TRUST NEGOTIATION USING HIDDEN CREDENTIALS

by

Evan Paul Child

A thesis submitted to the faculty of

Brigham Young University

In partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

July 2004

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Evan Paul Child

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

_____	_____
Date	Dr. Kent E. Seamons, Chair
_____	_____
Date	Dr. Yiu-Kai Dennis Ng
_____	_____
Date	Dr. Sean C. Warnick

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Evan Paul Child in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

_____
Date

_____
Kent E. Seamons
Chair, Graduate Committee

Accepted for the Department

_____
David W. Embley
Graduate Coordinator

Accepted for the College

_____
G. Rex Bryce
Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

TRUST NEGOTIATION USING HIDDEN CREDENTIALS

Evan Paul Child

Department of Computer Science

Master of Science

In large-scale distributed environments, strangers with no pre-existing relationship often seek to perform transactions with each other. Online users may utilize trust negotiation to authenticate and authorize strangers. Traditional trust negotiation allows users to exchange policies and prove ownership of credentials to gain access to resources or services. These credentials can contain private information, such as a name or social security number. Trust negotiation protects sensitive credentials by relying on a secure channel and requiring satisfaction of the policy protecting the credential before disclosure of that credential. However, the secure channel trust negotiation typically relies on, TLS, must disclose credentials during session establishment without any authentication or protection from eavesdroppers. This disclosure potentially violates the privacy of the participants.

Hidden credentials are a new approach to trust negotiation that allows parties to prove ownership of credentials without actually disclosing them. In this way, a user can protect his credentials during TLS session establishment. Additionally, hidden credentials are suitable for use over insecure protocols, such as email, since they do not require a secure channel for authentication.

This thesis advances the application of hidden credentials in three ways. First, it defines a general-purpose hidden credentials message format standard for use in a variety of usage models. Second, it overcomes privacy limitations in TLS authentication through the design and implementation of an extension to TLS authentication that utilizes hidden credentials. Third, it provides increased privacy and authentication capabilities to email users through the design and implementation of an extension to email that utilizes hidden credentials.

ACKNOWLEDGEMENTS

I am very grateful to my wife, who has been an amazing support to me throughout our marriage and especially during this time in the Masters program. I am grateful for my sons Elijah and Brigham, who have brought much happiness into my life. I am thankful to have had Dr. Kent Seamons as my advisor; his advice and feedback have been invaluable and his patience unending. I am grateful for his vision to see what I could achieve. Finally, I cannot adequately express the gratitude I feel to my Father in Heaven for the many opportunities I have been given.

This research was supported by funding from DARPA through SSC-SD grant number N66001-01-1-8908, the National Science Foundation under grant no. CCR-0325951 and prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Because of the vastness of the Internet, strangers with no pre-existing relationship often seek to perform transactions with each other. These transactions may be of a sensitive nature, and require some type of authentication and/or authorization before either party is willing to grant access to a service or resource. Typically, in role-based access control, a server maps identities to certain roles. A resource's access control policy requires that an entity identify itself, which allows the server to decide whether or not to grant access to the resource. However, since the parties participating in Internet transactions may be strangers, they most likely reside in different security domains and lack a pre-established relationship. This prevents the server from establishing a meaningful mapping between a supplied username and role in the local security domain.

Trust negotiation provides a way to overcome the problem of authentication between disparate security domains by allowing parties to use non-forgeable digital credentials to establish trust. The parties establish trust by mapping the attributes found in these credentials to roles in their local security domain. The parties iteratively exchange credentials until the policy protecting the resource is satisfied and the server grants access to the requesting client. The iterative approach provides incremental authentication for each side to prevent disclosure of sensitive credentials to unauthorized parties required to satisfy a policy. One approach to performing trust negotiation, referred to as traditional trust negotiation throughout this thesis, involves the exchange of credentials and policies between a client and server over a secure channel.

Traditional trust negotiation [33] relies upon a secure channel to exchange digital policies and credentials. This secure channel is necessary to provide confidentiality for

the exchanged credentials, which can be sensitive. One way to establish a secure channel is through a secure protocol such as transport layer security (TLS) [6]. However, protocols such as email (SMTP, POP3, IMAP), the hypertext transfer protocol (HTTP), and the file transfer protocol (FTP) do not provide a secure channel. In addition, although TLS is secure, the authentication methods within TLS expose the credentials used during session establishment. Consequently, these authentication methods do not afford much privacy for either the client or server. Because these protocols are ubiquitous on the Internet today, extending them to perform trust negotiation securely and privately can provide great privacy benefits to users on the Internet.

Protecting the privacy of information in digital certificates is critical to the continued growth of the Internet. Credentials can contain sensitive personal information. Verisign, a certificate authority, issued around 3,000,000 certificates containing sensitive demographic information between 1997 and 2000 [24] including country, zip code, age, and gender. Almost ninety percent of American Internet users say they are concerned about general threats to their privacy online [32]. As a result, better methods to protect credential privacy are needed. The words certificate and credential are used interchangeably throughout this thesis.

This thesis research leverages hidden credentials [14], a new method for doing trust negotiation without actually showing any credentials or policies. Credentials and policies may contain sensitive information that the owner does not wish to reveal. However, showing these credentials and policies is currently required in traditional trust negotiation.

The primary goals for this thesis are to enhance user privacy by:

1. Creating a generic hidden credentials message format for use in a wide range of protocols
2. Protecting certificates during TLS session establishment using hidden credentials
3. Providing a method to do trust negotiation securely over insecure protocols such as email (SMTP, POP3, IMAP), HTTP, and FTP using hidden credentials

Hidden credentials utilize identity-based encryption (IBE) [3] to avoid certificate and policy disclosure when authenticating another party in a transaction. For example, Alice wishes to send a message to Bob, but she only wants Bob to be able to open it if he is a manager at the store where he works. Alice encrypts the email using the key derived from the string "manager" || "bob@store.com", where "||" signifies concatenation. Bob's unique identifier, his email address, is also known as a nym. Bob does not need to send his manager certificate to Alice to prove he is a store manager.

Bob must decrypt the message using the private keys of his credentials that correspond to the public keys Alice used to encrypt the message. Hidden credentials differ from IBE primarily because they support the use of complex encryption policies that require more than one credential, which IBE does not. If Alice does not send the policy, (i.e., which credentials Bob must use to decrypt the email message), Bob must try all of his credentials to determine which set of credentials satisfies the policy. Alice may facilitate the decryption of the email by sending Bob the entire policy, or hints as to which credentials to use. By allowing parties to avoid disclosing policies and credentials,

hidden credentials are an excellent way to afford the client and server greater privacy on the Internet.

Because hidden credentials are a new approach to trust negotiation, a standard message format does not exist. For security and interoperability, a standard message format is necessary to thwart potential attacks and provide a guide for implementers. Therefore, this thesis proposes a hidden credentials message format that mitigates the risks associated with common threats in a network environment. It also presents a threat analysis to show how to reduce these risks.

TLS establishes a secure channel through several different authentication methods to ensure channel confidentiality and integrity. However, these authentication methods do not protect the client or server's credentials used during session establishment from an eavesdropper. This credential disclosure may constitute a violation of privacy if sensitive information is contained in the credentials. Because of the inherent protection of credentials and policies, hidden credentials can ensure additional privacy beyond that provided by current authentication methods in TLS. This thesis proposes a way to enhance the privacy of TLS using hidden credentials.

Email is widely used for exchanging messages between two or more parties. Performing trust negotiation securely over email is valuable because of the large number of inter-organizational email exchanges. One survey estimates that 85% of the companies using the Internet use it for inter-company email [10]. The vast majority of these interactions most likely occur between organizations in disparate security domains. These interactions typically do little or nothing to authenticate and authorize the involved parties, despite the possible sensitive information in the messages. Consequently, being

able to perform trust negotiation over email can help organizations with no pre-existing relationship to easily authenticate and authorize each other when exchanging email. This thesis proposes a way to use hidden credentials to negotiate trust over email. Throughout the remainder of the thesis, email refers to three insecure email protocols: SMTP, POP3, and IMAP. The vast majority of email sent today utilizes these protocols and is not secure.

1.1 Thesis Statement

Because online privacy is critical to almost ninety percent of Americans on the Internet today [32], they need the ability to protect their privacy and authenticate strangers during online transactions. Trust negotiation helps provide privacy protection and authentication between parties with no previous relationship. Hidden credentials are a new way to perform trust negotiation that limits the information disclosed during an interaction. Integrating hidden credentials into email and TLS enables powerful privacy safeguards and robust authentication methods.

1.2 Motivating Scenarios

The following scenarios show how integrating hidden credentials with TLS and email can be beneficial to privacy:

Scenario 1: Hidden credentials integrated with email for policy fulfillment protection

Jimmy, claiming to be a member of a government agency, telephones Elijah, a VP at a government defense contractor, and requests information regarding one of the top-secret communication modules the company is building. Elijah agrees to send him the document via email. However, Jimmy will not reveal what department he is from, as this is classified information. Elijah encrypts the message using the combination of Jimmy's

email address and the following attributes: “FBI_Agent”, “CIA_Agent”, “WhiteHouse_staff”, and “Dept_Homeland_Security”. The encryption procedure ensures that Jimmy will only be able to open the document if he is actually a member of one of the aforementioned groups. Elijah accomplishes all of this without having to see Jimmy’s credentials. Elijah does not learn how Jimmy fulfills the policy, if at all.

Scenario 2: Hidden credentials integrated with email to solve the cyclic policy dependency problem

CIA agents Daniel and Brigham wish to engage in a secure online chat to talk about related investigations. Both agents have policies governing the disclosure of their CIA credentials that state that they will only reveal the credential to another CIA agent who is a member of a special task force. With traditional trust negotiation, these parties would fail to negotiate trust since neither would be able to satisfy the other’s policy without violating their own. Hidden credentials can overcome this cyclic policy dependency by serving as the authentication method within TLS prior to their conversation. Agent Daniel can use hidden credentials to encrypt the message to agent Brigham with the keys derived from the attributes, “CIA_Agent” and “Special_Task_Force” along with agent Brigham’s unique identifier. Thus, agent Daniel does not disclose the fact that he is looking for a CIA agent, but still allows agent Brigham to fulfill his policy. Agent Brigham may respond with a similarly encrypted message and once they have authenticated each other, the TLS session continues and Daniel and Brigham chat securely.

Chapter 2 Foundational Material

2.1 Definitions

- *Authenticity* is the ability to verify that a message is from the expected person.
- *Confidentiality* assures that only the intended message recipient can read a message.
- *Integrity* guarantees that nothing has altered a message in transit.
- *Privacy* is a more generic term that encompasses confidentiality, but extends it to include avoiding other types of potential revelations, especially credential attribute values and policy requirements.
- *Access control* is the method whereby administrators make certain that only authorized users can gain access to specific resources or services.
- *Availability* refers to the accessibility of a service or resource. For example, a denial-of-service attack can limit the availability of a service.

2.2 Trust Negotiation

Trust negotiation allows parties with no previous relationship to establish trust through the exchange of digital credentials and access control policies. This section provides an overview of the way trust negotiation works. It explains some of the recent advances made to further trust negotiation research and discusses the current implementation of trust negotiation from the Internet Security Research Lab (ISRL) at BYU known as TrustBuilder.

In a transaction requiring trust negotiation, a client requests a protected resource or service from a server. Both parties can use non-forgeable digital attribute credentials, which assert some attribute about the possessor, to satisfy policies supplied by the other

negotiating party. Often credentials may be sensitive, thus requiring a policy to protect them. In addition, the policy can contain sensitive information that requires protection by another policy. Both sides exchange credentials and policies until the desired level of trust is achieved, and the resource is disclosed or access is denied.

Several interesting properties of trust negotiation make it a useful paradigm for authentication. First, since trust negotiation does not require a pre-existing relationship, authentication between disparate security domains can occur with no pre-registration, such as a username and password. Second, a user has fine-grained control over credential disclosure. Third, a trust agent administers and enforces the policies protecting credentials and policies on behalf of the user. The automated protection of credentials and policies maintains a high level of privacy without requiring that the user interactively approve each disclosure.

Researchers in the ISRL have advanced trust negotiation in several different ways. Content-triggered trust negotiation [13] allows for automatic protection of sensitive outbound data based on content analysis. Using selective disclosure [17] with X.509v3 digital certificates is an effective way to limit the disclosure of attributes not pertinent to a trust negotiation. Surrogate trust negotiation [29] allows low-powered devices to have their home machines store their sensitive digital credentials and negotiate trust on their behalf. Intra-session trust facilitates multiple requests per session [16] without trust re-negotiation. Inter-session trust allows a party to request access to a resource in a new, separate session, without trust re-negotiation [4].

TrustBuilder is an implementation of trust negotiation by the ISRL. It relies upon X.509v3 digital certificates and the Trust Establishment (TE) [11] policy compliance

checker developed by IBM. The policies are in an XML format required by TE. Typically, credentials and policies flow back and forth across the transport protocol over which the two parties are negotiating. Various protocols in which trust negotiation has been implemented include HTTP [9], SSH [30], and TLS. Researchers in the ISRL implemented TrustBuilder in Java, though it can interact with clients through Java or an XML-based remote procedure call framework such as SOAP. Figure 1 illustrates the architecture.

Figure 1 – Example Trust Negotiation Architecture

2.3 Identity-Based Encryption

Hidden credentials rely on identity-based encryption (IBE) [3]. IBE defines functions that allow anyone to calculate or derive a public key, and only the CA to calculate the private key. A user desiring a credential certifying a particular attribute

submits this value to the CA, along with their unique identifier, or nym. For example, Bob desires a credential with attribute “software_engineer” for his nym “bob@company.com”. He submits this request to the CA, who must authenticate that Bob has those attributes and release the private key accordingly. If the CA issues the key to Bob, he can now decrypt any message encrypted with “software_engineerbob@company.com”. If Alice wishes to obtain the public key associated with Bob’s private key, she must derive it with the CA’s public key from the string “software_engineerbob@company.com”. Alice can now encrypt a message with the appropriate public key, send it to Bob, and Bob can decrypt it with the corresponding private key. It is not necessary that Bob obtain his private key before Alice derives the public key. In fact, Bob can request the private key after he has received the message.

2.4 Hidden Credentials

2.4.1 Overview of Hidden Credentials

Hidden credentials allow a party to encrypt a resource so that only individuals with the correct attribute credentials can decrypt it. They also provide the ability to mask policies, and avoid disclosure of credentials. In addition, hidden credentials provide a constant bound on the number of message exchanges required between negotiating parties for most interactions. Typically, there will be a single round where the parties exchange nyms, and then another round where the parties exchange messages encrypted according to the respective policies guarding the requested resource or service. This section gives a more in depth introduction to hidden credentials and contextualizes their usage.

Hidden credentials encrypt messages according to Boolean expression policies. An example is as follows:

$$P = (C_1 \text{ AND } C_2) \text{ OR } (C_3 \text{ AND } C_4)).$$

In this policy, C_1, C_2, C_3, and C_4 represent the credentials required to satisfy the policy. For example, to decrypt a message encrypted according to policy P, the recipient must possess C_1 and C_2, or they must possess C_3 and C_4. The way to express this policy when encrypting is to first encrypt resource R with C_1, and encrypt the resulting ciphertext with C_2 to represent the first sub-expression $(C_1 \text{ AND } C_2)$. Next, encrypt R with C_3 and the resulting ciphertext with C_4 to represent the sub-expression $(C_3 \text{ AND } C_4)$. Finally, concatenate both sub-expressions, separated by a delimiter, to achieve the final, fully encrypted ciphertext.

To build upon the afore-mentioned example, Alice wishes to send an email to Bob. Alice desires that Bob be a member of the IEEE Printing Committee and be a software engineer or program manager at the company where he works. The following expression represents Alice's generic policy:

P = ("IEEE_PrintingCommittee" AND "software_engineer") OR

("IEEE_PrintingCommittee" AND "program_manager")

The actual policy expression Alice uses to encrypt the message using Bob's nym is the following:

P = ("IEEE_PrintingCommitteebob@company.com"

AND "software_engineerbob@ company.com") OR

("IEEE_PrintingCommitteebob@ company.com"

AND "program_managerbob@ company.com")

Alice will derive the public keys associated with these respective attribute + nym combinations, encrypt the message according to the policy, and send the result to Bob.

If Alice has not given Bob any policy hints, he must try all of his credentials to discover the correct decryption path. Each combination of credentials in the policy represents a path, or fulfillment path, to satisfy the policy. Figure 2 illustrates the notion of policy fulfillment paths. For Bob to decrypt the email, he must have the private key associated with the public key for each of the elements in a fulfillment path of the policy. He can know when he has successfully decrypted the message by searching for elements that he knows must occur in a properly formatted plaintext message.

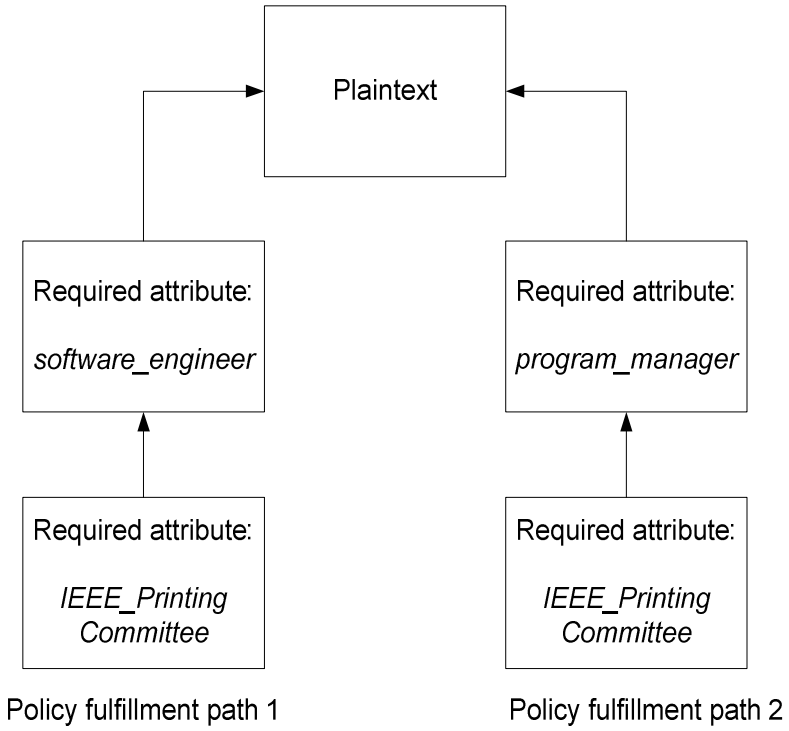


Figure 2 – Example of Policy Fulfillment Paths

2.4.2 Properties of Hidden Credentials

There are multiple beneficial privacy and performance properties gained by using hidden credentials. Three properties are especially relevant to this thesis. First, hidden

credentials solve the cyclic policy dependency problem. Second, they protect credentials and policies by not requiring disclosure of the former and limiting leakage of the latter. Third, hidden credentials may require fewer round trip message exchanges than a comparable traditional trust negotiation.

Hidden credentials solve the cyclic policy dependency problem by not requiring either party to violate their policy if there is a cyclic dependency in the policies. The following is a summary of the example originally given in Section 1.2 that illustrates the cyclic policy dependency problem.

> Two CIA agents, Daniel and Brigham, wish to communicate, but both have policies protecting disclosure of their CIA credential. For security purposes, their policies only allow disclosure of their CIA credential to another CIA agent. They seemingly are at an impasse since neither is willing to disclose their credential without the other going first.

Hidden credentials solve the cyclic policy dependency problem by allowing either agent to send a message that does not disclose their credential or the policy, but only someone who has the appropriate CIA credential can decrypt it. For example, Daniel sends a message to Brigham encrypted with the public key derived from the identifier "CIA_Agent" concatenated with "Brigham@cia.gov". Thus, if Brigham has the appropriate "CIA_Agent" credential, he will be able to decrypt the message. Brigham will then send a message back to Daniel encrypted in a likewise fashion. Thus, hidden credentials solve the cyclic policy dependency problem by allowing a party to send a message knowing that if the receiving party does not have the necessary credentials, they will learn nothing about the decryption policy. Although hidden credentials solve the

cyclic policy dependency problem, they do not guarantee a fair exchange of messages. Even if Daniel understands the message Brigham sends to him, he is not obligated to respond. Brigham may never learn if Daniel is an agent, but Brigham can know that Daniel will only be able to open the message if he is one.

2.4.3 Scenarios for privacy protection with hidden credentials

Hidden credentials also allow clients and servers to encrypt messages, prove ownership of credentials, and mask the policies needed to decrypt the messages without actually sending either the credentials or the policies. They prevent leakage of information regarding credential possession to unauthorized parties. The following list illustrates the different possible scenarios in terms of possession of credentials when two parties interact:

1. Client does not have credentials to satisfy server's policy, and server does not have credentials to satisfy client's policy.
2. Client has credentials to satisfy the server's policy, the server does not have the credentials to satisfy the client's policy.
3. Client does not have credentials to satisfy the server's policy, the server does have the necessary credentials to satisfy the client's policy.
4. Client and server both have the necessary credentials to satisfy each other's policies.

The following are examples of each scenario to illustrate how hidden credentials protect credentials and policies.

Scenario 1: Malfoy, who is not a CIA agent, wishes to find out if Carol, who is also not a CIA agent, possesses a CIA credential. He sends a hidden credentials message

encrypted with Carol's nym and the string "CIA_Agent", derived using the public key of the CIA. Carol attempts to decrypt the message, but since she is not a CIA agent, she does not have the necessary credentials. Additionally, Carol does not even know what credentials are required to decrypt the message. Therefore, she decides to send a message back to Malfoy to prevent him from learning whether she actually possesses the credential with which he encrypted his message. By sending a message back, encrypted with a NAK (a non-existent credential), Malfoy does not know if Carol did not understand his original message, or if he does not possess the right credentials to decrypt the response. Therefore, nothing is revealed to either party about what credentials they do or do not possess. Hereafter, NAK policy encrypted messages are referred to as NAK messages, or messages encrypted with a NAK.

Scenario 2: Brigham, who is a CIA agent, wishes to find out if Carol, who is not a CIA agent, possesses a CIA credential. He encrypts the message using the key derived from Carol's nym, the string "CIA_Agent", and the public key of the CIA. Carol, upon receiving the message, attempts to decrypt it. When she cannot decrypt it, she sends a response encrypted with a NAK. Because Brigham already knows the necessary policy to decrypt a response encrypted with a CIA credential, he can reasonably assume that Carol is not a CIA agent, since he could not decrypt the message. However, if he did not have this prior knowledge of the policy, he could make no such assumption.

Scenario 3: Malfoy, who is not a CIA agent, wishes to find out if Brigham, who is a CIA agent, possesses a CIA credential. He sends a message encrypted with the key derived from Brigham's nym, the string "CIA_Agent", and the public key of the CIA. Brigham receives the message, and decrypts it. He then responds with a message

encrypted with the key derived from Malfoy's nym, and the string "CIA_Agent". Malfoy cannot decrypt the message, therefore he does not know if Brigham responded with a NAK message, or a legitimately encrypted CIA agent message. Neither party learns anything about the other in terms of what credentials they have.

Scenario 4: Brigham, who is a CIA agent, wishes to find out if Elijah, who is also a CIA agent, possesses a CIA credential. He sends a message encrypted with the key derived from Elijah's nym, the string "CIA_Agent", and the public key of the CIA. Elijah receives the message and successfully decrypts it. He then constructs a similar message using Brigham's nym and sends it back. Brigham then decrypts the message and discovers that Elijah possesses a CIA credential, since he was able to respond to the request properly. At this point, Brigham knows that Elijah is a CIA agent, but Elijah does not yet possess the knowledge as to whether Brigham is an agent. However, if their interaction continues, and Brigham sends Elijah a well-formatted message that Elijah is able to decrypt and verify, Elijah knows that Brigham is a CIA agent.

Hidden credentials also provide the ability for interacting parties to fulfill monotonic Boolean policy expressions with two round-trip exchanges of messages. First, both parties must exchange nyms, unless previously exchanged. Second, they must exchange messages encrypted according to the appropriate policy. Each side can attempt to fulfill the policy by simply trying to decrypt the message sent from one party to another. In traditional trust negotiation, a policy would theoretically require multiple rounds of messages sent back and forth, depending on whether the policy had another policy protecting it. Thus, hidden credentials can help reduce the number of round trips required to negotiate trust to two, a nym exchange round and a message exchange round.

Chapter 3 System Design

The primary goals for this thesis are to enhance user privacy by:

1. Creating a generic hidden credentials message format for use in a wide range of protocols
2. Protecting certificates during TLS session establishment using hidden credentials
3. Providing a method to do trust negotiation securely over insecure protocols such as email (SMTP, POP3, IMAP), HTTP, and FTP using hidden credentials

To achieve these goals, four primary areas of research are proposed:

1. Design a general-purpose message format for hidden credentials
2. Design a method to utilize hidden credentials as an authentication method in TLS
3. Design a method to integrate hidden credentials into email
4. Provide prototype implementations for these protocols utilizing the developed message format

The following sections present a discussion of some of the usage models considered while designing the message format. They provide an explanation of the hidden credentials message format. They also include descriptions of the integrations into the TLS and email protocols. Chapter 4 presents the implementation details for each of these protocols.

3.1 Usage Models

There are three usage models for hidden credentials messages. The first usage model is a single, one-way message in which no immediate response is necessary. The second is a single round exchange in which the client sends one message and receives one reply. The third is a session-based interaction predicated upon successful completion of the hidden credentials authentication. Each of the aforementioned usage models is important in the privacy sensitive context of a hidden credentials message exchange.

One of the features of hidden credentials is that Alice can prevent Bob from knowing how or if she was able to satisfy his policy, insofar as the circumstances allow that information to be concealed. It is possible for Alice to conceal this information in the first two usage models. However, in the third model, if Alice and Bob successfully authenticate, they necessarily reveal that they satisfied each other's policies since continuing the session is based on successful authentication. Neither Alice nor Bob learns anything if one of them does not possess the necessary credentials.

To exemplify the first scenario, suppose Alice wishes to send a single email to Bob without expecting a reply. Alice never learns whether Bob satisfied the policy and was able to successfully decrypt the message.

An example of the second scenario is an HTTP request/response where Alice sends a request encrypted with the appropriate policy and Bob encrypts the response according to his policy. However, if Alice cannot successfully decrypt the reply, she learns nothing regarding what credentials Bob may have. Alice does not know if Bob understood her message and sent a legitimate reply that she is not authorized to decrypt,

or whether Bob did not understand the original message and is sending a NAK message back.

The third scenario is a session-based protocol, such as TLS, where a long-running session follows successful authentication. When authentication is successful, both Alice and Bob know that the other has satisfied their policy since they continue with their session. If Bob cannot satisfy the policy, he may bluff by sending a NAK message. Alice cannot understand this message, but not knowing if it is a NAK message or a message she does not have the credentials to decrypt, she does not want to reveal that she could not satisfy the policy. Therefore, she returns a NAK message. At this point, neither party is willing to admit that they could not understand the other's latest message, so they continue bluffing, sending NAK messages back and forth until either a pre-defined number of rounds, or one party terminates the connection.

3.2 Hidden Credentials Message Format

Designing message formats for security protocols is a non-trivial process. There exist many possibilities for people to sign incorrect data, send ambiguous messages, transmit re-playable data messages or authentication material, allow digital identity theft (spoofing), incorrectly authenticate others, inadvertently expose private information, or other types of security problems. Many protocols once considered secure have later been found to have significant flaws. Even secure protocols may have implementation vulnerabilities [19].

This section identifies several eavesdropping and man-in-the-middle threats relevant to hidden credentials. A description of the message format for hidden credentials attempts to mitigate or eliminate these threats. Chapter 5 contains a detailed threat

analysis for these and other more general threats and relates them to hidden credentials. Throughout this chapter, Alice plays the role of client, or sender, and Bob fills the role of server, or recipient.

The following are design goals for the message format:

1. Thwart certain types of eavesdropper and man-in-the-middle attacks
2. Establish a standard message format for all usage models
3. Extensibility

The focus of the first goal is to prevent eavesdropper and man-in-the-middle attacks using cryptographic algorithms. The purpose of the second goal is to set a standard for hidden credentials to allow disparate implementations on diverse platforms to interoperate. Another part of the second goal is to allow library reuse across the different usage models. Finally, the third goal is desirable to support future protocol enhancements.

There are two parts of a hidden credentials message: the external and the internal. The external message format facilitates version interoperability and ease of transmission. The internal message format reduces the threat of man-in-the-middle attacks and ensures message timeliness. It also provides structure to the plaintext for correct decryption recognition.

3.2.1 External Message Format

The external message format is a wrapper for the encrypted internal part. Figure 3 is an example of the external portion of a hidden credentials message in XML. In the discussion that follows, "field" denotes an XML tag that can contain a set of values. Each message contains an `hc_external` field, which contains all the other fields in the

external format. The `version` field allows future implementations to identify previous version messages.

The `fulfillment_paths` field contains the `fulfillment_path` fields, the roots of all the different ways to decrypt the message. Each `fulfillment_path` field represents a path in the policy used to encrypt this message. Bob must examine these paths and potentially try each one of them to decrypt the message successfully.

```
<hc_external>
        <version>1.0</version>
        <fulfillment_paths>
                <fulfillment_path>
                        <ciphertext>as;kldfj;dlkfjasdlfk</ciphertext>
                </fulfillment_path>
                <fulfillment_path>
                        <ciphertext>as;kldfj;dlkfjasdlfk</ciphertext>
                </fulfillment_path>
        </fulfillment_paths>
        <content encoding="base64"
                encryption-algorithm="rc4">ak/
.uqoi;ejksralmcz.,xbn.,mzD;aosijeu09821304986u234o</content>
</hc_external>
```

Figure 3 – Example of External Fields in a Hidden Credentials Message

To create a hidden credentials message, Alice generates a random symmetric encryption key for encrypting the internal portion of the message using a symmetric key algorithm such as RC4 [25] or AES [8]. She puts the encrypted internal portion of the hidden credentials message in the `content` field. It has an attribute that specifies the encoding, such as base64. She encrypts the symmetric key using hidden credentials according to the policy protecting disclosure of the plaintext message.

Alice encrypts the message with a separate symmetric key because hidden credentials utilize public key algorithms, which are often too computationally expensive to encrypt long documents [26]. Encrypting with a symmetric key, and then encrypting the relatively short symmetric key with hidden credentials, decreases the encryption time.

The security community does not recommend direct encryption of message plaintext using public key algorithms because of potential security vulnerabilities. Public key cryptography is vulnerable to chosen-plaintext attacks [26]. A chosen-plaintext attack occurs when an attacker captures a ciphertext and attempts to guess the original message plaintext by encrypting a chosen plaintext with the public key. If the resulting ciphertext is the same as the captured ciphertext, the attacker knows the plaintext.

A summary of the external fields in a hidden credentials message is contained in Table 1.

Field	**Description**
`Version`	Specifies the version of the message
`Fulfillment_paths`	Contains the different fulfillment_path fields wherewith the recipient may decrypt the message key
`Fulfillment_path`	Contains a path in the policy for decrypting the message
`Content`	Contains the encrypted internal portion of the message

Table 1 – Summary of External Fields of a Hidden Credentials Message

3.2.2 Internal Message Format

The internal portion of the hidden credentials message contains several different fields designed to ensure message timeliness while preventing certain types of man-in-the-middle attacks. Figure 4 contains an example of the internal portion.

```
<hc_internal>
        <authentication-token>10923lfasdgh</authentication-token>
        <sending-nym>my_nym</sending-nym>
        <receiving-nym>his_nym</receiving-nym>
        <timestamp>12/14/1976:05:02:05.113</timestamp>
        <session-id>382940172358</session-id>
        <message-id>28384910</message-id>
        <plaintext encoding="base64">My voice is my passport.</plaintext>
</hc_internal>
```

Figure 4 – Example of Internal Fields in a Hidden Credentials Message

The `authentication-token` field allows Alice to authenticate proactively to Bob without requiring a response message. Typically, hidden credentials returns a requested resource encrypted according to the protecting policy. However, for services that do not return a resource, Alice may proactively authenticate herself as an optimization to avoid further messages. One required assumption is that Alice must know the policy protecting the resource that she is requesting. If this assumption is valid, then she may proactively authenticate herself. To authenticate herself, she must prove ownership of the private keys associated with the public keys of the attributes required by the policy protecting the service she is requesting.

The following example illustrates how Alice can create the authentication tokens for proactive authentication. Bob is a bank. Alice is a client of Bob. Alice wishes to transfer $100 from her checking account to her savings account. This service, known as the transfer service, is protected by policy ***A***, which requires proof of ownership of a driver's license, a birth certificate, or a passport of the account holder. Therefore, Alice must prove ownership of at least one of those items to satisfy policy ***A*** so that Bob will execute the transfer. Alice creates the transfer request and inserts it into the internal portion of the hidden credentials message. She also creates and signs the authentication

tokens for each credential required by policy $\boldsymbol{A}$ that she possesses. In this way, Alice may proactively authenticate herself to Bob by proving ownership of one of the necessary credentials without any response required.

Bob may not be authorized to view Alice's passport credential or birth certificate, only her driver's license credential. Therefore, after creating the authentication token for each respective credential, Alice encrypts each authentication token according to the policies $\boldsymbol{P_P}$, $\boldsymbol{P_{DL}}$, and $\boldsymbol{P_{BC}}$, which protect disclosure of the credentials used to create each token. In this way, Bob can only see the authentication tokens for which he has the necessary credentials and can satisfy the respective policy. Alice does not reveal anything about what private keys she may have used in creating the authentication tokens. In other words, she does not reveal to Bob that she possesses a passport if he is not authorized to see it, even though she sent proof of possession of a passport.

Alice must create cryptographically secure authentication tokens. One way Alice can create the tokens is by signing, with her private key, a SHA-1[7] hash of the concatenation of the public key of Bob's CA and the internal message minus the authentication token fields, as illustrated in Equation 1. If Alice will use more than one CA to encrypt the external message, she concatenates all respective public keys in numerical order. Alice then encrypts the signature for each credential with the policy

$$Auth\_token_{passport} = HC_E(SIGN(SHA-1(Msg_{internal} \parallel PublicKey_{CA_{1..N}})), P_P, nym_{Bob})$$

Equation 1 – Creation of the Passport Authentication Token

protecting disclosure of the attribute associated with that private key, e.g. policy $\boldsymbol{P_P}$ protecting her passport credential. Bob decrypts the authentication tokens using the public keys derived from the attributes in policies $\boldsymbol{P_P}$, $\boldsymbol{P_{DL}}$, and $\boldsymbol{P_{BC}}$. At that point, Bob

must decrypt the signature using the public key of the passport, driver's license, or birth certificate. He compares the value of the decrypted signature against the SHA-1 hash of his CA's public key and the decrypted internal message. If they are equal, he knows that Alice possesses the credentials required by the policy protecting the transfer service.

Bob must not be able to impersonate Alice by using her authentication tokens to authenticate to another server. If Bob were to forward the message on to another server Carol, she would be able to detect that the authentication tokens were not intended for her and discard the message. Carol could detect this since the SHA-1 hash of her CA's public key and the internal message do not match the decryption of the authentication tokens. If Carol and Bob are using different CAs, the values will not match since the CA public key used in the calculation of the token will be different for both parties. If Carol and Bob are using the same CA, the tokens will not match since the nyms used in the internal message will not match the respective sending and receiving parties. In this fashion, the authentication tokens prevent Bob from impersonating Alice to another server.

The `sending-nym` field allows Bob to verify that the nym received during the initial nym exchange with Alice is legitimate. Bob is able to detect if Malfoy substitutes his own nym for Alice's nym during the nym exchange. If Malfoy is someone the CA already trusts by possessing the appropriate attribute credentials, he could decrypt the response from Bob. If not, he can disrupt communication and neither Alice nor Bob could detect why. A more complete description and example of this threat is given in the "Insider Attack, Same Credentials as Client" section of Chapter 5.

The `receiving-nym` field helps prevent replay attacks in one-way messages with authentication tokens. It does this by ensuring that when the hash of the internal

message is calculated with the `receiving-nym` in place, Bob cannot forward the request to anyone else trusted by the same CA with his same attributes. He cannot forward them because the `receiving-nym` field would not match the actual recipient's nym, and they would reject the message.

The `timestamp` field also prevents replay attacks. Bob must verify that the timestamp is within the allowable range for transmission delay. This assumes Alice and Bob have closely synchronized clocks. The primary purpose of the `session-id` field is to allow Alice and Bob to have multiple rounds of exchanges, while still tying all exchanges to the same session. This can be particularly helpful when exchanges are over email and there is no notion of a single session on a single physical TCP connection.

The `message-id` field helps to thwart replay attacks in single and multiple-round scenarios. Even though the timestamp helps prevent replay attacks, it is possible for a malicious user to duplicate a message immediately, still within Bob's receive window. In a transaction sensitive to this type of attack, such as a funds transfer request to a bank, this is a critical field. Bob must keep a record of all the message-ids received in a session during a valid time interval in order to detect a replay attack.

The message format provided by the internal portion of a hidden credentials message allows Bob to verify whether he has decrypted Alice's message properly. If Bob finds certain required fields that are present in a correctly decrypted message, he has confidence that he decrypted it properly. Depending on the number of characters which are in the message alphabet (i.e., all the different possible characters, such as A-Z, a-z, 0-9, /, ., +, -, =, _, etc.) the probability is low for the occurrence of a string representing a certain field in an incorrectly decrypted message.

Table 2 contains a summary of the internal fields in a hidden credentials message.

Field	Description
`Authentication-token`	Method for client to proactively authenticate herself to the server
`Receiving-nym`	Nym of the person to whom the message will be sent
`Sending-nym`	Nym of the person who actually sent the message
`Timestamp`	Timestamp of when the message was generated
`Session-id`	Unique identifier to allow perpetuation of session beyond physical connections, such as might be necessary in email
`Message-id`	Unique identifier for each message in a session
`Plaintext`	Actual contents of the message

Table 2 – Summary of Internal Fields of a Hidden Credentials Message

The following is a summary of all the steps required to create a hidden credentials message. This assumes that Alice and Bob have previously exchanged nyms.

1. Alice creates the plaintext message: "My voice is my passport."
2. Alice creates the internal hidden credentials message and puts the plaintext message in the `plaintext` field. She also inserts the current time, Bob's nym, her own nym, a session-id, and a message-id.
3. Alice creates an authentication token.
    a. First, she computes the SHA-1 hash of the concatenation of the internal hidden credentials message and the CA public key used to derive the required attribute credential public keys.
    b. Next, she signs the result from step a with the private key of the attribute credential of which she wishes to prove ownership.

c. Finally, she encrypts the result from step b with the policy protecting disclosure of the attribute credential used to sign the hash in step b and inserts that into the internal message in an `authentication-token` field.

d. Alice repeats steps a, b, and c for each authentication token.

4. Alice generates a random symmetric key and encrypts the internal portion, including the authentication tokens, with that key. She then inserts the resulting ciphertext into the `content` field of the external hidden credentials message format.

5. Alice encrypts the random symmetric key using IBE according to the policy protecting the message and inserts each resulting policy path in `fulfillment_path` fields.

6. Alice sends the hidden credentials message to Bob.

3.3 HCTLS Protocol Design

This section presents the design of the hidden credentials TLS (HCTLS) protocol. The protocol leverages the TLS handshake protocol shown in Figure 5. It also provides an explanation of the TLS handshake protocol, followed by a proposal to utilize hidden credentials as an authentication method in TLS.

The primary design goals for extending the TLS handshake protocol to support hidden credentials are:

1. Enhance privacy for both client and server during a TLS handshake

2. Maintain backward compatibility so that HCTLS implementations can interact with TLS implementations

To achieve the first goal, it was necessary to modify the TLS handshake protocol to avoid the privacy violations. To accomplish the second goal, careful consideration was given to the modifications of TLS to provide backward compatibility. Sections 3.3.1 and 3.3.2 explain the TLS handshake protocol and the HCTLS modifications to the TLS handshake protocol.

3.3.1 TLS Handshake Protocol

There are several purposes for the TLS handshake protocol. First, the client and server negotiate cryptographic capabilities, as well as establish a session-id. Second, the client and server may authenticate each other. Finally, they establish the cryptographic keys used for the duration of the session to protect messages. The following paragraphs explain each message shown in Figure 5 in detail.

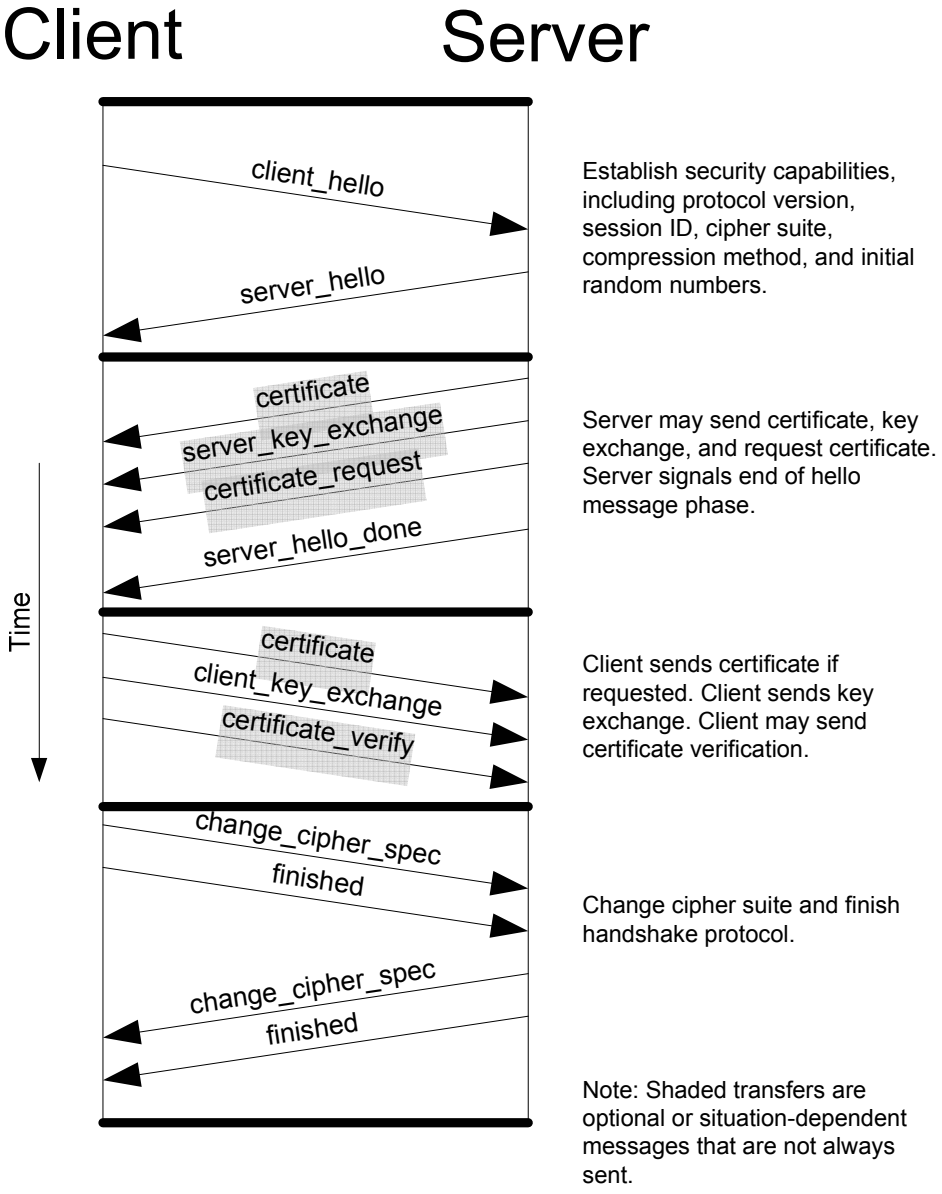


Figure 5 – TLS Handshake Protocol, taken from [28]

The TLS handshake protocol allows the client and server to negotiate cryptographic algorithms for use during the session. First, the client sends a *client_hello* message that includes his TLS version number, a random nonce, a session-id, a list of cipher suites he supports in decreasing order of preference, and a list of compression methods. Each cipher suite consists of the combinations of key exchange methods and cryptographic algorithms the client supports. The server receives the *client_hello*

message and constructs the *server_hello* message. In the *server_hello* message, the server establishes a session-id, generates a nonce, selects the cipher suite to be used during the session, and decides which compression algorithm to use. Thus, the server ultimately decides what cipher suite and compression method will be used and responds with that information in the *server_hello* message.

The client and server may optionally authenticate each other during the TLS handshake protocol. Although there are several different authentication methods available in TLS, a description of ephemeral Diffie-Hellman is given since that is what HCTLS uses. Server authentication begins when the server sends its certificate in the *certificate* message. The server sends the *server_key_exchange* message containing the Diffie-Hellman parameters, establishing the key material for deriving the encryption keys for use later in the TLS session. This message is signed with the server's private key associated with the public key sent in the *certificate* message. If the server requires client authentication, the *certificate_request* message is sent to request a certificate from the client. The server then sends a *server_hello_done* message to indicate completion of the server hello.

The client receives the messages from the server and verifies that the server has sent a certificate from a CA that the client trusts. If the server has requested a certificate for client authentication, the client then sends his certificate to the server in a *certificate* message. Following the optional *certificate* message, the client must send a *client_key_exchange* message that contains the necessary Diffie-Hellman parameters. If the client sent a certificate in the previous *certificate* message, he must include a proof of ownership of the private key associated with that certificate in a *certificate_verify*

message. Finally, the client sends *change_cipher_spec* and *finished* messages to indicate completion of his turn. The *finished* message verifies that the client was able to understand all previous messages and correctly calculated the corresponding encryption keys.

The server calculates the respective keys from the messages the client sends. He verifies the proof of ownership the client sent in the *certificate_verify* message, as well as the *finished* message to make sure both parties calculated the encryption keys correctly. He responds with a *change_cipher_spec* message and a *finished* message to prove to the client his correct derivation of the keys and to indicate completion of the handshake protocol.

3.3.2 HCTLS Modifications to TLS handshake protocol

HCTLS is designed to enhance client and server privacy, while maintaining backward compatibility with the original TLS specification. HCTLS avoids the privacy violations in TLS by eliminating the flow of certificates and using hidden credentials to authenticate the exchanged Diffie-Hellman parameters.

In the HCTLS protocol design, the client and server exchange nyms in the *client_hello* message and the *server_hello* message. They maintain these nyms for the duration of the session. RFC 3546 [2] describes how the client and server hello messages may be modified to include additional extensions not in the hello messages in the original TLS specification [6]. The client may send his nym in an additional extension type field in the *client_hello* message. The server is required to accept the field, even if he does not understand it. If the server does not understand it, the TLS connection may continue, but no messages pursuant to the extensions may be sent. If the server does not support hidden

credentials but the client does, this would not break an existing TLS implementation. However, the client would not be able to use hidden credentials as the authentication method. If the client does not send a nym in the *client_hello* message, the server must not respond with a nym in the *server_hello* message. This allows HCTLS servers to interact with non-HCTLS clients.

If the client wishes to use hidden credentials as the authentication method, he sends the hidden credentials cipher suite as the first suite that he supports for authentication in the initial *client_hello* message. Subsequently, the server chooses a cipher suite from the list the client sent for the connection and responds accordingly. If the server does not support the hidden credentials cipher suite, he will not select it for the TLS session. Consequently, the HCTLS design maintains backward compatibility in the protocol during cipher suite negotiation.

The following is a proposed cipher suite to be added to TLS to support hidden credentials. It is called "TLS_DHE_HC_WITH_AES_SHA1" for TLS with Diffie-Hellman key exchange (DHE), using hidden credentials (HC), with advanced encryption standard (AES) as the symmetric encryption algorithm, and secure hash algorithm 1 (SHA-1) as the hashing function.

3.3.3 Explanation of Diffie-Hellman Key Exchange

In HCTLS, the client (Alice) and server (Bob) perform a Diffie-Hellman key exchange to establish the session key parameters for the TLS session. Diffie-Hellman allows users to establish a session key over an insecure network. The security of Diffie-Hellman depends on the difficulty of computing discrete logarithms. This section

explains the Diffie-Hellman key exchange protocol and later relates it to TLS and HCTLS.

In the Diffie-Hellman key exchange, shown in Figure 6, there are two publicly known values, q and a. To establish a shared secret key, Alice and Bob both generate a random number X_A and X_B, respectively. At this point, Alice computes $Y_A = a^{X_A} \bmod q$. Bob also calculates $Y_B = a^{X_B} \bmod q$ and both sides exchange Y values in the clear. Each side then calculates the secret key K by taking $K = (Y_B)^{X_A} \bmod q$ or $K = (Y_A)^{X_B} \bmod q$, since $K = (Y_B)^{X_A} = (Y_A)^{X_B}$. In this way, both sides can compute the secret shared key without concern that an eavesdropper can recover the key.

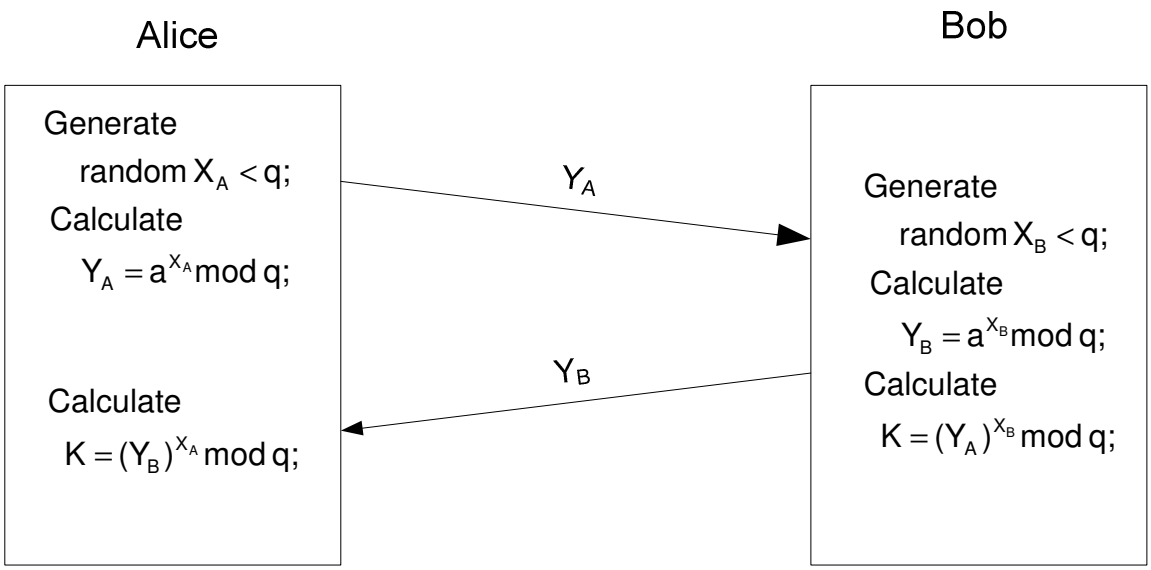


Figure 6 – Diffie-Hellman Key Exchange, adapted from [28]

However, the Diffie-Hellman key exchange is subject to a man-in-the-middle attack. If a malicious user Malfoy is able to intercept and modify messages between Alice and Bob, he can substitute his own values for Y_A and Y_B, essentially establishing an

independent Diffie-Hellman session with both sides of the connection. In this situation, neither Alice nor Bob can detect the attacker.

When a Diffie-Hellman key exchange takes place in the TLS handshake protocol, there are three different authentication methods. First, the parties may agree to engage in anonymous Diffie-Hellman, where no authentication takes place. Second, the parties may engage in fixed Diffie-Hellman, where the values for Y_x are fixed inside a public key certificate that has been signed by a trusted CA. Third, both sides could exchange key values encrypted or just signed using private RSA or DSS keys. This would allow the other side to decrypt the values or verify the signatures using the respective public keys that a trusted third party has signed. Using the second or third methods, the protocol thwarts both eavesdropping and man-in-the-middle attacks.

Relating the Diffie-Hellman key exchange to the TLS handshake protocol, the Diffie-Hellman parameters are sent in the *server_key_exchange* and *client_key_exchange* messages. In a regular TLS handshake protocol, both sides send these values in the clear with no encryption; however, they are signed using either the respective party's private RSA or DSS key. In the HCTLS protocol design, each side encrypts the parameter values using their respective policies protecting TLS session establishment. Thus, each *key_exchange* message is wrapped inside a hidden credentials message. By using hidden credentials, no *certificate* message is required, since no certificates are sent.

HCTLS also supports a more fine-grained access control model. If Alice requests a resource ***C*** from Bob, for which she has not previously satisfied the policy during the TLS session, Bob may request a rehandshake. In the rehandshake, Bob may send Alice new Diffie-Hellman parameters for subsequent use in the TLS session. He encrypts these

according to the policy protecting resource ***C***. If Alice can correctly decrypt the new Diffie-Hellman parameters and continue the TLS session, she may obtain the resource. Alternatively, Bob could simply encrypt resource ***C*** according to its access control policy and send it to Alice. If Alice has the correct credentials, she will be able to decrypt it.

The HCTLS protocol design accomplishes the goals of enhancing privacy in the TLS handshake protocol and maintaining backward compatibility with existing implementations. In the original TLS handshake protocol, either party may be required to disclose a certificate containing sensitive information. By not disclosing credentials or policies, HCTLS affords more privacy to both the client and server. HCTLS maintains backward compatibility by using the extension type field in the client and server hello messages, and only allowing the use of the hidden credentials cipher suite when both sides support it. Thus, HCTLS implementations should not break existing TLS implementations.

3.4 HCEmail Protocol Design

This section presents the design of the hidden credentials email (HCEmail) integration. Email is typically sent using the simple mail transfer protocol (SMTP) [23] and retrieved by a user's mail client using either post office protocol version 3 (POP3) [21] or internet message access protocol (IMAP) [5]. Figure 7 depicts a high-level example of HCEmail. In this figure, Bob sends an email to alice@fbi.gov, which Alice will only be able to decrypt if she can satisfy policy P_B. Alice may then send a response to Bob that he will only be able to decrypt if he can satisfy policy P_A.

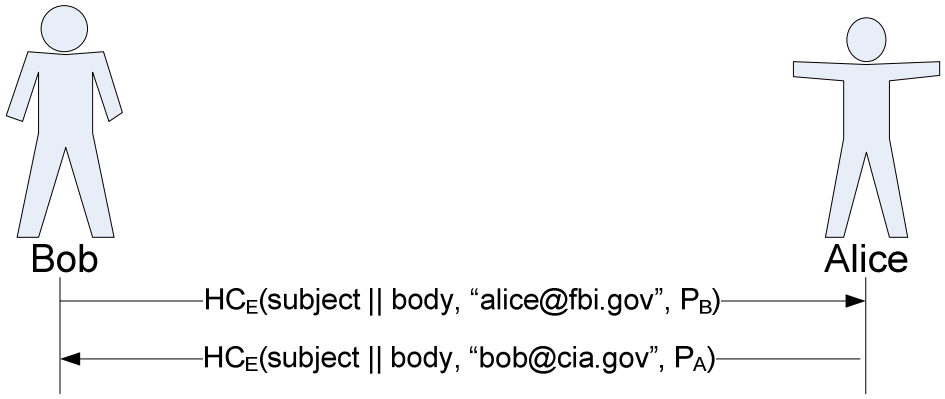


Figure 7 – Email Exchange Using Hidden Credentials

The following are the design goals for HCEmail:

1. Enhance privacy and provide security to insecure email protocols.
2. Maintain compatibility with existing email applications and email transfer protocols.

Alice, desiring to send a message to Bob, derives the public key associated with Bob's nym, bob@cia.gov. Alice composes the plaintext email message. She creates the internal portion of the hidden credentials message and encrypts it using a randomly generated symmetric key. She then encrypts the key according to the policy protecting the information in the email. After wrapping the resulting ciphertext in the hidden credentials external format, she sends it to Bob. Bob may only decrypt the message if he possesses the required attribute credentials.

HCEmail enhances the privacy and security of email by allowing users to secure their email using encryption and protect the decryption policy. A person's public key may be easily derived by knowing their email address and their CA's public key. This is particularly useful in the context of hidden credentials and trust negotiation, where interactions typically take place between strangers with no prior relationship. No

communication must occur before sending a message to an individual, other than to obtain that individual's email address.

HCEmail maintains compatibility with existing mail servers and mail transfer agents by utilizing an email proxy server to handle message formatting. The hidden credentials message is sent in the body of the email with simple delimiters specifying the start and end of the hidden credentials message, as shown in Figure 8.

```
From: Evan Child [childe@cs.byu.edu]
Sent: Tuesday, June 22, 2004 10:33 AM
To: childe@cs.byu.edu
Subject: Hidden Credentials Message

---------- Begin Hidden Credentials Message ----------

<?xml version="1.0" encoding="utf-8"?>
<hc_external>
        <version>1.0</version>
        <fulfillment_paths>
                <fulfillment_path>
                        <ciphertext
encoding="base64">as;kldfj;dlkfjasdlfk</ciphertext>
                </fulfillment_path>
                <fulfillment_path>
                        <ciphertext
encoding="base64">as;kldfj;dlkfjasdlfk</ciphertext>
                </fulfillment_path>
        </fulfillment_paths>
        <content encoding="base64"
encryption-algorithm="rc4">ak/.uqoi;ejksralmcz.,xbn.,mzD;aosijeu09821304986u
234o</content>
</hc_external>

---------- End Hidden Credentials Message ----------
```

Figure 8 – Example of an HCEmail message

Because email is independent of the transport protocol, email bodies may include any transmittable data. To maintain compatibility with the majority of email servers on the Internet today, transmittable is defined as 7-bit ASCII character encoding. Hidden credentials utilize XML; therefore, they require no special transport protocol for transmission, since XML may be represented in 7-bit ASCII. A hidden credentials message may be transmitted independent of whether the sender and receiver utilize SMTP, POP3, IMAP, or other mail transfer protocols.

HCEmail accomplishes the goals of enhancing privacy and security and maintaining backward compatibility with existing email protocols. By using hidden credentials, HCEmail affords the user additional privacy by not disclosing credentials. Users may also authenticate one another. By utilizing XML and a simple embedded message format, HCEmail also maintains backward compatibility with existing email protocols and applications. The HCEmail design represents the first time trust negotiation has been implemented securely in email.

Chapter 4 System Implementation

This chapter discusses the HCEmail and HCTLS implementations. Both of these systems rely on the message format presented in Chapter 3. In addition to HCEmail and HCTLS, the existing implementation of hidden credentials was extended to support the new message format.

The hidden credentials message format was designed to be useful across multiple protocols and usage models without any modification. HCEmail and HCTLS represent the three usage models presented in Chapter 3. HCEmail can be used for the first two models, namely one-way message and single round message exchange. HCTLS represents the session-based model, where session establishment is predicated upon successful authentication. By showing integration into both protocols, the message format is shown to be effective regardless of the protocol or usage model.

The HCTLS and HCEmail prototypes do not implement the authentication tokens described earlier in this thesis. Researchers in the ISRL are currently investigating IBE signature techniques. Existing signature algorithms, such as RSA, can also be used to implement the authentication token.

4.1 Hidden Credentials Extensions

The message format has been integrated into the existing hidden credentials implementation. Robert Bradshaw of the ISRL created a prototype hidden credentials system, as detailed in [14]. For this thesis, Bradshaw's implementation was extended to support the new message format, as well as XML policies. Because Bradshaw implemented hidden credentials in a library format, any application can utilize the library.

This allows applications to benefit from the library, regardless of usage model or protocol.

4.2 HCTLS Integration with PureTLS

The HCTLS implementation leverages PureTLS (see http://www.rtfm.com/puretls), a java-based implementation of the TLS protocol. The HCTLS implementation extends the handshake protocol of PureTLS to support using hidden credentials as an authentication method for executing a Diffie-Hellman key exchange.

HCTLS clients may contact non-HCTLS servers without breaking them. Similarly, HCTLS servers can accept connections from non-HCTLS clients and establish a normal TLS session. Thus, HCTLS supports compatibility with non-HCTLS applications.

4.3 HCEmail Integration with JAMES

The hidden credentials email implementation utilizes the Java Apache Mail Enterprise Server (JAMES), a 100% pure Java SMTP and POP3 email server, as an email proxy. In addition to being a traditional mail server, JAMES is an email application platform. Email is processed according to special applications called *mailets*. The HCEmail implementation integrates hidden credentials encryption and decryption into a *mailet*. As email arrives at the mail server, either inbound or outbound, it is processed according to the specified policy protecting it. All inbound email messages are checked to see if they are hidden credentials messages by looking for the email message format specified in Section 3.4. Each user that wishes to use hidden credentials will run a JAMES server with the hidden credentials *mailet*.

Users must reconfigure their email clients to point to JAMES, instead of their email server, e.g., mail.cs.byu.edu, to use HCEmail. They must also input their login information for their regular email server into JAMES to permit the downloading of their email. In this way, HCEmail allows users to utilize hidden credentials with no functional changes to their email clients or servers.

Figure 9 illustrates the HCEmail integration. In this figure, Alice and Bob have a mail client and a JAMES server, acting as an email proxy, running on their local computers. Email 1 shows the inbound message from Alice to Bob as Alice has encrypted it and before Bob has decrypted it. Email 2 shows the inbound message from Bob to Alice after Alice has decrypted the message.

Hess et al. [13] integrated traditional trust negotiation into email as a proof of concept implementation. However, the implementation was not secure or private since it did not address credential ownership verification and it exchanged credentials in the clear.

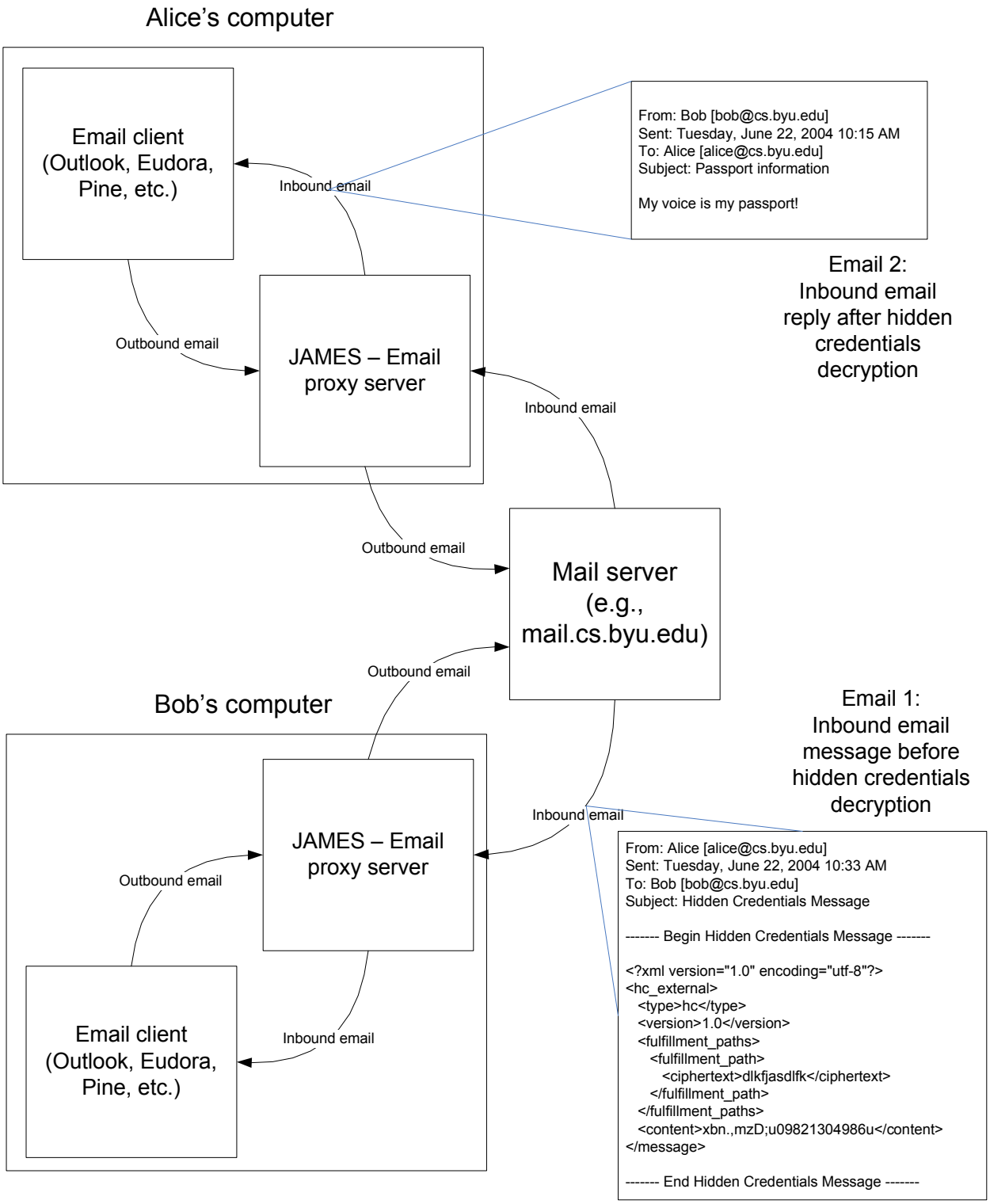


Figure 9 – Architecture of HCEmail Implementation

Chapter 5 Threat Modeling

Disciplines such as mathematics and physics utilize "proofs" to establish the validity of their results. Security and privacy are much more subjective. No formal process exists for proving that a protocol is secure. Although many parts of the mathematics underlying encryption algorithms are known to be extremely difficult to break, vulnerabilities can exist elsewhere. In [27], Schneier describes threat modeling as "…a way to start making sense of the vulnerability landscape." By analyzing and modeling various threats, the security of a protocol can be estimated and the risk associated with deploying it better understood.

This chapter discusses different types of attacks to illuminate the strengths and vulnerabilities of hidden credentials. The attacks detailed in this chapter are the most probable types of attacks that would typically be executable in a network environment. They are not a comprehensive examination of all types of attacks; such a list would be extremely long and contain many scenarios that would not be practical in this context.

Sections 5.1-5.3 accomplish three items:

1. Analyze the hidden credentials message format with regard to the various types of attacks mentioned below.
2. Analyze the hidden credentials message format integration into TLS.
3. Analyze the hidden credentials message format integration into email.

5.1 Hidden Credentials Message Format

5.1.1 Replay Attack

A replay attack occurs when an eavesdropper listens to a message and replays or resends it later. For example, a malicious person may capture a funds transfer message in

transit and replay it later. A hidden credentials message has three fields that help prevent this particular type of attack: the `timestamp` field, the `message-id` field, and the `receiving-nym` field.

The `timestamp` field assures timely delivery of a message. The `timestamp` field contains the time at which Alice, the client, initially creates the message. If Bob, the server, receives the message outside of the required time window, he rejects the message.

The `message-id` field is an identifier for each message. This field need not be unique for all messages within a session, only unique within the session receive window. Bob must save each `message-id` for at least as long as the receive window size for a message. After receiving each message, Bob must verify that the `message-id` in the message has not been previously received in the current receive window. If the `message-id` is a duplicate, he must reject the message.

The `receiving-nym` field contains the nym of the intended recipient. Bob verifies that he is the intended recipient of each message he receives. If a different value is in the `receiving-nym` field, Bob should reject the message. Using the `receiving-nym` field, Bob's ability to impersonate Alice, by replaying the message to different server, is limited.

5.1.2 Reflection Attack

A reflection attack occurs when an attacker replays a signed message received from a client or server in order to impersonate the client or server to another individual. Figure 10 illustrates one way to accomplish this attack. Malfoy convinces Alice to sign what Bob wanted Malfoy to sign. In this way, Malfoy can fool Bob into believing that he is Alice. A hidden credentials message is not subject to such an attack because there is no

notion of a challenge/response message to prove ownership of the respective keys. With hidden credentials, Alice encrypts a message according to the appropriate policy, and Bob or Malfoy can only decrypt it if they possess the correct credentials.

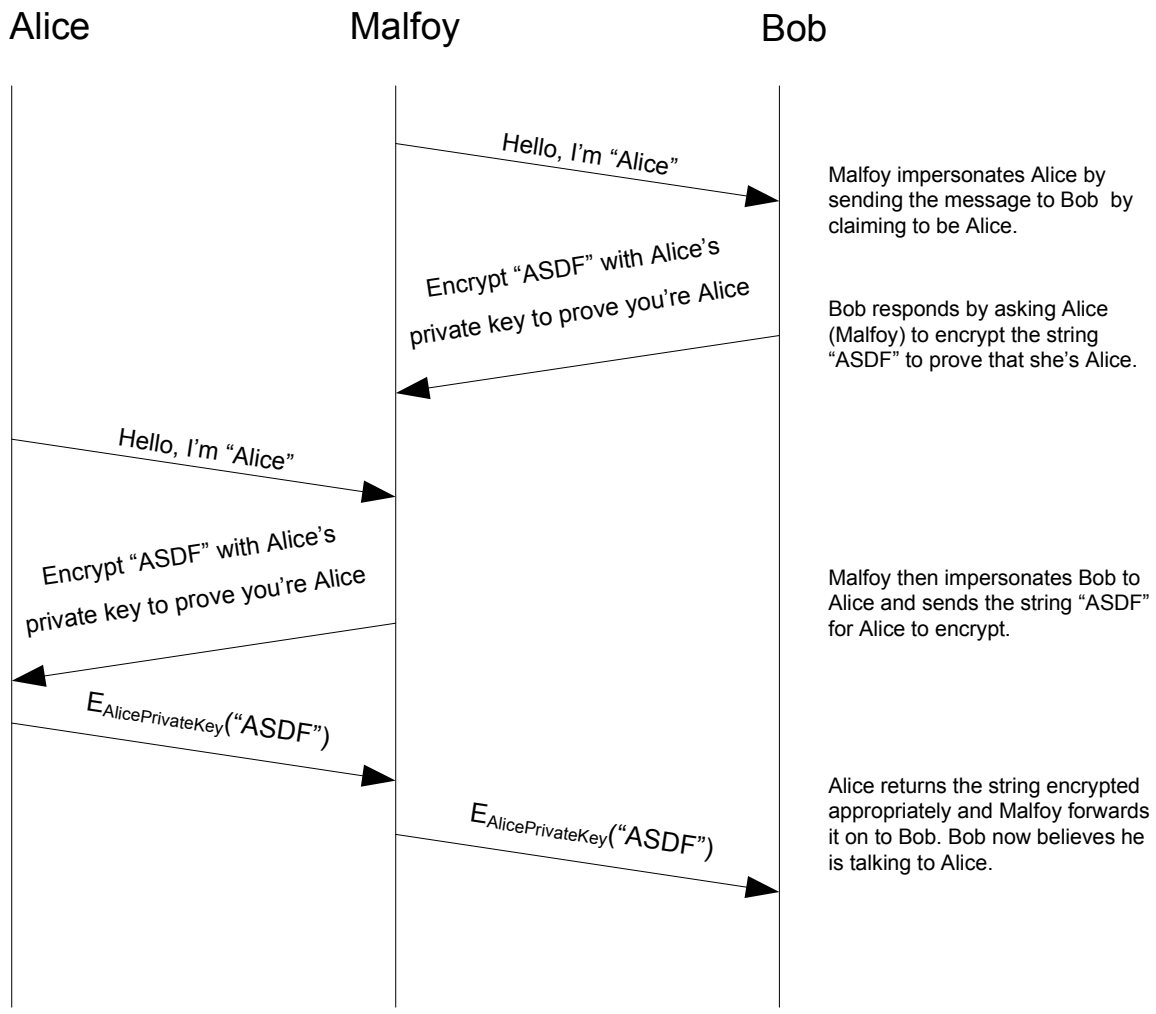


Figure 10 – Example of a Reflection Attack

5.1.3 Cryptanalysis of Ciphertext

Cryptanalysis of ciphertext is a hard task. It generally requires an advanced understanding of encryption algorithms, access to large amounts of computing power,

and luck. Hidden credentials currently use the advanced encryption standard (AES) for encrypting a message, typically using a 128-bit AES key. They utilize elliptic curve cryptography (ECC) to encrypt the AES key. The security community, assuming correct usage and sufficient key sizes, considers both AES and ECC secure against all cryptanalytic attacks. Possible attacks include chosen-ciphertext and differential cryptanalysis.

5.1.4 Compromise of Private Key

Each credential has an associated private key. A compromised private key means that somehow somebody other than the owner obtained the private key. In this situation, the malicious user can read all past and future communications encrypted with the key and can impersonate the owner in transactions with other servers.

A compromised private key is a serious issue. However, a user can mitigate the effects of a compromised private key in several ways. First, the user can place the associated public key on a certificate revocation list (CRL), assuming he knows that it has been compromised. Applications that use public keys signed by a particular CA should periodically check the revocation status of a presented public key. Second, private keys can have a built-in revocation date. For example, instead of using the key based on the email address “software_engineerbob@company.com”, a user could use the key derived from “software_engineerbob@company.com-2004”. In this way, the key is only good for a single year. An even more paranoid approach would use the key “software_engineerbob@company.com-05-31-2004” for a key that is only valid for a single day. Both of these techniques can significantly reduce the effects of a compromised private key.

Numerous ways exist to compromise a private key file. Private key files exist in various locations such as in a file on a hard drive, on a smart card, or on a USB key. Each location has its advantages and drawbacks in terms of security and accessibility.

A hard drive is usually a general-purpose, large storage device in a computer and is the most accessible storage medium to store private keys. The private key is typically stored encrypted, which makes it difficult to retrieve it, even if a malicious individual obtains the private key file. However, if it is encrypted, the user must remember the decryption passphrase when it is used, thus decreasing usability. In addition, poor passphrase selection for the encrypted file can reduce the effectiveness of the encryption, since the key would vulnerable to an off-line dictionary attack.

A smart card is a tamper resistant device, typically the size of a credit card, designed to store private keys and execute signing and encryption algorithms. A smart card is relatively secure, since the private key does not leave the card and all encryption operations occur locally on the card. In addition, if smart cards are lost, the private keys are difficult to obtain because of their tamper-resistant nature. Unless a passphrase protects the smart card, the attacker may impersonate the actual owner of the card. Each computer where the private key is used must have some type of smart card reader, potentially inconveniencing a user if no reader is available. In addition, to increase security, the smart card may be protected with a passphrase, which can inhibit usability and reduce security when poor pass phrases are selected.

USB keys are like portable hard drives and have no built-in tamper-resistance to protect the private key. Typically, the owner downloads the key to the computer because

USB keys have no processor to execute encryption and signing algorithms. The risk of a compromised key increases when the key is temporarily stored on the computer.

5.1.5 CA Key Attack

There are two different ways to execute a CA key attack. The first, and most obvious, is to compromise the CA private key used to issue credentials. With HC, the CA issues credentials using its private key. The compromise of a CA's private key permits a malicious user to issue all keys and read all communications from everyone that relies on that CA.

The second type of attack is to keep a user from obtaining the correct CA public key. In order to encrypt a message according to a particular policy with hidden credentials, the client must be able to derive all keys associated with the necessary attribute credentials. To derive the keys, the client must have the public key of the certifying CA. If a malicious user can substitute his own CA public key for a legitimate CA public key, he could then decrypt any message from the users that encrypt against his CA public key. These attacks are difficult to thwart, and are inherent in all public key infrastructure (PKI) based systems. The scenarios in this thesis assume secure CA public key distribution.

5.1.6 Malicious CA

A malicious CA may impersonate any users that rely on it by generating the private keys issued to those users. The future work section of Chapter 7 details the measures that a user can take to mitigate the effects of a malicious CA. Because of their reliance on IBE, hidden credentials are currently vulnerable to this type of attack.

5.1.7 Compromise of Session Key

The session key is the temporary key randomly generated for use only during the current session. To compromise a session key, an attacker may perform several kinds of attacks, depending on the encryption algorithm. Different types of attacks on encryption algorithms include side-channel, brute force key attacks, power consumption cryptanalysis, acoustic cryptanalysis, and statistical cryptanalysis. These attacks require extraordinary circumstances to succeed, such as physical access to the computer or special hardware. Each of these attacks is difficult to execute.

In addition, the compromise of a session key is of relatively limited value to an attacker, since the key's life span is short. Consequently, the risk from this type of attack is typically low.

5.1.8 Insider Attack, Same Credentials as Client

The following example details an insider attack from someone who possesses credentials asserting the same attributes as the client. Bob is a CIA agent. Frank is a CIA group manager, and Bob's superior. Malfoy is a CIA agent, and an undercover Russian spy. Malfoy and Bob both have the same CIA credentials. Thus, Malfoy is an insider; someone already trusted by the CIA's CA. Bob and Frank do a nym exchange in which Malfoy, as a man-in-the-middle, substitutes his own nym for Bob's nym. Bob then sends an encrypted message to Frank. Malfoy cannot read the message that Bob sends to Frank. However, since Frank encrypts the reply with Malfoy's nym instead of Bob's, Malfoy can read the reply from Frank.

The hidden credentials message format proposed in this thesis thwarts this attack by utilizing a sending-nym field in the internal portion of the message. When Frank decrypts the message, he can check that the nym contained in the sending-nym field

matches the nym he received in the nym exchange. If the nyms do not match, Frank does not send a reply, (and begins to investigate who might be doing this…).

5.1.9 Insider Attack, Same Credentials as Server

The following example details an insider attack from someone who possesses credentials asserting the same attributes as the server. Bob is a CIA agent. Frank is a CIA group manager, and Bob's superior. Malfoy is also a CIA group manager, but not Bob's superior, and an undercover Russian spy. Malfoy and Frank both have the same CIA credentials. Thus, Malfoy is an insider; someone already trusted by the CIA's CA. Bob and Frank do a nym exchange in which Malfoy, as a man-in-the-middle, substitutes his own nym for Frank's nym. Bob then sends an encrypted message to Frank. Malfoy can read the message that Bob sends to Frank, since he possesses the same credentials as Frank, and the message is encrypted against his nym instead of Frank's nym. In fact, Frank may never know that a message was intended for him, as Malfoy can pretend to be Frank. This is a type of spoofing attack. Several varieties of spoofing attacks are problems with hidden credentials, traditional trust negotiation, and TLS.

5.1.10 Message Integrity Attack

In cryptographic messages, it is common to include a message authentication code (MAC) to check the integrity of the message when it arrives at the destination. However, with a hidden credentials message, this is not necessary. If an attacker were to change any part of the ciphertext contained in the message, it would not decrypt properly. Since the internal portion of the message is well formatted, it is easy for the recipient to detect if the message has been decrypted properly or not. However, the recipient may not know if it was modified in transit, or if he just did not have the correct credentials to decrypt it.

An attacker could execute a type of denial of service (DOS) attack by simply modifying the message as it passed by. This is an area of future research, detailed in Chapter 7.

5.1.11 Denial of Service Attack

A malicious user executing a denial of service (DOS) attack prevents legitimate users from accessing a resource or service. With hidden credentials, it is possible to flood the server with many requests for service. These requests could be randomly generated and have no legitimate decryption. However, the server must try each message to see if it is valid, causing a heavy CPU load on the server. Because of this heavy load, valid requests for service may be delayed or rejected.

As mentioned in section 5.1.10, it is possible to modify a message in transit so that it is no longer valid. By doing this, a malicious user prevents the server from decrypting the message properly. Hidden credentials, like most other protocols, are vulnerable to DOS attacks.

5.1.12 Social Engineering Attack

A social engineering attack is a malicious person attempting to gain access to a secret, such as a password or private key file, by convincing a legitimate owner of the password or file to release it to them. For example, someone may call an employee at a large corporation pretending to be a member of the technical support help desk. The attack perpetrator will say that there has been a problem with the user's account and that if the user will simply give the "support desk" his password, the "support desk" will be able to rectify the problem. This is a difficult attack to thwart and is very common today. The best way to prevent social engineering attacks is to educate users. Like most security protocols and practices, hidden credentials are vulnerable to this type of attack.

5.2 HCTLS

HCTLS is impervious to the same threats as hidden credentials. This section argues that the security of HCTLS is comparable to that of TLS, with increased privacy for the user.

TLS supports various key exchange methods. Since HCTLS uses the Diffie-Hellman key exchange, this thesis will analyze and compare the modified version to the original version in the TLS protocol. In the typical Diffie-Hellman key exchange, the client and server values Y_c and Y_s do not need to be encrypted. Although to prevent a man-in-the-middle attack, the values need to be signed with an RSA or DSS key. A public key certificate is sent in the *certificate* message for the client or the server to use in verifying the signature. As shown in section 3.3, an eavesdropper cannot discover the secret key from these values. In the HCTLS implementation, these values are encrypted using hidden credentials instead of signed using RSA or DSS. The purpose of the encryption is to authenticate the other party in the connection. Each party must decrypt the encrypted Y_x message properly and calculate the key. If the connection succeeds, both sides know that the other satisfied the policy successfully. Thus, the HCTLS integration increases the authentication capabilities of the TLS handshake protocol by using hidden credentials to authenticate the receiver of the Y_x parameter values.

In addition to enhancing the authentication capabilities of the TLS handshake protocol, HCTLS affords the user greater privacy compared to TLS. For example, in either the RSA key exchange method, or the original Diffie-Hellman key exchange method in the TLS handshake protocol, the server discloses a public key certificate. The server may optionally require a certificate from the client. The HCTLS protocol avoids

this potential privacy violation since it uses hidden credentials, which require no transmission of policies or certificates. By eliminating the need to send a public key certificate, the HCTLS protocol affords the user greater privacy.

5.3 HCEmail

HCEmail maintains the security and privacy properties mentioned in the previous section. There are no changes to the protocols underlying email, including SMTP, POP3, and IMAP. HCEmail uses the email body to embed the hidden credentials message so that it can be transmittable over regular email. Because HCEmail only uses the transport protocols for email, its security and privacy are dependent on the hidden credentials message format alone. The previous sections in this chapter have shown that the hidden credentials message format is secure and private against most types of attacks. Therefore, HCEmail gains the same powerful security and privacy safeguards as the hidden credentials message format.

HCEmail, like HCTLS, may be vulnerable to certain spoofing attacks as described in section 5.1.9. Thus, parties must use caution when exchanging nyms. One effective way to combat this threat is to have some external method for verifying that a particular nym belongs to the person with whom the party wishes to interact.

However, email and TLS also have inherent advantages that help avoid the attacks described in section 5.1.9. When Alice wants to send an email to Bob, she must typically know Bob's email address beforehand. Thus, deriving the public key is trivial, since Alice can usually verify visually that bob@bobscompany.com is Bob's email address. Similarly, the nym exchange is inherent to HCTLS since Alice knows that the nym for bob.com must be "bob.com" before she even connects.

Chapter 6 Related Work

Hidden credentials protect privacy during online negotiations. HCTLS protects the disclosure of certificates during the TLS handshake protocol. Several other protocols that exhibit similar properties to HCTLS are TNT [12], SPSL [22], and secret handshakes [1]. These protocols make changes to TLS to achieve greater privacy than those inherent in TLS. An in depth analysis of each of these protocols and a comparison can help highlight similarities and elucidate the contributions of HCTLS. This chapter explains each protocol, compares it to HCTLS, and also discusses related work in secure email.

6.1 Trust Negotiation in TLS

Trust negotiation in TLS (TNT) is a protocol created by researchers from the ISRL to add the ability to perform traditional trust negotiation within the TLS protocol. Through modification of the rehandshake protocol within TLS, this protocol provides the ability to perform trust negotiation over a secure connection. TNT exchanges policies and credentials over a secure channel to provide attribute authentication for each party.

TNT makes several changes to the TLS rehandshake protocol to be able to perform traditional trust negotiation in TLS. TNT assumes that a secure session has already been established when a client or server issues a rehandshake request. Thus, all messages involved in the trust negotiation are encrypted under the current TLS session. The TLS specification states that either side may issue a rehandshake request at any time.

In traditional trust negotiation, the client and server use strategies for determining what credentials they will disclose. The client and server must agree upon a strategy family to assure interoperable credential disclosure strategies. The *server_hello* and

client_hello messages have a TrustNegotiationStrategyFamily field added to them for allowing the client and server to agree upon what strategy family to use.

The resource request is submitted over a secure connection and is not accessible to eavesdroppers. TNT adds a *policy* message for sending a policy that the other party must fulfill to gain access to the requested resource. Policies contain hints about what credentials the opposing party must use to gain access to the resource. Both sides exchange policies, credentials, and credential verification messages in typical trust negotiation fashion until the policy guarding the requested resource is satisfied and the resource can be disclosed. The TNT system design integrates the changes to the rehandshake protocol so that the implementation can maintain backward compatibility with TLS.

There are several important differences between HCTLS and TNT. Since TNT only performs trust negotiation in the TLS rehandshake, the standard TLS handshake protocol must have previously executed and succeeded. The standard TLS handshake protocol exchanges certificates in the clear to establish a secure session. This exchange of certificates in the clear can be a violation of privacy, if the credentials are sensitive. Hidden credentials avoid these privacy violations by not sending any certificates.

Although either side of a TNT connection may issue a rehandshake request at any time, one side must send a policy first, after requesting a rehandshake, to initiate the negotiation. This policy may contain sensitive information. Hidden credentials also avoid this potential privacy violation by not sending any policies, but instead encrypt the request according to the policy. In this way, if the receiving party cannot decrypt the message, they learn nothing about the policy.

Traditional trust negotiation in TNT requires proof of policy fulfillment by sending the necessary attribute credentials and signatures. The hidden credentials protocol inside of HCTLS does not require any credential disclosures. Instead, the key material necessary for establishing the session is encrypted according to the policy protecting it, and sent to the other party. In this way, no credential disclosures occur at any point in the negotiation.

6.2 Secure and Private Socket Layer

Secure and private socket layer protocol (SPSL) identifies potential violations of privacy through public key certificate disclosure during the TLS handshake protocol. It also allows the client to prevent the server from tracking what documents the client accesses during the TLS session. It proposes countermeasures to combat the violations of privacy by introducing *crypto certificates*. Crypto certificates, combined with the SPSL protocol, hide the identity of the user while still allowing the resource provider to authenticate the necessary attributes. This work focuses primarily on transactions that occur over TLS in which a client requests a protected service or resource.

SPSL targets a slightly different usage model than hidden credentials. SPSL focuses on anonymous group membership authentication. Hidden credentials allow parties in disparate organizations to authenticate and authorize each other. Both protocols recognize the privacy violations that occur in the TLS handshake protocol and attempt to solve the problem by changing or eliminating the plaintext certificate messages. SPSL has the advantage of disallowing the server from knowing which user is currently accessing its database. Hidden credentials never need to reveal the credentials used for authentication.

SPSL extends the TLS protocol to provide untraceability and privacy to clients in a TLS session. As noted previously in the thesis, the TLS handshake protocol transmits public key certificates in the clear. This represents a potential violation of the privacy for the information contained in these certificates. It also allows the server to track the usage patterns and habits of individual users. SPSL attempts to solve these problems with crypto certificates and anonymous authentication. The SPSL paper proposes crypto certificates, a new type of certificate that allows selective disclosure of attributes in X.509v3 certificates.

The authors of the SPSL protocol use the following example to illustrate their usage model: “Users pay a monthly fee to access data from a medical publication database. The database contains links to information regarding various diseases. The database manager implements access control using X.509v3 certificates and SSL/TLS.” In this way, the database manager can grant access to those who have paid the monthly fee. However, this also allows the database manager to track what documents each individual user is accessing. “This information is not essential to the role of the service manager, which consists in making sure that only qualified users access the database” [22].

SPSL defines a mask to consist of a subset of the public key certificates of authorized users for a particular resource. From the preceding example, the mask would be a subset of the users who had paid for the monthly service. Either the client or server may select the mask. A unique mask identifier may be associated with each mask to allow a client and server to agree upon a mask without transmitting the entire mask for each interaction.

SPSL defines four new cipher suites for TLS for varying degrees of privacy during the TLS handshake protocol. SPSL maintains the `NULL` suite for backward compatibility with the standard TLS handshake protocol. In the `WEAK` cipher suite, SPSL clients must send their public key certificate encrypted with the server's public key. This provides minimal privacy and intentionally does not achieve the paper's goal of untraceability, since the server can track all resources accessed on that connection and associate them with the public key used during TLS session establishment. In the `MEDIUM` cipher suite, SPSL proposes the use of an anonymous authentication protocol. This protocol allows either the client or the server to select the subset of certificates, (the mask), and lets the client prove ownership of the private key associated with one of the public keys in the mask. However, the proof does not reveal which public/private key pair was used. The `STRONG` cipher suite allows the client to present the mask encrypted with the server's public key, thus preventing eavesdroppers from knowing the mask.

Although this protocol prevents the ability of the server to detect which public/private key pair was used to authenticate, a malicious server may still discover this information. Suppose a server publishes a list containing fake certificates to force a client to contact the server through some external means to rectify the omission of their public key from the list. The server then gives the client a new list of certificates with bogus certificates and only a single valid certificate – that of the user who contacted them about the omission. When the client can prove ownership of a key in this list, the server knows who is accessing the documents for the duration of the TLS session.

SPSL requires a large amount of bandwidth. For each resource that a client requests at the application level, the server requests a rehandshake in the TLS session.

The server must then transfer the public key certificates associated with the requested resource to the client, which may be a large database of users. The client must then select a subset of these certificates to return to the server as the mask and prove ownership of a private key associated with one of the public keys in the mask. For greater privacy and less traceability, the client may select a greater number of public key certificates in the subset he sends to the server.

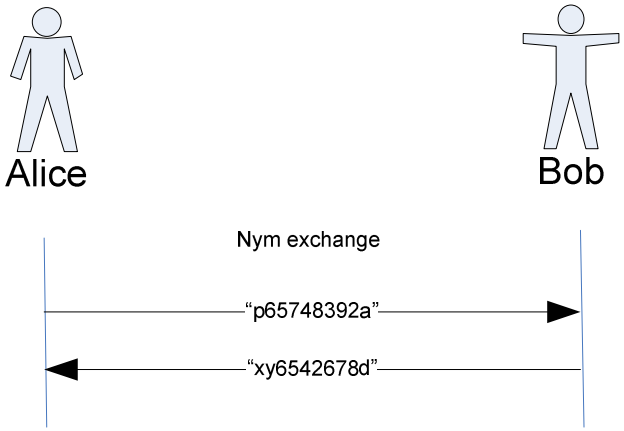
For performance advantages, the server may publish the masks beforehand for users to cache. However, this may violate the server's privacy by requiring that the server reveal what resources are available on its site.

6.3 Secret Handshakes

Secret handshakes allow individuals in the same group to authenticate each other without anyone else knowing that they are performing an authentication. In addition, if a group member attempts to authenticate to a non-member, the non-member will not understand the attempt. In this way, group members can secretly authenticate each other and no one outside of the group can authenticate anyone inside. The authors define what a secret handshake is and show how their implementation achieves these goals. Secret handshakes also provide a solution to the cyclic policy dependency problem.

The authors utilize various paring-based cryptographic schemes to perform secret handshakes. Two parties wishing to authenticate each other exchange pseudonyms. Using pairing agreements, each side calculates a session key. This session key will only be accessible to both if each side has the necessary attributes. For example, suppose police officer Bob pulls Alice over. Alice wishes to authenticate Bob to ensure that he is a police officer. Bob wishes to verify that Alice possesses a valid driver's license

credential. Figure 11 shows how both sides exchange pseudonyms and each side calculates the session key using a hash (H_1) of the pseudonym concatenated with an attribute. Alice calculates the key from "xy6542678d" || "-cop" using her private key T_A to authenticate Bob as a police officer. Similarly, Bob calculates the key from "p65748392a" || "-driver" and his private key T_B. The bilinear properties of $\hat{e}$ allow these two calculations to be equal. If Bob and Alice can communicate using the shared keys K_A and K_B, they have successfully authenticated the respective attributes.



$$K_A = \hat{e}(H_1("xy6542678d - cop"), T_A) \qquad K_B = \hat{e}(T_B, H_1("p65748392a - driver"))$$

Figure 11 – Example of Secret Handshake Protocol

Secret handshakes are integrated into the TLS handshake protocol to provide authentication between strangers in the same group. This section will describe the properties of the authentication without delving into the mathematical details. The secret handshakes authentication method requires modification of two TLS handshake messages: the *server_key_exchange* and the *client_key_exchange*. The authentication method allows authentication between strangers and the exchange of pre-master-secret key material. This key material is later used to calculate the master secret from which all necessary keys are derived for the TLS protocol.

It is interesting to note that secret handshakes and hidden credentials attempt to solve similar, but different problems. Secret handshakes facilitate group members attempting to authenticate each other so that non-group members cannot discover that an authentication is taking place. Hidden credentials allow inter and intra-group authentication without trying to mask the fact that an authentication is taking place. They protect disclosure of any policies or credentials during the authentication process. Hidden credentials use a trust negotiation paradigm to authenticate strangers in open systems, whereas secret handshakes deal with authenticating strangers in closed systems.

One key distinction between secret handshakes and hidden credentials is that secret handshakes do not use complex policies to authenticate strangers. Authentication is only based on group membership, authorized by a single CA. Hidden credentials utilize attribute-based policies to be able to denote complex Boolean expressions.

6.4 PGP, S/MIME, SecureMail

The most common ways to send secure email today include PGP [15] and S/MIME. These protocols rely on public key certificates that do not provide any type of inherent privacy protection and do not prohibit, but rather encourage, the disclosure of their credentials. Trust negotiation assumes that these credentials may be extremely sensitive and hidden credentials take the notion of sensitive credentials one step further by allowing proof of ownership of a credential without actually disclosing it. In addition, PGP and S/MIME do not take into account the notion of a complex policy protecting either a resource on the server, or a credential on the client. However, PGP and S/MIME are common on the Internet today.

Voltage SecureMail (http://www.voltage.com/products/securemail.htm) provides the ability for users to utilize identity-based encryption to secure email. Voltage SecureMail is an industrial implementation of IBE for email developed by Voltage Security. Voltage SecureMail utilizes a simple message format. Based on examples on their website, it appears that the application simply encrypts the message using the public key derived from the respective email address. It then includes the encrypted ciphertext in the body of the email. One advantage of Voltage SecureMail is that it is very simple to understand and relatively easy to integrate into an existing email infrastructure. Hidden credentials, on the other hand, are more complicated. However, despite their higher complexity level, they also provide additional flexibility through complex policy encryption for email messages.

Chapter 7 Conclusions and Future Work

To reiterate, the primary goals of this thesis are to enhance user privacy by:

1. Creating a generic hidden credentials message format for use in a wide range of protocols
2. Protecting certificates during TLS session establishment using hidden credentials
3. Providing a method to do trust negotiation securely over insecure protocols such as email (SMTP, POP3, IMAP), HTTP, and FTP using hidden credentials

These goals were accomplished according to the following three steps: First, a hidden credentials message format was proposed that prevents numerous types of attacks on hidden credentials message exchanges. Second, a secure way to integrate hidden credentials with the TLS protocol was proposed to protect sensitive credentials during session establishment. Finally, a way to integrate hidden credentials with email to provide authentication and privacy was shown.

7.1 Contributions

The novel contributions of the thesis are:

1. Establishment of a standard message format for hidden credentials
2. Implementation of HCTLS
3. Implementation of HCEmail
4. Addition of an authentication-token field to provide proactive authentication
5. Addition of sending-nym and receiving-nym fields to prevent man-in-the-middle attacks during nym exchange

6. Detailed threat model for hidden credentials

Standard Message Format

Establishing a message format for hidden credentials facilitates the adoption of hidden credentials. Establishing this format helps hidden credentials implementers by providing a standard to develop interoperable software. Because the format was designed independently from a specific protocol, all protocols that use it benefit from its security and privacy properties.

Implementation of HCTLS

This thesis proposes a design for integrating hidden credentials into the TLS handshake protocol. This is beneficial for several reasons. First, it provides a prototype implementation of the hidden credentials message format in an existing security protocol. Second, it provides additional privacy properties beyond what the current TLS handshake protocol provides. Third, it affords enhanced privacy when compared to other TLS-extending protocols such as SPSL, TNT, and secret handshakes.

Implementation of HCEmail

The integration of hidden credentials into email is the first secure implementation of email-based trust negotiation. Email is a particularly useful protocol in which to integrate hidden credentials since it is inherently insecure. Additionally, email allows for simple public key distribution by using email addresses as nyms, from which email senders may derive public keys. The integration of the authentication token in the hidden credentials message format supports mutual authentication following a one-way email.

Authentication-token field

In the message format, the authentication-token field was created to allow for proactive authentication. In the original hidden credentials protocol [14], the only way for the server to authenticate a client was to send a reply encrypted according to the server's policy. The client authenticated himself by correctly decrypting the message. For a one-way message exchange, client authentication was not possible since no response is returned to the client. If the client knows the policy protecting the resource a priori, he can send proof of his ability to satisfy the policy, encrypted inside the original message.

Sending-nym and receiving-nym fields

The sending-nym and receiving-nym fields are novel additions to the message format that provide protection against man-in-the-middle attacks. These fields are not standard in other security protocols, but are critical to hidden credentials message exchanges. Inside each encrypted message, the sending-nym field contains the nym of the person sending the message. This prevents a man-in-the-middle from substituting his nym during the hidden credentials nym exchange and thus being able to decrypt any response. The receiving-nym field includes the nym of the person receiving the message. The purpose of the receiving-nym field is to prevent the server from attempting to impersonate the client to another server.

Detailed threat model for hidden credentials

Hidden credentials are a new approach to trust negotiation that affords more privacy to the user than traditional trust negotiation. Because hidden credentials are a relatively new authentication method, a detailed threat model can help identify the risk from possible security threats. By identifying the risk associated with most types of

practical threats, the threat model helps build confidence in the hidden credentials protocol. The threat model presented in Chapter 5 is a detailed examination of the most common types of attacks.

7.2 Future Work

There are numerous opportunities for future work within the hidden credentials realm. A robust credential format and template must be established to allow interoperable construction of attribute + nym pairs. The current credential format is simple, promoting ease of use, but it is not easily extensible should credentials require additional data. Other protocols, such as SSH, Kerberos, and SAML [31], could benefit significantly from integration with hidden credentials by facilitating privacy-preserving exchanges that require no credential disclosure.

Developing an effective checksum is an area for future research. There are two reasons a message recipient may not be able to read a decrypted message. First, he may not possess the attribute credentials necessary to satisfy the policy and correctly decrypt the message. Second, the message may have been modified in transit by a faulty network. A mechanism using a checksum could help the message receiver identify if anything happened to the message in transit. If the recipient had this knowledge, he could send a retransmission request if the network corrupted the message or drop the connection if he does not have the correct credentials.

Mitigating the effects of a malicious CA is a challenging area of research for hidden credentials. Because a CA can impersonate any user that relies on it, a malicious CA can be very harmful to a public key cryptosystem. However, there may be ways to lessen the effect of a malicious CA by each user generating his own CA. Then each

authenticating party should authenticate against the traditional CA and the user's CA. The reliable establishment of the public key of the user's CA by the authenticating party thwarts most malicious regular CA impersonation attacks. The semantics and protocol for this type of interaction deserve exploration and definition.

Another significant opportunity for future work exists in the scenario where a prolonged session depends on hidden credentials authentication. When Bob cannot satisfy a policy, he will bluff by sending a NAK message. Alice cannot understand this message, but does not know if the response was encrypted with a NAK message or if it is a message she is not authorized to decrypt. She does not want to reveal the fact that she could not satisfy the policy, so she also sends a NAK message back. At this point, neither party is willing to admit that they could not understand the other's message, so they continue bluffing, sending NAK messages back and forth. The problem is that neither party wants to disclose to the other that they do not possess the credentials required to decrypt the most recent message. The simple solution to this problem is for both parties to agree to limit the number of rounds before beginning the negotiation. However, if the parties select a limit that is too low, it is possible they will not be able to finish the negotiation before the reaching the limit. Another aspect to consider is that a malicious user may attempt to execute a DOS attack by setting the round-limit high and using NAK messages throughout the interaction. Finding an appropriate balance between large and small round-limits is an area for future research.

Bibliography

[1] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret Handshakes from Pairing-Based Key Agreements. *2003 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.

[2] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. IETF Request for Comments: 3546, June 2003.

[3] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. Extended abstract in proceedings of *Crypto 2001, Advances in Cryptology*, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 213-229, August 2001.

[4] T. Chan. Inter-session Trust Preservation in Trust Negotiation. M.S. thesis, Computer Science Department, Brigham Young University, June 2004.

[5] M. Crispin. Internet Message Access Protocol – Version 4. IETF Request for Comments: 3501, March 2003.

[6] T. Dierks and C. Allen. The TLS Protocol Version 1.0. IETF Request for Comments: 2246, January 1999.

[7] Federal Information Processing Standards (FIPS) – 180-1. Secure Hash Standard, April 1995.

[8] Federal Information Processing Standards (FIPS) – 197. Advanced Encryption Standard, November 2001.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. IETF Request for Comments: 2616, June 1999.

[10] Foundation for Information Technology Education. Email survey. (http://www.edfoundation.org/internetusagesurvey.htm).

[11] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. *2000 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[12] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith. Advanced Client/Server Authentication in TLS. *Network and Distributed System Security Symposium*, San Diego, CA, February 2002.

[13] A. Hess. Content-triggered Trust Negotiation. M.S. thesis, Computer Science Department, Brigham Young University, February 2003.

[14] J. Holt, R. Bradshaw, K. E. Seamons, and H. Orman. Hidden Credentials. *2nd ACM Workshop on Privacy in the Electronic Society*, Washington, DC, October 2003.

[15] http://www.pgpi.org/doc/pgpintro/ -- taken from Chapter 1 of the document *Introduction to Cryptography* in the PGP 6.5.1 documentation. Copyright © 1990-1999 Network Associates, Inc.

[16] J. Jacobson. Session-Level Access Control for Automated Trust Negotiation. M.S. thesis, Computer Science Department, Brigham Young University, July 2003.

[17] R. Jarvis. Protecting Sensitive Credential Content During Trust Negotiation. M.S. thesis, Computer Science Department, Brigham Young University, April 2003.

[18] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). IETF Request for Comments: 1510, September 1993.

[19] M. Kolsek. Alert: Bypass Warnings For Invalid SSL Certificates. NTBugtraq, June 2000.

http://www.ntbugtraq.com/default.asp?pid=36&sid=1&A2=ind0006&L=ntbugtraq&F=P&S=&P=2339

[20] N. Li, W. Du, and D. Boneh. Oblivious Signature-Based Envelope. *ACM Symposium on Principles of Distributed Computing (PODC 2003)*, Boston, MA, July 2003.

[21] J. Myers. Post Office Protocol Version 3. IETF Request for Comments: 1939, May 1996.

[22] P. Persiano and I. Visconti. User Privacy Issues Regarding Certificates and the TLS Protocol. *7th ACM Conference on Computer and Communications Security (CCS)*, Athens, Greece, November 2000.

[23] J. Postel. Simple Mail Transfer Protocol. IETF Request for Comments: 821, August 1982.

[24] S. G. Renfro. CZAG: Privacy Leak in X.509 Certificates. *11th USENIX Security Symposium*, San Francisco, CA, August 2002.

[25] R. Rivest. RC4 Stream Cipher. RSA Data Security, 1987.

[26] B. Schneier. *Applied Cryptography, 2nd edition*. John Wiley & Sons, New York, NY, 1996.

[27] B. Schneier. *Secrets and Lies: Digital Security in a Networked World.* John Wiley & Sons, New York, NY, 2000.

[28] W. Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, March 2000.

[29] T. Sundelin. Surrogate Trust Negotiation. M.S. thesis, Computer Science Department, Brigham Young University, July 2003.

[30] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH protocol architecture. I-D draft-ietf-secsh-architecture-13.txt. September 2002.

[31] Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1 (sstc-saml-tech-overview-1.1-04) Draft 04. March 2004.

[32] A. Westin. E-Commerce Privacy Survey. 1998. http://www.privacyexchange.org/iss/surveys/ecommsum.html

[33] W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 2000.

Appendix

Section A Hidden Credential Internal Message XML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://isrl.cs.byu.edu"
  xmlns="http://isrl.cs.byu.edu">
  <xs:element name="internal">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="timestamp" minOccurs="1" maxOccurs="1"/>
           <xs:element ref="authentication-token" minOccurs="1" maxOccurs="1" />
        <xs:element ref="sending-nym" minOccurs="1" maxOccurs="1" />
        <xs:element ref="receiving-nym" minOccurs="1" maxOccurs="1" />
        <xs:element ref="session-id" minOccurs="1" maxOccurs="1" />
        <xs:element ref="message-id" minOccurs="1" maxOccurs="1" />
        <xs:element ref="plaintext" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="authentication-token">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="plaintext">
    <xs:complexType mixed="true">
              <xs:attribute name="encoding" type="xs:NMTOKEN" use="optional"
default="none"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="receiving-nym">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="sending-nym">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="session-id" default="0">
    <xs:complexType mixed="true" />
  </xs:element>
```

```
  <xs:element name="message-id" default="0">
    <xs:complexType mixed="true" />
  </xs:element>

  <xs:element name="timestamp">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="date" />
          <xs:element name="time" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Section B Hidden Credential External Message XML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://isrl.cs.byu.edu"
  elementFormDefault="qualified"
  xmlns="http://isrl.cs.byu.edu">

  <xs:element name="content">
    <xs:complexType mixed="true">
      <xs:attribute name="encryption-algorithm" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="encoding" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="fulfillment_path">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ciphertext" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="fulfillment_paths">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="fulfillment_path" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
  <xs:element name="hc_external">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="type" minOccurs="1" maxOccurs="1" />
        <xs:element ref="version" minOccurs="1" maxOccurs="1" />
        <xs:element ref="fulfillment_paths" minOccurs="1" maxOccurs="1" />
        <xs:element ref="content" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="version">
    <xs:complexType mixed="true" />
  </xs:element>

</xs:schema>
```