Simple Public Key Infrastructure Analysis Protocol Analysis and Design

Alexander G. Vidergar

SIMPLE PUBLIC KEY INFRASTRUCTURE PROTOCOL ANALYSIS AND DESIGN

THESIS

Alexander G. Vidergar, 1Lt, USAF

AFIT/GCE/ENG/05-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/05-07

SIMPLE PUBLIC KEY INFRASTRUCTURE PROTOCOL ANALYSIS AND DESIGN

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Alexander G. Vidergar, BS

1Lt, USAF

March 2005

AFIT/GCE/ENG/05-07

SIMPLE PUBLIC KEY INFRASTRUCTURE PROTOCOL ANALYSIS AND DESIGN

Alexander G. Vidergar, BS

First Lieutenant, USAF

Approved:

/signed/	10 Mar 05
Maj Robert P. Graham, Jr., Ph. D.(Chairman)	Date
/signed/	10 Mar 05
Dr. Henry Potoczny (Member)	Date
/signed/	10 Mar 05
Dr. Richard Raines (Member)	date

Acknowledgments

I would like to thank my advisor for providing much needed guidance and support; my friends and family both in state and beyond for providing a sound link in the magnetic chain of humanity; the Krebs cycle, without it none of this would have been possible; and finally, the people in the corner for making time for my appointments.

Alex Vidergar

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCE/ENG/05-07

Abstract

Secure electronic communication is based on secrecy, authentication and authorization. One means of assuring a communication has these properties is to use Public Key Cryptography (PKC). The framework consisting of standards, protocols and instructions that make PKC usable in communication applications is called a Public Key Infrastructure (PKI). This thesis aims at proving the applicability of the Simple Public Key Infrastructure (SPKI) as a means of PKC.

The strand space approach of Guttman and Thayer is used to provide an appropriate model for analysis. A Diffie-Hellman strand space model is combined with mixed strand space proof methods for proving the correctness of multiple protocols operating in the same context. The result is the public key mixed strand space model. This model is ideal for the analysis of SPKI applications operating as sub-protocols of an implementing application.

This thesis then models the popular Internet Transport Layer Security (TLS) protocol as a public key mixed strand space model. The model includes the integration of SPKI certificates. To accommodate the functionality of SPKI, a new protocol is designed for certificate validation, the Certificate Chain Validation Protocol (CCV). The CCV protocol operates as a sub-protocol to TLS and provides online certificate validation.

The security of the TLS protocol integrated with SPKI certificates and sub-protocols is then analyzed to prove its security properties. The results show that the modified TLS protocol exhibits the same security guarantees in isolation as it does when executing its own sub-protocols and the SPKI Certificate Chain Validation protocol.

SIMPLE PUBLIC KEY INFRASTRUCTURE PROTOCOL ANALYSIS AND DESIGN

I. Introduction

Diffie and Hellman provided the foundation of public key cryptography in the 1970s in (1) and it has been used in electronic communications ever since. However, as often as it has been used successfully, poor implementations have resulted in its exploitation. It has become increasingly clear that the strength of a cryptographic system rests not only on the mathematics of cryptography but also on the protocols and implementation methods used in application design.

This work focuses on the cryptographic framework provided by the Simple Public Key Infrastructure (SPKI). This highly customizable and flexible standard implements a robust public key infrastructure aimed at overcoming the shortcomings of current X.509-based architectures. Integrating SPKI into an existing application or building a new one must be done with diligence. Despite its strong cryptography, SPKI is just as vulnerable to the shortcomings of poor protocol design as any security framework.

This thesis demonstrates the use of strand space, the formal protocol analysis method developed by Thayer, et al (2). The strand space formalism is used as a means of integrating SPKI into the Transport Layer Security Protocol (TLS). The security properties of TLS are proven and a new protocol, to accommodate the validation of SPKI certificates, is designed. These results provide the solution to gaps in previous SPKI secure web implementations, which assumed execution in a secure environment (27; 12). Furthermore, the method used here encourages up front consideration to security protocol design by incorporating analysis into the development process. As pointed out by Arkin, Stender and McGraw in their article on software penetration testing (3:84-87), too often security matters are taken into consideration late in the design process. However, if care is taken when developing a protocol, strand space proofs can be direct implications from the design.

Chapter 2 identifies the context of cryptography this project focuses on, the motivation for SPKI development and the basic foundation of the strand space formalism. Chapter 3 identifies how strand space was tailored to accommodate SPKI analysis and how it is intended to prove properties about multiple protocols operating in the same environment. Chapter 4 provides theorems and their proofs based on TLS security properties and the execution of sub-protocols within the context of TLS. Additionally, Chapter 4 contains the design of the new Certificate Validation protocol. The final chapter provides a summary of the analysis, future work and conclusions.

II. Public Key Cryptography and Protocol Analysis

2.1 Security & Cryptography

The National Research Council (NRC) reveals in (4: Part I) just how critical security in electronic communication is to individuals, companies and the government as a whole. The essence of this security is to prevent the interception, disruption and alteration of a communication between two or more principals. A modern day solution to these problems is Public Key Cryptography (PKC). Developed in the seminal paper by Whitefield Diffie and Martin Hellman (1), PKC offers solutions for secrecy, authentication and non-repudiation, among other desired security goals of the NRC proposal.

As PKC has developed, it has become common practice to use the Alice-Bob scenario as meaningful, albeit informal means of discussing protocol interactions and the security desires of principals executing a protocol. The nomenclature is introduced here as it will be used throughout the analysis and is found throughout cryptography literature. The scenario posits two principals, Alice and Bob, denoted in short hand as A and B. Alice wishes to communicate some message to Bob. Depending on the sensitivity of the information, Alice may want to authenticate Bob (or vice versa), ensure secrecy of the communication or have Bob prove to her that he is authorized to receive the message.

The goal of a security protocol is generally to provide some combination of secrecy, authentication or authorization. To begin, secrecy is the most basic of security properties. A secret communication is intended to be impossible for a third principal, referred to as a penetrator, to decipher. Consequently a protocol that is secret is assumed to be impervious to eavesdropping.

Authentication, on the other hand, provides Alice with some degree of certainty that she is in fact speaking with Bob. A ‘man in the middle’ attack such as the one discussed in (5:8-10), would allow a penetrator to trick Alice into thinking she is talking with Bob when in fact she is communicating with the penetrator. A message that can be guaranteed through some cryptographic means to come from a particular party is said to be authentic. A stronger version of authentication is non-repudiation. A non-reputable message is provably sent and received from the appropriate principals.

Beyond authentication and secrecy, is authorization. Perhaps Bob authenticates Alice, and they communicate securely, however Bob wants to know weather or not Alice should be allowed to know what he is about to tell her. If he can determine Alice has permission to receive the information, she is said to be authorized.

Public Key Cryptography offers a flexible solution to authentication and secrecy. Each principal obtains a key-pair consisting of a public and a private key. Although the keys are different, they exhibit mathematical properties such that a message encrypted with one can only be decrypted by the other. The mathematics driving this functionality is rooted in number theory and expressed thoroughly in (6:175-178, 275-280).

In PKC, each principal keeps one key secret and publishes the other to be publicly available to other principals. In a public key environment, if Alice wishes to communicate to Bob secretly, she encrypts her message with Bob’s public key. If Bob’s private key is uncompromised, he is the only one that can decrypt and read Alice’s message.

PKC also provides the ability to sign messages. Signing is the converse of an encryption. A principal signs a message by using its private key to encrypt it. Due to the property of the PKC key pair, everyone with access to the public key can then decipher and read the message. However, if Alice signs a message with an uncompromised key and supplies it to Bob, then Bob can be positive that Alice is the only one that could have created that message. Of course in this situation the communication, although authentically from Alice, provides no secrecy, as all principals that can hear the message can use Alice's public key to decrypt it. To provide secrecy to this interaction, the signed message would need to be encrypted with a private or secret key.

Signatures enable the use of certificates. A certificate is a message signed by a principal and is a guarantee from that principal. Often times trusted principals are an authority of some type. For example a name authority provides certificates that assert guarantees of a principal's identity. A certificate issued by a certificate authority (CA) is assumed to be true based on the trust of that CA. A certificate can be applied to ownership of a resource, an identity or some other type of electronic relationship. For example, take a scenario such that both Alice and Bob trust a third principal, Cyril. In this instance, Cyril provides certificates to Alice and Bob. Alice's certificate contains her public key and the same is true for Bob's certificate. When Alice sends a message signed with her private key, Bob can decrypts it with her public key provided by Cyril's certificate. If Bob needs additional proof, he can ask Cyril to verify the authenticity of the certificate used by Alice. In either case, Bob now knows or assumes that Alice is not only who she says she is, but also she is who the CA, Cyril, claims her to be. This scenario is simplistic, but it illustrates that a certificate provides a guarantee based on the level of trust for the CA.

2.2 Public Key Infrastructure

A Public Key Infrastructure (PKI) is the combination of standards, procedures and protocols used to issue and revoke keys or authenticate keys that are in used in public key cryptography (7). The design of this infrastructure is just as important as the cryptography itself. If there is a flaw in any part of the infrastructure, then the integrity of the system as a whole is jeopardized.

For clarity, it is convenient to define common cryptographic terms. A *key* is assumed to be either the public or private part of a key pair used in PKC. In general the only keys PKC protocols openly use are the public keys, whereas private keys are stored safely and only used their owners. Symmetric keys will be scarcely used; however, when they are used they are identified with qualified names such as a session keys or long-term keys. These are assumed to be generated in a secure manner or stored in a way that prohibits their exposure.

Certificates, issued by Certificate Authorities, provide a guarantee about their owner. Most commonly certificates provide guarantees about authorization, group membership or identities. Regardless of what a certificate is issued to certify, all certificates are a binding between a principal’s public key and the information representing the certificate type.

A principal is a user of the system. Principals are one of two types, regular or penetrator. A regular principal follows the protocols and standards of the PKI and operates within system parameters. A penetrator is a malicious principal that drives to undermine the security provided by the PKI.

A protocol is the functional description of principal interaction. A protocol defines what information is sent, what format it is in and when to send it. Generally, protocols are built with one or more security goals in mind. Thus, the execution of a safe protocol will provide some combination of authentication, secrecy or authorization. However, a protocol that attempts too much can lead to a cumbersome implementation. If an infrastructure based on this is considered too burdensome to use, it is as useless as not having a secure infrastructure at all (8).

2.2.1 X.509.

X.509 is the de facto standard for Internet PKI. X.509 establishes the framework for a centrally controlled directory of cryptographic keys and users. The directory is managed by Certificate Authorities, which carry out the procedures for supplying and validating certificates. A corresponding authority in the structure is the Naming Authority (NA). The NA controls the scheme with which a CA issues certificates to a particular name. Often times it is convenient for NAs and CAs to be the same entity, however, this can preclude independent CAs from sharing a single NA, which degrades the continuity of the global directory X.509 relies upon (9).

X.509 certificates bind a public key to a name of a principal with a global directory of names. This type of certificate is called a name certificate. X.509 relies on a unique name for each user participating in the PKI. This empowers the infrastructure to issue certificates and provide definitive guarantees as to the registered identity of individuals. When a certificate is under question, all one needs to do is refer to the global directory and all the necessary information is provided concerning that certificate and the individual it was issued to.

In practice, X.509 has revealed a number of shortcomings. Privacy is an important issue in cryptography; however, the existence of a global hierarchy inherently lacks privacy. For example, a company that requires its employees to have some degree of anonymity may want them to utilize PKC via the X.509 hierarchy. In doing so they would identify themselves as an employee of that company. However, to protect against this type of information leak, the company may choose not to register a portion of its employees in the global directory. Although this protects that group's privacy, it diminishes the inherent strength of the directory by decreasing the number of participants.

X.509 authentications rely on having on individuals registered to unique names. This restricts usable names and also presents a considerable challenge to issue unique certificates. Due to the global scope of the directory, it is a sizeable configuration challenge to coordinate non-repeating names for all users.

Currently in draft form, the X.509 standard overall is a fairly complicated and cumbersome standard. As a result, it has been subject to wide interpretation. There are no guarantees between different implementations that certificates will be formed in the same manner, be processed in the same way, or even be accepted by all applications using them.

This overview is only meant to reveal some of the clear problems with the X.509 standard and justify why others are motivated to propose new standards. ` more thorough review of X.509 and public key infrastructures is available in (10: 3-9), and the technical reader is directed to the standard itself (9).

Carl Ellison, et al (8), in light of X.509 shortcomings, has proposed an alternative infrastructure to support public key cryptography. They aimed at creating a PKI that is extensible, robust and easy to use. This new PKI is aptly named the Simple Public Key infrastructure (SPKI).

2.2.2 SPKI/SDSI.

In short, SPKI is just as its name suggests, a simple PKI. It was developed concurrently with the Simple Distributed Security Infrastructure (SDSI), a standard for defining certificates. Based on the same principles, these two standards eventually merged to form SPKI/SDSI. For brevity, the pair is most commonly referred to as simply SPKI.

Traditionally, certificates have been a binding between a name and a key. However, Ellison, et al, point out in (11:7-8) that a key holder's name is rarely of security interest. Rather, it is argued, the authorizations of that person are inherently more useful. Thus, in addition to name certificates, SPKI uses authorization certificates. This novel concept allows an authority to associate an authorization directly to a principal. The certificate thus is a binding of an authorization to a key and offers an explicit and customizable assurance of authorization.

Authorization certificates enable a CA to provide anonymity and privacy to the users of the PKI. If the identity of the principal is not contested but rather only a user's authorization to perform an action, a user's identity can remain private. One application of this is secret balloting. If keys are bound with no identifying information and distributed blindly, then participants may use them to vote without revealing their identity. This is made possible by using separate certificates for identity and authorization. Where traditional certificates merge identity and authorization, SPKI breaks these into two types of certificates, which offers the flexibility for unique situations such as anonymous voting or group authorizations.

SPKI governs authorization through customizable authorization tags. There are very few limitations to these tags in order for them to be easily applied in a diverse range of applications. This flexibility, however, comes with a price. The issuers of the certificates must have intimate knowledge of the authorization requirements of the system. This problem is compounded by the distribution of certificates via delegation.

In the SPKI framework, each principal is empowered with the ability to issue certificates. In short, each principal is a certificate authority for any resource it controls. If a resource requires secrecy, authentication and authorization, then certificates granting access to that resource reflect that all three security properties must be fulfilled to validate access.

Intuitively, each principal is the keeper of his or her own resources. Therefore must manage them and provide certificates that will make sense when supplying access to those resources. Since certificates are issued by all principals, there is no need to coordinate with other principals to ensure unique names. Just like the authorization certificates, each principal has its own frame of reference concerning principals it will work with. Each principal's perspective is referred to as a namespace. A namespace represents the domain of that principal. A principal that controls a university will have a large namespace, while a student within that university may have a very limited one.

The management of a large namespace is simplified by the ability to delegate authority through SPKI certificates. Using the university example again, the principal in charge of a university can delegate university authority to its various departments, and in turn the departments can delegate it down to their own divisions. Ultimately, a student in the delegation chain may be issued a certificate that identifies him or her as a student of the university in the computer science department in the graduate school.

An authorization certificate has a delegation bit to manage the user of delegation. If the delegation bit is enabled, then the principal issued that authorization can delegate it to principals in its own namespace. In the university example, the university initially supplies an authorization with the delegation bit enabled. In turn each level of the hierarchy re-issues the authorization with the enabled bit. Finally, the student receives a certificate with a disabled delegation bit. Thus, although the student is authorized to access student resources, that student does not have the ability to delegate that authorization to anyone else. In this way, a local hierarchy is built with the delegation of authority beginning at the university level and ultimately ending with a student who no longer has the ability to delegate.

Once delegation has taken place and meaningful keys are distributed, there still remains the challenge of proving a certificate was issued from the proper chain of authority. This process is called *certificate chain discovery*. In essence it follows a transitive property. For example, imagine three principals A, B, and C. A is the owner of a resource, and grants authority to B to delegate access to that resource. B delegates authorization to C to use the resource. In turn, C asks to use the resource, and is challenged by A. In response to the challenge, C provides a certificate chain that shows the delegation from A→B→C. A reduces the certificate chain to a simple form A→C, and is convinced that C should be authorized to access the resource.

Due to the potential problem of determining the appropriate chain to supply given a challenge, a good deal of research has been conducted on this topic (12;13;14) and has resulted in a tractable and efficient algorithm for certificate chain discovery.

2.3 Protocol Analysis

Modern protocol proof methods are generally founded on two seminal papers with regards to abstraction and focus. The Dolev-Yao model, proposed originally in 1981 (15), abstracts protocol messages into a term algebra. In applied cryptography messages are bit-strings, however, abstracting them into terms allows the focus of the analysis to be on the protocol itself. These terms are then applied to a first order equational logic, which allows them to be manipulated as they would in a cryptographic protocol.

Because attacks on cryptographic systems generally avoid challenging the mathematics cryptography is built upon (6), but rather exploit protocol design, it is possible for poor protocols to undermine the very security they intend to provide. (16) provides numerous examples of security protocols that are now defunct or have undergone numerous fixes to prevent simple intrusions. Many protocol attacks are a result of incomplete, misguided or misinterpreted proofs of protocol correctness.

Along this vein, Woo and Lam argue that the problem with protocol analysis is that it fails to separate correctness and verification. Instead, it includes too much in a single analysis, without considering the importance of individual parts. This focus, they argue, only contributes to complicated proofs that are often misunderstood.

Consequently, Woo and Lam break down analysis into two elements, secrecy and correspondence. Secrecy is the notion of maintaining data integrity in the presence of a penetrator. Integrity includes a secret message remaining secret and unaltered from when it was sent. Correspondence, on the other hand, is what has been referred to thus far as authentication. By dividing these two notions it is possible to obtain flexible analysis for a diverse range of protocols.

Based upon Dolev-Yao and supported by the Woo-Lam notion of separation of secrecy and authentication, Thayer, et al, developed the strand space formalism for protocol analysis (2).

2.3.1 Strand Space.

The strand space formalism establishes an inductive base used to prove theorems about protocol correctness. This section provides the basic definitions of strand space, although a more thorough description is available in the defining papers (2; 17; 18; 21; 28; 29).

The strand space formalism imposes a Dolev-Yao style set theory on protocol messages. All data used and communicated by a protocol is a member in the set A. This set is then specialized into disjoint sets for more accurate models of data. In particular, A contains the subsets of keys K and texts T. The language of strand space is then built freely from these sets using operators appropriate to the protocol being analyzed (2).

The encryption operator takes any term g from A, a key k from K, and outputs an encryption $\{g\}$k that is in A, but not in either K or T. The join operator simply takes two terms, *g* and *h*, concatenates them and results in a new term in *A*. These sets allow a proof to reason about which sets of information are available to each principal and penetrator of a protocol. As protocols become more specialized, these operators can be expanded, such as in (17).

Furthermore, a sub-term relationship $\sqsubset$ is defined over terms. The result of the join operator is a term which contains the two sub-terms used to create it. On the other hand, the sub-term of an encryption is only the payload of the cipher text and not the key used to create it. Formally, the sub-term relationship $\sqsubset$ is the smallest inductive relation such that it has the properties of Table 1 (23:11-23).

Table 1 Sub-term relationships of terms

Relationship	**Elucidation**
$t \sqsubset t$	*t* is a sub-term of itself
$t \sqsubset \{g\}k$ if $t \sqsubset g$	*t* is a sub-term of an encryption only if it is a sub-term of the payload of that encryption.
$t \sqsubset gh$ if $t \sqsubset g$ OR $t \sqsubset h$	*t* is a sub-term of a concatenation if it is a sub-term of either of the terms composing the concatenation.

The strand space formalism models a protocol into a set of nodes and edges. Each principal of a protocol is modeled by a subset of these nodes and edges forming a graph structure called a strand. A *strand* consists of a series of temporally ordered nodes connected by intra-strand edges $\Rightarrow$ representing a series of actions. These actions will always be appropriate for the type of strand they are on and can include the deciphering of an encryption, the concatenation of terms or the separation of terms. Each node of a strand is associated exclusively with the reception or the transmission of a message from that node's strand to or from another strand. Inter-strand communication is represented by inter-strand edges $\rightarrow$. An inter-strand edge always includes a term from the set *A* representing information communicated to or from that node.

A principal is represented by one or more strands. Several strands linked together constitute a strand space representing all possible communications between connected strands. Subsets of this graph, called bundles, are more manageable and are used to accurately and minimally represent protocol principal interactions. Appendix A provides a simple example of a protocol and its strand space representation.

These and other formal definitions constitute the foundation of the strand space formalism (2: 6-15):

A strand space Σ is composed of the following:

nodes – a tuple $\langle s,i \rangle$, where *s* is a strand, $s \in \Sigma$ and *i* is an integer $1 \leq i \leq$ length of the strand. The set of *nodes* is denoted $\mathcal{N}$. The *node* $\langle s,i \rangle$ belongs to the strand *s* and every *node* belongs to exactly one strand.

terms – if n = $\langle s,i \rangle \in \mathcal{N}$ then index(*n*) = *I* and strand(*n*) = *s*. Then term(*n*) is the *i*th signed *term* in the trace of *s*. That is the *i*th *term* communicated between this strand and another.

inter-strand edges – an edge $n_1 \rightarrow n_2$ if and only if term(n_1) = +*t* and term(n_2) = -*t* for some $t \in A$. The sign of a term indicates weather it has been sent (positive) or received *(n*egative). An inter-strand edge captures a causal relationship between strands.

intra-strand edges – When *n*1 = $\langle s,i \rangle$ and *n*2 = $\langle s,i+1 \rangle$ and *n*1, *n*2 $\in \mathcal{N}$, there is and edge $n_1 \Rightarrow n_2$. This type of edge expresses that *n*1 is an immediate causal predecessor of *n*2 on the strand *s*. The set *n′* is used to denote all predecessors of a node on a single strand.

occurrences – a term *t* is said to *occur* on a node *n* if and only if $t \sqsubset$ term*(n*).

entry points – the node $n \in \mathcal{N}$ is an *entry point* for a set of terms *I* if and only if term*(n*) = +*t* for some $t \in I$, and whenever term*(n′*) $\not\in I$.

originations – *A* term *t originates* on $n \in \mathcal{N}$ if and only if *n* is an entry point for the set I =$\{t' : t \sqsubset t'\}$.

unique originations – a term *t* is *uniquely originating* if and only if *t originates* on a unique $n \in \mathcal{N}$.

bundles – a bundle $\mathcal{C}$ consists of a finite subset of nodes, inter-strand edges and intra-strand edges of a given strand space. If the node $n_2 \in \mathcal{C}$ and term*(*n_2) is negative then there is a unique $n_1 \in \mathcal{C}$ such that $n_1 \rightarrow n_2$. Furthermore, if $n_1 \Rightarrow n_2$ is in Σ and $n_2 \in \mathcal{C}$ then $n_1 \Rightarrow n2$ is in $\mathcal{C}$. Finally, this subset of Σ must be acyclic to be considered a bundle.

It is worthwhile noting that the properties of a bundle allow a partial order relation to be defined. Thus, $\preceq_{\mathcal{C}}$ is the reflexive, anti-symmetric, transitive closure of the edges of a bundle. This ordering ensures that every non-empty subset of nodes in a bundle has a minimal member with respect to $\preceq_{\mathcal{C}}$. Furthermore any term, t, of the minimal node of a bundle must have a positive sign, and is an originating occurrence of t.

Strand space was developed to prove properties not only about protocols in an isolated environment, but also can represent multi-protocols in a single strand space. A strand space representing a primary protocol and any number of sub-protocols is called a *mixed* strand space. A protocol used in the presence of sub-protocols is called a *base* protocol. A base protocol can have any number of sub-protocols that may influence the security properties of the all the protocols it interacts with.

In contrast to this hierarchy of sub-protocols, is the analysis hierarchy. The subject of an analysis is called the *primary* protocol. Any other protocol operating in the same strand space as a primary protocol is called a *secondary* protocol. The primary protocol is used to produce the foundation of definitions, properties and rules with which to compare secondary protocols. A base protocol may intuitively seem like the best choice for a primary protocol, however, this does not have to be the case.

More formally, in a mixed strand space Σ, the set Σ_1 is the set of all primary regular strands. The remaining regular strands Σ_2 are secondary strands and constitute the set difference of all primary strands and the set of regular strands. Thus, the mixed strand space Σ is composed of the disjoint sets of primary strands Σ_1, secondary strands Σ_2 and penetrator strands $\mathcal{P}_\Sigma$.

The strand space model is further extended in (17) to provide a means for analyzing the Diffie-Hellman (DH) protocol. A protocol is said to be *conservative* with regard to DH if a generated key arises on a regular node only when the values used to create it arise on regular nodes. Furthermore, a protocol is said to be *silent* with respect to DH if no DH generated key originates on a regular node. It is also convenient to conclude that a bundle over a protocol that is *silent* and *conservative* with respect to DH, and only has DH constructors that arise on regular strands will never originate a DH generated key.

The strand space formalism allows the application of induction to prove the properties of bundles representing protocol interactions. Various proofs that use this formalism are published in (2). As strand space has developed it has expanded to include a proof method to more easily apply the formalism to protocol analysis: the authentication test.

2.3.1.1 Authentication Tests

The authentication test method for strand space analysis was established to expedite secrecy and authentication proofs (18). Authentication tests do this by building on strand space theory and an assumption referred to as the *normal form lemma*. The normal form lemma simply limits the actions of the penetrator to non-trivial manipulations of messages. Among other things this means that if a term is encrypted with a key the penetrator does not have, then it cannot be deciphered. It has been proven that enforcing this restriction on the penetrator only forces an ordering of actions, but does not limit what can be accomplished by the penetrator. The useful consequence of this is that if a term is sent encrypted with a safe key then it can only be altered by a regular participant. Thus, if that term is returned altered, it can be assumed that a regular participant received it, altered it, and sent it back. Such a term is what Thayer, et al, defines as *a test component* and the actions taken on it as an *authentication test*.

Authentication tests rely on two additional assumptions, proven elsewhere (18), regarding keys used in cryptography. First, messages sent with a key known by the penetrator can be manipulated by the penetrator. In addition, keys that are sent encrypted with a key known by the penetrator then become a key known by the penetrator. These are the only means the penetrator has of obtaining keys. Conversely, a key is considered *safe* if it is either never uttered by a principal in the protocol or is only uttered as a sub-term of a term encrypted with a key that is not known by the penetrator.

In order to reason about messages sent between strands, it is necessary to deal with the atomic pieces of terms. These are defined as *components.* A term t_0 is a component of a term t if it is a *sub-term* of t, if it is not a concatenated term, and no concatenated term in t is equal to t_0 Less formally, components are either an atomic value or they are an encryption.

A component is considered *new* at a node in a strand if it is a component of that node, but it is not a component of any previous node on that strand. The fact that it may have been a sub-term of a larger component previously in the strand makes no difference. Since the component was not a visible component of previous terms, it is new at its first non-sub-term appearance. Thus, it is an intuitive assumption that if a component occurs new on a regular node, then the strand has either generated, encrypted or decrypted information to supply the new component.

The authentication tests themselves revolve around the transformation of components. An edge $n_1 \Rightarrow n_2$ is a *transformed edge* for a term *t*, if n_1 is positive and n_2 is negative, *t* is a *sub-term* of n_1 and there is a new component t_2 of n_2 which contains *t* as a *sub-term*. On the other hand an edge is a *transforming edge* if n_1 is negative and n_2 is positive, *t* is a sub-term of n_1 and there is a new component of n_2 which contains *t* as a *sub-term*. Figure 1 shows visually a simple example of both types of edges.

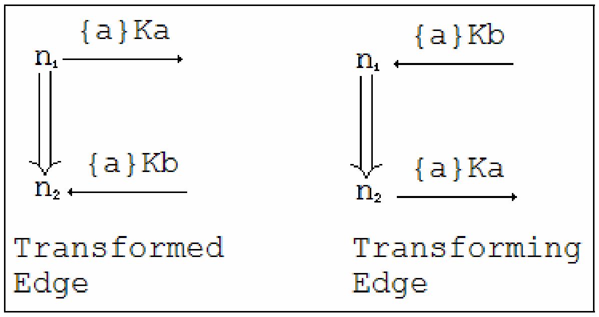


Figure 1 Strand space edges

Components of interest in proofs are known as *test components*. A component *c*, is a *test component* for a term *t* at a node if *t* is a *sub-term* of *c* and *c* is a component of that node, and the term *c* is not a proper *sub-term* of a component of any other node in the strand space. Combining this with the previous edge definitions, an edge between two nodes ($n_1 \Rightarrow n_2$) is a test for term *t* if *t* uniquely originates at node n_1 and the edge between n_1 and n_2 is a transformed edge for *t*.

A test component is used as a challenge to another participant in a protocol. One way to offer this challenge is to take a uniquely originating value, such as a nonce, encrypt it and send it to another principal. At this point, the challenge is to see if the principal is able to decipher it, this is called an *outgoing test*. If instead the value is sent in the clear and is expected to be returned in an encrypted form, the interaction is an *incoming test*. Depending on the properties of the messages being sent, these two authentication tests can offer a variety of guarantees to one or both of the parties.

Authentication tests provide guarantees about the existence of regular strands receiving sent messages. If a bundle includes an outgoing test for a component then there exists a corresponding set of nodes from another regular strand that constitute a *transforming edge* for that component. Additionally, if a component *c* only occurs as a sub-term of an encrypted term *t* of the regular strand including the transforming edge, and *t* is not a proper *sub-term* of any regular component, and the key used to encrypt it is not known by the penetrator, then there must be a negative regular node with *t* as a component.

The corresponding assertions are also valid for incoming tests. Given a bundle and a term *t* that is an incoming test for *a* within that bundle, then there exist regular nodes that *t* is a component of and there is an edge corresponding to those nodes that is a transforming edge for *a*.

One final definition is required for a third authentication test, the unsolicited test. A negative node *n* is considered an *unsolicited test* for an encrypted term *t*, if *t* is a test component for any *a* in *n* and the encrypted key is not known by the penetrator. Thus, given a bundle with a node *n* which receives an unsolicited test for an encrypted term *t*, then there must exist a positive regular node from another strand such that *t* is a component of that node.

These authentication tests are combined with the properties of the terms being sent in a protocol to prove authentication and secrecy guarantees. Detailed examples are available in (18).

2.3.1.2 Penetrator Strands

Regular strands are those that represent a legitimate and accurate run of a protocol. In a two-principal protocol, regular strands are generally called the initiator and responder strands. Protocols that include a trusted server will also include a server strand. This naming method of strands is not formal; however, it is employed to make reasoning about protocols more easily understood. The other type of strand useful to protocol analysis is the penetrator strand.

In strand space the penetrator is represented by eight types of penetrator strands. These strand types are separate in order to differentiate what a penetrator can and cannot do in an attempt to infiltrate a protocol. Roughly speaking, the eight strands capture the penetrator’s ability to block messages, generate messages, join messages from parts of other messages, and apply encryption or decryption. These effectively model what typical protocol attacks consist of and thus reveal these weaknesses during protocol analysis. The eight strands are represented by a single letter, M, F, T, C, S, K, E, or D. Their definitions follow.

M : Sending a message.

F : Receiving a message.

T : Receiving a message and sending it out to two other strands.

C : Concatenation. Receiving two different terms, and sending the concatenation of them to another strand.

S : Separation. Receiving a concatenation of terms, separating them, and sending each out individually.

K : Key. Sending a key as a term, all keys sent this way are assumed already known to the penetrator.

E : Encryption. Receive a key, and a term, and send out the term encrypted with the key.

D : Decryption. Receive in inverse key, and an encrypted term. Decrypt the term, and send out the unencrypted term.

Because the model of the penetrator is not protocol specific, it can be applied to any protocol under analysis. More importantly, it can help determine what information the penetrator can learn or how the penetrator can trick other strands into believing they are talking with someone else.

An infiltrated strand space is the union of regular and penetrator strands. Bundles carved out of infiltrated strand spaces reveal the weakness or conversely the strengths of a protocol under attack. Such an analysis is demonstrated in (2; 17; 18; 21; 28; 29).

In summary, a protocol is a sequence of interactions between principals and potential penetrators of the protocol. In strand space, each participant (regular and penetrator) is represented by at least one strand. A strand is built representing each principal's actions; messages sent are represented by an edge between two nodes on different strands. Each node has a causal relation to other nodes in the strand space, either within its own strand or to a node of another strand. A bundle is a sub graph of the strand space. Properties of bundles accommodate inductive proofs due to their partial ordering. A bundle representing a protocol may include only principal strands or principal strands entwined with penetrator strands. When bundles exhibit very specific properties, they allow assertions to be made about other strands in the protocol. These assertions are made through the application of authentication tests and prove the secrecy or authentication of the messages and principals, respectively, involved in the interaction. It is also important to note that in strand space assertions made about a protocol proven with authentication tests will be correct regardless of the presence of penetrator strands within its bundle.

III. Analysis Methodology

The SPKI standard provides a cryptographic framework to build secure applications. The framework supplies certificates that are used for authentication, secrecy and authorization. However, developed with the goal of flexibility in mind, the SPKI standard has minimal protocol specifications. Due to the sensitivity of information used in cryptographic applications, it is important for them to rely on formal proofs of the security properties they purport to provide. Consequently, it is necessary to supply these proofs before a secure application can be relied upon.

Furthermore, protocol design is a delicate matter. Overlooking minor details in a protocol can result in exposing the cryptographic system to catastrophic intrusions. A common practice is to imbed a new protocol into one that has already been proven correct. However, this apparently innocuous integration can completely undermine the security of the existing protocol. An example of this is the Neumann-Stubblebine protocol. The Neumann-Stubblebine protocol provides proper authentication on its own; however, upon execution of its session resume sub-protocol, the authentication of the original is undermined (17).

Since SPKI is an enabling technology, it will generally never be implemented as a stand-alone application, but rather, will be integrated into another application. That environment will utilize its own protocols and SPKI will be adjusted to its needs. This integration must be proven to supply the necessary security properties before it can be trusted. To prove that the implementing application and SPKI harmonize their security goals, it is necessary to prove the base protocol in an isolated environment in addition to in an environment running sub-protocols such as SPKI.

This chapter identifies which components of the strand space formalism can be assembled to provide a proof foundation for SPKI applications. Using this base and the strand space method of mixing protocol in a single analysis, the method is well-suited for base and sub-protocol execution in the same strand space. Furthermore, the methodology guides protocol design to ensure the security properties of the primary protocol are retained.

3.1 The PKI Strand Space

Due to the unique nature of cryptographic protocols, the strand space formalism must be adjusted to accommodate any unique operations of a protocol before it can be applied in analysis. This is done by adding additional operators to the term building logic and by adding new types of penetrator strands representing the actions of the added operators.

The strand space here is constructed for an environment conducive to DOD applications which would benefit from the security measures supplied by a public key infrastructure. In particular, it is a hybrid of the strand space model used in stand-alone Diffie-Hellman analysis (17) and in a mixed-protocol environment (28).

To build this strand space, the term algebra presented in Chapter 2 must be expanded to include PKI operations used in the Diffie-Hellman (DH) protocol (17). This is accomplished by adding the disjoint subset D to the set of all terms A. D represents DH exchange values. These are the values used to coordinate the shared secret key in a

DH exchange and the generated key from the DH algorithm. In addition, let H represent the range of a one way hash function and assume that H is another disjoint subset of *A*.

The operators available on all terms must also be expanded (17). Operators for hashing, signing and creating DH keys are included. All operators of the DH strand space are listed in Table 2.

Table 2 Strand space operators

Operator	**Meaning**	**Notation**
hash : $A \rightarrow \mathrm{H}$	Injective hash function	hash(t)
encr : $K \times A \rightarrow A$	Encryption	$\{t\}k$
sign : $K \times A \rightarrow A$	Signature	[t]k
join : $A \times A \rightarrow A$	Concatenation	gh
DH : $D \times D \rightarrow K$	Diffie-Hellman Calculation	f(ds,dc) $\rightarrow k'$

In addition to the term algebra, the PKI penetrator is given functionality as appropriate to the PKI strand space. The new model of the penetrator defined in following section.

3.1.1 PKI Penetrator.

The PKI penetrator model is the standard strand space penetrator, with the addition of public key operations in terms of the F, σ, X and H strands. Table 3 provides the type of strands used to model penetrator actions.

Table 3 PKI penetrator strand types and signatures

Strand designator and type	**Strand signature**
M – simple text message	< +*t* >
C – concatenation of terms	<-*g*, -*h*, +*gh*>
S – separation of terms	< -*gh*, +*g*, +*h* >
k – key tell	< +*k* >
E – encryption of terms	< -*k*, -*h*, +{*h*}*k* >
D – decryption of terms	< -*k*, -{*h*}*k*, +*h* >
σ – signature of terms	< -*k*, -*h*, + [*h*]*k* >
X – extraction of signed terms	< -[*h*]*k*, +*h* >
F – create fresh Diffie-Hellman value	< +ds >
H – compute injective one way hash	< -*g*, +hash(*g*) >

This model works on the assumption that the encryption being used by the regular strands is strong. The strong encryption assumption is that it is impossible for the penetrator to guess a key used in an encryption or signature unless he already has that key. Similarly, hashing functions are assumed to be irreversible and unpredictable, such as the MD5 (19) and SHA-1 (20).

3.3 Protocol Independence though Disjoint Encryption

In order to mix two or more protocols successfully, it must be proven that each protocol is safe in isolation and that the execution of all protocols together is also safe. One way to accomplish the latter is to prove that the protocols to be mixed are independent of one another. If so, then it can be concluded without further proofs that all security properties a protocol exhibits in isolation are also valid in the mixed environment.

Protocol independence requires further extensions to the strand space formalism. These were introduced in (28) as the concept of *respect*, but have been refined (21) to form a more thorough model called *independence*.

Two crucial definitions are those of privacy and fullness. An atomic value $a \in T \cup K$ is said to be *private* in Σ if it never originates in the set of secondary strands Σ_2 or the set of penetrator strands Σ_P. If it is not private, then it is considered *public*. Furthermore, a concatenated value *gh* is *public* if *g* and *h* are *public* and an encrypted value $\{h\}k$ is *public* if *h* and *k* are. Additionally, a strand space Σ is said to be *full* if every atomic value $a \in T \cup K$ that originates on Σ_2 also originates on some M-strand or K-strand in Σ. Intuitively, this defines a strand space with a penetrator that is fully capable of listening to messages and operating with them according to his defined actions. Thus, a full strand space is one with a penetrator strand entwined with regular strands to accommodate the full extent of the penetrator's abilities.

Disjoint Outbound Encryption – A strand space Σ has DOE if and only if given a positive node $n_1 \in \Sigma_1$, a negative node $n_2 \in \Sigma_2$ and a *private* term $a \sqsubset \{h\}k$ such that $\{h\}k \sqsubset \text{term}(n_1)$ and $\{h\}k \sqsubset \text{term}(n_2)$, then there is no positive node n_x such that $n_2 \Rightarrow^+ n_x$ and *a* occurs in a new component of n_x (21).

Disjoint Inbound Encryption – A strand space Σ has DIE if for all negative $n_1 \in \Sigma_1$, positive $n_2 \in \Sigma_2$ and for all $\{h\}k$, if $\{h\}k \sqsubset \text{term}(n_1)$, then $\{h\}k \not\sqsubset t_0$ for any new component t_0 of n_2 (21).

Disjoint Encryption – A strand space Σ has disjoint encryption if it has both disjoint inbound and disjoint outbound encryption (21).

If Σ is full and has disjoint encryption, then Σ_1 is *independent* of the set Σ_2. A simple case of independence occurs when the set of keys used to encrypt terms in Σ_1 are disjoint from those used in Σ_2. Intuitively, if no similar keys are used by both protocols, then independence is trivially true as the conditions are never challenged.

IV. An Analysis of the TLS Protocol

4.1 Transport Layer Security Protocol

Arguably one of the most widely used protocols in secure Internet communications is the Transport Layer Security protocol (TLS) (22: 1-2). TLS is a layered protocol designed to provide secret communication between two principals while offering the ability to authenticate each participant (23).

The two layers of the TLS protocol are the TLS Handshake protocol and the TLS Record protocol. The Handshake protocol is used to allow the principals to agree on what protocols to use for symmetric key generation. The Record protocol then uses the newly generated symmetric key to exchange information securely. RFC 2246 specifies the Change Cipher Spec and Alert protocols, in addition to the Handshake and Record protocols; however, for this work the functionality of these are assumed to be subsumed by the Handshake protocol.

The TLS Handshake protocol is a simple protocol designed to coordinate the usage of other protocols. During the Handshake protocol, principals negotiate not only which key exchange protocol to use, but also the parameters to use while running these sub-protocols. The two sub-protocols used for key exchange are the Diffie-Hellman public key agreement protocol and the RSA key exchange protocol (24). Once agreed upon, the key exchange protocol is executed as a sub-protocol. After a session key is generated from a successful key agreement protocol, the Handshake protocol concludes and TLS protocol will continue by executing the Record protocol or abort as is appropriate.

To apply strand space, the TLS protocol was abstracted into Dolev-Yao style terms (15). The abstraction allows the strand space proofs to reason how the protocol operates rather than obfuscate the analysis with implementation details.

The model used in this analysis was designed to focus on the interaction of cryptographic terms in the TLS protocol. Thus, the details of clear text messages identified in the TLS standard (23) are abstracted to a simple terms as they have limited impact on the underlying cryptography. A similar abstraction was made in (17).

The TLS protocol itself is abstracted into one of two base protocols, either the SAP, Server Authentication Protocol (Figure 2) or the SCAP, Server & Client Authentication Protocol (Figure 3). Sub-protocols executed within the context of these base protocols are the Resume Session (Figure 4) and the Certificate Chain Validation protocols.

4.1.1 TLS: Server Authentication Protocol.

The TLS Server Authentication protocol is depicted in Figure 2. The client initiates the exchange by sending a message with two components in order to negotiate which protocol to use for the key exchange. T_c is a list of client-supported protocols and parameters. If the client desires to resume an old session, it also sends the preferred session ID, S_c. However, most frequently the S_c message is a null message, indicating a new session needs to be established. This first exchange is a minimized representation of the ClientHello message in the TLS standard (23). Tc is assumed to include DH.

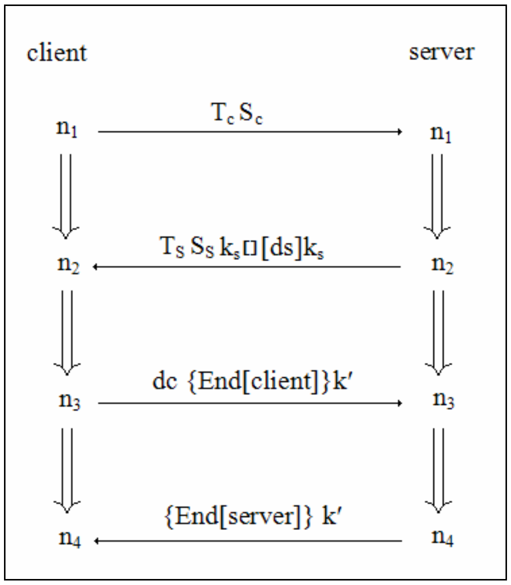


Figure 2 TLS Server Authentication

The server either recognizes the old session ID and executes the Resume Session protocol, or it continues on. If the server continues with the Server Authentication protocol, the server responds with T_s to specify the protocols and parameters to be used for the remainder of the exchange. The server also sends the session ID, S_S, to be used as a reference for this connection. T_s is assumed to specify DH.

Additionally, the server replies with a certificate chain, ks[], of SPKI certificates detailing its identity. Each link of the chain is a certificate that binds the server's public key to a name or authorization. The chain in its entirety, ks[], is a logical implication of certificates proving the server's identity from a client-verifiable source.

Furthermore, the fresh DH values are signed with the server's private key and presented to the client to negotiate a shared secret. These exchanges represent the ServerHello, Certificate, ServerKeyExchange and ServerHelloDone messages defined in the TLS standard.

The third message is the unsigned DH response, dc, and a hashed message digest, End[client]. The digest is signed with the newly calculated shared secret k'. This exchange represents the ClientKeyExchange, ChangeCipherSpec, and ClientFinished messages.

The final exchange returns a hashed digest, End[server], that allows the client to verify the details of the exchange were done according to the standard and that there have been no alterations.

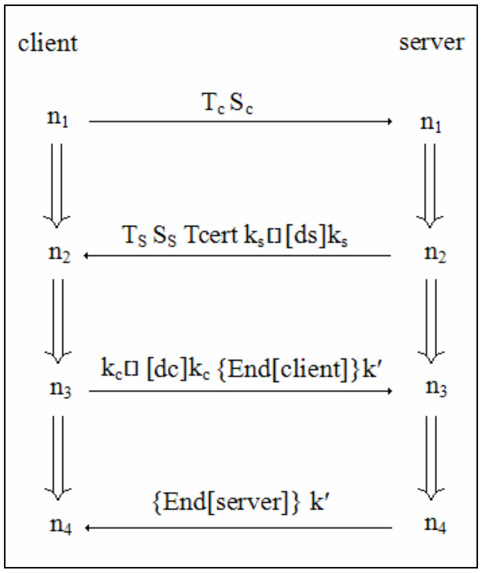
The strand space traces of the principals in this protocol, Client and Server, are provided in Table 4. They are identical with the exception that their signs are reversed.

Table 4 Server Authentication Protocol Principals

Principal Signature	**Strand Trace**
Client[T_c, S_c, T_s, S_s, ks[], ds, dc]	$\langle$+ T_c, S_c, -$T_sS_s$ks[][ds]k_s, +dc{End[]}k', -{End[]}k' $\rangle$
Server[T_c, S_c, T_s, S_s, ks[], ds, dc]	$\langle$– T_c, S_c, +$T_sS_s$ks[][ds]k_s, -dc{End[]}k', +{End[]}k' $\rangle$

4.1.2 TLS: Server & Client Authentication Protocol.

The Server & Client Authentication version, shown in Figure 3, is only subtly different from the previous protocol. The Server adds an additional request, Tcert, specifying what type of certificate must be produced by the client to complete the exchange. Furthermore, the client returns a certificate chain of its own kc[] and signs its half of the DH value with a private key. The exchange then ends just as the Server Authentication protocol does.



client
server
n1
$T_c S_c$
n1
n2
$T_S S_S$ Tcert k_s□ [ds]k_s
n2
n3
k_c□ [dc]k_c {End[client]}k′
n3
n4
{End[server]} k′
n4

Figure 3 TLS Server & Client Authentication

The strand traces for regular principals for the Server & Client Authentication protocol are provided in Table 5.

Table 5 Server and Client Authentication Protocol Principals

Principal Signature	**Strand Trace**
Client[T_c, S_c, T_s, S_s ks[], ds, kc[], dc]	$\langle$+ T_cS_c, -$T_sT_{cert}S_s$ks[][ds]k_s, +kc[][dc]k_c\{End[]\}k', -\{End[]\}k' $\rangle$
Server[T_c, S_c, T_s, S_s ks[], kc[], ds, dc]	$\langle$− T_cS_c, -$T_sT_{cert}S_s$ks[][ds]k_s, -kc[][dc]k_c\{End[]\}k', +\{End[]\}k' $\rangle$

4.1.3 TLS: Resume Session Protocol.

The final TLS protocol being analyzed is the Resume Session protocol. Unlike the other two, this protocol can only be executed following a successful execution of either the Server Authentication or Server & Client Authentication protocol. Instead of establishing a new session ID, this exchange restarts a session based on a previously negotiated secret key. The ability for the server to decline a resume request and the hash message digest End[] are the controls of this protocol to prevent abuse and limit feasibility of attacks. The strand space trace is depicted in Figure 4 and principal traces in Table 6.

Table 6 Resume Session Protocol Principals

Principal Signature	**Strand Trace**
Client[T_c, S_c, T_s, S_s]	$\langle + T_c, S_c, -T_sS_s\{End[]\}k', +\{End[]\}k' \rangle$
Server[T_c, S_c, T_s, S_s]	$\langle - T_c, S_c, +T_sS_s\{End[]\}k', -\{End[]\}k' \rangle$

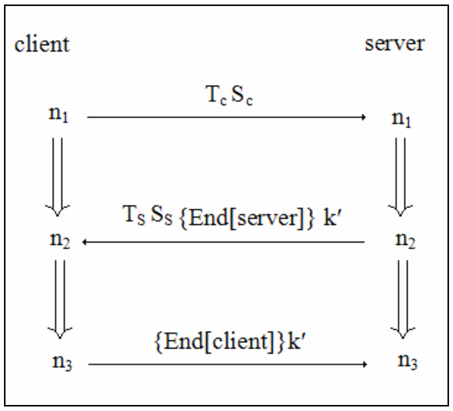


Figure 4 Resume Session

The End[] function warrants further explanation. First, as mentioned above, it is a symbolic representation of the final messages sent between participants in the TLS protocol. The actual composition of these messages varies depending on the negotiated parameters; however, the properties remain the same. Each is constructed using a mix of pseudo-random functions, MD5, and SHA hashes. It is assumed that these digest values uniquely originate at their source node. A message produced by a server is constructed with a label indicating as such, whereas a message created by a client is also appropriately labeled before the hashing takes place to prevent the confusion, replay attack or replacement of one instance with another.

The uniqueness of each instance of the End[] function can be illustrated with the composition of its inputs. An example of a set of inputs would include the generated session key k', a specification of what role the message is being sent under (client or server), then an ordered array of previously sent messages history[] and an array of previously sent secrets secrets[]. The result of this function is a uniquely originating value N_x that is unpredictable to a penetrator but reproducible by a principal with all the correct information. The input and outputs of the End[] function are illustrated in Table 7.

Table 7 End[] Function Inputs for Server & Client Authentication Protocol

Message	Input	History[]	Secrets[]	Output
3	k', client, history[], secrets[]	T_sS_s, T_sS_sTcert ks[][ds] k_s	k'	N_0
4	k', server, history[], secrets[]	T_sS_s, T_sS_sTcert ks[][ds]k_s, kc[][dc] $k_c\{ N_0\}k'$	k', N_0	N_1
x	k', <role> history[] secrets	T_sS_s, T_sS_sTcert ks[][ds]k_s, kc[][dc] $k_c\{ N_0\}k'$ … …$\{N_{x-1}\}k'$	k', N_0, … … N_{x-1}	N_x

The aim of the End function is to provide a unique, secure, non-reversible digest. A received digest can then be compared to a self-generated one in order to confirm all the messages and secrets shared between two principals. Although there have been recent concerns with the MD5 hash (25; 26), it is still considered to be a safe means of creating digests.

4.2 SPKI Integration into TLS

During the analysis, reference to TLS will imply a reference to the abstract Handshake protocol as described below. The SAP and SCAP are assumed to be the base protocols and will be executed using the Diffie-Hellman key agreement algorithm using SPKI name certificates in place of X.509 certificates.

TLS currently uses X.509 certificates, which are functionally identical to SPKI name certificates. Consequently, SPKI certifications can be easily substituted (27). Furthermore, the strand space theory is ideal for the analysis of public key protocols and in particular to the Diffie-Hellman key agreement protocol (17). The integrity of TLS executing with sub-protocols is a concern; however, it fits the model of the mixed strand space formalism defined in (28) and can be analyzed in a mixed strand space.

4.2.1 Certificate Chain Validation Protocol Design.

In the standard TLS, there is no way for the client to specify what type of certificate he requires the server to supply for authentication. In practice, this is accommodated by the use of global or far-reaching certificate authorities that are for the most part universally recognized. In an SPKI TLS application, an unprompted certificate chain may end at a certificate authority the client has no knowledge of. To accommodate this problem, it will be assumed that the initial message exchange used to coordinate which protocols to use will also coordinate a certificate base for the server to provide a certificate chain. For example, the standard ClientHello message contains two arrays specifying cipher suites and compression methods (23:34-35). To accommodate SPKI TLS, a third array of high level authorities can be included to identify which sources of trust this client will work with. Thus, the server still retains the option of choosing how the connection is made, what methods of security to implement and in addition can decide if it can provide a certificate chain to any of the client's known authorities. Adding this certificate chain to this analysis is hidden by the degree of abstraction; however, if it were explicitly represented it would appear in the first message sent from the client concatenated with T_c (Figure 3).

Furthermore, the standard TLS protocol assumes that certificates can be validated offline. However, realistically, offline validation introduces security concerns in the form of stale revocation lists and compromised keys. To overcome these problems, certificates can be checked online, thus eliminating delays in revocation and validation. TLS can use SPKI to solve this problem if it is augmented to accommodate the frequent validation of certificates via the SPKI hierarchy of certificate authorities. This will necessarily include the addition of a protocol between a client and an authority which may need to execute as a sub-protocol. The inclusion of this protocol will allow TLS to accommodate online validation. The SPKI standard currently has no protocol specification for certificate chain validation, and thus one is developed here and will be integrated into the TLS model used in analysis.

The term *certificate chain discovery* appears in several contexts with regard to SPKI and thus it is important to distinguish them. First, the SPKI standard provides a tractable algorithm called Certificate Chain Discovery for a principal to sift through owned certificates in order to provide a minimal certificate chain. The second use of this term is in the TLS standard. A principal uses certificate chains, similar to those of SPKI, to trace a certificate from its CA to the principal providing it. In order to avoid confusion, the protocol designed here for certificate chain discovery will be called the Certificate Chain Validation protocol (CCV).

CCV requires two regular participants, a solicitor and an authority. It is assumed that the authority is an issuer of certificates, whereas the solicitor is providing a certificate to be verified.

The solicitor requires the authority to be authenticated. By authenticating the authority, the solicitor is assured of the authority's identity and hence assured of that authority's response to the validity of the certificates. It is assumed that a regular principal playing the role of the authority will respond only to certificates it has created and will respond only with accurate assessments of presented certificates.

The following procedure uses authentication tests as a guide to protocol design (29). To satisfy the security goals of the CCV protocol, the solicitor provides an incoming test for the authority to validate. The easiest method of supplying a test is to provide a certificate verifiable by the authority along with a nonce N_s to be returned signed with the authority's private key. If the set of keys used in CCV is not known to the penetrator and the incoming test is returned then, this test verifies the certificate. By Theorem 2 it is deduced that only a regular participant could have returned a signed certificate validation. Based on the assumed behavior of regular principals, only a certificate authority would sign and return such a value. Furthermore, the only certificate authority with the nonce N_s and access to the private key corresponding to the certificate being validated is the principal acting as the authority. Therefore, the certificate is validated by the authority and the solicitor can continue with the remaining certificates until the chain is completely verified. The strand space representation of the CCV protocol is depicted in Figure 5 and the principal traces are in Table 8.

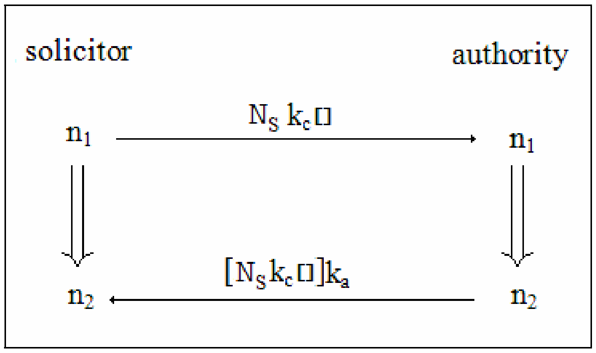


Figure 5 Certificate Chain Validation Protocol

Table 8 Certificate Chain Validation Protocol Principals

Principal Signature	**Strand Trace**
Solicitor[N_S kc[]]	$\langle + N_S\ kc[], - [N_S\ kc[]]k_a \rangle$
Authority[N_S kc[]]	$\langle - N_S\ kc[], + [N_S\ kc[]]k_a \rangle$

4.3 TLS Strand Space Analysis

Analysis of the TLS strand space proceeds as follows. Strand space proofs are applied to the base protocols of TLS in isolation to prove their security properties. Next, the sub-protocols operating under the base model are proven to have protocol independence. This allows the conclusion that all security properties of the TLS protocol are maintained even when operating with the CCV and Resume Session protocols.

The theorems below prove authentication for the TLS base protocols, Server Authentication Protocol and Server & Client Authentication protocol. All of the following proofs assume the conditions listed in Table 9.

Table 9 Strand space assumptions

A bundle $\mathcal{C}$ over the TLS strand space
Signed messages and encryption form a disjoint set to the set of keys used in the protocol
The set of keys known to the penetrator $K_{\mathcal{P}}$ is disjoint from the set of keys K
The set of penetrator known DH values is disjoint from those used in the protocol
Hashing functions are chosen such that they are computationally infeasible to find two distinct inputs which hash to a common value and if given a hash value, it is similarly infeasible to predict the inputs (30)
The Diffie-Hellman problem is hard

Theorem 1 Origination of Cryptographic Terms

If $k \in K$, and for all nodes in $\mathcal{C}$, k is never a sub-term of that node, then any term $h \in A$ signed or encrypted with k must originate on a regular strand.

Proof: If $\{h\}k$ or $[h]k$ originate on an adversary strand, they must originate on either an E or σ strand, as they are the only strands that create new cryptographic terms. For these strands to do this, $k \in K_{\mathcal{P}}$. However, this contradicts the assumption that $K_{\mathcal{P}}$ is disjoint from the set K.

Theorem 2 Signature Origination

If $k \in K$, and for all nodes in $\mathcal{C}$, k is never a sub-term of that node, then for any $[h]k \in A$, if $[h]k$ originates on any node then there is exactly one regular principal that created that value, and furthermore the principal associated with the originating strand corresponds to the public key used to read h.

Proof: Based on Theorem 1, $[h]k$ must originate on a regular strand. Furthermore, only regular strands have access to their own private keys. Thus, since each principal is represented uniquely by the inverse of their private key, any term signed is guaranteed to come from exactly one principal strand in the strand space.

4.3.1 Isolated TLS.

Theorem 3 SAP Authentication

Let $\mathcal{C}$ be a bundle consisting of SAP Client, SAP Server, and penetrator strands. Assume that k' is not known to the penetrator. Then, if $\mathcal{C}$ contains some strand Client[T_c, S_c, T_s, S_s, ks[], ds, dc] then $\mathcal{C}$ must also contain a strand Server[T_c, S_c, T_s, S_s, ks[], ds, dc] with a height of four whose identity is associated with the public key k_s. Furthermore, the server's actions are explicitly in response to the client's and all messages of the protocol are received un-altered.

Proof: The second message of the protocol contains a signed DH constructor. Based on Theorem 2, the reception of this term by the client assures the client that this DH constructor was created by the server whose public key is k_s. Knowing that this was created by that particular server, however, is not sufficient: it must also be proven that the message was produced with the intent to continue this particular run of the protocol. This is accomplished with an authentication test.

The edge connecting n_2 and n_3 of the client strand is an outgoing test for the value of the End[] function, N_0. The test component for this authentication test is thus $\{N_0\}k'$. Since the DH problem is hard, the only two principals capable of creating the key k' are those participating in the exchange. Furthermore, because N_0 is an ingredient of k', the only two principals capable of creating N_0, are also those participating in the exchange. Since regular principals act according to the protocol and according to a regular protocol run, N_0 is only contained in this encrypted message. Therefore, it can be deduced that N_0 uniquely originates at node n_3 of the client strand.

Additionally, the test component $\{N_0\}k'$ never appears as a proper sub-term of any regular component. Assume for a moment that $\{N_0\}k'$ is a proper sub-term of test components which use it as an ingredient. As a sub-term, $\{N_0\}k'$ is obtainable through some combination of decryption, separation, joining or concatenation. In the SAP protocol, the only use of $\{N_0\}k'$ is as an ingredient to the hash functions. Thus, to obtain this component, the inverse of the End[] function must be calculated. However, this contradicts the assumption that the hash functions used are probabilistically unpredictable and irreversible. Therefore, $\{N_0\}k'$ is not a proper sub-term of any node in the bundle.

To fulfill the outgoing test, the server must decipher the message and return N_0 in a context outside of the encryption it was received in, i.e. the server must provide a transforming edge on the test component. This is accomplished in message four sent by the server between nodes n_3 and n_4. The new context is provided by the End[] function result N_1. Using N_0 as an ingredient for N_1, it is presented in a new context only producible by principals who know the key k'. By returning N_1, the server proves that he has deciphered the message and created a response to the test component recently.

To review, the original test component of message three is never the sub-term of a previous node, its sub-terms uniquely originate on the node sending it and the sub-term N_0 is returned in a new context outside of the sent component. Therefore, this component qualifies as a test component, nodes n_3 to n_4 represent an outgoing test and thus the client authenticates the server as not only the server associated with the key $k_{s,}$ but also the other participant in this particular protocol execution.

Furthermore, since regular participants only act according to protocol specifications, the only server node that produces a message of this composition is n_4. Therefore the server strand must have a height of at least four. Additionally, because the history array used to build N_1 includes all previously sent messages and is built with secret values only known by the participants of this protocol, the client is provided with the guarantee that all messages sent and received have not been altered by a third party. If the messages where altered, the verification of hash values would reveal this and the protocol would not continue.

Thus, the SAP protocol assures the client of three things: first, the server's identity is that associated with k_s; second, that server is executing this protocol in response to the client's requests and not some other run of the protocol; and third, all four messages are received un-altered by any third party.

Theorem 4 SCAP Server Authentication

Let $\mathcal{C}$ be a bundle consisting of SCAP Client, SCAP Server, and penetrator strands. Then, if $\mathcal{C}$ contains some strand Client[T_c, S_c, T_s, S_s, ks[], ds, kc[], dc] then $\mathcal{C}$ must also contain a strand Server[T_c, S_c, T_s, S_s, ks[], kc[], ds, dc] with a height of four whose identity is associated with the public key k_s. Furthermore, the server's actions are explicitly in response to the client's and all messages of the protocol are received un-altered.

Proof: The proof for this theorem is analogous to that of Theorem 3. The only cryptographic difference in the SCAP and SAP protocols is that the DH constructor provided by the client is now a signed term. However, that signed term only becomes important when proving client authentication.

To begin, the signed DH constructor supplied by the server provides the client with sufficient proof that the sender of that term was the server associated with the key k_s via Theorem 2.

Similarly, $\{N_0\}k'$ provides an authentication test for the server. Through the same logic as before, this component is never the sub-term of a previous node, its sub-terms uniquely originate on the node sending it and the sub-term N0 is returned in a new context outside the sent component. Consequently, the client can authenticate the server as participating in this particular execution of the protocol and as the particular server associated with k_s.

Once again the composition of the message digest provides a guarantee of message agreement. Thus, the SCAP protocol assures the client that the server is k_s, is running this protocol in response to the client's messages, and all messages have been received un-altered.

Because the SCAP protocol is designed to provide mutual authentication it is also necessary to prove that SCAP authenticates the existence of a particular client given a server strand.

Theorem 5 SCAP Client Authentication

Let $\mathcal{C}$ be a bundle consisting of SCAP Client, SCAP Server and penetrator strands. Assume that k' is not known to the penetrator. Then, if $\mathcal{C}$ contains some strand Server[T_c, S_c, T_s, S_s, ks[], kc[], ds, dc], then $\mathcal{C}$ must also contain a strand Client[T_c, S_c, T_s, S_s, ks[], ds, kc[], dc] of height at least three.

Proof: Theorem 2 allows the server to identify the DH constructor supplied in message three as an incoming value provided at some previous time by the client associated with the public key k_c. However, authentication once again must be deduced from an authentication test via a test component.

The authentication takes place between nodes n_2 and n_3 of the server strand and is an incoming test. In this case the test value a is message digest N_0.

The test value N_0 encrypted with k' is the test component that fulfills the incoming authentication test. To qualify as a test component, $\{N_0\}k'$ must not be the proper sub-term of any other term in the bundle and must contain a uniquely originating value as a sub-term. Using the same logic as in Theorems 3 and 4 this component is not a proper sub-term of any term in $\mathcal{C}$.

To show that N_0 is a uniquely originating value, recall that the ingredients of N_0 include the generated secret key k'. Because the DH problem is hard, the only principals that know this key are the regular principals executing this protocol. Therefore, the generation of the digest using this key is a unique value. Furthermore, since regular principals act only according to the protocol standard, through inspection of the strand trace it can be seen that only the client strand will create the message digest N_0. Thus, the test value N_0 is a uniquely originating value in the strand space.

Additionally, for the same reasons as shown in previous theorems, N_0 also provides agreement on all previously sent messages. Thus, the server is assured the client is that associated with the key k_c, is responding to this particular execution of the protocol, agrees with the first three messages of the SCAP protocol and is of height at least three.

As an interesting side note, despite the strong agreement on the first three messages the server has no guarantee the client receives the last message. Because there is no response to the final message, let alone one including a means of authentication, it is impossible for the server to know if the client received the last message. This, however, does nothing to weaken the authentication of the previous messages and is more of an inconvenience peculiar to the sender of the final message of a protocol.

4.3.2 Isolated Resume.

Theorem 6 Resume Server Authentication

Let $\mathcal{C}$ be a bundle consisting of Resume Session client, server and penetrator strands in addition to SCAP or SAP strands. If $\mathcal{C}$ contains some client strand Client[T_c, S_c, T_s, S_s] then $\mathcal{C}$ must also contain a strand Server[T_c, S_c, T_s, S_s] of height at least two that agrees on all messages of a particular SCAP or SAP protocol.

Proof: The edge connecting nodes n_1 and n_2 acts as an incoming authentication test to authenticate the server to the client. The test value is the result of the message digest N_{x+1}, where x is the number of previously sent message digests from the protocol session being resumed. The test component for this test is the encrypted portion of message 2, $\{N_{x+1}\}k'$. To be a test component, it cannot by a proper sub-term of any other messages in $\mathcal{C}$ and must contain a value a that uniquely originates on the n_1.

Assume $\{N_{x+1}\}k'$ is a proper sub-term of a message in $\mathcal{C}$. Through observation it is clear that this component is not included in any other message other than as an ingredient. In particular as an ingredient in message digest N_{x+2}. To extract this from the digest the DH key k' must be compromised and the digest must be reversed. However, this contradicts the assumption that the DH problem is hard and that the hash functions are irreversible. Thus $\{N_{x+1}\}k'$ is not a proper sub-term of any other messages in $\mathcal{C}$.

The node n_1 does not supply explicitly a uniquely originating value, however, it does provide a constructor such a value. Message one of the protocol provides an additional entry to the history[] array used to compose N_{x+1}. Based on the assumption that hash functions are unpredictable N_{x+1} must be a uniquely originating value.

Therefore, the component $\{N_{x+1}\}k'$ is a test component for the incoming test over the nodes n_1 and n_2 of the client strand and nodes n_1 and n_2 of the server strand are the transforming edge which satisfy the test. This allows the client to conclude that this execution of the Resume protocol is in response to the messages the client initiated.

A validated message digest, as discussed in previous theorems, indicates agreement on all previously sent messages. In this case a message digest will be built with the messages from a previous SCAP or SAP session. Because both of these protocols authenticate the server, the successful execution of either reveals to the client the identity of the server, k_s, via Theorem 3 or Theorem 4 respectively.

Thus, given a client strand, there must exist a server strand of at least height two that is participating in this particular execution of the protocol, is authenticated as the server corresponding to the public key k_s of the resumed session and agrees on all previously sent messages of the same resumed session.

Theorem 7 Resume Client Authentication.

Let $\mathcal{C}$ be a bundle consisting of Resume Session client, server and penetrator strands in addition to either SCAP or SAP strands containing a complete run of the respective protocol. If $\mathcal{C}$ contains some server strand Server[T_c, S_c, T_s, S_s] then $\mathcal{C}$ must also contain some strand Client[T_c, S_c, T_s, S_s] that agrees on all messages of resumed protocol.

Proof: This proof is very similar to that of Theorem 3. The test being accomplished is an outgoing test over the nodes n_2 and n_3. The test value is N_{x+1} and the test component is $\{N_{x+1}\}k'$, where x is the number of previously sent message digests from the protocol session being resumed. Based on the same arguments for the message digest in Theorem 3, this value is not a proper sub-term and contains a uniquely originating value in the digest $N_{x+1.}$

The corresponding client strand must extract this value and return it in a new context via a transforming edge. This is done in message three between nodes n_2 and n_3 of the client strand. The new context is provided by the digest N_{x+2}. Because this value uses the test value as an ingredient and is encrypted with the shared secret key k', the client proves that he has deciphered the message and created an appropriate response to the test. Therefore, this component authenticates the client as the principal executing the other half of this particular protocol run.

The identity of the client, however, is a more delicate matter. Through inspection of the history array used in constructing message digests, the server can search for a client supplied DH constructor. If the included strands are SCAP strands, then the DH constructor will be signed with the key k_c. In this case, the server can authenticate not only that the client is running this particular session of the protocol but also that the client's identity is that associated with k_c. On the other hand, if the strands are a SAP execution, then the DH constructor is not signed and the best the server can do is to guarantee that this execution is being accomplished with the same client as the resumed protocol, but cannot authenticate the identity of that client.

4.3.3 Isolated CCV.

Theorem 8 CCV Authentication.

Let $\mathcal{C}$ be a bundle consisting of CCV Solicitor, CCV Authority and penetrator strands then, if $\mathcal{C}$ contains some strand Solicitor[N_S kc[]], then $\mathcal{C}$ must also contain a strand Authority [N_S kc[]] and the assessment of the link of the certificate chain, kc[], is accurate.

Proof: This protocol was built with an authentication test in mind and thus is a relatively straight forward proof. The test value is the uniquely originating nonce Ns supplied in message one of the protocol. It is assumed that the solicitor can provide an unpredictable nonce N_s such that the value is uniquely originating. The authentication test is an incoming test over the edges n_1 and n_2. The test component returned is the entire second message of the protocol $\{N_s\text{kc}[]\}k_a$. Furthermore, due to the simplicity of the protocol from inspection it can be seen that no other messages contain the component $\{N_s\text{kc}[]\}k_a$ as a sub-term. Thus, the component qualifies as a test component because it includes a uniquely originating value and is not a sub-term of any other message in $\mathcal{C}$.

In order to complete this incoming test, the authority must provide a transforming edge. The new value N_s is received in the clear-text of message one. It is placed in a new cryptographic context within the test component thus satisfying the transforming edge and completing the authentication test. Theorem 2 applied to message two identifies the authority as the particular principal associated with $k_{a.}$

Therefore, the solicitor, through an incoming test, authenticates the authority as running this protocol in response to the protocol initiation by the solicitor and the authority is the particular principal associated with the key k_a. Recall the assumption originally made in Section 4.2.1 that a regular principal playing the role of the authority will respond only to certificates it has created and will respond only with accurate assessments of presented certificates. Based on this assumption, the assessment of this link of the certificate chain is valid. Thus, given a solicitor strand executing the CCV protocol, there exists an authority k_a that corresponds with that particular execution of the protocol and provides a valid certificate assessment.

4.4 TLS Protocol Independence

To prove protocol independence the base and the sub-protocols must be compared against the properties required for disjoint encryption. The two base protocols will be the SAP and the SCAP protocols. To prove that each protocol execution is independent, it is required to prove that the two base protocols are independent of each sub-protocol. Once these relationships are proven to be independent, it can be concluded that the TLS protocol running CCV and Resume protocols have the same security properties as the TLS protocol running in isolation.

4.4.1 SAP, Resume & CCV.

To prove that the SAP-Resume-CCV strand space retains the security properties of the SAP strand space, it must be shown that all secondary strands exhibit disjoint encryption with SAP. The set of secondary strands Σ_2 consists of a proper subset of the strands from the Resume (Σ_{res}) and CCV (Σ_{ccv}) protocols.

To begin, the CCV protocol was designed to provide the needed security assurances while avoiding potential problems with the SAP protocol. Consequently, it shares no terms either as sub-terms or terms with the same signature as those in the SAP protocol. Similarly, it does not originate any texts or keys vital to the privacy of the primary protocol. That is to say, there are no encrypted terms t such that $t \sqsubset \text{term}(n_1)$ and $t \sqsubset \text{term}(n_2)$, for $n_1 \in \Sigma_1$ and $n_2 \in \Sigma_{\text{ccv}}$. The same is true for disjoint inbound encryption. Therefore, the CCV exhibits disjoint encryption with the SAP.

To complete the proof, the set of strands of the Resume Session protocol must also be proven to be disjoint with the SAP. No new components of the Resume Session protocol contain, as sub-terms, private values from the SAP protocol. The construction of the message digest uses private values, however, they are not retrievable from the components, and thus not considered sub-terms. Therefore, there is no positive node in Σ_{res} that uses private values as sub-term of new components, therefore, the Resume Session protocol exhibits disjoint outbound encryption with respect to SAP.

For disjoint inbound encryption there cannot be a node such that a private encryption occurs as a new component on Σ_{res} that also occurs on the SAP. The only encrypted component generated by the Resume Session protocol is $\{\text{End[server]}\}k'$. Although this component matches the signature of components found in the primary protocol, i.e. in messages three and four, the nature of the End function means that this is a new component. That is to say that this is a uniquely originating component that is distinguishable from all other occurrences of the message $\{\text{End[server]}\}k'$. Consequently, the Resume Session protocol exhibits disjoint outbound encryption.

Since the Resume Session protocol exhibits both inbound and outbound disjoint encryption it also exhibits disjoint encryption. Furthermore because all subsets of Σ_2 exhibit disjoint encryption with SAP, Σ_2 exhibits disjoint encryption with SAP.

Finally, since no atomic text or key originates on secondary nodes of the mixed strand space, there is an equivalent strand space which has an arbitrary amount of M and K strands. Since these values are incapable of interacting with the Σ_2, the strand space is said to be full. Therefore, all protocols of the mixed SAP strand space are considered to be independent. Thus, all properties of these protocols are preserved when executing the SAP with Resume Session and CCV as sub-protocols.

4.4.2 SCAP, Resume & CCV.

The proof of protocol independence for the SCAP is identical to that of the SAP proof. This is provable by identifying the difference in terms of the SAP and SCAP protocols and showing that these provide no new terms that interfere with disjoint encryption in the sub-protocols.

Table 10 Term differences in TLS base protocols

New Term in SCAP	**Corresponding term in SAP**
Tcert	-
kc[]	-
[dc]kc	dc

The terms which differ in the SCAP and the SCAP are identified in Table 10. None of these terms are secret values and they share no common signatures with the sub-protocols Resume Session and CCV. Therefore, Σ_2 exhibits disjoint outbound encryption. Similarly, since none of these are private encryptions, there cannot be a new component in Σ_2 that produces a private encryption that is in this set. Therefore Σ_2 also exhibits disjoint inbound encryption. Furthermore, the set of secondary strands Σ_2

exhibits disjoint encryption to the SCAP. The same reasoning as above allows this mixed strand space to be considered full. Finally, a full strand space with disjoint encryption exhibits protocol independence. Consequently, the SCAP protocol is independent of the Resume Session and CCV protocols and thus the mixed strand space that contains all of these protocols exhibits the same properties as any of the protocols in isolation.

V. Final Words

5.1 Summary

The goal of this work was to assemble a method to integrate the Simple Public Key Infrastructure (SPKI) into an application and provide security analysis of the result. Strand space authentication tests were used to provide a guide to SPKI application design. The authentication tests supplied a well-grounded formal method of protocol generation and analysis. The result was that the security properties of the new protocol designed were direct implications of the security goals.

Overall, strand space was found to be an excellent tool for SPKI analysis because it provided an explicit model of PKI protocol environments as well as a means for analyzing the protocols interacting with one another. In unison, these properties are an ideal model for SPKI protocol design and analysis.

As a demonstration of the assembled model, SPKI was integrated into the Transport Layer Security protocol. The TLS protocol was chosen due to its popularity, its security goals, and its reliance on a PKI. The two TLS protocols examined where the Server Authentication Protocol and the Server & Client Authentication protocol. As anticipated, each was proven in isolation to provide authentication to their respective principals.

The strand space based method of authentication tests as a means of protocol design furthered the example by providing a way to integrate SPKI functionality into the

TLS protocol. Authentication tests were used to create the Certificate Chain Validation protocol used to validate certificates being used by the TLS protocol.

Furthermore, utilizing the ability of strand space to analyze multiple protocols operating in the same environment TLS was proven to be independent of its sub-protocols. This then led to the proofs that TLS running its own Session Resume protocol and the newly created CCV protocol has the same security properties as TLS operating in isolation.

5.2 Future Work

5.2.1 Security Policy Design.

A key feature of SPKI is its ability to delegate authority for access control, certificate distribution or any other definable authorization. Due to this flexibility, there is the additional concern of security policy design. When is it reasonable to delegate an authorization? Furthermore, is there an ideal method for developing the hierarchy of delegation? Perhaps an inductive proof method can be derived to establish a logic for building the desired hierarchy or possibly a means of recursively building and checking the tree during policy design and certificate distribution.

5.2.3 Performance Based Analysis.

Using SPKI in place of other public key technologies may result in additional overhead. One such overhead could be excessive network traffic. In the TLS example explored above, execution of the certificate chain validation protocol will result in network traffic. If each principal of a network is required to check a dozen certificates per interaction the network traffic may build. A model of this traffic and the demands it places on a network could be a valid topic of future SPKI research.

5.2.4 Hardware Demands.

Since the SPKI standard allows for principals to create their own keys, will this place further demands on the hardware necessary for SPKI applications, and if so to what extent? If a central key authority is to issue keys and certificates, what will be the network congestion as a result of requests to verify generated certificates or keys?

5.2 Conclusions

SPKI is an intriguing and flexible standard that provides an excellent framework for PKC. Due to the brevity of its specification, a good deal of care must be taken in application integration. However, if attention is paid to an application's desired security goals SPKI can offer a provable and flexible solution.

The strand space theory has proven to be an invaluable tool both in the analysis of protocols and their design. Strand space graphs are a clear and concise means of representing both simple and obscure protocols. Furthermore, the inductive nature of strand space proofs provide not only analysis of protocol correctness, but also reasoning why a protocol may fail. This concept can then be carried into the analysis and construction of new protocols to avoid the same security problems in the future.

Appendix A: Strand Space Protocol Example

The following is the strand space representation of the Simple Protocol. The Simple Protocol consists of two brothers, Cyril and Dmitri passing information to and from one another. Dmitri begins by asking Cyril to send him the location of their secret hideout (M_1). Cyril responds by encrypting a message detailing where the hideout is using Dmitri's public key, $\{M_2\}K_D$. Dmitri in turn encrypts a new message telling Cyril what to bring to the hideout at their next meeting $\{M_3\}K_C$. The graphical representation of this protocol in strand space is in Figure 6.

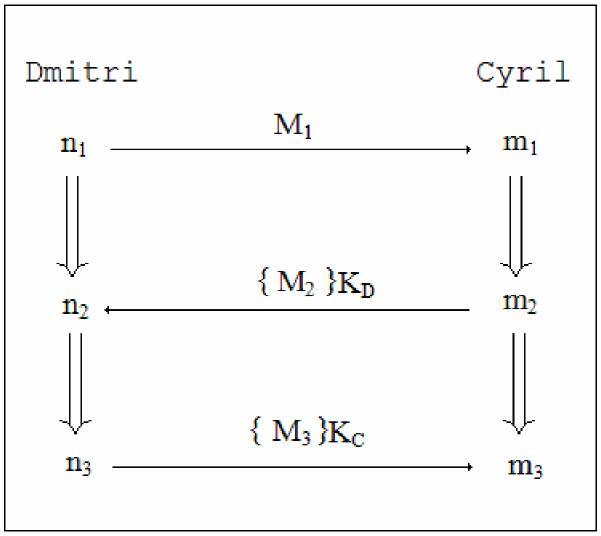


Figure 6 the Simple Protocol

Nodes n_1 and m_1 both share the same term M_1. However, n_1 is a positive (sending) node and m_1 is a negative (receiving) node. The vertical path from n_1 to n_3 represents the strand of Dmitri. The edge between n_2 and n_3 represents Dmitri deciphering Cyril's encryption extracting the information and enciphering his own message to send off. The actual act of sending the messages is captures along the edges between the *n* and *m* nodes.

Bibliography

1 Diffie, W. and M. E. Hellman “Multiuser Cryptographic Techniques,” *Proceedings of AFIPS National Computer Conference*. 109-112. Montvale, NJ: AFIPS Press, 1976.

2 Thayer, Javier, Jonathan Herzog and Joshua Guttman. “Strand Spaces: Proving Security Protocols Correct,” *The Journal of Computer Security*, 7: 191-230 (1999).

3 Arkin, Brad, Scott Stender and Gary McGraw. “Building Security in: Software Penetration Testing,” *IEEE Computer Society Security and Privacy Magazine*, Vol 3 No. 1, 84-87 (January-February 2005).

4 Dam, Kenneth W and Herbert S. Lin. *Cryptography’s Role in Securing the Information Society*. Washington D.C.: National Academy Press, 1996.

5 Asokan, N., Niemi Valterri and Kaisa Nyberg. *Man-in-the-middle in tunneled authentication protocols*: Technical Report 2002. Finland: Nokia Research Center, 11 November 2002.

6 Shoup, Victor *A Computational Introduction to Number Theory and Algebra* Cambridge University Press, 2005.

7 Benantar, M. “The Internet Public Key Infrastructure,” *IBM Systems Journal*, Vol 40: Num 3, 2001.

8 Ellison, Carl. “SPKI Requirements.” Request for Comments 2692 to Internet Engineering Task Force, Network Working Group, Sept 1999.

9 Housley, R, W. Ford, W. Polk and D. Solo. “Internet X.509 Public Key Infrastructure Certificate and CRL Profile.” Request for Comments 2459 to Internet Engineering Task Force, Network Working Group, Jan 1999.

10 Gerck, E. “Overview of Certification Systems: X.509, CA, PGP and SKIP,” *Proceedings of Black Hat Briefings 1999*, July 1999.

11 Ellison, Carl, B. Frantz, B Lampson, R. Rivest, B. Thomas and T. Ylonen. “SPKI Certificate Theory.” Request for Comments 2693 to Internet Engineering Task Force, Network Working Group, Sept 1999.

12 Clarke, Dwain. *SPKI/SDSI HTTP Server/Certificate Chain Validation in SPKI/SDSI*. MS thesis Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, September 2001.

13 Clarke, Dwain, J-E. Elien, C. Ellison, M. Fredette, A. Morcos, R. Rivest. “Certificate chain Validation in SPKI/SDSI.” *Journal of Computer Security*. 9:4 285-322 (July 2001).

14 Elien, Jean-Emile. *Certificate Validation Using SPKI/SDSI 2.0 Certificates*. MS thesis Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, May 1998.

15 Dolev D and A.C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, Vol. IT-29, No. 2:198-208 (March 1983).

16 Clark, John and Jeremy Jacob. *A Survey of Authentication Protocol Literature: Version* 1.0. Unpublished Technical Report, Department of Computer Science, University of York, UK, Nov 1997. Available at the URL: www-users.cs.york.ac.uk/~jac/papers/drareviewps.ps.

17 Herzog, Jonathan. “The Diffie-Hellman Key Agreement Scheme in the Strand-Space model,” *Proceedings of the 16th IEEE Computer Security Foundations Workshop*. IEEE CS Press, June 2003.

18 Guttman, Joshua and Javier Thayer. “Authentication Tests,” *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

19 Ron Rivest. “The MD5 Message-Digest Algorithm.” Request for Comments 1321 to Internet Engineering Task Force, Network Working Group. April 1992.

20 National Institute of Standards and Technology. *Secure Hash Standard*. PUB 180-1. Federal Information Processing Standards Publications, May 1993.

21 Guttman, Joshua and Javier Thayer. “Protocol Independence through Disjoint Encryption” *Proceedings of the 13th IEEE Computer Security Foundations Workshop*. Cambridge MA, July 2000.

22 Portmann, Marius and Aruna Seneviratne. “Selective Security for TLS,” *Proceedings of the Ninth IEEE International Conference on Networks.* IEEE CS Press, October 2001.

23 Dierks, T. and C. Allen, “The TLS Protocol Version 1.0.” Request for Comments 2246 to Internet Engineering Task Force, Network Working Group, Jan 1999.

24 Yasinsac, Alec and Justin Childs. “Analyzing Internet Security Protocols,” *Proceedings of the Sixth IEEE International Symposium on High Assurance Systems Engineering*, October 2001.

25 Kaminsky, Dan. “MD5 To Be Considered Harmful Someday.” Unpublished Report Avaya Inc., 6 December 2004.

26 Wang, Xiaoyun, Dengguo Feng, Xuejia Lai and Hongbo Yu, *Collisions for hash functions md4, md5, HAVAL-128 and RIPEMD*. School of Mathematics and System Science, Shandong University China, 17 August 2004.

27 Maywah, Andrew. *An Implementation of a Secure Web Client Using SPKI/SDSI Certificates*. MS thesis Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, May 2000.

28 Thayer, Javier, Jonathan Herzog and Joshua Guttman. “Mixed Strand Spaces,” *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*. Washington D.C, IEEE CS 1999.

29 Guttman, Joshua “Security Protocol Design via Authentication Tests,” *Proceedings of the Computer Security Foundations Workshop.* Cape Breton Nova Scotia, June 2002.

30 Menezes, Alfred J. and others. *Handbook of Applied Cryptography*. New York: CRC Press, 1997.

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 074-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* 21-03-2005	**2. REPORT TYPE** Master's Thesis		**3. DATES COVERED** *(From – To)* Aug 2003 – Mar 2005
4. TITLE AND SUBTITLE Simple Public Key Infrastructure Analysis Protocol Analysis and Design			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Vidergar, Alexander G., First Lieutenant, USAF			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 641 WPAFB OH 45433-8865			**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/05-07
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Mr. Sylvan Pinsky pinsky@thematrix.ncsc.mil National Security Agency (NSA/C43) Fort Meade, MD 2755-6000 Phone: 410-854-6191			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			
13. SUPPLEMENTARY NOTES			

14. ABSTRACT

Secure electronic communication is based on secrecy, authentication and authorization. One means of assuring a communication has these properties is to use Public Key Cryptography (PKC). The framework consisting of standards, protocols and instructions that make PKC usable in communication applications is called a Public Key Infrastructure (PKI). This thesis aims at proving the applicability of the Simple Public Key Infrastructure (SPKI) as a means of PKC.

The strand space approach of Guttman and Thayer is used to provide an appropriate model for analysis. A Diffie-Hellman strand space model is combined with mixed strand space proof methods for proving the correctness of multiple protocols operating in the same context. The result is the public key mixed strand space model. This model is ideal for the analysis of SPKI applications operating as sub-protocols of an implementing application.

This thesis then models the popular Internet Transport Layer Security (TLS) protocol as a public key mixed strand space model. The model includes the integration of SPKI certificates. To accommodate the functionality of SPKI, a new protocol is designed for certificate validation, the Certificate Chain Validation Protocol (CCV). The CCV protocol operates as a sub-protocol to TLS and provides online certificate validation.

The security of the TLS protocol integrated with SPKI certificates and sub-protocols is then analyzed to prove its security properties. The results show that the modified TLS protocol exhibits the same security guarantees in isolation as it does when executing its own sub-protocols and the SPKI Certificate Chain Validation protocol

15. SUBJECT TERMS
strand space, authentication tests, SPKI, SDSI, TLS, SSL, public key infrastructure

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Graham, Robert, Maj, USAF
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	62	(937) 255-3636, ext 4715 (Robert.graham@afit.edu)