. University of Michigan has about 50,000 users, so we need to be able to support that. For a better estimate, we retrieved logs from the *umich* KDC and measured the number of AS_REQs that correspond to ticket granting ticket requests. Table 4.1 shows a summary of AS_REQs from one of the *umich* KDCs. Just as an example, on 4/13/99, we counted 605,477 AS_REQs. Not all all those requests correspond to users' initial login. We have to separate users' TGT requests from the TGT requests issued by services that use user's passwords to complete the requests. For example, a total of 525,088 requests come from email (IMAP), Dialin, and Web login servers. Requests that come from IMAP servers are due to misconfigured email client applications that run on Windows and Macs. They are configured to prompt the user for his password and then send the password in the clear to the IMAP server. UNIX email applications are Kerberized. Assuming proper behavior from email clients and the use of X.509 authentication on the Web, the KCA needs to handle 143,918 AS_REQ requests per day. By looking at the logs, we can show the distribution of AS_REQ through out the the day.

We will compare the performance of the KCA to the performance of a KDC. We will also compare the performance of the KCA to the performance of another certification authority, understanding that it is often difficult to compare against commercial version of the software (and the majority of CAs happen to be commercial products).

We predict that public key operations to be the bottleneck in the KCA performance. In this case, hardware accelerators can be used to boost the performance. Additionally, to better handle the load replication of the service should be considered. Both issues will not be considered in this work as they have been thoroughly studied elsewhere.



1-4	Kerberos login
5. *Alice* → *KCA*:	KCA_TKT, Auth, K_{pub}, $\text{MAC}_{K_{A,KCA}}(K_{pub})$, $(\text{Auth})_{K_{priv}}$
6. *KCA* → *Name Service*:	Kerberized LDAP query for *Alice*
7. *Name Service* → *KCA*:	Identity Attributes (e.g., full name)
8. *KCA* → *Prot Service*:	Kerberized PTS query for *Alice*
9. *Prot Service* → *KCA*:	Group membership, Credentials
10. *KCA* → *Alice*:	Certificate(s), $\text{MAC}_{K_{A,KCA}}$(certificate(s))

Figure 4.4: **Extended KX.509 protocol.** The basic protocol presented in Figure 4.2 is extended to query additional identity information from a naming service (e.g., LDAP database), and query a protection service (e.g., AFS PTS) for group membership or authorization information. Steps 6-9 are Kerberos authenticated.

4.5.2 Naming

Problem statement: How can we accommodate differences in the namespace format between PKI (X.509 format) and Kerberos (UNIX id)?

Naming of entities is a crucial component of a secure distributed system. Entities in a system require a naming convention and a mechanism to verify or establish identity. The format and verification process go hand in hand. We propose to investigate namespace management problem by evaluating the current scheme, looking for alternatives, and extending the scheme to include more flexible and expressive namespace. We noted earlier that instead of creating a new namespace, we borrow one from Kerberos. This solution offers the least amount of flexibility. Thus, a more PK-like identity format might be desired for the certificate. Figure 4.4 shows an extension to the KX.509 protocol that makes use of other services to create a client certificate. The solution that we currently implement is: upon receiving a request from the client, the KCA contacts a directory service and retrieves additional information about the Kerberos principal. The KCA Kerberos authenticates to a directory service and using LDAP retrieves (world-readable) information about the principal.

KCA's responsibility is to bind a public key to an identity. However, since we are starting from a Kerberos identity, the binding between Kerberos identity and PK identity might be provided by a service different other than KCA. We propose to separate the two binding processes allowing for greater flexibility, separation of responsibilities, and reducing the complexity and load of the KCA. So, another possible solution that we plan to explore is to allow the client to

present the identity information to be included in the certificate. Here, the client's identity information is vouched for by some other service, e.g., it might be signed by a naming service. In this case, a security policy must exist at the KCA to enforce the trust relationship between the naming service and the KCA. This solution allows for greater user information privacy. Users should be able to set a policy that specifies who has access to their personal information.

Questions to consider:

- As a solution, building and populating a new database with information allows us greater flexibility in choosing security options. However, bringing up a new database has many disadvantages: duplication of work, keeping the information in both databases consistent, expensive and, in practice, unrealistic to deploy.
- Access control mechanisms of the existing directory services support restricted access. For a given attribute, a list of users or applications is specified along with access level and the authentication method required to access this attribute. However, the directory service does not support the functionality of certifying the identity information. Making modifications to a complex software is not an easy task.

Alternatively, we might make no assumptions about the meaning of the identity information present in the certificate, the client's self identity (nickname) could differ from the registered identity.

Problem statement: What is the relationship between the Kerberos and PK credentials? Should a Kerberos realm correspond to a (Kerberos defined) PK domain or should a more complex structure be considered?

In an environment such as University of Michigan, a single KCA can serve multiple administrative domains. Issues to be considered include: uniqueness of user ids. It is assumed that uniquenames are unique within the university, e.g., *aglo@engin.umich.edu* and *aglo@umich.edu* are the same principal. It is not clear if this property should be assumed in general. In this case, the KCA's client identification responsibilities should be clearly identified by the security policy. From the authentication step, the KCA is aware of the client's originating domain. This domain might be either used in the certificate, or a generic domain (e.g., *umich.edu*). The two policies reflect different threat models. Using the latter adds to the KCA's responsibilities to check for uniqueness of principal identifiers but relieves services that use the certificates from this burden. This presents an advantage of enforcing the check at one place rather than having each of the end services do it. On the other hand, this precludes fine-grained access control based on the user's actual domain. In effect, a generic domain in the certificate provides a single signon to multiple domains which might not be desirable in all cases.

4.5.3 Attribute Certificates

Problem statement: Issuing general attribute certificates is well understood. However, issuing certificates with protected (i.e., encrypted) information and

specifying a way to revealing that information securely to authorized parties is not so well studied.

We propose to extend the KX.509 protocol to issue other types of public key based credentials, e.g., attribute and role-based certificates, KeyNote assertions [12, 13, 14], and SPKI/ SDSI certificates [20, 36, 101]. An application that requires attribute certificates is described in Chapter 6, where authorization decision is based on a signed group membership statement received with the request. We propose to investigate the feasibility and usefulness of *protected* certificates.

In some cases, the user's identity is confidential and needs to be protected during transmission. For example, in SSL, both user and server certificates travel in the clear. But suppose we produced a secret identity certificate whose *subject* field is encrypted with a key that is revealed only when needed. The KCA encrypts the identity with a symmetric key which is in turn encrypted with the client's public key and returned to the client along with the certificate. We propose to investigate the ability to use such certificates by Web browsers and servers and how the symmetric key can be revealed to the server. Two possibilities suggest themselves. First, a client can reveal the key. Second, a server can present this special certificate to a server and request that the identity be revealed.

Other confidential attributes might need to be protected by encrypting the information. Imagine an attribute certificate in which some of the attributes are encrypted, possibly under different symmetric keys. The same certificate can be presented to different services and different keys can be disclosed to get access to resource while not disclosing full capabilities to each of the individual services and not requiring users to acquire, store, and use multiple certificates/capabilities.

4.5.4 Key Management Service

Problem statement: Is it feasible and practical to do short-term key management operations other than key creation?

Another possible avenue for future work is to extend the work to provide a comprehensive public key management service that includes registration, retrieval, private key recovery and key escrowing. The architecture for such service is not straightforward. It would a mistake use short-lived certificates for digital signature because of the short lifetime. Their usefulness is further impaired by the fact that the user's public key might have not yet been registered if the user has not logged in that day. Also, to make sure old keys are not used during the retrieval query, the logout procedure must always request that the current keys be revoked.

With short-lived certificates it is hard to send secure email. If a certificate is only valid for a day, then the recipient will be unable to verify the signature or unseal encrypted message after the certificate expires. We propose to investigate solutions to solve this problem. One idea could be to have long term public and private keys stored at a server and retrieved when needed with the use of short

term keys. The drawback of this solution is that the certification authority has to be trusted in the same manner as the KDC and eliminates non-repudiation because the assumption about the private key has changed. We propose to investigate the potential of this design as a solution for long term keys and digital signatures. The question of where and how the long term private key is used needs to be address. Public keys could be retrieved from the server and used to verify signatures or encrypt data. Decryption and verification could be by the server, so that private keys do not have to leave the protected server. Such a server may become a bottleneck and an attraction for hackers.

A different solution we propose to explore is to build long term keys from short term keys by reusing the same public and private key and re-validating them each time a user logs in. A created certificate still has a short lifetime but the public key could be registered with a server and used. A key server can assign a probable life to the key-based on its previous status. Then when somebody queries the server for the public key, the CA issues a certificate with this probable lifetime and returns back to the requestor. This presents a dilemma: should a client elect to reuse the same key pair, where should those keys be stored?

Another idea is to have a key server provide multiple keys that differ in their security properties. For example, one type could be short-term keys which are the most secure keys. Every time the user logs in, he registers the new junk keys. These keys can only be used when a user is logged in. They have the highest value of freshness and are least likely to have been compromised. On the other hand, the server can store a pair of long term keys which are least/less secure keys depending on where the private key is stored and how frequently it is used.

Chapter 5

Kerberized Credential Translation

5.1 Introduction

Access control for Web space is often viewed in terms of gating access to Web pages where the job of the Web server is limited to simple file reads. The functionality provided by Web servers has grown considerably making it the most popular technology on the Internet. With the expansion of the Internet, many new kinds of services are accessible from the Web, increasing Web servers' importance and scope. For example, a Web server may serve information stored in backend databases. A Web interface to backend services is considered to be more user-friendly and accessible compared to predominant text-based interfaces.

The possibilities opened by the use of a Web server to access a variety of backend services pose challenging questions on how to retain access control of backend services. A Web server could potentially become another access control decision point, increasing the burden on the server and its administrators. It would have to comply with the same security requirements as all of the backend services it fronts, increasing its potential as a place for system compromise.

A solution that provides end-to-end authorization would allow the end service to retain control over the authorization decisions. Furthermore, it would obviate constructing and maintaining consistent replicas of authorization policies.

In practice, authorization mechanisms are tied to authentication mechanism: end-to-end authorization requires end-to-end authentication. A mismatch in authentication mechanisms prevents a Web server from using authorization mechanisms provided by backend servers. While Web servers support SSL authentication with certificates, this does not provide credentials for access to AFS file servers, LDAP directory servers, and KPOP/IMAP mail servers, which use Kerberos for client authentication. To provide end-to-end authorization, we address the problem of end-to-end authentication.

We motivate the end-to-end authentication problem by considering the following scenario:

Alice attends the University of Michigan, where she enjoys access to a variety of computing services. One of the most commonly used services is AFS file service, which is protected by Kerberos. Alice, being a very private person, does not want others to have access to her files. Through the access control mechanisms provided by AFS, she limits access to specific users. But if these users prefer to access Alice's files through the Web, then the flexibility of AFS access controls disappear.

Web presence for other Kerberized services also suffers. For example, Alice would like to manage her umich.edu X.500 directory entry from a browser. The directory is stored in an LDAP directory that uses Kerberos authentication to control read and write access. Alice would also like to read mail from a browser; this too requires that the Web server authenticates as Alice to the Kerberized mail server.

If an AFS client is running on Alice's workstation, a simple solution presents itself. Instead of making an HTTP request, a user can access AFS file space directly with `file://localhost/afs/`$\cdots$. But it is fair to say that most machines do not run AFS. Also, the solution fails to provide a general mechanism for accessing services from the Web; browsers cannot anticipate all possible service access types.

In this scenario, end-to-end authentication presents the question of how to convey Kerberos credentials to the Web server. One solution is for the client to acquire the needed credentials and delegate them to the Web server. A frequently used solution is to send a Kerberos identity and password through SSL, but this gives unlimited power to the Web server to impersonate users, a significant risk. It is also hazardous to expect a user to know when it is safe to give her password to a Web server.

Kerberos supports a mechanism for delegation of rights. However, browsers do not support any form of delegation. A practical solution is needed that works with existing software and is easy to deploy, administer, and maintain. The process should demand minimal interaction with a user, providing transparent access to resources. To limit misuse of user's credentials, the Web server must be constrained in its actions. Furthermore, a central, easily administered location for enforcing security policies controlling the Web server's actions is required.

While many backend services use Kerberos for authentication, Web servers use SSL to authenticate with public key cryptography. We address the mismatch of authentication credentials between the Web server and Kerberized service by introducing a new service that translates PK credentials to Kerberos tickets. The Web server engages in proxy authentication. The process consists of SSL client authentication, a request to a credential translation service, and finally authentication to the Kerberized service on a user's behalf.

This chapter is organized as follows. Sections 6.2 and 5.2 provide background material and discusses related work in the area of web access control. Section 6.4.1 presents an architecture for access to Kerberized services through a browser. Section 5.3.4 gives a security analysis of the systems. Section 5.4 gives implementation details. Section 5.5 describes performance.

5.2 Related Work

This section describes related work on distributed authorization and interoperability among authentication mechanisms. Many efforts have focused on creating formal systems that allow reasoning about delegated (restricted) rights and express (general) authorization statements. Many researchers have focused on creating powerful and expressive languages for making and verifying security assertions efficiently. Among them are SPKI/SDSI [20, 36, 101], PolicyMaker, and its successor KeyNote [12, 13, 14], GAA API [106], Akenti [128], and Neuman's proxied authorization [93]. Applications that lack an authorization mechanism of their own greatly benefit from these mechanisms. However, our goal is to make use of already existing authorization mechanisms at the backend services.

Authors of capability file names [100] propose to create and distribute to potential users capabilities that include the name and rights to the file given. In order to get access to a file, a user submits the received capability that is verified by a proxy daemon that gates access to the file systems. They do not carefully discuss the issue of initial distribution of capabilities. Instead they assume an out-of-band secure mechanism such as secure email would be an acceptable solution.

There is a simple alternative solution to enable the Web server to act on a user's behalf. A user can send his password (securely, of course) to the Web server. The solution has been implemented as an Apache module [8, 118, 115]. In this case, the Web server is given an unlimited power to impersonate users, a significant security risk.

There are several projects that propose to use Kerberos for Web authentication without sending user passwords. Minotaur [85] depends on a client side plugin to acquire a service ticket for a Web server. However, it has been shown that, in its current design, Minotaur's handling of HTTP POST is insecure. Another system, called SideCar [99], achieves Kerberos authentication by talking to a dedicated process on a client's machine. Failure to start the daemon process prevents the client from being able to do Web authentication. Yet another solution makes use of the extension to TLS cipher suites that includes Kerberos as an authentication mechanism [55].

Kerberos authentication to a Web server is not enough for end-to-end authorization. There must be support for delegating Kerberos credentials after the client authenticates to the Web server, which is addressed by Jackson *et al.* in their proposal on how to delegate credentials (currently, Kerberos and X509 certificates) in TLS [58]. There are a few problems with considering this approach as a solution. First, no browsers currently support Kerberized TLS. There is an implementation of Kerberized TLS [116] that relies on a local proxy, but browsers are often limited to a single proxy, complicating system management. Furthermore, the description of the exact content of the protocol is vague, making it hard to validate the security of the protocol.

Tuecke *et al.* [130] propose a specific delegation mechanism that allows a user to delegate an identity certificate to a third party. The receiver must engage in a special verification process that validates these certificates to identify the real

sender. Authentication to a commodity server with these certificates cannot be considered secure, as each entity in the delegated path serves as a certification authority and can create a certificate under whatever identity it pleases.

The problem with delegation is that the client may be tricked into requesting a ticket by a rogue server. It has been repeatedly demonstrated that we cannot always trust a valid server's certificate, most recently by the Microsoft/VeriSign debacle [87]. Delegation places a large administrative burden on the client. First, a client must be able to understand and apply security policies to determine whether or not to forward his credentials. To avoid the hassle, users frequently allow for unlimited and unchecked delegation. It is not reasonable to assume that for each compromised Web server each user will update her security policy to address the problem. Lastly, browser support for restricted delegation always leaves us wishing for more.

5.2.1 Performance Studies

Apostolopuolos *et al.* [9] investigated the cost of setting up the SSL connection. For small HTTP transfers the overhead from the SSL handshake is significant but for large requests (1Mbytes or more), the overhead is mostly due to encryption and authentication. Client authentication was omitted so the results do not reflect an additional private key encryption operation. Furthermore, a self-signed server certificate was used as opposed to a certificate signed by a certification authority. In this experiment the verification of the CA's certificate was not accounted for. SPECweb96 benchmark was used to assess the performance. A regular Web server (Apache 1.3) can handle about 250 requests per second while the a secured server can only handle about 85 (with 100% session keys reuse). Caching of certificates and modified SSL protocol that reduces the number of round trips is presented and evaluated.

Further improvements (15% to 50%) are achieved by caching and reusing SSL session keys [47]. 10ms is required to establish a TCP connection and 40ms for a new SSL connection or 10ms for the reused one. Distributions of time spent on either TCP, non-cached SSL, cached SSL, or HTTP GETs connections for several Web sites are presented.

To speed up performance several designs [25, 86] proposed to offload the RSA operations to a dedicated server with specialized hardware.

Coarfa *et al.* [21] analyzed the performance of an SSL-secured Web server (Apache 1.3 with mod_ssl 2.7 based on OpenSSL 0.9.5a). The experiments studied the effect of the CPU speed by varying the speed between 500MHz and 933MHz. Some of the results are summarized in Table 5.1. Two workloads were studied. One trace was taken from an e-commerce Web server with the mixed (secure and insecure) traffic of the average size of 7KB and with the estimate of one full handshake to every twelve requests. The other trace was from the departmental Web server assumed to serve only secure documents of the average size of 46KB. Not surprisingly they note that overall performance of the Web server is drastically reduced while servicing secure Web pages. In the e-commerce trace the largest performance cost was due to the public key

operations. For this case hardware accelerators shown to be extremely effective (over 100% improvement). However, in the data transfer trace while the cryptographic operations were the dominant cost, there was a factor of 2 difference in times spent doing the RSA and at the same time the amount of non-SSL related costs were doubled. The effectiveness of the faster CPU also depends on the workload. The data transfer trace benefits greatly from the increase the CPU speed as opposed to the use of the accelerator.

Trace	Apache (ops)	Apache+SSL (ops)	RSA (%)	non-SSL (%)
e-com	1370	147	58	10
data	610	149	23	29
e-com	2200	261	57	12
data	885	259	20	32

Table 5.1: **SSL Performance.** The summary of results presented in [21] are presented as two workloads (e-com and data) evaluated on 500MHz shown in the first two rows and 933MHz in the other rows. The first two columns show the number of hits serviced by the Web server. The second two columns represents the percentage of time spent doing specified operations.

Other results state that SSL appears to be CPU bound, as optimizations intended to reduce network traffic have little effect on the server throughput. Also, the cost associated with setting up the SSL connection (TCP connection establishment, data structure initialization used by SSL has greater impact on the server throughput compared to the cost of the data transfer.

For none-SSL operations the throughput for the data transfer trace is lower due its higher data transfer size. The interesting fact is that the throughput of the SSL-secured Web server for both traces is about the same (148 hits/sec) but the times spent on RSA operations is dramatically different. The e-commerce trace is doing many more full handshake (equivalent to more RSA operations) while at the same time the data transfer trace spends the same time encrypting the reply.

5.3 Design

Our goal is to design, implement, and deploy a system that allows users access to Kerberized services through a Web server while making use of existing infrastructures and security policies. We define two different authentication environments: Kerberos space and PK space. We assume that all or some users and services have identities in each of the spaces. For example, a user can have both Kerberos and PK identity while a backend service has only Kerberos identity (both are implicit assumptions based on the stated goal of the system). The system contains four components: a client (represented by a Web browser), a Web server, a backend server, and a trusted third party (referred as a Kerberized Credential Translator). Section 5.3.2 discusses client authentication and

the Web server's responsibilities in meeting user requests. Section 5.3.3 introduces our Kerberized Credential Translator, an extension to TGS that converts PK credentials to Kerberos tickets.

5.3.1 Design Criteria

The following considerations guide our design.

- **Enable easy administration** for both users and system administrators. Administration and management of software is difficult and frequently results in security compromise[1] of the very systems that administrators are trying to protect. Added features should not require user interaction. For example, users should not be forced to actively (i.e., by typing in passwords) or passively (i.e., running additional processes) obtain additional credentials. For example, configuring additional daemon processes (e.g., like in SideCar described in Section 5.2) correctly and preventing malicious and uneducated users from interfering with correct execution of such processes makes the job of a system administrator more difficult and thus overall hurting the security of the whole system.
- **Use off-the-shelf software** as much as possible. The modifications to the components such as Web browsers and servers should use the provided extensibility features of the products such as add-on modules. Modifications to the browser software is inadvisable for a couple of reasons. First, the most widely deployed and used Web browsers (Internet Explorer and Netscape Navigator) are commercial products thus changes to the code not possible. Second, the difficulties with installing upgrades (or additional software) on a large number of client machines are well known and recognized. Modifications to the server software are considered to be less severe. Furthermore, the existence and wide use of the open source Apache Web server [2] with its modular and easily extensible design makes it easy to provide new features.
- **Use existing security infrastructure and mechanisms** for authentication and authorization, i.e., use SSL for secure Web connections, preserve authentication (Kerberos) and authorization mechanisms of the backend services.
- **Restrict and control Web server actions** through authorization mechanisms. The Web server is vulnerable to attacks, so it must be constrained in the actions it is allowed to take on a user's behalf. The system must provide a central, easily administered location for policy decisions regarding Web server's actions.

We make the following assumptions about the system as a whole and in particular about security of the system.

[1] For example, Fu *et al.* [45] showed how an incorrectly configured Web server allowed for authorize users authenticate to the system

[2] 2001 Netcraft Web Server Survey found that 56% of the web sites on the Internet are using Apache[8]

- **Physical security of the services**. We assume the Web server has an adequate physical security. Also, we assume that the Kerberized Credential Translator, described in Section 5.3.3, has physical security comparable to the KDC.
- **Minimal PKI functionality**. We are not trying to solve PKI problems such as reliable and efficient key revocation. This leads to the following additional assumptions.
  - We assume the ability to instantiate a root certification authority, be it a self-signed CA certificate or one signed by an acknowledged root CA, such as VeriSign.
  - We assume the CA certificate can be distributed efficiently and securely. All the client machines need to have such a certificate installed in their Web browser CA certificate list (unless the certificate is signed by one of the well acknowledged root CAs). All other servers in the system need to possess the CA certificate.
  - We assume the root certificate can be revoked.[3] A mechanism is needed that notifies all clients and servers.
  - We assume the (long-lived) certificates issued to the services can be revoked.

5.3.2 Web Server

This section describes the Web server's role in processing a request for a Kerberized service. Our goal is to provide the Web server with a means to access resources on a user's behalf. We built a Web server plugin that engages in proxy authentication by performing the following actions: (i) authenticates the user, (ii) requests Kerberos credentials from a credential translator, and (iii) fulfills the user's request by accessing a Kerberized service.

Client authentication takes place in the SSL handshake. We assume *Alice* possesses a certificate verifiable by the Web server, i.e., the certificate must be issued by a certification authority trusted by the Web server. A client can obtain such a certificate through the KX.509 protocol. However, the use KX.509 protocol is not required, a client can acquire an X.509 identity certificate through other channels.

The Web server records a transcript of the handshake, then presents the captured transcript to a Kerberized Credential Translator (described in Section 5.3.3) for verification, if successful, receives and caches Kerberos credentials, and finally uses them to access a Kerberized service.

The intuition behind capturing the handshake is to provide the evidence to the KCT of the client authentication by the Web server. The details of the captured handshake are shown in Figure 5.1. The SSL transcript contains all the message up to and including CertificateVerify, which serves as the proof of client's knowledge of the private key. Other expects of the handshake such as hello messages are used to prevent replayed attacks.

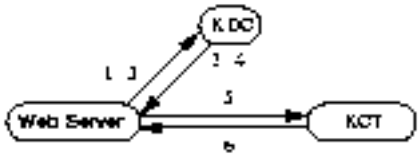
[3] We know it usually cannot.

1. *Client* → *Server*: CLIENT HELLO (CH):
   Version, Random Num, SessionID, CipherSuites
2. *Server* → *Client*: SERVER HELLO (SH):
   Version, Random Num, SessionID, CipherSuites
3. *Server* → *Client*: SERVER CERTIFICATE (SC):
   X.509 certificate
4. *Server* → *Client*: SERVER CERTIFICATE REQ (SCR):
   Cert Type, CA chain
5. *Client* → *Server*: CLIENT CERTIFICATE (CC):
   X.509 certificate
6. *Client* → *Server*: CLIENT KEY EXCHANGE (CKE):
   $[\text{Key material}]_{K_{WSPK}}$
7. *Client* → *Server*: CERTIFICATE VERIFY (CV):
   $[\text{Hash}_{K_{MK}}(\text{CH,SH,SC,SCR,CC,CKE,CV})]_{K_{private}}$

Figure 5.1: **SSL transcript.** The details about each of the SSL messages are presented. Some of the message were omitted, such as SERVERHELLODONE and SERVERKEYEXCHANGE (and others). CERTIFICATEVERIFY messages contains a signed hash of all the messages seen up to this step. (An abbreviation for each of the SSL messages is used to identified each message.) This message is used to do client authentication and is not present otherwise. We denote $K_{private}$ to be the user's private key. K_{MK} is the key generated from the key material sent by the client in CLIENTKEYEXCHANGE. We call it the *SSL session key*.

- CLIENTHELLO carries a version, random number (first four bytes occupied by a timestamp), session id, which allows the user to resume a previous session, and cipher suites.
- SERVERHELLO confirms the version and either creates a new session id if this is a new session or agrees to continue a previously established session, thus forgoing the rest of the handshake. SERVERHELLO states the cipher suite to be used. A server sends its CERTIFICATE and requests the user's in CERTIFICATEREQUEST. SERVERHELLODONE specifies the end of the negotiation phase.
- A client sends her public key certificate in CERTIFICATE. CLIENTKEYEXCHANGE message contains the session information encrypted with the server's public key, K_{WSPK}. Key material included in this message depends on the key exchange protocol. For example, in the case of RSA, a client generates a premaster secret that both parties use to generate key (encryption and digest) material, including the master key, K_{MK}, some times also referred to as the *SSL session key*. A client also sends CERTIFICATEVERIFY, which includes a key-based digest of all the messages prior to this one signed with the client's private key. The server uses the public key from the client's certificate to verify the client's identity.

Web server to improve performance caches user's Kerberos credentials. The



1-4	Original Kerberos done once per lifetime of a session
5. *Web Server* → *KCT*:	TKT, Auth, SSL transcript, $\{\text{MK, Service}\}_{K_{WS,KCT}}$
6. *KCT* → *Web Server*:	TKT=$\{$*Alice*, *Service*, $\text{K}_{WS,Service}\}_{K_{Service}}$, $\{\text{K}_{WS,Service}, \text{T}\}_{K_{WS,KCT}}$

Figure 5.2: **Credential translation protocol.** Steps 1-4, not shown, indicate Kerberos authentication of the Web server. They are performed once per the lifetime of a service ticket for the KCT service. Steps 5 and 6 show the conversation with the KCT. *Service* is the requested backend service. Depending on the version of SSL, an SSL secret key, MK is included in the request to the KCT.

lifetime of the service ticket issued by the credential translator should be short, minimizing potential misuse of credential. At the same time, the service ticket should have a lifetime long enough that multiple requests from the user do not incur the cost of getting a service ticket each time. A compromise of the Web server enables the intruder to use the currently cached credentials and to acquire credentials on the user's behalf for any of the requests to this compromised Web server.

5.3.3 Kerberized Credential Translator

We define a *Credential Translator (CT)* as a service that converts one type of credential into another. In this section, we introduce a Kerberized credential translator (KCT) that converts PK credentials to Kerberos credentials.

Figure 5.2 shows the KCT protocol. The protocol depends on the version of the SSL used due to differences in the format of the CertificateVerify message between the SSL and TLS. Currently, most browsers and servers still use SSL, thus we will describe the KCT protocol based on it. A discussion about the impact of using TLS instead of SSL is provided later.

First, the Web server authenticates to the KCT by presenting a service ticket, $\{$*Web Server*, KCT, $\text{K}_{WS,KCT}\}_{K_{KCT}}$, and the corresponding authenticator, $\{\text{T}\}_{K_{WS,KCT}}$. Along with its Kerberos credentials, the Web server sends the SSL transcript, the name of the service ticket being requested, and the SSL session key. After validating the Web server's credentials, the KCT performs the following steps:

- Validates user and server certificates and checks that each was signed by

a trusted CA.

- Verifies client's signature in CertificateVerify by recomputing the hash of the handshake messages up to CertificateVerify and comparing it to the corresponding part of the SSL handshake.
- Verifies that the identity inside of the server's certificate matches the Kerberos identity. This step is needed to ensure that the Web server participated in the SSL handshake.
- Assures the freshness of the transcript, by checking the freshness of a timestamp or a nonce present in the hello messages. In the latter case, the Web server acquires a nonce from the KCT and includes it in ServerHello.
- Generates a service ticket for the user.
- Encrypts the session key included in the service ticket under the Web server's session key, $K_{WS,KCT}$.
- Returns the ticket, authenticator, and encrypted session key to the Web server.
- Logs the transaction for postmortem auditing.

We see that the KCT needs access to the database of service keys maintained by the KDC. Consequently, the KCT requires the same physical security as the KDC. In practice, we run the KCT on the same hardware as the KDC, which achieves the physical security requirement and sidesteps the challenge of consistent replication of the Kerberos database.

5.3.4 Security Analysis

In this section, we discuss the security of the design of the system. First, we introduce a model of the system, then we sketch a threat model and attacks we consider to be appropriate for the environment.

Threat Model that we consider has following assumptions. First, we state the general threat assumptions such as communication medium is insecure. Any threats that apply to SSL (e.g., man-in-the-middle) and Kerberos (e.g., key DB compromise) protocols apply here. We list the sets of beliefs (trust assumptions) each participants has about the system and how each can misbehave.

- **Clients:** are not trusted by the servers and thus required to authenticate (in turn, mutual authentication is assumed). Clients trust the Web server to acquire the appropriate credentials for a given request. Client machines can be compromised.
- **Web server:** is not trusted by the client (or the trusted-third party) to perform any requests on client's behalf unless specifically asked by the client in the form of initiated SSL-protected request. The Web server must be constrained on the type of services it is allowed to ask. The Web server must be prevented from mounting a (non-Kerberos related) replay attacks against the reuse of client's credentials.
- **Backend server:** provides services only to authorized clients, thus it authenticates and checks for authorization for all requests.
- **Trusted-third party:** is a trusted entity by all other participants in the Kerberos space. However, it is not assumed to be trusted in the PK space.

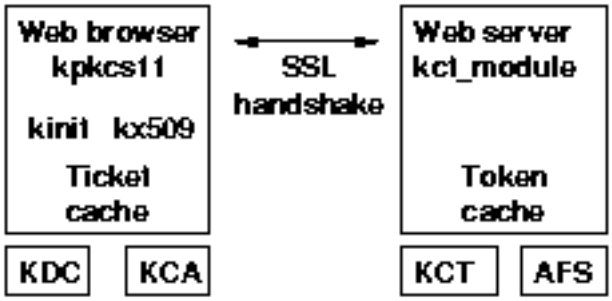


Figure 5.3: **WebAFS architecture.** We show details of architectural components present in the implementation of the proposed system. The new components are: kpkcs11, kx509, KCA, kct_module, and KCT. The first three components are for credential translation from Kerberos to PK credentials. The last two effect translation in the other direction.

- **Adversary:** can eavesdrop on all communications, thus able to capture all SSL handshakes. We assume an adversary has valid Kerberos and PK identities.

Attacks we are trying to defend against:

- Unauthorized user gaining access to a backend service.
- Web server reusing the SSL handshake.
- Web server requesting Kerberos credentials without client's request.
- Web server requesting unnecessary Kerberos credentials.
- KCT eavesdropping on the communication between the Web browser and the Web server.

5.4 Application: WebAFS

We have implemented a prototype that allows a user to submit requests to a Web server that accesses a Kerberized AFS file server on the user's behalf. An overview of the system is shown in Figure 5.3. It combines both systems: KX.509 and KCT. We now look more closely at the problems that arise from differences in the SSL protocol specifications and implementations, and from harsh browser realities, which make the solution more complex and introduce delays.

To enable the server to act on a user's behalf, we added a module to the Apache Web server, under 2000 lines of code. The initial version of the system was implemented to work with a version of the `OpenSSL` (versions 0.9.5 through 0.9.6c) library modified to save the SSL transcript. Modifications to the library are minimal (under 200 lines of code) and include a new data structure and calls to a function that saves the incoming and outgoing handshake messages.

`OpenSSL` version 0.9.7 (still unreleased version) includes a specialized callback

mechanism that allows us to *listen* on the handshake without actually modifying the library. However, to be able to capture the on-going handshake Apache's `mod_ssl` module (which is Apache's implementation of an SSL server) requires further (minor) modifications. There exist a solution that does not requires additional modification to the existing code but it pays the penalty of engaging in an additional SSL handshake.

In our prototype, we use timestamps present in SSL handshake to check the freshness of the handshake. Unfortunately, SSLv2 does not include timestamps in the hello messages. Worse yet, Navigator by default starts the SSL handshake with an SSLv2 ClientHello message. Only after receiving the reply from the Web server suggesting the use of SSLv3 does the browser switch to the higher version. The resulting handshake is overall a valid handshake, but lacks an SSLv3 client timestamp. To get the timestamp, we require the Web server to request renegotiation.

Renegotiation is another special feature of the SSL protocol. To reduce the risk of key compromise, the SSL protocol supports renegotiation of the security context. If the server wishes to start a (new) SSL session (which not necessarily corresponds to a new connection), it sends an empty ServerHello message to the client. However, it is always a client's responsibility or choice to actually initiate a new handshake by sending a ClientHello message, i.e., a client can refuse to engage in a new SSL handshake or just ignore the requests.

SSL specifications allow renegotiation only after the ongoing handshake is complete, so two full SSL handshakes must take place.

In the KCT protocol based on SSL the SSL session key is revealed to the KCT. It gives the credential translator the power to eavesdrop, so we require the Web server to request renegotiation to establish a new session key, one that is not known to the KCT. This is a trade-off between security and performance[4].

Establishing an SSL session requires sophisticated cryptographic calculations and numerous protocol messages. To minimize the overhead of these calculations and messages, SSL provides a mechanism by which two parties can reuse previously negotiated SSL parameters. With this method, the parties do not repeat the cryptographic operations, they simply resume an earlier session. The user proposes to resume a previous session by including that session's SessionID value in ClientHello. It is up to the server to decide whether to allow the reuse of the session. We call this a *partial* SSL handshake. This feature of the protocol requires special attention.

When a partial SSL handshake happens, the Web server checks if AFS credentials are cached; if so, then the server proceeds with the AFS request. Otherwise, the Web server forces an SSL renegotiation followed by a full SSL handshake. After creating a transcript, the Web server, as before, submits a request to the KCT for an AFS service ticket.

As of this writing, the MIT Kerberos libraries are not thread-safe, so the KCT cannot be implemented as a multithreaded application. To improve per-

[4]One could argue that because the KCT is as powerful as the KDC and can impersonate any user, then the KCT itself can place a request to a Kerberized service, and, thus, the KCT can be trusted with the knowledge of the SSL session key.

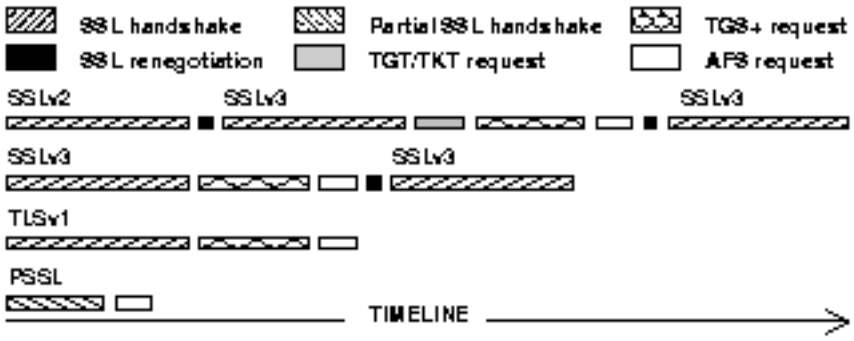


Figure 5.4: **Timelines for WebAFS requests.** We show the components of a user request in four scenarios illustrated as timelines. The legend identifies each of the components involved. We consider all the different versions of an SSL protocol, v2, v3, TLSv1, and a partial handshake. Access to an AFS file server is used as an example.

formance, we spawn a process to handle incoming requests. To achieve the required physical security, we run the KCT on the same hardware as the KDC. Implementation of the KCT is under 2000 lines of code.

5.5 Performance

In this section we discuss the performance of the system by examining the cost of making a request to a Web server, which, in turn, requests a service from a backend server on a user's behalf. The experiments described in this section were performed on an unloaded Intel 133MHz Pentium workstation running RedHat Linux 6.2 (kernel version 2.2). Our focus is on understanding overhead induced by the system, so all the components were executed on the same hardware to avoid network and file access delays.

The software was tested against commodity browsers, but it is hard to glean detailed measurements from commercial software, so we used `OpenSSL` tools to mimic the browser's actions. We used `OpenSSL`'s generic SSL/TLS client, called `s_client`. All requests were made for a 1K file. For each of the test cases 30 trials were measured and averaged.

We define a *browser session* to be the time from launch to termination of the browser application. We define a *server session* to be the time from the first request to a Web server until the termination of the browser application. Within a browser session a user starts multiple server sessions. Requests for different files from the same Web server fall into a single server session. Requests to different Web servers are associated with different server sessions.

Figure 5.4 shows the breakdown of a user's request into the basic compo-

nents. Four scenarios are illustrated as timelines. We describe each of the scenarios in detail and point out which ones are more common. Table 5.2 summaries the end-to-end delays seen by the user for different types of requests. We divide requests into two groups, depending on whether user's credentials are cached at the Web server.

No cached credentials. First, we consider the cases where user's credentials are not cached. This happens when a user is making the first request to the Web server or when her credentials have been evicted from the Web server's LRU cache.

- **Once a day:** *SSLv2 hello no TGT* and *SSLv3 hello no TGT*. In these two scenarios, the Web server has stale credentials so the user's request gets penalized by the time needed by the Web server to get new Kerberos credentials. The lifetime of our Web server's TGT is 24 hours.
- **Once per server session:** *SSLv2 hello 1st request*. When contacting a Web server for the first time, the default behavior of Navigator is to start with an SSLv2 ClientHello message. Until the browser is restarted, all subsequent requests will start with an SSLv3 ClientHello. This scenario measures the overhead of the three handshakes and a KCT request. The first additional handshake produces a valid timestamp in the ClientHello message. The second additional handshake renegotiates the SSL session key, which was revealed to the KCT.
- **Most common request:** *SSLv3 hello request*. Explorer starts with an SSLv3 ClientHello. Any requests from this browser fall either into this category or the partial handshake.

Cached credentials. We now review the scenarios where the user's credentials are cached at the Web server. Caching is important because it saves the overhead of getting Kerberos credentials. Furthermore, no SSL renegotiation plus handshake is needed at the end. The only overhead the system imposes is that associated with token management.

- **Frequent:** *Partial handshake cached credentials*. The lifetime of the session key negotiated in the full handshake is configurable by the web server. If more than one request is made within five minutes of a full handshake, a partial handshake takes place. (Five minutes is a default value used by Apache Web servers). We can safely assume that user's credentials are already cached at that point. The time required for a partial handshake is considerably smaller than for a full handshake. The frequency of these requests depends on the user's access pattern.
- **Common:** *SSLv3/TLSv1 cached credentials*. Once the user contacts a Web server, her credentials are cached until they get evicted due to expired lifetime or lack of space. When requests to the Web server are separated by more than five minutes, a user experiences end-to-end delay presented in last row of Table 5.2.
- **Unlikely:** *SSLv2 hello cached credentials*. The browser sends an SSLv2 ClientHello message to the Web server if it never contacted it within the current browser session. However, it is still possible for the user's credentials to be cached at the Web server if the user restarted the browser

within the lifetime of the cached credentials.

End-to-End	Time(s)
SSLv2 hello no TGT	4.08
SSLv2 hello 1st request	4.04
SSLv2 cached creds	2.50
SSLv3 hello no TGT	2.86
SSLv3 hello request	2.80
SSLv3 cached creds	1.25

Table 5.2: **End-to-end delays.** Each of the scenarios represents a possible user request. We measured end-to-end latency seen by the user.

To summarize, an SSL handshake costs 1.25 seconds. Delays associated with refreshing a TGT and making KCT requests are small: 0.02 and 0.26 seconds, respectively. In the most common case, credentials are cached and SSLv3 connections are used, so the system incurs negligible overhead. Further testing in more complex environments is necessary and will be done in the future. However, these preliminary results are encouraging.

5.6 Future Work

In this section we outline the scope of future research. Section 5.6.1 proposes to enhance the KCT's authorization policy and investigate how to allow users control delegation. Section 5.6.2 proposes extensions of the design to other applications (e.g., Telnet) and other types of credentials (e.g., certificates). Section 5.6.3 proposes further performance studies to evaluate the design.

5.6.1 Security Policy Issues

The authorization model of the credential translator is primitive and is the focus of our future work. The current model supports generic access control lists: for each Web server there is an entry listing the Kerberized services for which it can request tickets. We propose to investigate whether or not an ACL-based policy is sufficient for the KCT.

It might be desirable to have access control based on the principal on whose behalf the Web server is requesting credentials. For example, for some principals less (or more) services are available in this scheme. The policy must take then into account three entities: the principal, the Web server, and the service. Authorization could be based on the principal group membership, only if a principal belongs to a particular group, the Web server is allowed to the specified service, or the Web server's group membership, only if it is a departmental Web server, the Web server is allowed to request tickets to the specified service. The authorization decision might depend on other attributes, e.g., time of day. In this case, if the Web server is providing a registration service for the users, then

the KCT can enforce control over the time period this service is available for specific users. This might be useful if registration is offered periodically, so that the Web server does not need to be configured to handle the on and off periods.

Authorization could be based on specific attributes of the principals. It might be desired to be able to make a decision about a principal from the a different (but trusted) domain than that of the KCT. For example, a user with the certificate given to *aglo@engin.umich.edu* is trying to access a resource (e.g., file) in the *umich.edu* domain. This is closely related to the namespace issue discussed in Section 4.5.2. An more interesting scenario is to allow the user with the certificate *aglo@mit.edu* to access a resource in the *umich.edu* domain. A simple solution could be to have a policy that states: anybody from the *mit.edu* domain is allowed access to the AFS file server with the predefined identity (e.g., *mituser*). A question about general PKI to Kerberos identity mapping needs further research.

Another interesting issue that requires further research and related to the KCT authorization decision is whether or not and how to allow principals to control which services the Web server is allowed to access on their behalf. One solution could be for each request to prompt the user for his permission to let the Web server act on user's behalf and somehow propagate that decision over to the KCT. However, as has already been shown with cookies, users do not like to be bothered. Instead, browsers provide coarse-grained configuration options to accommodate cookies. In theory, the same could be done for expressing rules about Web servers' actions. Two problems arise. First, we lack the means to state such policies. Second, we lack the means to propagate policy to the KCT securely. Major modifications to the browser and Web server software are required to achieve both. An alternative solution could be to transfer policy requirements inside the attribute certificate, or encoded as attributes inside of the identity certificates. Only the identity certificate is used in the current infrastructure, so a proper mechanism for transferring the attribute certificate is required.

The KCT is the central place where junk keys are used. While short term certificates eases revocation problems, the KCT offers a place to eliminate the problem all together. When the need to revoke a certificate arises, it is simply reflected at the KCT.

Several other issues need to be addressed. In the current model, the captured handshake does not reflect the type of request that comes after the secure connection is established. Another issue is multi-staged request: a single request from the user might require the Web server to perform multiple actions, possibly communicate with multiple Kerberized services. It might be more efficient to extend the protocol to request multiple credentials at once.

5.6.2 Extending Credential Translation

Credential translation need not apply only to Web traffic but can be extended to any SSL-enabled client and SSL-enabled server communication, e.g., SSL-enabled Telnet. Assuming a user has a certificate on his local computer, we can

thus obviate the need to send his password over the network. A user can use his certificate, mutually authenticate with the remote host (telnetd process), and empower it to act on his behalf (e.g., acquire Kerberos tickets).

Furthermore, credential translation need not be limited to producing Kerberos credentials. For example, a Web server can request a certificate on client's behalf. A simple solution is presented in the next section where the Web server requests Kerberos credentials to talk to the KCA and then initiates a kx509 protocol. The two steps could be combined into one.

5.6.3 Performance

We need to perform further performance studies measuring page-serving throughput under trace-driven workloads. The throughput of the Web server can be measure in terms of the number of connections or number of bytes transferred. We already studied the end-to-end client delays imposed by the proposed system. We need to study how the system effects overall performance of the Web server.

Chapter 6

Practical Distributed Authorization

6.1 Introduction

Reliable high speed end-to-end network services are increasingly important for scientific collaborators, whether separated by large distances or located just across campus. Our experience shows that long haul networks demonstrate good performance (thanks to overprovisioning), but the *last mile* - from the edge of the campus network to the desktop – is often a network bottleneck.

Quality of Service functionality is a common feature of network hardware. Recent studies show the viability and usefulness of these features to control network resources. Currently, configuration of network hardware is done by hand. While several efforts [57, 109] are attempting to produce standard protocols to enable automated configuration across network administrative domains, it is not clear yet which protocol(s) will be embraced.

Our work aims to service the need for an automated network reservation system to provide reliable last mile networking for video, audio, and large data transfers. Reliable end-to-end network service between participants is achieved by reserving network resources within the end point institution networks, coupled with the demonstrated good performance of the interconnecting long haul networks where no network resource reservation is needed.

In automating network configuration, security of all communications is vital. Network hardware is a prime target for malicious hackers, because controlling the routing and resource allocation of a network enables all other attacks. What makes this security problem difficult is the cross-domain nature of end-to-end network resource allocation. Requesting end-to-end network resource allocation between the local domain and a remote domain, a user needs to be authenticated and authorized in both domains before the request can be granted.

Our work is based on the Globus General-purpose Architecture for Reservation and Allocation (GARA) system [30, 32, 31, 40]. The goal of the GARA

architecture is to create a flexible solution that satisfies requirements of different types of resources (networks, CPUs, disks, etc.), while providing a convenient interface for users to create both advance and immediate reservations. GARA uses the Globus Grid Security Infrastructure (GSI) [17] for authentication and authorization. An attractive feature of GSI is that it performs cross-domain authentication by rellying on a Public Key Infrastructure (PKI) and requiring users to have long term public key (PK) credentials.

GSI provides coarse-grained access control. A flat file, called the *gridmap* file, stores mappings from PK credentials (Distinguished Names, (DN)) to local user names. A user is allowed access to Globus services if there is an entry corresponding to this user in the *gridmap* file. This all-or-nothing access control policy is extremely limiting. Authorization decisions in QoS are based on many parameters such as the amount of available bandwidth, time of day, system load, and others. We propose to control resource usage with a policy engine and expressive security policies.

We introduce a practical system that shows a design and implementation of GARA services that offer automated network reservation services to users. We leverage solutions proposed in the previous two chapters and use GARA services as an application of the proposed protocols. The contributions of this systems are twofold. First, we provide a fine-grained cross-domain authorization for GARA that leverages existing institutional security and group services, with universal access for users. We identify and discuss issues involved. Second, we eliminate the need for long term PK credentials and associated overheads that are required by other systems. We describe the implementation of an easy and convenient Web interface for making reservation requests.

The remainder of this chapter is organized as follows. Section 6.2 describes the GARA architecture. Section 6.3 discusses related work. Section 6.4.1 describes the KX509 and KCT services described in Chapters 4 and 5, and shows how they allow ubiquitous access to GARA by enabling a reservation to be made via the Web, obviating the need to install Globus software on workstations. Section 6.4.2 presents an architecture for distributed authorization that employs a shared namespace, delegated authorization through secure and trusted channels and a signed authorization payload, and the policy engine used to make the authorization decision. Section 6.5 is a step by step description of the enhanced GARA system. Section 6.6 identifies issues that require further research.

6.2 Overview of GARA Architecture

GARA relies on Globus GSI for security mechanisms, i.e., distributed authentication and coarse-grained authorization. To support distributed computing environments GSI relies on the public key cryptography. It requires users to have long term PK credentials signed by a certification authority (CA). Long term keys are used to create proxy certificates that are used for requesting services. At each resource a global to local identity mapping is enforced to provide limited authorization. It also allows for a resource to specify security policies

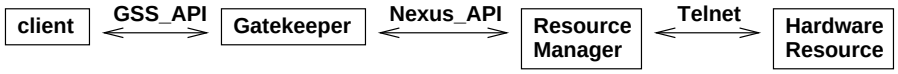


Figure 6.1: **GARA Architecture.** This figure shows the process of reserving local network resources with the GARA architecture. Long term user PK credentials are stored in the client workstation file system. A Globus-Gara Service consists of a gatekeeper and a resource manager all residing on a single machine. On each link a communication mechanism is specified

based on locally defined principals. To secure communication, GSI uses GSS-API [69] an authentication mechanism defined by the SSL protocol. Figure 6.1 shows a GARA reservation request step by step. It is assumed that the user has already acquired the long term certificate.

1. The user generates a proxy certificate and signs it with his long term key.
2. The user runs the GARA client program and inputs reservation parameters via a command line interface. A GSSAPI_SSLEAY secured connection is established between the user and the gatekeeper. The user is authenticated with the proxy credentials generated in Step 1. The reservation request parameters are passed in the RSL (Resource Specification Language) form [127]:

   ```
   &(reservation-type=network)(start-time=997212110)(duration=5)
   (endpoint-a=141.211.92.130)(endpoint-b=141.211.92.248)
   (bandwidth=5)(protocol=tcp)
   ```

3. The gatekeeper checks for an entry in the *gridmap* file that matches the Distinguished Name field in the proxy certificate used to authenticate the user. This file is GSI access control mechanism. A user is either authorized to use the globus system and place any kinds of reservations or use the resources or not. A fine-grained authorization mechanism is obviously lacking here.
4. The gatekeeper in turn forwards the RSL to the resource manager using the Nexus API [60, 63][1]. *diffserv_manager* is a specific type of a resource manager responsible for a particular type of quality of service reservation.
5. The *diffserv_manager* checks its configuration files for the two endpoints (source and destination IP addresses) and on the availability of network bandwidth.
6. The *diffserv_manager* configures the routers. Currently, an Expect script Telnet's to the appropriate routers and sets up the flow [2]. Expect [37] is

[1] Currently, Nexus protocol does not offer any security and becomes a security hole in the request reservation path. Communication between the gatekeeper and a resource manager should be secured using GSS_API.

[2] Using Telnet between the resource manager and the resource is not secure and needs to be replaced by secure protocol. We choose to use SSH to securely communicate with the routers.

a tool for automating interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, etc.

6.3 Related Work

The Globus MyProxy [98] initiative provides a trusted server to store user's delegated credentials, indexed by a tag and a password. Later, a service can contact a MyProxy server, present a tag and a password and receive corresponding credentials (e.g., certificate or Kerberos ticket) on a client's behalf. Each service requires a different tag and password, forcing users to manage many passwords. This approach requires users to type in their passwords into HTML forms. HTML forms are easily reproduced by a malicious hacker who collects passwords. He can obtain a certificate, signed by one of the default Certificate Authorities supported by browsers, and run a Web server, providing a spoofed login HTML form. However, by examining the credential, an activity most users do not bother doing, the user can tell the spoofed login form.

The Grid Portal Architecture [54] is a Web interface to Grid Computing resources that uses MyProxy Services for client authentication. The GARA Web interface differs from the Grid Portal in several important ways. Access in our scheme is via done an https stream and requires mutual SSL authentication, which in turn requires a user certificate, thus obviating the need for users to type passwords in HTML forms, as it is done in the Grid Portal.

The Community Access Service (CAS) [97] is a proposed Grid authorization service that the user calls prior to making a request for Grid resources. CAS returns a signed capability to indicate a successful authorization request. The capability is then added to the Grid resource request.

The GARA client is designed to contact each end domain GARA service. In the future, GARA client will contact the first GARA service, which in turn will contact other bandwidth brokers (BB), needed for the end-to-end reservation. Sander *et al.* discusses the bandwidth broker to bandwidth broker protocol in [107]. The Simple Inter-Domain Bandwidth Broker Specification (SIBBS) [109] is a simple request-response bandwidth broker to bandwidth broker broker protocol being developed by the Internet2 QBone Signaling Design Team. It is anticipated that GARA will be an early development code base for SIBBS.

Authentication Authorization Accounting and Auditing (AAAA) research group produced Internet drafts on distributed authorization requirements [38], framework [133] and architecture [24].

6.4 Design

6.4.1 GARA Web Interface

Many sites, such as the University of Michigan, lack a PKI, but they do have an installed Kerberos [94] base. The University of Michigan has developed a service that allows users to access Grid resources based on their Kerberos credentials.

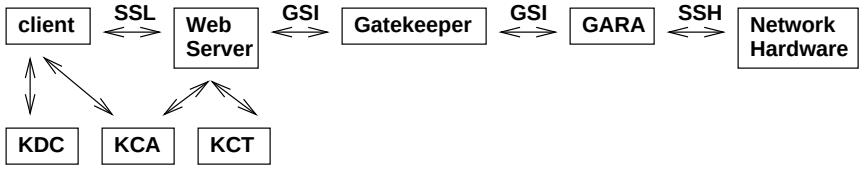


Figure 6.2: **KX509 GARA Web Interface.** This figure how local network resources are reserved with the GARA Web interface. KX509 junk keys replace long term PK credentials.

The KX509 [34, 67] system translates Kerberos credentials into short-lived PK credentials, or *junk keys*, which in turn can be used by browsers for mutual SSL authentication or by GSI for Globus authentication.

Junk keys have several advantages over traditional long-lived PK credentials. They have short lifetimes, so the revocation problem [10] is largely obviated. While, in a traditional PKI, long term credentials put the ease of user mobility in question, KX509 users can obtain new junk keys at each workstation.

KX.509 creates a new public/private keypair and sends the public key to a Kerberized Certificate Authority (KCA) over a Kerberos secured channel. Using the presented public key, the KCA creates and signs a short term X.509 identity certificate.

In order to make network resource reservations convenient for users, we built a GARA Web Interface. A user makes a reservation by filling out a GARA network reservation Web form. All requests are SSL protected and require mutual authentication. As oppose to a traditional password-based user authentication, we use short-lived user certificates, priorly acquired with KX.509. After the Web server authenticates the user, it contacts a Kerberized Credential Translation (KCT) [67] server, presents appropriate credentials, and requests Kerberos credentials on the user's behalf. Next, the Web server runs KX509 on the user's behalf, which creates a new junk key for the user on the Web server. This junk key is then used to create Globus proxy credentials. GARA client code resides on the Web server and uses Globus proxy credentials. Figure 6.2 gives an overview of the GARA Web Interface.

6.4.2 Distributed Authorization Design

In a cross domain distributed authorization scheme, authorization decisions are made even if the requestor and resources reside in separate domains. Often authorization decisions are made by a policy engine that applies policy rules to a set of input attributes. These attributes might include user attributes such as group membership or environmental attributes such as time of day. Attribute information can come from a variety of sources: local services, environment, configurations, or attached to the resource request. We separate the authorization process into two phases: gathering of attributes and running of the policy engine.

In designing the distributed authorization system, we must address the location where the authorization decision takes place. We discuss how the use of shared namespace and delegated credentials are the key to creating a practical authorization scheme. We also believe in utilizing existing local authorization services to require as little replication of information as possible.

Location of authorization decision: The question that needs to be answered is: where is the best place in GARA to make the authorization decision? Three possible locations exist: Web server, gatekeeper, and resource manager.

Prior to initiating any contact with the desired resource, the Web server can contact an authorization service and provide user's identity and resource request information. Having such an authorization service would perforce need to have a policy for each resource and information about each user. However, this choice presents extra communications when the resource is not available, or when fine-grained authorization is not required.

The gatekeeper is expected to handle numerous requests, so performing the authorization decision at the gatekeeper could have an impact on the gatekeeper's performance. At the gatekeeper, it is still unknown if the resource is available, so as above, the extra communication and work to make an authorization decision could be wasted effort. We conclude that adding authorization at the gatekeeper would be counter productive.

The best place to enforce authorization in the GARA architecture is at the resource manager where each service is capable of stating, enforcing, and modifying its policies without depending on the administration of the Globus architecture at large.

Shared namespace. Central to any authorization service design is the formation of an attribute namespace that is understood by policy engines. Frequently, the primary concern in the authorization decision is related to a group membership question: does this user belong to appropriate groups? Consequently, a security policy would enforce the restricted membership for specific actions. Within a domain, the statement of group membership is well defined. Both user identity information and a group namespace are available locally.

A shared group namespace, presented to policy engines in multiple domains and used to control access to resources in multiple domains, is defined by a number of groups with common names across domains. In its existing group service, each domain creates groups with these names and manages user membership as any local group. Other attributes presented to the distributed policy engines such as the amount of requested bandwidth or start-time of the request are already encapsulated in a shared namespace in that they are coded as name,value pairs in the request.

Signed authorization payload. At the remote service, we do not add a callback to the local group service to determine group membership, instead authorization information is added to the existing resource request.

The local GARA resource manager queries the local group membership service for the subset of shared namespace groups in which the requestor is a member, and passes the group list along with the request parameters to its policy engine to make an authorization decision. If the request succeeds, the local

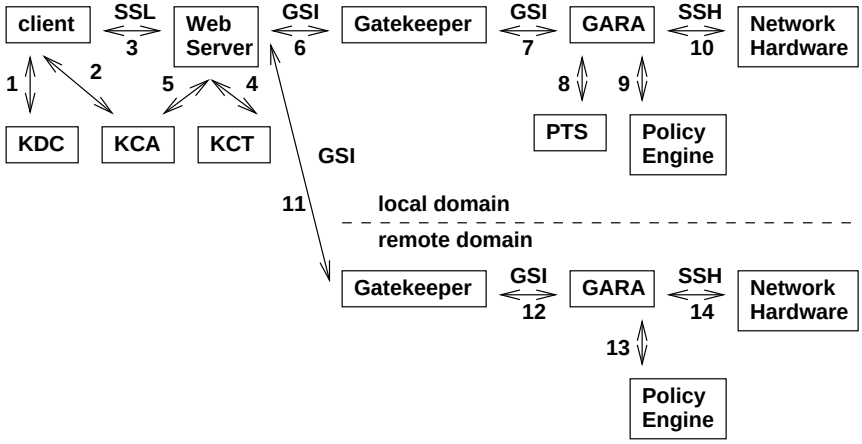


Figure 6.3: **Network Resource Reservation Data Flow.** KDC is a Kerberos Key Distribution Center. KCA is a Kerberized Key Signer. KCT is a Kerberized Credential Translator. KDC and KCT must share hardware because both require access to the Kerberos database. Steps 1,2,4 and 5 use Kerberos authentication.

GARA resource manager creates an *authorization payload* consisting of the requestor's distinguished name and the group list. To secure the authorization payload, we require the local GARA resource manager to sign the authorization payload before adding it to the reservation reply returned to the GARA client running on the Web server. The reservation request is forwarded to the remote GARA who validates the signature on the received authorization information before passing the group list as input to its policy engine. Thus we piggy-back the group membership information on the existing reservation request communication.

Policy Engine. After all the requestor's attributes such as group membership and request parameters have been established, the fine-grained authorization decision can be made. In general, policy engines accept attribute-value pairs as input, compare the input attributes to a set of policy rules, and return a pass/fail response. The granularity of the authorization decision is embodied in the complexity of the policy rules that must be satisfied by the input attribute-value pairs. To allow different policy engines, the authorization callout has a generic API that passes information about the requestor and the action to the policy engine. We chose KeyNote [12, 13, 14] for our policy engine because of its flexibility and easy availability.

6.5 Implementation

We successfully demonstrated our modifications to GARA by reserving bandwidth for a video application running between the University of Michigan and

CERN[3]. Bandwidth is reserved by filling in a Web form served by a modified Apache Web server that runs the GARA client. The GARA client communicates with separate GARA services at each endpoint domain, as shown in Figure 6.3. The GARA services use KeyNote authorization policies configured to require bounded request parameters for bandwidth, time and duration. Group membership is also required. We demonstrated that if any of the policy parameters are not satisfied, e.g. too much requested bandwidth or incorrect AFS PTS group membership, the reservation fails.

A successful reservation results in configuring the end domain Cisco ingress routers with the appropriate Committed Access Rate (CAR) rate_limit, which marks the packets and polices the flow. The participating routers are statically configured with WRED, Cisco's implementation of the Random Early Detection (RED) class of congestion avoidance algorithms.

What follows is a step by step description of an end-to-end network reservation using the enhanced GARA, also illustrated in Figure 6.3.

1. User (locally) executes *kinit* and acquires Kerberos credentials.
2. User (locally) executes *kx509* and acquires junk keys.
3. Using a browser, a user makes an https request for the network resource reservation page. The junk key, obtained in Step 2, is used for mutual SSL authentication. Network reservation parameters such as source and destination IP address, desired bandwidth, start time are entered into a form and sent to the Web server.
4. The Web server *kct_module* makes a Kerberos authenticated request to the KCT and acquires a service ticket for the KCA service on the user's behalf.
5. The Web server *kx509_module* acquires and caches junk keys on behalf of the user as in Step 2. Then, the Web server *globus_proxy_init module* uses the newly created keys to create user's Globus proxy certificate.
6. The Web server *gara_module* constructs a reservation request to the local gatekeeper using the Globus GSSAPI_SSLEAY protocol and the proxy certificate. The local GARA gatekeeper looks for an entry in the *gridmap* file that matches the corresponding distinguished name – a field in the Globus proxy certificate (from Step 5). The DN and local id are attached to the RSL (Resource Specification Language) string.

   ```
   &(reservation-type=network)(start-time=997212110)(duration=5)
   (endpoint-a=141.211.92.130)(endpoint-b=141.211.92.248)
   (bandwidth=5)(protocol=tcp)(client-name=aglo)
   (client-dn=287f42c19122ec08ddc679ba259070f9)
   ```

   The last two attribute value pairs (client-name and client-dn) are the two fields not present in the original GARA RSL reservation request. DN is a considerably long string, so we use a hashed value of the DN instead.

[3]European Organization for Nuclear Research

7. Using the Nexus API for interprocess communication, the local gatekeeper forwards the RSL to the resource manager (*diffserv_manager*).

8. The local id is passed to the group_membership function, which performs one of several actions, depending on configuration and on whether the request is from a local or remote user. If the authorization data in this RSL is null, the request is from a user in the local realm. In our settings, group membership function queries a Protection Server (PTS) – a part of AFS. The call can be easily replaced by a query to any local group service such as an LDAP service or a flat file. The call is performed over an authenticated channel using the *diffserv_manager*'s identity. Prior to accepting any connections, *diffserv_manager* acquires AFS tokens needed to authenticate with the PTS server.

9. The group membership information acquired in the previous step, the reservation parameters, and a policy file are passed to the KeyNote policy engine that makes an authorization decision. If the request does not satisfy the current security policy, an error is returned back to the client and the rest of the steps are not executed.

   We now describe the basic pseudocode for the KeyNote policy engine call with a group membership action condition. Figure 6.4 shows the main actions of the resource manager. Figure 6.5 provides details of the *authorization_decision* function.

   - *requester*: the requesting principle's identifier.
   - *action_description*: the data structure describing an action contains attribute value pairs that are included in the request. For example, "$system\_load \leq 70$" specifies an environment condition stating that the current system load must not exceed 70%.
   - *SN_groups*: the groups in the action_description, also added as attribute value pairs describing the action. For example, "umich_staff = yes" states the request is a member of the *umich_staff* group.
   - *policy*: the data structure describing local policy, typically read from a local file.
   - *credentials*: the data structure with any relevant credentials, typically sent along with the request by the requesting principal. Before making use of these credentials, their validity must be confirmed by verifying a signature included in the credential data structure.

   Figure 6.4 shows the main actions of *diffserv_manager*.

   Figure 6.5 provides details of the *authorization_decision* function.

   Figure 6.6 shows an example of a KeyNote top level security policy that allows the action if the following conditions hold: an application domain is called *gara* and the requested operation is *reservation* for the resource of type *bandwidth*. Furthermore, if this is a local request, then bandwidth

```
SN_groups = retrieve_group_membership(requestor);
result = authorization_decision(requestor, action_description, policy,
 credentials);

if(result == "allowed") do the requested action
else report action is not allowed
```

Figure 6.4: Pseudo-code for the authorization mechanism in *diffserv_manager* .

```
session_id = kn_init();
retrieve_policy(&policy);
kn_add_assertion(session_id, policy);

for all attribute/value pairs
  kn_add_action(session_id, attr, value);

result = kn_do_query(session_id);
```

Figure 6.5: Pseudo-code for the call to the KeyNote Policy Engine

for more than 100Mb is not allowed. If the request is from a remote user, then amount greater than 10Mb is not allowed. If the current time is after hours, then no restriction on bandwidth is enforced. The requestor must be a member of *grid_bw* group.

```
keynote-version: 2
local-constants: ADMIN_UM = "x509-base64:MIICrzCC"
authorizer: "POLICY"
licensees: ADMIN_UM
conditions: app_domain == "gara" &&
  operation == "reservation" &&
  type == "bandwidth" &&
  ((location == "local" && @amount <= 100) ||
  (location == "remote" && @amount <= 10) ||
  time == "night") && grid_bw == "yes");
```

Figure 6.6: Trusted assertion stating KeyNote top level security policy. Note that the value of the key has been truncated.

If the KeyNote policy engine states that the action is not allowed, no reservation is made by the local *diffserv_manager* and an authorization failure is returned to the Web server. As the result, the reservation protocol returns an authorization error back to the client. A success value is returned to the client only if both local and remote authorization have succeeded.

10. Same as Step 7.

11. Same as Step 9.

12. Same as Step 10.

This design lets us express many policies, including who can request which network resources and when such requests are valid. In the example we presented, the authorization payload is signed by one certificate, the remote GARA *diffserv manager*. More broadly, a site may require the authorization payload to contain assertions from other services. For example, a site might require that users be U.S. citizens, verified by some specific Certificate Authority. Signed assertions can be added to the authorization payload to accommodate such requirements.

6.6 Future Work

In this section we outline the scope of future research.

6.6.1 Implementation Issues

In the current implementation, the size of the RSL string is fixed. We might need to send a chain of certificates, where the size of one certificate is usually 1 KB. Because the size of the chain is not known in advanced, the size of the RSL should be variable.

6.6.2 Policy Management

Writing KeyNote assertions is not intuitive. Furthermore, there is no assertion management provided by the KeyNote library. Support for dynamic configuration of KeyNote policies is needed.

6.6.3 Controlled Delegation

In Section 5.6.2 we mentioned that credential translation can be extended to create credentials other than Kerberos ones. We propose to study the benefit of combining the KCT and KCA steps into one. Undoubtedly, it will improve a performance by reducing the request latency seen by the client. Doing so also allows us to exercise additional control over the certificate that is being created. A special certificate can be issued that prohibits delegation. In the current scheme, the Web server gets a certificate equivalent in power to the ticket granting ticket in Kerberos. The Web server has unlimited power to impersonate the user in the public key environment. KCT was the solution to restrict the Web server from having the unlimited power. A similar scheme is required in this new environment.

6.6.4 Namespace Management

We need to investigate the feasibility of the common namespace management.

Chapter 7

Research Plan

In this chapter, we summaries the future work and outline a schedule for completing the thesis.

- *August 2002*: Complete the performance study of the KCA. Complete the investigation of alternating naming and corresponding KCA's security policies and propose an adequate design. Complete the investigation and propose the design for issuing attribute certificates.
- *October 2002*: Complete the implementation and performance study of the extended KCA. Complete the investigation of the appropriate KCT's authorization scheme. Complete the investigation and propose a design of the key management scheme for short-term certificates.
- *December 2002*: Complete the implementation and evaluation of the authorization scheme for the KCT. Complete the extended functionality of the KCT. Complete the implementation of the key management scheme.
- *February 2003*: Complete policy management of KeyNote assertions. Complete the investigation of the symmetry between the symmetric and asymmetric authetiation protocols.
- *May 2003*: Complete the thesis.

Bibliography

[1] A Digital Library Authentication and Authorization Architecture. `http://www.ucop.edu/irc/cdl/tasw/Authentication/Architecture-3_W95.pdf`.

[2] M. Abadi. On SDSI's linked local name spaces. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, 1997. 16

[3] M. Abadi. Two facets of authentication. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 27–32, 1998. 15

[4] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering*, 22(1):6–15, 1996. 16

[5] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, August 1991. 15

[6] W.A. Adamson, J. Rees, and P. Honeyman. Joining security realms: Single login for NetWare and Kerberos. In *Proceedings of 5th USENIX Security Symposium*, June 1995. 27

[7] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Proceedings of "Advances in Cryptology – EUROCRYPT'95", Lecture Notes in Computer Science*, August 1995. 16

[8] Apache web server. `http://www.apache.org`. 39, 42

[9] G. Apostolopuolos, V. Perris, and D. Saha. Transport layer security: how much does it really cost? In *Proceedings of the 8th Conference on Computer Communications*, March 1999. 40

[10] Andre Arnes. *Public Key Certificate Revocation Schemes*. PhD thesis, Norwegian University of Science and Technology, Kingson, Ontario, Canada, February 2000. 25, 59

[11] A. Birrell, B. Lampson, R. Needham, and M. Schroeder. A global authentication service without global trust. In *Proceedings of IEEE Symposium on Security and Privacy*, April 1986. 16

[12] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust management system version 2. RFC 2704, September 1999. 35, 39, 61

[13] M. Blaze, J. Feigenbaum, and A. Keromytis. Keynote: Trust management for public-key infrastructure. In *Proceedings Cambridge 1998 Security Protocols International Workshop*, April 1998. 35, 39, 61

[14] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of Financial Cryptography*, February 1998. 35, 39, 61

[15] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990. 15, 16

[16] M. Burrows, M. Abadi, and R. Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks*. IEEE Computer Society Press, 1996. 15, 16

[17] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, and J. Volmer. A national-scale authentication infrastructure. *IEEE computer*, 33(12):60–66, December 2000. 56

[18] D. Cheriton and T. Mann. Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems*, 7(2):147–183, May 1989. 16, 28

[19] J. Clark and J. Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, 1995. 16

[20] D. Clarke, J. Elien, C. Ellison, F. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. To be published, November 1999. 28, 35, 39

[21] C. Coarfa, P. Druschel, and D. Wallach. Performance analysis of TLS web servers. In *Proceedings of the 8th Network and Distributed System Security Symposium*, February 2002. 40, 41

[22] D. Davis and D. Geer. Kerberos security with clocks adrift. In *Proceedings of the 5th USENIX Security Symposium*, June 1995. 16

[23] D. Davis and D. Geer. Kerberos security with clock adrift: History, protocols, and implementation. *Computing Systems*, 9(1):29–46, 1996. 16

[24] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. Generic AAA architecture. RFC 2903, August 2000. 58

[25] D. Dean, T. Berson, M. Franklin, D. Smetters, and M. Spreitzer. Cryptology as a network service. In *Proceedings of the 7th Network and Distributed System Security Symposium*, February 2001. 40

[26] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981. 11, 16

[27] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999. 12

[28] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transaction on Information Theory*, 22:644–654, November 1976. 13

[29] W. Diffie, P. van Oorschot, and M. Weiner. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, 2:107–125, 1992. 16

[30] Documentation: A Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`. 55

[31] Documentation: Administrators Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`. 55

[32] Documentation: Programmers Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`. 55

[33] D. Dolev and A. Yao. On the security of public-key protocols. *Communications of the ACM*, 29:198–208, 1983. 15

[34] W. Doster, M. Watts, and D. Hyde. The KX.509 protocol. Technical Report 01-2, Center for Information Technology Integration, University of Michigan, February 2001. 29, 59

[35] C. Ellison. Establishing identity without certification authorities. In *Proceedings of the 6th Annual USENIX Security Symposium*, July 1996. 16

[36] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, September 1999. 35, 39

[37] Expect. `http://expect.nist.gov`. 57

[38] S. Farrell, J. Vollbrecht, P. Calhoun, L. Gommans, G. Gross, B. de Bruijin, C. de Laat, M. Holdrege, and D. Spence. AAA authorization requirements. RFC 2906, August 2000. 58

[39] ITU-T (formerly CCITT) Information Technology Open Systems Interconnection. Recommendation X.509: The directory authentication framework, December 1988. 13, 16, 28

[40] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the 8th International Workshop on Quality of Service (IWQQOS 2000)*, June 2000. 55

[41] B. Fox and B. LaMacchia. Certificate revocation: Mechanisms and meaning. In *Proceedings of Financial Cryptography*, February 1998. 25

[42] B. Fox and B. LaMacchia. Online certificate status checking in financial transactions: the case for re-issuance. In *Proceedings of Financial Cryptography*, February 1999. 26

[43] A. Freier, P. Karton, and P. Kocher. Secure Socket Layer 3.0. Internet Draft, March 1996. 12

[44] A. Freier, P. Karton, and P. Kocher. The SSL protocol version 3.0, March 1996. Netscape Communications Corporation. 12

[45] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, August 2001. 42

[46] K. Gaarder and E. Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3:81–98, 1991. 15

[47] A. Goldberg, R. Buff, and A. Schmitt. Secure web server performance dramatically improved by caching SSL session keys. In *Proceedings of the Workshop on Internet Server Performance*, June 1998. 40

[48] D. Gollmann. What do we mean by entity authentication? In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996. 15

[49] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1990. 15

[50] R. Gorrieri and P. Syverson. Varieties of authentication. Panel Discussion during the 11th IEEE Computer Security Foundations Workshop, June 1998. 15

[51] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure, certificate and CRL profile. Internet Draft, June 1998. 26

[52] R. Housley, W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 2459, April 2002. 25

[53] J. Howell. *Naming and sharing resources across administrative boundaries*. PhD thesis, Darthmouth College, 2000. 16

[54] Grid Computing Portal: `http://hotpage.npaci.edu/cgi-bin/hotpage_top.cgi`. 58

[55] M. Hur and A. Medvinsky. Kerberos cipher suites in Transport Layer Security (TLS). Internet Draft, May 2001. 39

[56] T. Hwang and Y-H. Chen. On the security of SPLICE/AS the authentication system in WIDE internet. *Information Processing Letters*, 53:97–101, 1995. 16

[57] IETF Internet Traffic Engineering Working Group. `http://www.ietf.org/html.charters/ tewg-charter.html`. 55

[58] K. Jackson, S. Tuecke, and D. Engert. TLS delegation protocol. Internet Draft, February 2001. draft-ggf-tls-delegation-09.txt. 39

[59] R. Kailar and V. Gligor. On belief evolution in authentication protocols. In *Proceedings of the Computer Security Foundations Workshop IV*, 1991. 15

[60] N. Karonis, C. Kesselman, G. Koenig, and S. Tueke. A secure communication infrastructure for high-performance distributed computing. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, August 1997. 57

[61] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994. 14

[62] S. Kent. Privacy enhancements for internet electronic mail, certificate based key management. RFC 1422, February 1993. 28

[63] C. Kesselman and S. Tueke. Managing security in high-performance distributed environment. *Cluster Computing*, 1(1):95–107, 1998. 57

[64] P. Kocher. On certificate revocation and validation. In *Proceedings of Financial Cryptography*, February 1998. 26

[65] L. Kohnfelder. Towards a practical public-key cryptosystem. Bachelor's thesis, EECS Department, Massachusetts Institute of Technology, May 1978. 13

[66] D. Kormann and A. Rubin. Risks of the passport single signon protocol. *Computer Networks*, 33:51–58, 2000. 27

[67] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to web access control. In *Proceedings of the 10th USENIX Security Symposium*, August 2001. 29, 59

[68] B. Lampson. Designing a global name service. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, August 1986. 16, 28

[69] J. Linn. Generic security service application program interface, version 2, update 1. RFC 2743, October 2000. 57

[70] D. Longley and S. Rigby. Use of expert systems in the analysis of key management systems. *Security and Protection in Information Systems*, pages 213–224, 1989. 14

[71] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995. 12, 16

[72] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, January 1997. 16

[73] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997. 15

[74] A. Malpani and P. Hoffman. Simple certificate validation protocol. Internet Draft, August 1999. 26

[75] W. Mao and C. Boyd. Towards a formal analysis of security protocols. In *Proceedings of the Computer Security Foundations Workshop VI*, 1993. 15

[76] P. McDaniel and S. Jamin. Windowed certificate revocation. In *Proceedings of IEEE INFOCOM*, March 2000. 26

[77] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1989. 15

[78] C. Meadows. Representing parital knowledge in an algebraic security model. In *Proceedings of the Computer Security Foundation Workshop III*, June, 1990. 15

[79] C. Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991. 15

[80] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992. 14, 15

[81] C. Meadows. Formal verification of cryptographic protocols. In *Proceedings of "Advances in Cryptology – ASIACRYPT'94"*, 1994. 14, 15

[82] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Techonology, 1983. 15

[83] S. Micali. Efficient certificate revocation. Technical Report Technical Memo, MIT/LCS/TM-542b, Massachussetts Institute of Technology, March 1996. 26

[84] J. Millen, S. Clark, and S. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987. 14

[85] Project Minotaur: Kerberizing the Web, software at Carnegie Mellon University. `http://andrew2.andrew.cmu.edu/minotaur`. 39

[86] R. Mraz. Secure blue: an architecture for a high volume SSL internet server. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001. 40

[87] Microsoft Security Bulletin MS01-017. Erroneous VeriSign-issued digital certificates pose spoofing hazard, March 2001. 40

[88] M. Myers. Revocation: Options and challenges. In *Proceedings of Financial Cryptography*, February 1999. 26

[89] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure: Online certificate status protocol. RFC 2560, June 1999. 26

[90] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium*, January 1998. 26

[91] R. Needham and M. Shroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993 – 999, December 1978. 11, 17

[92] D. Nessett. A critique of the burrows, abadi and needham logic. *Operating System Review*, 24(2):35–38, April 1990. 15

[93] C. Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributing Computing Systems*, May 1993. 39

[94] C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994. 11, 58

[95] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating System Review*, 21(1):8–10, January 1987. 15

[96] .Net Passport 2.0 Technical Overview, 2000. `http://www.microsoft.com/myservices/passport/technical.asp` . 27

[97] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *IEEE Workshop on Policies for Distributed Systems and Networks*, 2002. submitted. 58

[98] MyProxy project. `http://dast.nlanr.net/ Projects/MyProxy`. 58

[99] SideCar project. Software at Cornell University. `http://www.cit.cornell.edu/ kerberos/sidecar.html`. 39

[100] J.T. Regan and C.D. Jensen. Capability file names: Separating authorisation from user management in an internet file system. In *Proceedings of the 10th USENIX Security Symposium*, August, 2001. 39

[101] R. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rump session, 1996. 28, 35, 39

[102] R. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21:120–126, February 1978. 13

[103] M. Roe. Certification authority requirements. PASSWORD Document R2.5, November 1992. 28, 30

[104] M. Roe and W. Schneider et al. Service requirements. PASSWORD Document R1.1, August 1992. 28

[105] A. Rubin. *Nonmonotonic cryptographic protocols*. PhD thesis, University of Michigan, 1994. 14

[106] T. Ryutov and C. Neuman. Representation and evaluation of security policies for distributed system services. In *Proceedings of the DISCEX*, January 2000. 39

[107] V. Sander, W. A. Adamson, I. Foster, and A. Roy. End-to-end provision of policy information for network qos. In *Proceedings of the 10th Symposium on High Performance Distributed Computing*, August 2001. 58

[108] J. Schiller and D. Atkins. Scaling the web of trust: Combining kerberos and pgp to provide large scale authentication. In *Proceedings of the USENIX Winter Technical Conference*, January 1995. 27

[109] SIBBS. The simple inter-domain bandwidth broker specification. `http://qbone.internet2.edu/bb/`. 55, 58

[110] D. Sidhu. Authentication protocols for computer networks. *Computer Networks and ISDN Systems*, 11:297–310, 1986. 14

[111] Netscape Single Singon, 2000. `http://developer.netscape.com/docs/manuals/security/SSO/sso.htmii#1053955`. 27

[112] M. Sirbu and J. Chuang. Distributed authentication in Kerberos using public key cryptography. In *Symposium On Network and Distributed System Security*, February 1997. 27

[113] E. Snekkenes. Exploring the BAN approach to protocol analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1991. 15

[114] E. Snekkenes. Roles in cryptographic protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1992. 15

[115] D. Song. Kerberized WWW access. `http://www.monkey.org/~dugsong/krb-www`. 39

[116] V. Staats. Kerberized TLS, June 2000. Private communication. 39

[117] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, February 1988. 11

[118] Stone Cold Software. Apache Kerberos Module. `http://stonecold.unity.ncsu.edu`. 39

[119] S. Stubblebine and R. Wright. An authentication logic supporting synchronization, revocation, and recency. In *Proceedings of the 3rd ACM Conference on Computer and Communication Security*, March 1996. 16

[120] P. Syverson. A logic for cryptographic protocol analysis. Technical Report 9305, NRL Formal Report, December 1990. 15

[121] P. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1:317–330, 1992. 15

[122] P. Syverson. Adding time to a logic of authentication. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, November 1993. 15

[123] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the IEEE Computer Security Foundations Workshop VII*, 1994. 16

[124] P. Syverson. Limitations on design principles for public key protocols. In *Proceedings of IEEE Symposium on Security and Privacy*, May 1996. 16

[125] P. Syverson and C. Meadows. A logical language for specifying cryptographic protocol requirements. In *Proceedings of the IEEE Symposium in Security and Privacy*, May 1993. 15

[126] International Telegraph and Telephone Consultative Committee (CCITT). The directory. Recommendations X.500, X.501, X.509, X.511, X.518-X.521, 1988. 16

[127] The Globus Resource Specification Language RSL v1.0. `http://www.globus.org/gram/rsl_spec1.html`. 57

[128] M. Thompson, W. Johnson, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate based access control for widely distributed resources. In *Proceedings of the 8th USENIX Security Symposium*, August 1999. 39

[129] D. Trcek and B.J. Blazic. Certification infrastructure reference procedures. NIST PKI Technical Working Group (W. Burr, Ed.), NISTIR 5788, NIST, September 1995. 28

[130] S. Tuecke, D. Engert, and M. Thompson. Internet X.509 public key infrastructure impersonation certificate profile. Internet Draft, February 2001. draft-ggf-x509-impersonation-05.txt. 39

[131] B. Tung, C. Neuman, and J. Wray. Public key cryptography for initial authentication in Kerberos. Internet Draft, April 2000. 26

[132] V. Varadharajan. Verification of network security protocols. *Computers and Security*, 8(8):693–708, 1989. 14

[133] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijin, C. de Laat, M. Holdrege, and D. Spence. AAA authorization framework. RFC 2904, August 2000. 58

[134] T. Woo and S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992. 16

[135] T. Woo and S. Lam. A sematic model for authentication protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1993. 15

[136] T. Woo and S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994. 16

[137] A. Yasinsac. *A Formal Semantics for Evaluating Cryptographic Protocols*. PhD thesis, University of Virginia, 1996. 14