The following paper was originally published in the
Proceedings of the Second USENIX Workshop on Electronic Commerce
Oakland, California, November 1996

Verifying Cryptographic Protocols for Electronic Commerce

Randall W. Lichota, Hughes
Grace L. Hammonds, AGCS, Inc.
Stephen H. Brackin, Area Systems, Inc.

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Verifying Cryptographic Protocols for Electronic Commerce

Dr. Randall W. Lichota
Hughes Technical Services Company, P.O. Box 3310, Fullerton, CA 92834-3310
lichota@stars1.hanscom.af.mil

Dr. Grace L. Hammonds,
AGCS, Inc., 91 Montvale Avenue, Stoneham, MA 02180-3616
hammonds@dockmaster.ncsc.mil

Dr. Stephen H. Brackin
Arca Systems, Inc., 303 E. Yates St., Ithaca, NY 14850
Brackin@arca.com

ABSTRACT

This paper describes the Convince toolset for detecting common errors in cryptographic protocols, protocols of the sort used in electronic commerce. We describe using Convince to analyze confidentiality, authentication, and key distribution in a recently developed protocol proposed for incorporation into a network bill-payment system, a public-key version of the Kerberos authentication protocol. Convince incorporates a "belief logic" formalism into a theorem-proving environment that automatically proves whether a protocol can meet its goals. Convince allows an analyst to model a protocol using a tool originally designed for Computer-Aided Software Engineering (CASE).

1.0 INTRODUCTION[1]

As electronic commerce on the Internet experiences explosive growth, so does the number of security protocols for safeguarding business transactions. Almost without exception, these protocols use cryptography, in the form of symmetric- and/or public-key algorithms.[2] Using encryption does not guarantee protection, though. A protocol must be free of flaws that an electronic thief can exploit. Through such devices as clever replays and modifications of messages, legitimate parties to a protocol can be tricked into thinking they are communicating with each other when they are actually communicating with the thief.

While the use of formal methods does not necessarily result in detection of all such flaws, it increases the level of confidence in protocols for electronic commerce. This paper describes an automated toolset, Convince, that facilitates the analysis of cryptographic protocols by systematically checking a number of their essential security properties.

In general, cryptographic protocols use encryption to protect the confidentiality and/or integrity of message data, and to verify the identity of (i.e., *authenticate*) one or more of the parties involved in message transfers. To confirm that each message transfer in a protocol performs its intended security functions, one must ask questions such as the following:

[1] This work has been sponsored by the Air Force Materiel Command, Electronic Systems Center/Software Center (ESC/AXS), at Hanscom AFB, MA, and funded by Rome Laboratory, through contract numbers F19628-92-C-0006 and F19628-92-C-0008.

[2] Some of the more widely publicized protocols of this type include the Secure Sockets Layer (SSL), Secure Hypertext Transfer Protocol (S-HTTP), Private Communications Technology (PCT), and Secure Electronic Payment Protocol (SEPP). [BERN96]

a. Can the sender be confident that the data being sent has the expected properties.
b. Can the sender and receiver be confident that the confidentiality and integrity of the data are preserved in transit?
c. Can the receiver be confident who sent the data?
d. Can the sender later be confident that the intended party received the data sent?

Assuming that the cryptographic algorithms used are themselves relatively "safe"[3], the answers to these questions depend on whether the parties to the protocol can *convince* themselves that the protocol provides the necessary assurances.

During the past decade, researchers have developed *belief logics* [BUR90, GON90, SYV94] that formalize inferences about what protocol parties "can be confident" of regarding authentication properties of protocols.[4] Constructing formal proofs from a belief logic thus gives a means of testing whether a protocol serves its intended functions.

Convince incorporates a belief logic into a specialized automatic theorem-proving environment. In this environment, a protocol designer or analyst uses Computer-Aided Software Engineering (CASE) tools as a front end to a formal theorem prover. Convince makes the formal verification process similar to debugging software. An analyst creates a protocol model (the "code"), specifies its associated initial conditions and goals (identifies the "code's" expected behavior), and makes incremental revisions to the model until the goals are either proved or the protocol is judged to be fatally flawed (the "code" executes correctly or is abandoned). Convince makes it possible to maximize the early detection of security-related design errors, without requiring a lot of theorem-proving expertise.

Convince's CASE-based interface is implemented using Interactive Development Environments' Software Through Pictures™ [IDE94a] system, which allows an analyst to model a protocol using a combination of familiar graphical and textual notations. Convince's proof process is implemented using a well-known Higher Order Logic (HOL) [GOR93] theorem prover.

[3] The strength of encryption algorithms is not covered by Convince.

[4] While the emphasis in belief logics is on authentication, their rules implicitly address basic aspects of confidentiality and integrity.

We used Convince to analyze aspects of confidentiality, authentication, and key distribution in a recently proposed public-key version of the Kerberos authentication protocol, which the remainder of this paper will refer to as *PK Kerberos*. The PK Kerberos protocol is a component of the NetBill system for secure electronic commerce between on-line customers and merchants of on-line goods (e.g., reports) [COX95]. This protocol is being proposed as an Internet standard [CHU96].

Within NetBill, PK Kerberos is used to establish the initial authentication between customer and merchant. Consequently, we examined this protocol from two points of view: whether it is secure for the purpose for which it is intended (providing authentication services for NetBill); and whether it is reasonable for use in more general contexts (as would be expected for an Internet standard).

This work is part of a series of efforts, begun under the Air Force's Portable, Reusable, Integrated Software Modules (PRISM) program, to identify emerging technologies that are ready to be incorporated into ongoing Air Force programs. Convince development came after the review of a Rome Laboratory research prototype, the Romulus Verification Environment [ORA94]. This review clearly established the value of protocol analysis based on belief logic, but in order to effectively interact with Romulus, the user had to have specialized knowledge of its HOL-based theorem-proving environment. We quickly recognized that the effort needed to acquire this specialized knowledge would limit user acceptance. We also considered other protocol analysis tools, described in Section 5, but each of these also had serious limitations.

The remainder of this paper is organized as follows: Section 2 gives an overview of Convince's theoretical foundation, its belief logic; Section 3 gives an overview of Convince's software components; Section 4 describes using Convince to model and analyze PK Kerberos; Section 5 gives an overview of related work; and Section 6 gives our conclusions and recommendations for future work.

2.0 CONVINCE'S BELIEF LOGIC

Like all other belief logics, the Convince belief logic grew out of the BAN logic developed by Burrows, Abadi, and Needham [BUR90]. In the BAN logic, an authentication protocol is transformed into a sequence of logical statements that are then analyzed.

Gong, Needham, and Yahalom developed another belief logic, the GNY logic, based on BAN but expressed at a lower level of abstraction [GON90]. This makes it able to identify a somewhat larger class of protocol flaws.
Gong then discovered that it is possible to specify and "verify" protocols, using the original GNY logic, that are impossible or unreasonable to implement, resulting in situations where the causality of beliefs is not preserved [GON91]. He developed conditions for excluding these "infeasible" protocols.

The Romulus prototype [ORA94] implemented part of the GNY logic, in HOL, and implemented Gong's refinement to the original GNY logic.

Brackin [BRA96a] subsequently developed a HOL implementation of the full GNY logic, including Gong's refinement, and developed logics extending this logic. One of these extensions, called BGNY, is the foundation for the Convince toolset. It covers protocols using symmetric- and public-key encryption, ordinary and key-dependent hash codes, key-exchange algorithms, multiple encryption and hash algorithms, and protocols using hash codes as keys.

At a high level, BGNY is a set of rules identifying the conditions under which protocol participants can obtain data and draw conclusions about this data and other protocol participants. While most of the BGNY rules are based on GNY, there are omissions, additions, and modifications. The omitted rules reflect making more restrictive use of the concepts of "conveyance" and "trust" (see Table 1 below). The new and modified rules implement extensions to the GNY logic, remove unnecessary restrictions in the GNY logic, and correct errors in the GNY logic [BRA96a].

In the following informal descriptions of sample BGNY rules, the rules describe how a principal B can obtain data sent in encrypted form:

Rule P1:

> If B receives a message M, then B possesses M.

Rule P4:

> If B possesses a decryption algorithm and a key, then B possesses the result of applying this decryption algorithm, with this key, to any message it possesses.

Rule P7:

> If B possesses the result of applying a decryption function, with a key, to a message encrypted with the corresponding encryption function and key, then B possesses the decrypted message.

While a complete description of the BGNY logic is beyond the scope of this paper, Table 1 lists the logical statements and symbols used in the discussions that follow. These constructs are part of Convince's Intermediate Specification Language (ISL) [BRA97]. ISL is used to describe protocols and their expected authentication properties, as well as their principals and these principals' initial conditions.

3.0 CONVINCE SOFTWARE

Three major software tools lie at the heart of Convince: the Software through Pictures™ (StP), version 2.0, Object Modeling Tool (OMT), a Higher Order Logic (HOL) theorem prover, and a translator, based on LEX and YACC, to convert ISL specifications into HOL specifications.[5] Figure 1 depicts the process, and the data flow between software components, when a user analyzes a protocol. The dashed lines show where user input is required. As the figure indicates, once the protocol is specified, most of the remaining work is done automatically.

From a textual or other description of the protocol, the user creates a model — a high-level representation — under StP/OMT. This model identifies the important attributes of the principals, messages, and encryption services (e.g., keys and other parameters), used in the protocol.

From this model, Convince generates an ISL specification, which provides a representation of all the defined elements of the protocol. Convince translates the ISL representation into an internal HOL specification, processes the HOL specification to create a HOL theory of the protocol, and executes a set of functions that automatically make deductions in this theory from the rules in the BGNY logic.

Convince produces screen output telling whether it proved all the goals. If it cannot automatically prove a goal, Convince displays the goal to the user and terminates its theorem-proving process. In this case, it

[5] The Convince components are hosted on Sun SPARCstation platforms, running SunOS 4.1.3.

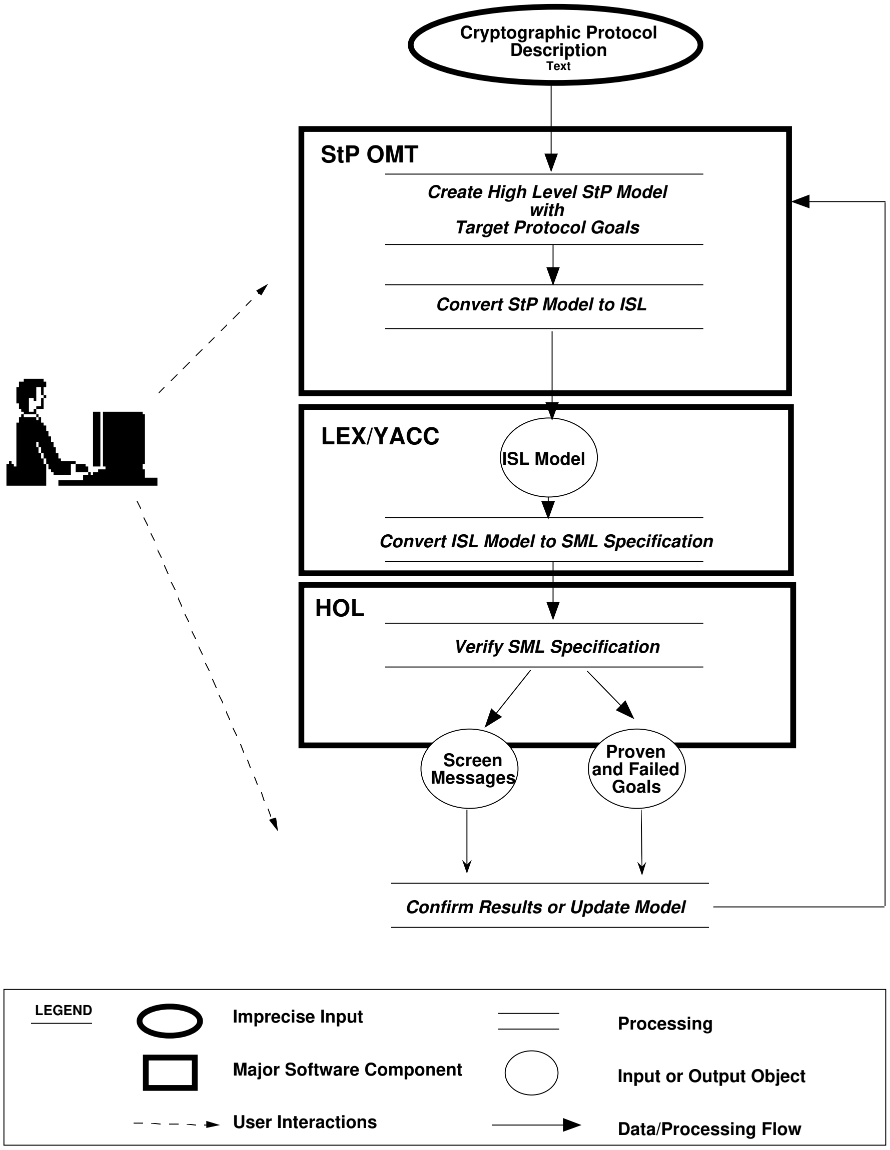

Figure 1. Convince Process and Data Flow

StP/OMT is a CASE tool that facilitates the development of software systems via the Object Modeling Technique (OMT) methodology [IDE94b]. The latter was originally developed by James Rumbaugh and his associates at General Electric in the late 1980's. More recently, this methodology has been enhanced through the efforts of Jacobson, Booch, and Rumbaugh. While the methodology has focused primarily on the development of custom software implementations, we adapted it for the modeling and analysis of authentication protocols. In the process, we enhanced the methodology by employing a somewhat richer notation necessary for describing the salient characteristics of authentication protocols.

StP/OMT automates the application of the OMT methodology by providing a set of graphical editors that can be used to create a model of an authentication protocol, providing views of the model at different levels of abstraction. We describe three below: Use Cases, Event Traces, and Dynamic Models.

produces ISL output files describing proved and unproved goals and subgoals.

A goal failure can indicate that either the protocol's specified initial conditions are insufficient, or there is some other error in the formal description of the protocol, or the protocol is flawed. The user can repeat the model/analyze process, making changes to the model, until becoming satisfied that either the protocol is flawed or does not have a flaw detectable by the BGNY logic.

3.1 StP/OMT COMPONENT

Use Case models are an extension to the original OMT formalism. We adapted this notation to provide a structure for organizing protocol specifications. We use Event Trace diagrams to describe the sequence of message transfers that comprise an authentication protocol. Each Event Trace consists of a context object, a set of object classes, and a set of directed line segments that denote message transfers. The user labels each message transfer with text denoting the nature of the message (e.g., "request for public key") and the protocol stage at which the transfer occurs. Dynamic models are used to depict the state of each principal between message transfers.

In order to completely describe the properties of authentication protocols, we had to extend the notation provided by OMT. We did this primarily by using annotations. An annotation represents additional protocol information that is associated with StP/OMT model elements. The model elements that require annotations include principals, message transfers, context objects, and states.

SYNTAX	SEMANTICS
NOUNS	
P, Q	Principals, or parties to the protocol
x	Data item or message
s	Statement
k	Key
alg	Algorithm for encryption, signature, or hash (message digest)
NOTATION	
x || s	Often the fact that a principal sends a data item x reflects the principal's current state of mind. A statement s concerning this state could be associated with the data item to denote the meaning of sending the data item.
H(x)	Function, normally used to identify a hash (or digest) of x , such as with the MD5 algorithm.
{ x } alg(x)	Encrypt content x with key K using algorithm alg, such as RSA or DES
{ H(x) } alg(k)	"Sign" x by hashing x and encrypting result with the associated key k
[x] (H, alg)(k)	"Sign" x by hashing x and encrypting result with the associated key k, then append result to x in the clear
P -> Q: x	P sends message x to Q
STATEMENTS	
P Receives x	Information is received by P, possibly after performing some computation such as decryption.
P Possesses x	P is able to repeat this data item in future messages in the current session. P would be considered to possesses a data item which is possible to compute from the data items P already holds.
P Conveyed x	P originated and subsequently transmitted x
Fresh x	Data item x has never been used in a previous message. An example is a "nonce" -- a random number generated for the purpose of being fresh
P Recognizes x	P is said to recognize the data item x if P has expectations about the content of x (e.g., presence of a particular identifier, structure, or other form of redundancy).
P Believes s	P believes or is entitled to believe that statement s holds.
Trustworthy P	If P was the source of a data item with an associated statement, this is adequate reason for believing that the associated statement is true.
SharedSecret P Q k	P and Q may properly use k to prove each other's identity. They may also use it as (or derive from it) an encryption key for secure communication.
PrivateKey Q alg k	The key k used in conjunction with the algorithm alg is one of Q's private keys.
PublicKey Q alg k	The key k used in conjunction with the algorithm alg is one of the principal Q's public keys.

Table 1. Elements of ISL Syntax and Semantics

An annotation associated with a principal denotes the name to use for the principal in message descriptions, initial conditions, and goal statements. This allows one to use longer, more descriptive principal names in OMT diagrams, while using shorter, equivalent names in formulas.

For message transfers, annotations represent the structure of messages conveyed between principals.

Annotations associated with context objects represent definitions of cryptographic and hash functions, keys, principal names, and other variables (e.g., timestamps and nonces).

Annotations associated with a principal's state correspond to ISL statements. In the case of start states, annotations represent initial conditions assumed to be true at the start of a protocol execution. Because they represent initial conditions, these annotations are limited to statements constructed from the Received and Believes operators. Annotations associated with other states (i.e., intermediate and end states) are not so restricted; these may be composed from any of the ISL statements.

Examples of an Event Trace diagram, Dynamic Model, and associated annotations are given in Figures 2 and 3.

3.2 LEX/YACC

LEX and YACC are standard UNIX utilities used to implement a parser to convert formats inside Convince. ISL is a textual language whose syntax is a superset of the annotation syntax employed under StP/OMT. ISL specifications are generated from a Convince model via a simple command option invoking the parser. ISL specifications have four major sections:

1. A set of definitions for certain data types, including principals, algorithms, and keys;
2. A set of initial conditions, indicating data items and beliefs of principals;
3. A sequence of message transfers denoting the protocol steps, or stages; and
4. A set of goal conditions showing what the protocol should achieve from the point of view of the principals.

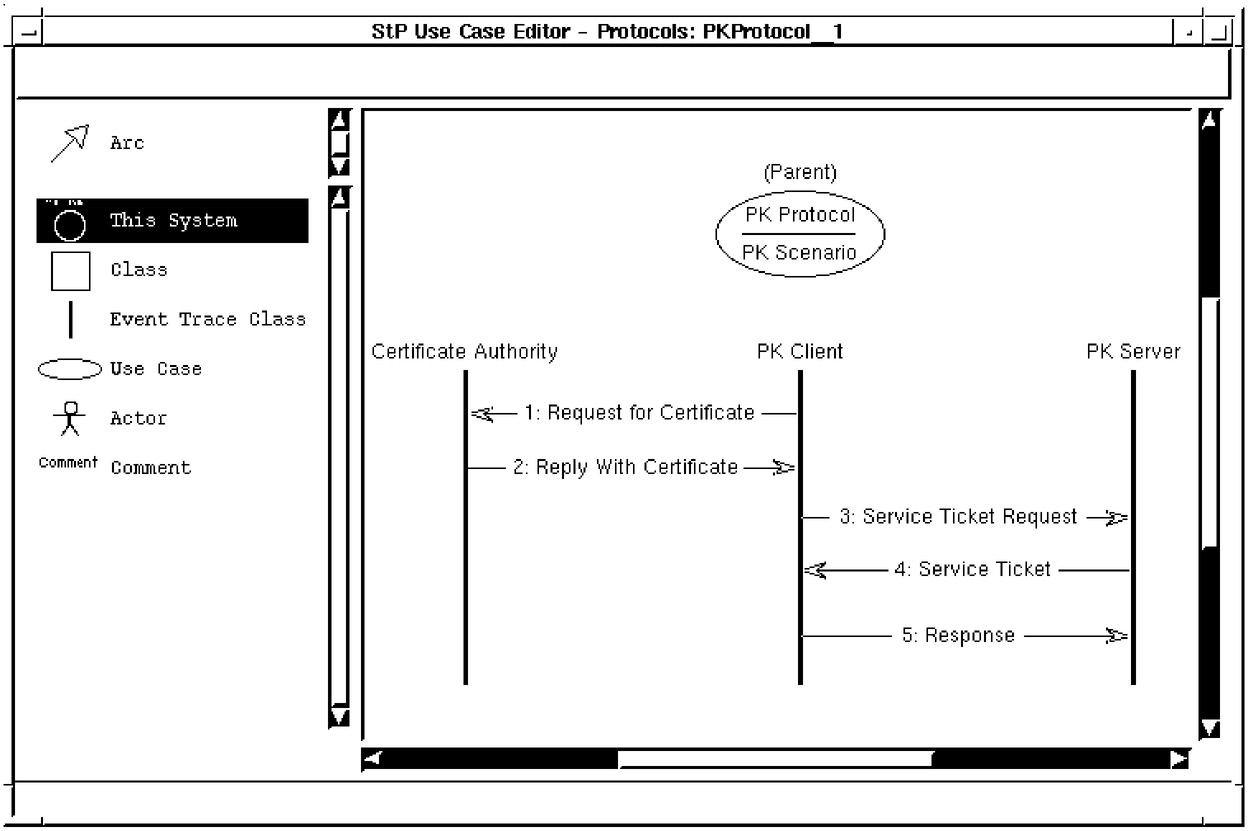

Figure 2. Event Trace Diagram

Goal conditions are numbered according to the transfer stages defined in the protocol model. The number of a goal condition is the stage of the message transfer expected to cause the goal to become true.

An example of a complete ISL specification for PK Kerberos is given in Appendix A.

In Convince, verification of an authentication protocol uses Higher Order Logic (HOL). This necessitates translating the ISL specification into a HOL internal form prior to the actual proof process. The LEX/YACC translator makes this translation. It produces HOL code that defines a theory of the protocol and invokes the automatic proof process.

3.3 HOL COMPONENT

The core of Convince is the Highter Order Logic (HOL) implementation of the BGNY logic together with a proof procedure that automates the construction of proofs in this logic. The proof procedure checks whether the protocol's goals follow from the protocol's definition and the rules of the BGNY logic. If a goal's proof fails, the problem might be an error in the initial assumptions, an overly ambitious goal, or a security flaw in the protocol.

Convince's output files listing proved and unproved goals and subgoals, in ISL, help identify the cause of proof failure.

4.0 EXAMPLE: PK KERBEROS

To illustrate how Convince can be used to model and analyze cryptographic protocols that support electronic commerce, we provide the example of PK Kerberos [CHU96], a public-key version of the Kerberos authentication protocol [STE88].

All versions of Kerberos seek to establish secure communication between two parties while maintaining confidentiality and data integrity and detecting masquerading and replays. In earlier versions of Kerberos, a centralized Key Distribution Center (KDC) authenticates a user through symmetric-key encryption, then gives this user a shared key for subsequent

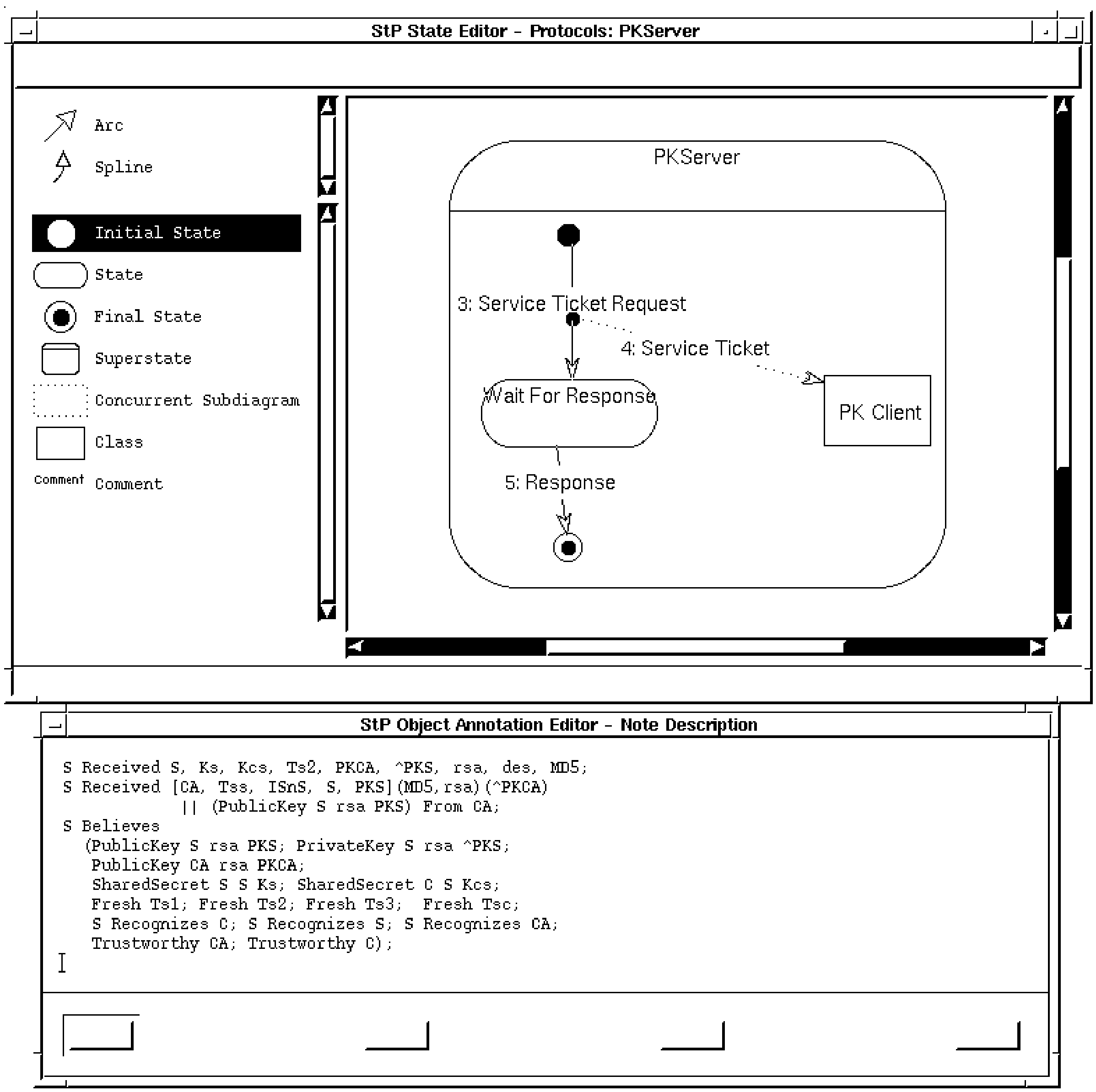

Figure 3. Dynamic Model and Annotations

communications with other parties. This makes the KDC a potential bottleneck in the system, as well as a single point of failure that could disrupt the entire system if compromised

PK Kerberos attempts to overcome this weakness by employing Public Key Certificates based on the X.509 standard [CCI88].[6] After the initial authentication, PK Kerberos continues as Kerberos does, with the exchange of symmetric keys to be used for later communication.

4.1 PK KERBEROS PROTOCOL

[6] Full implementation of these certificates will later involve an infrastructure to support the creation and initial distribution of these certificates, but they are available today for both public and private users.

The PK Kerberos protocol involves three parties: a client C, a server S, and a certificate authority CA.[7] Initially, C requests S's public-key certificate from CA. In a series of message exchanges, C receives S’s public key from CA, then, using this public key along with its own private key, requests and obtains a symmetric key for later use. By the end of the exchange, both C and S can believe that they have correctly identified each other, using certificates that they trust, and the key they share is known only to themselves.

In the model of PK Kerberos shown below, we have excluded certain fields that would normally be present in the protocol and in X.509 certificates: message IDs; encryption, signature, and message-digest algorithms; version numbers; compromise key lists; and certificate serial numbers. While these fields are needed for an implementation, they are not relevant for determining the security properties of interest, i.e., confidentiality, integrity, and authentication. In another simplification, we leave out the validity periods for keys, assuming that the protocol is running when the keys are valid.

The protocol’s description uses the following terms, along with the BGNY/ISL notation in Table 1:

C	Client
S	Server
CA	Certification Authority
CertificateX	Public-key certificate of X, defined below, signed by an authorized Certification Authority
Ts#	Time stamp number #; Ts1 is also a proxy for a current validity interval
Kr	Symmetric key to be used as a one-time session key
Kcs	Symmetric key to be used as a long-term session key
Ks	Symmetric key known only to S and used to protect tickets
PKC, PKS	Public keys for C and S
^PKC, ^PKS	Private keys for C and S
MD5	Hashing algorithm
rsa, des	Public- and symmetric-key encryption /decryption algorithms

[7] The inclusion of the CA is optional; the source for S’s public-key certificate could be S itself. For the purpose of this analysis, we use CA as both the repository for certificates and the authority that verifies their integrity. This option allows us to explore issues of levels of trust, with CA having the highest level

“authdata” is defined as data used to help authenticate C to S:
authdata = S, CertificateS, Ts1, Kr

The public-key certificate for a principal X is defined as follows:
CertificateX = CA, Ts#, X, PKX
{H(CA, Ts#, X, PKX)}rsa(^PKCA)

“CA” is the certification authority for the certificate; CA serves as the certificate repository in our model.

The transactions in PK Kerberos are as follows:

1. C requests S’s public-key certificate; C could request it directly from S, but in our model asks CA.

2. C receives the requested public-key certificate.

3. C uses S’s public key to encrypt a new temporary symmetric key, Kr, for one-time use by S, along with C’s own public-key certificate and a signature created by encrypting the hash of Kr along with S’s public-key certificate and a timestamp. The ISL statement associated with this signature asserts that C believes Kr will be known only to itself and S.

4. S decrypts the message to obtain Kr, and checks the signature to confirm that Kr came from the C named in the enclosed certificate. S creates a long-term symmetric key, Kcs, for itself and C, and sends it, encrypted under Kr, back to C. S also sends a “ticket” with Kcs, C's name, a timestamp, and possibly other security information not shown in the model (e.g., file access rights). S encrypts this “ticket” with Ks; C is to return this encrypted ticket when making later requests from S.

5. C returns a timestamp encrypted with Kcs to confirm that it received Kcs. C also returns the encrypted ticket for additional validation.

4.2 INITIAL CONDITIONS AND GOALS

The initial conditions for this protocol consist of all the received items and beliefs that the analyst assumes are held by the principals at the start of the protocol. Typical initial conditions are that the principals hold their own public and private keys, and that they trust the appropriate authority that dispenses these keys. A complete list is included in Appendix A.

Goal conditions should express the underlying purpose of the protocol's exchanges, such as that the principals believe they each possess a common symmetric key. The following shows the major goals for PK Kerberos. The numbers represent the protocol stages after which the associated goals should be true.

```
2. C Believes PublicKey S rsa PKS;
3. S Possesses Kr;
   S Believes
    (SharedSecret C S Kr;
     C Possesses Kr;
     C Believes SharedSecret C S Kr);
4. C Possesses Kcs;
   C Believes
    (SharedSecret C S Kcs;
     S Possesses Kcs;
     S Believes SharedSecret C S Kcs);
5. S Believes
    (C Possesses Kcs;
     C Believes SharedSecret C S Kcs;
     SharedSecret C S Kr;
     C Possesses Kr;
     C Believes SharedSecret C S Kr);
```

After the second transaction, for instance, C should have reason to believe that it has a bona-fide public key for S. By the third transaction, S should possess the session key (Kr) that it believes is a shared secret between itself and C. By the last step, 5, S should believe C holds the shared symmetric key Kcs.

4.3 CONVERTING DESCRIPTIONS TO StP

From a description of the protocol, usually text, the user creates a protocol model by first defining the protocol elements within StP/OMT. The user then constructs a Use Case diagram and associates it with a specific protocol scenario. In our example , we model only a single scenario, shown in the Event Trace diagram in Figure 2. StP's Event Trace editor automatically provides a context object, here labeled as PK Protocol. The user next adds the vertical bars representing the principals, and labels them accordingly. The user adds a set of directed line segments to denote the message transfers that occur as part of the protocol scenario, and labels each message transfer with a text string denoting the nature of the message (e.g., "request for public key") and the stage of the protocol at which the transfer occurs.

After completing the Event Trace Diagram, the user constructs a Dynamic Model for each of the principals. As shown in Figure 3, the Dynamic Model for S in PK Kerberos is a state transition sequence. The start state is represented as a solid circle, the intermediate state as a rounded rectangle, and the end state as a bull's eye. Transitions between states are represented by directed lines whose labels denote the received events responsible for triggering the transitions. Message transfers that are produced by the principal are represented as output events. These are associated with directed lines connecting a state transition to the principal who is the recipient of the message.

Generally speaking, the start state of a Dynamic Model corresponds to a subset of the initial conditions for the protocol. Accordingly, for each start state, the user provides annotations that represent the initial conditions of the corresponding principal. In Convince, these conditions are limited to statements of belief or reception. The initial conditions of principal S are shown as a sequence of ISL statements at the bottom of Figure 3.

After adding the initial conditions to the model, the user provides annotations for the intermediate and end states. These annotations represent goals for the protocol (e.g., C and S share a certain symmetric key), which should become true once the protocol reaches a specific state.

4.4 CONVERTING StP TO ISL

Once the initial conditions, transactions, and goals have been input, the user directs Convince to convert the model to an ISL specification, then invoke the translation and proof processes. This is done with a single menu selection from StP/OMT. The full ISL specification for PK Kerberos is given in Appendix A.

The LEX/YACC and HOL subsystems of Convince can be used without Convince's StP interface. To do so, the user prepares an ISL specification directly, as a text file, and gives the name of this file as a command-line argument to the LEX/YACC translator, which invokes the proof process.

4.5 RUNNING THE VERIFIER

Convince attempts to verify a model by proving that it meets both its user-specified goals and a standard set of goals, originally derived from the GNY logic, that encompass all protocol properties that are typically of interest [BRA96b].

During the first few iterations of creating or modifying a protocol model and seeing if Convince

proves that it meets its goals, proof failures will typically result from insufficient initial conditions, such as a principal not possessing a needed algorithm. This was the case with our analysis of PK Kerberos. Insufficient initial conditions relating to possession often result in protocol feasibility failures (i.e., a principal attempting to send something it does not possess) [GON91].[8]

In PK Kerberos, the most significant proof failure due to an insufficient initial condition involved S's having to trust C to create the temporary symmetric key Kr. Our original model did not include this condition, and the proof failed at the subgoal of S believing Kr is a shared secret. Even though this key is for one-time use, production of weak or guessable keys by C could cause vulnerabilities in the protocol. Within the context of NetBill, C will be executing software with a predefined algorithm for creating these temporary keys, which is expected to limit their vulnerability. In more general contexts, this assumption should be examined closely.

Problems due to insufficient initial conditions are generally easy to correct once the reason for proof failure is identified. Convince's output files giving lists of proved and unproved standard goals, and their proved and unproved subgoals, are useful for this purpose. It should be noted, however, that some initial conditions might impose constraints on an implementation that are unacceptable.

In addition to problems that result from insufficient initial conditions, proof failures can result from inadequate or inappropriate associations of properties, expressed via ISL statements, with messages. As a rule of thumb, encrypted messages used to convey keys that are shared secrets should include an associated statement expressing this fact.

We call the types of errors noted above *setup* errors because they are due to the specific form of the model being constructed and do not necessarily show flaws in the protocol itself. Similarly, apparently redundant information in a protocol, which we found in the PK Kerberos example, might not cause security flaws.

In translating the English descriptions of the PK Kerberos example into ISL, we uncovered a particular aspect of the protocol that demonstrated the need for one of our extensions to the GNY logic. In stage 4, S sends out an encrypted copy of a ticket that only S can decrypt, along with the same and more information in a form that is readable by C. In stage 5, C uses the information available to it to prepare an appropriate authenticator, and sends that authenticator, along with the ticket that only S can decrypt, back to S. This is necessary because S has forgotten everything except the key it used to encrypt "send this back to me" copies of the tickets it has sent out in the last few hours. S uses this key to decrypt these tickets when they are sent back to it, to confirm that they were originally from S and go with the authenticators sent back with them. So rather than remember each ticket or a hash of this ticket, S remembers only the key Ks it uses to encrypt these tickets.

This is not expressible in the GNY logic, which assumes that principals remember everything for the length of a protocol run; every principal has perfect memory of the messages it has sent or received. For a potential attacker, this is a good, conservative assumption, but for legitimate protocol principals, PK Kerberos shows that it might not be true.

In total, it took us about 3 days of tool use, spread over a couple of weeks, to resolve all the problems in our model of PK Kerberos. Once the model was finished, the conversion to ISL and production of all the proofs took less than 5 minutes on a Sun SPARCstation 20.

In the course of our analysis, we proved that by the end of PK Kerberos the keys are securely in place with the parties authenticated to each other, but this requires that the client be trusted to create a sufficiently strong symmetric session key. We concluded that the protocol contains elements that, while appropriate for NetBill, might be unnecessary or insufficient for use in other contexts. For example, in some environments, encryption keys should only be generated by a high-integrity source.

5.0 RELATED WORK

Romulus [ORA94] represents an early effort to automate the analysis of authentication protocols via theorem proving. Romulus implements belief logic, in HOL, in the form of a theory of authentication, crypto_90. This implementation requires that a user create protocol models in HOL, with all initial assumptions, protocol actions, initial conditions, and goals expressed as HOL statements. The user produces proofs by applying HOL tactics, by hand, using rules defined in crypto_90. A typical verification strategy is first proving a set of simple conditions that can be

[8] Additional forms of insufficient initial conditions we encountered in modeling other protocols include beliefs relating to "freshness" (e.g., recent timestamps), recognition of key message elements (e.g., principal names), trust, and properties of keys (e.g., that a principal's public key is believed to be what it is).

applied repeatedly as lemmas in the proof of the goals. Romulus depends heavily on the HOL environment, which necessitates that a user be thoroughly familiar with HOL. In addition, crypto_90 is limited to symmetric-key encryption; it does not handle public-key encryption, signatures, hash functions, key-exchange functions, or the use of hash codes as keys.

More recently, Tom Schubert and his students at Portland State University [SCH95] created a HOL implementation of a variant of the SVO belief logic [SYV94]. The SVO logic represents an effort to harmonize different belief logics into a single unified formalism. Schubert and his students used the resulting tool to specify and verify the correctness of key escrow protocols. As a byproduct of this work, they created an infrastructure to support interactive proofs in HOL. They have made plans to extend this tool by incorporating semi-automated decision procedures.

The NRL Protocol Analyzer [MEA91] and the MITRE Interrogator [MIL84] take somewhat different approaches to protocol analysis. They are rewrite systems that determine whether an undesirable state can be reached from a given set of initial states. An sample use of these tools would be determining whether any amount of manipulation of exposed data by an intruder could obtain a key that should be kept secret.

The NRL Protocol Analyzer differs from Convince in the type of goals it can handle; it attempts to show the absence of all possible flaws in a protocol, while Convince attempts to establish that certain beliefs are justified. The NRL Protocol Analyzer is a low-level tool that requires a user to define the undesirable states.

The Interrogator, which was built by Millen, generates a large number of paths through the protocol ending in an insecure state, and sees whether any of these begin in a valid initial state. It does not prove that there are no such paths, and finds flaws similar to those found by the NRL Protocol Analyzer.

The Interrogator and NRL Analyzer are in principal capable of finding more flaws in cryptographic protocols than Convince is, but they are drastically much slower and drastically more difficult to use. Protocol analyses using them typically take weeks or months of effort by these tools' developers.

The tools described above are tailored specifically for analyzing cryptographic protocols. While in theory one could apply belief logics to an analysis of the strengths and weaknesses of non-cryptographic protocols, this would require changing the rules that implicitly rely on properties of cryptographic functions.

Still, to varying extents, these tools share their underlying technologies with verification tools implemented for other domains; see [BUT95] for a description of some recent developments. While these other tools are too numerous to describe here, two are noteworthy. One of these, Murφ, incorporates a protocol description language and verifier for detecting design errors such as deadlocks [DIL92]. Another recent tool combines the formal language Estelle with Numerical Petri Nets (NPN) [JIR95]. In this system, protocols are first specified in Estelle and then translated into NPN specifications. These can then be analyzed using an NPN verifier.

6.0 SUMMARY & FUTURE DIRECTIONS

This paper has described a tool for modeling and analyzing authentication protocols. As the tool has matured, we have used it to analyze aspects of confidentiality, authentication, and key distribution in a recently proposed protocol, a public-key version of Kerberos. Our analysis brought questions regarding initial-conditions assumptions to light, including an issue of trust. Whether this is a problem depends on how the protocol is actually implemented and used.
We have provided this information to the authors of the PK Kerberos protocol and will support them with subsequent analyses if they change the protocol.

In general, the viability of proof-based analyses depends on the reasonableness of the proof goals, the scope of the logic, and whether a sufficient set of initial conditions are available. Convince has evolved to address the salient issues relating to the authentication protocols we have analyzed.

In translating English descriptions into ISL, we have been able to quickly identify those aspects of protocols that are not clear. We are following ongoing efforts in the protocol analysis community to develop a standard protocol specification language.

In the future, we plan to extend Convince to provide more detailed guidance, through its graphical user interface, for identifying the cause of proof failures. We also plan to explore how Convince might be used in conjunction with other types of protocol analysis tools.

7.0 ACKNOWLEDGMENTS

The authors would like to thank John Chuang and Marvin Sirbu, of the Carnegie Mellon University Engineering and Public Policy Department, for allowing us to use a preliminary version of the Public Key Kerberos protocol. We would also like to credit Rae Burns and Yi-Fang Koh, whose use of StP as a front end to a design tool for multilevel secure databases was the inspiration for our use of StP. We also express our appreciation to Jack Wool and Joe Giordano for their support of this effort.

8.0 REFERENCES

[BERN96] T. Bernstein, A. B. Bhimani, E. Schultz, and Carol A. Siegel, *Internet Security for Business*. New York: Wiley Computer Publishing, 1996, pp. 251-255.

[BRA96a] S. H. Brackin, “A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols”. *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, County Kerry, Ireland, 1996.

[BRA96b] S. H. Brackin, “Deciding Cryptographic Protocol Adequacy with HOL: The Implementation”. *International Conference on Theorem Proving in Higher Order Logics*, 1996.

[BRA97] S. H. Brackin, “An Interface Specification Language for Automatically Analyzing Cryptographic Protocols”. To be presented at the Internet Society Symposium on Network and Distributed System Security, February 1997.

[BUR90] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication". *ACM Transactions on Computer Systems*, 8(1), 1990.

[BUT95] R. Butler, J. Caldwell, V. Carreno, C. Holloway, P. Miner, and B. Di Vito, “NASA Langley’s Research and Technology Transfer Program in Formal Methods”. In *Proceedings of the Tenth Annual Conference on Computer Assurance*, pages 135 - 150, 1995.

[CCI88] CCITT, Recommendation X.509: The Directory Authentication Framework, 1988.

[CHU96] J. Chuang and M. Sirbu, Internet Draft, Update to RFC 1510, “Public-Key Based Ticket Granting Service in Kerberos”, Carnegie Mellon University/INI, draft-sirbu-kerb-ext-00.txt, May 6, 1996.

[COX95] Benjamin Cox, J. D. Tygar, Marvin Sirbu, “NetBill Security and Transaction Protocol”, *Proceedings of the First Workshop on Electronic Commerce*, 1995.

[DIL92] D. Dill, A. Drexler, A. Hu, and C. Yang, “Protocol Verification as a Hardware Design Aid”, 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1992.

[GON90] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols". In the *Proceedings of the 11th IEEE Symposium on Research in Security and Privacy*, pp. 234-248, 1990.

[GON91] L. Gong, "Handling infeasible specifications of cryptographic protocols". *Proceedings of the Computer Security Foundations Workshop IV*, pp. 99 - 102, 1991.

[GOR93] M. Gordon and T. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge, UK: Cambridge University Press, 1993.

[IDE94a] Interactive Development Environments, *Fundamentals of StP, Release 1*, February 1994.

[IDE94b] Interactive Development Environments, *Creating OMT Models, Release 1*, February 1994.

[JIR95] A. Jirachiefpattana and R. Lai, “An Estelle-NPN Based System for Protocol Verification”, *Proceedings, 10th Annual Conference on Computer Assurance*, pp. 135 - 150, 1995.

[MEA91] C. Meadows, “A System for the Specification and Analysis of Key Management Protocols”. In *Proceedings of the Symposium on Security and Privacy*, 1991, pp. 182-147.

[MIL84]J. Millen, “The Interrogator: A Tool for Cryptographic Protocol Analysis”. In *Proceedings of the Symposium on Security and Privacy*, pp. 134-141, 1984.

[ORA94] Odyssey Research Associates, *Romulus User's Manual*, March 1994.

[SCH95] T. Schubert and S. Mocas, A Mechanized Logic for Secure Key Escrow Protocol Verification, *International Workshop on Higher Order Logic and its Applications*, 1995.

[STE88] J.G. Steiner, C. Newuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems". In Proceedings of the USENIX Winter Conference, 1988, pp. 191 - 202.

[SYV94] P. Syverson and P. van Oorschot, “On Unifying Some Cryptographic Protocol Logics”. In *Proceedings of the Symposium on Security and Privacy,* Oakland, CA, 1994, pages 14-28.

APPENDIX A. PK KERBEROS ISL SPECIFICATION

```
DEFINITIONS:
PRINCIPALS: C,S,CA;
SYMMETRIC KEYS: Kr,Kcs,Ks;
PUBLIC KEYS: PKC,PKS,PKCA;
PRIVATE KEYS: ^PKC,^PKS,^PKCA;
OTHER: Ts1,Ts2,Ts3,Tsc,Tss;
ENCRYPT FUNCTIONS: des,rsa;
HASH FUNCTIONS: MD5;
des WITH ANYKEY HASINVERSE des WITH
ANYKEY;
rsa WITH ^PKCA HASINVERSE rsa WITH PKCA;
rsa WITH ^PKC HASINVERSE rsa WITH PKC;
rsa WITH PKS HASINVERSE rsa WITH ^PKS;

INITIALCONDITIONS:
CA Received rsa,MD5,CA,S,Tss,PKS,^PKCA;
CA Believes PublicKey S rsa PKS;
C Received
rsa,des,MD5,C,S,Ts1,Ts3,Kr,PKCA,^PKC;
C Received

[CA,Tsc,C,PKC](MD5,rsa)(^PKCA)||(PublicKe
y C rsa PKC) From CA;
C Believes
 (PublicKey CA rsa PKCA;
  SharedSecret C S Kr;
  Fresh Ts2; Fresh Tss;
  C Recognizes C; C Recognizes S;
  Trustworthy CA; Trustworthy S);
S Received
des,rsa,MD5,Ts2,Ks,Kcs,PKCA,^PKS;
S Received

[CA,Tss,S,PKS](MD5,rsa)(^PKCA)||(PublicKe
y S rsa PKS) From CA;
S Believes
 (PrivateKey S rsa ^PKS; PublicKey CA rsa
PKCA;
  SharedSecret S S Ks; SharedSecret C S
Kcs;
  Fresh Ts1; Fresh Ts3; Fresh Tsc;
  S Recognizes C; S Recognizes S;
  Trustworthy CA; Trustworthy C);

PROTOCOL:
1. C -> CA: S;
2. CA -> C:

[CA,Tss,S,PKS](MD5,rsa)(^PKCA)||(PublicKe
y S rsa PKS);
3. C -> S:
    S, Ts1,
    {Kr, C,

[CA,Tsc,C,PKC](MD5,rsa)(^PKCA)||(PublicKe
y C rsa PKC),
     [S, Ts1, Kr,

[CA,Tss,S,PKS](MD5,rsa)(^PKCA)||(PublicKe
y S rsa PKS)
     ](MD5,rsa)(^PKC)||(SharedSecret C S
Kr)
    }rsa(PKS);
4. S -> C:
    S,
    {Kcs,C,Ts2}des(Ks),
{C,S,Kr,Kcs,Ts2}des(Kr)||(SharedSecret C
S Kcs);
5. C -> S:
     S,
     {Kcs,C,Ts2}des(Ks),
     {C,Ts3}des(Kcs)||(SharedSecret C S
Kcs);

GOALS:
2. C Believes PublicKey S rsa PKS;
3. S Possesses Kr;
   S Believes
    (SharedSecret C S Kr;
     C Possesses Kr;
     C Believes SharedSecret C S Kr);
4. C Possesses Kcs;
   C Believes
    (SharedSecret C S Kcs;
     S Possesses Kcs;
     S Believes SharedSecret C S Kcs);
5. S Believes
    (C Possesses Kcs;
     C Believes SharedSecret C S Kcs;
     SharedSecret C S Kr;
     C Possesses Kr;
     C Believes SharedSecret C S Kr);
```