Empirical Evaluation of the Protocol Specification Language MSR 2.0

Rishav Bhowmick
School of Computer Science, Carnegie Mellon University Qatar
rishavb@cmu.edu

Advisor: Iliano Cervesato
School of Computer Science, Carnegie Mellon University Qatar
iliano@cmu.edu

Abstract

This research has the objective of giving an empirical evaluation of the security protocol specification language MSR (version 2.0). In other words, the research work answers the question how good is MSR 2.0 (MultiSet Rewriting) in specifying protocols. Protocols are needed everyday for secure information exchange. But, getting protocols right is hard. MSR 2.0 is a simple language to design protocols and test them. The methodology to find this was to represent 40 protocols from a commonly used repository of well-known protocols, the Clark-Jacob library. In the process of doing this, notes were made on the shortcomings and bugs were noted while relying on MSR 2.0 implementation. The findings are that the MSR language is good enough (succinct representation of protocols, easy to design simple and complicated protocols), but the MSR implementation needs improvement. Type reconstruction is not always possible, unclear error messages and bugs did not allow the MSR implementation from being efficient and less time consuming. The secondary goal was to find how easy it is to use by non-experts. The research resulted in showing that its not an impossible task for a non-expert to learn this language, but sufficient understanding and practice is required to master the language.

Table of Contents

1. Introduction

Cryptographic protocols are small distributed programs that ensure that information is exchanged securely over the Internet. For example, you want to send an e-mail to your friend and as usual you do not want someone else to read it. Your friend's computer and yours agree upon a set of rules (protocol) and securely exchange information. Dozens of protocols can be found in any computer and other networked device: well-known examples include SSL/TLS (the “s” in “https”), Kerberos and PGP [1,5,6]. Although small in size, protocols exhibit complex interactions, not always fully anticipated by their authors. Several protocols have been found to be flawed many years after their introduction (for example, SSL is on its third revision): this leaves the data they are supposed to protect open to attacks (which may lead to the disclosure of secret information or unauthorized access to such resources as a computer or a bank account). This is a significant problem for our society since, until cryptographic protocols can be certified to resist the most common types of attacks, commerce and communication over the Internet cannot be fully trusted.

The most promising countermeasure to this unfortunate state of affairs consists in building a mathematical proof that a given protocol does not have dangerous behaviors. The construction and verification of these proofs can often be automated. Although effective proof methods are rapidly being devised, not as much attention has been paid to the languages in which to express protocols for automated verification. Until a few years ago, such languages were ad-hoc, lacked expressiveness, and were so complex as to be understandable by just a few specialists. Before 90’s English was the only language used for specification of protocols. But the descriptions led to ambiguitites and imprecision. This meant in particular that new protocols may still be flawed because their developers were not able to use these languages and the verification techniques based on them. In the face of this situation, Cervesato recently proposed a family of new protocol specification language called MSR with well-understood foundations, wide applicability and apparent ease of use on the basis of a limited number of experiments.

The goal of this research was to provide the precise understanding of the actual usability of the language MSR 2.0. In other words, to evaluate MSR – how good is it in representing protocols and how easy it is to use by non-experts. Why was it

performed? Even if the generality and expressiveness of MSR are well-established, its ease of use for non-experts is still speculative since only its authors and a few collaborators have used it. In this project, 40 security protocols from a well-known protocol repository, the Clark-Jacob library [1], were represented in MSR 2.0. Notes were made on the shortcomings and the representations relied on MSR implementation. The research led to the conclusion that the MSR language is good enough but the MSR implementation needed improvement.

A point to be noted is that this research involves the student researcher as the subject of the experiment also. He is the non-expert who will give his opinions about how easy it is to use MSR 2.0. Furthermore, if MSR is mentioned, it refers MSR 2.0.

The rest of the report is structured in the following way. Section 2 talks about the background studies performed that helped in accomplishing the research. It also gives a good global insight into the topic. Section 3 talks about the technical approach in the project. Section 4 gives details about the result yielded from the research and section 5 concludes with some recommendation. The Appendix A contains all the work (40 protocols) specified in MSR 2.0 along with purpose of the protocol, informal specification and the findings after the protocol was specified in this language.

2. Background Studies

We start with some background studies performed before the research work took place.

2.1. Formal Specification of Protocols

Protocols can be found around us in most devices that communicate with one another. There are different types of protocols- Communication Protocols, which enable communication in the first place, and Security Protocols that ensure information is exchanged securely over a network, whether it is the World Wide Web or the Local Area Network. For example, computer and cell phones use protocols. Protocols can be defined as a set of rules agreed upon to allow the above process to take place.

The problem faced is that protocols are extremely hard to get right. This is because they are minuscule programs with extreme complex interactions (bugs can take years to discover). So there is a need of protocol specification language (design framework) that can describe the protocol—

- message flow
- message constituents
- operating environment
- protocol goals

Up to the late 90's English was used as a specification language (not a formal specification language). But it turned out to be long, tedious, ambiguous and incomplete specifications. Verification to small extent was not possible. Then, there came into picture formal specification languages like MSR and CAPSL (Common Authentication Protocol Specification Language). They are used to write more formal description of the protocols which can be implemented and verified. The protocols specified no longer have any ambiguity and lot of verification tasks like type-checking could be automated. They are flexible as they adapt to a protocol and apply to a wide class of protocols. Presently process calculus and the spi calculus are competitors for MSR.

2.2. A Brief History of MSR

MultiSet Rewriting or MSR is a simple model of distributed computing and an executable protocol specification language. MSR represents the state of a system as multiset of "facts" and its transitions as rewrite rules that describe how the system evolves based on local changes to the state. There are 3 generations of this language.

MSR 1, designed in 1999, had restricted ways of using rules for a protocol. Hence representation of real protocols was tedious and error prone. One of the main reasons MSR 1 was developed was to establish the fact that protocol verification is un-decidable.

Then, there came MSR 2, strongly typed, which can be used to describe the rules of a protocol and hence be used to represent them easily. This research used this version of MSR. Furthermore, the MSR 2 prototype was used, that is syntax checks were simplified, type reconstruction was supported and limited simulation was allowed.

MSR 3 is a prototype language being developed now, which is going to be the next generation of MSR.

There has been other works using MSR. One of the works by Cervesato and his collaborators involved using MSR 2 to represent Kerberos, a crucial authentication protocol within Microsoft Windows and many other computer systems. During this process they found a serious flaw which prompted the relevant international standards body to redesign elements of Kerberos and led Microsoft and other vendors to immediately release security patches for their products.

2.3. Using MSR 2.0 to represent protocols

The best way to understand how MSR works is to go through an example. So let’s look at the following simple protocol taken from the Clark-Jacob library of protocols [1].

ISO Symmetric Key Two-Pass Unilateral Authentication Protocol [1]

In this protocol principal A is authenticated by the verifier B by the means of challenge-response. Challenge-response authentication includes a question asked by the verifier to which the principal to be authenticated responds. In this case principal B, on receiving message 2, checks if A has sent Rb and B after decrypting the msg E(Kab : Rb, B, text2) using shared key Kab.

Informal Specification

1. B -> A: Rb, text1
2. A -> B: text3, E(Kab : Rb,B,text2)

Here A and B are two roles which participate in the protocol. Each role consists rules. In this case B sends the first message and receives the last one. That's 2 rules. A receives the first message and immediately sends the second one. That's one rule. As we will see in the MSR specification the roles are described as series of rules, linked only by symbol declarations in the preamble of the protocol.

Commonly Used Representations

- A -> B : E(K : N, Kx) — a sample rule
- A -> B — message sent from A to B
- Kxy — shared key between x and y
- Rb — Random number
- A — Alice (First principal)
- B — Bob (Second principal)
- E(K : N, Kx) — encrypting the nonce N and key Kx with key K

Given the above protocol, we can perform the following steps

1. List down the principals used in the protocol.
2. List down the types needed to describe the protocol - in this case its shared key, random number, text.
3. List down the functions needed - in this case there are two functions, enc (to encrypt) and net (that converts the message to a state, in other words passing the message over the network).

The above steps can be formally expressed in MSR as follows (see next page):

```
(reset) %to reset the context%
(
module ISOTwoPassUni
export * .
--- msg is a type for message
shrK  : type.   %declaring a type shrK for shared key %
randomNo: type.
randomNo  <: msg. % <: - subsort declaration, here randomNo is a
msg also%
enc : shrK -> msg -> msg. %function that takes in a shrK and a
message and
                              encrypts it to return an encrypted
message%
text: type.                                   text <: msg.
net : msg -> state. %function to convert the msg to state
(passing the                          message onto the network%

"% ... continued below%
```

Now we define the rules for each principal.

- Principal B sends out the first message where it creates a random number and passes a text along with it.
- Initially the state is empty to indicate that the rule is applicable no matter what is in the state. Other rules must have instances of their antecedent to be applicable, not this one.
- “exists” is used to show the creation of a particular value and “forall” means for any value
- This process is carried on for all the rules of each principal.
- As B is the final receiver of the protocol, the right hand side of the final rule is empty.

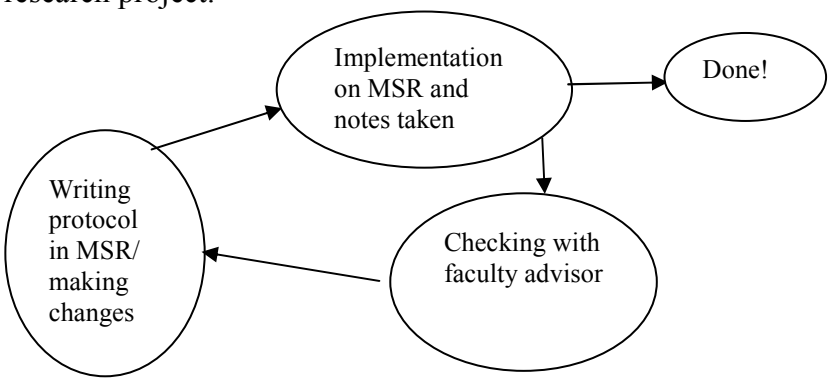
This is what the protocol looks like when finally represented in MSR where the preamble discussed above is reported without the comments (see next page):

```
%continued from top %
initiator:
forall B : princ. %role for any principal (the verifier)%
{
                  exists LA : randomNo. -> state. %local state
predicate%
  ISO2initRule1: %first rule%
  forall text1 : text.
  empty
  => exists rB : randomNo.
    net(rB & text1), LA rB.

 ISO2initRule3: %third rule%
  forall A : princ.
  forall KAB : shrK.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
    net(text3 & enc KAB (rB & B & text2)), LA rB
  => empty.
}

reciever:
forall A : princ. %role for any principal (the authenticated)%
{
  ISOinitRule2: %second rule%
  forall B : princ.
  forall KAB : shrK.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
   net(rB & text1)
  => net(text3 & enc KAB (rB & B & text2)).
}
) %end of protocol%
```

3. Methodology

The methodology implemented for this project involves the following stages:

1. **First look at MSR**
   After independently acquiring the background knowledge about cryptography and security necessary for the project, the next step was to get acquainted with MSR 2.0. This was a crucial stage where an untrained person had to master the key concepts of MSR to a point where he used it to autonomously describe protocols in this language. This first exposure took around three one-hour sessions which was less than expected 5 hours.
2. **Protocol Representations**
   Once the researcher was used to MSR syntax and semantics, he got hold of a well established repository of protocols called the Clark-Jacob library of security protocols. From this library, he started representing 40 protocols into MSR. Each time he was done with a protocol, it was checked by the faculty advisor and notes were made on the shortcomings and bugs found. This is shown below in the flowchart. The first 5-10 protocol representations were more of a learning experience. But later on, the process became a repetition. There were certain complicated protocols which needed more time and often needed help of the faculty advisor. The representations were based on MSR implementation [*See* Appendix]. They relied on the existing MSR 2 prototype rather than hand written. This phase took the most time and was the most crucial phase of the whole research project.



3. Representing a part of complete protocol

The third phase was to implement a small part of The Kerberos Network Authentication Service (V5). Only the first stage – reading the RFC (Request For Comments) of this protocol was accomplished. Due to time constraint, this phase could not be completed.

4. Results

The results can be split up according to the stages of the methodology:

1. **First look at MSR**
   This can be interpreted from the words of the subject of the research in the following quote:

"The first stage that involved learning about cryptography and security was pretty interesting and simple. I attended the Computer Security Forum 2007 at Venice. This experience gave me more insight both in depth and breadth in this field. Then we moved on to the phase of getting acquainted with MSR 2.0. My faculty advisor helped me through the basic concepts of MSR which would be enough to specify protocols.

Initially, the whole of MSR was like Greek to me. I used to have loads of questions. After my advisor answered to them I got some picture of how it works, but not a complete crystal clear one. Coming from a non-functional based programming background like Java or C, MSR was a whole new world, although I wasn't using the complete power of this protocol specification language. For example, the way functions work in MSR is very different from the ones that work in Java or C.

As we moved on to the next step of specifying protocols, things started to make more sense. The first 5-10 protocols representation was more of learning. Later on, things became easier. There were some protocols which needed a different perspective as they were more complicated and this is when I needed help from my advisor.

A point to be taken into account is that, after I took my functional programming class in ML (Meta Language), MSR started to make better sense. So I believe that, if someone is used to functional programming language, he/she can master it in very little time just to get used to the syntax. But if he/she is new to this field, it will take some time to understand the concept and syntax. But once he/she gets a hang of it, life is good."

- Rishav Bhowmick

2. Protocol Representations

The second phase included representations of 40 protocols from the Clark-Jacob library.

The following were the findings after this phase:

1. MSR language is good enough:
    - It represents the protocols succinctly with no ambiguity and presents a proper picture of the rules and principals in the protocol
    - It is easy to design not only simple but complicated protocols using MSR.
2. MSR implementation requires improvement:
    - Type reconstruction is not always possible. Type reconstruction is automatic computation of the type of an expression that does not contain type information. This mainly happened when the local state predicates were used.
    - MSR 2.0 has very unclear error messages. If at all it points to a position, the error is actually somewhere else. So this made debugging a long process.
    - Apart from these, certain bugs were found, that made no sense. For instance, while representing the Amended Needham Schroeder protocol, it gave unsolvable constraints error. But later, after doing a few changes and coming back to the same state, it worked.

The notes for each protocol can be found in the Appendix.

3. Representing a part of complete protocol

As this phase of research was not completed, no considerable result has yielded. The protocol was specified in long text of English and as a result most parts were ambiguous and it was a difficult read.

5. Conclusion and Recommendation

In a nutshell, a non-expert learnt the protocol specification language MSR 2.0 and evaluated it by representing 40 protocols and presented the findings as shown above and in the Appendix. The findings will be used to improve protocol specification languages, not only MSR.

6. References

1. Clark, John, and Jeremy Jacob. A Survey of Authentication Protocol Literature. Version 1.0. 1997.
2. Reich, Stefan. User's Guide to the MSR Implentation. University of Illinois, 2004.
3. Cervesato, Iliano. "Typed MSR: Syntax and Examples." Proceedings of the First International Workshop on Mathematical Methods, Models and Architectures of Computer Network Security (2001).
4. Cervesato, Iliano. "MSR 2.0: Language Definition and Programming Environment." (2007).
5. Kaufman, Charlie, Radia Perlman, and Mike Spencer. Network Security. Upper Saddle River: Prentice Hall PTR, 1995.
6. Gollmann, Dieter. Computer Security. Chichester: John Wiley & Sons, 2006.

7. Appendix A

This appendix contains all the 40 protocol specifications in MSR 2.0 as produced in this project. It can be viewed online at:
http://www.qatar.cmu.edu/%7Erishavb/research.shtml

Library of Security Protocol Specifications

- ISO Symmetric Key One-Pass Unilateral Authentication Protocol
- ISO Symmetric Key Two-Pass Unilateral Authentication Protocol
- ISO Symmetric Key Two-Pass Mutual Authentication Protocol
- ISO Symmetric Key Three-Pass Mutual Authentication Protocol
- Using Non-reversible function
- Andrew Secure RPC Protocol
- Denning Sacco
- Otway-Rees Protocol
- Wide Mouthed Frog Protocol
- Yahalom
- Carlsen
- ISO Five-Pass Authentication protocol
- Woo and Lam authentication (Pi_f) protocol
- Amended Needham Schroeder protocol
- Needham-Schroeder Signature Protocol
- Kerberos version 5 Protocol
- Neuman Stubblebine
- Kehne Langendorfer Schoenwalder
- Kao Chow Repeated Authentication Protocol
- ISO Public Key One-Pass Unilateral Authentication Protocol
- ISO Public Key Two-Pass Unilateral Authentication Protocol
- ISO Public Key Two-Pass Mutual Authentication Protocol
- ISO Public Key Three-Pass Mutual Authentication Protocol
- ISO Public Key Two-Pass Parallel Mutual Authentication Protocol
- Diffie Hellman Exchange
- Needham-Schroeder Public Key Protocol
- SPLICE/AS authentication Protocol
- Hwang and Chen's SPLICE/AS authentication Protocol

- Denning Sacco Key Distribution Protocol with Public Key
- CCITT X.509
- Shamir Rivest Adelman Three Pass protocol
- Encrypted Key Exchange
- Davis Swick Private Key Certificates
- Gong Mutual Authentication Protocol
- Bilateral Key Exchange with Public Key
- ISO One-Pass Unilateral Authentication Protocol with CCFs
- ISO Two-Pass Unilateral Authentication Protocol with CCFs
- ISO Two-Pass Mutual Authentication Protocol with CCFs
- ISO Three-Pass Mutual Authentication Protocol with CCFs

Commonly Used Representations

- A -> B : E(K : N, Kx) - a sample rule
- A -> B - Message sent from A to B
- Kx - public key of x
- Kx-1 - private key of x
- Kxy - shared key between x and y
- A - Alice (First principal)
- B - Bob (Second principal)
- S - Sue (Server)
- E(K : N, Kx) - encrypting the nonce N and key Kx with key K
- f (R) - function f applied on random number R

ISO Symmetric Key One-Pass Unilateral Authentication Protocol

Purpose

This protocol consists of a single message from A to B while sharing a secret key Kab.

Informal Specification

1. A -> B: text2, E(Kab:[Ta|Na],B,text1)

MSR Specification

```
--- Author: Rishav Bhowmick
--- Date: 27 May 2007
--- ISO Symmetric Key One-Pass Unilateral Authentication Protocol
---
---       A -> B: Text2, E(Kab:[Ta|Na],B,text1)
(reset)
(
module ISOOnePassUni
export * .

---msg : type.

---princ : type.                          princ  <: msg.
shrK  : type.                           shrK <: msg.
nonce : type.                          nonce  <: msg.
enc : shrK -> msg -> msg.
text: type.                  text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
  ISOinitRule1:
  forall B : princ.
  forall KAB : shrK.
  forall text2 : text.
```

```
  forall text1 : text.
    empty
  => exists NA : nonce.
     net (text2 & enc KAB (NA & B & text1)).
}


receiver:
forall B : princ.
{
  ISORecRule2:
  forall A : princ.
  forall KAB : shrK.
  forall text2 : text.
  forall text1 : text.
    net (text2 & enc KAB (NA & B & text1))
  => empty.
}

)
```

ISO Symmetric Key Two-Pass Unilateral Authentication Protocol

Purpose

In this protocol principal A is authenticated by the verifier B by the means of challenge-response.

Informal Specification

1. B -> A: Rb, text1
2. A -> B: text3, E(Kab:Rb,B,text2)

MSR Specification

```
--- Author: Rishav Bhowmick
--- Date: 28 May 2007
--- ISO Symmetric Key Two-Pass Unilateral Authentication Protocol
---
---     B -> A: Rb,Text1
---       A -> B: Text3, E(Kab:Ra,B,text2)
(reset)
(
module ISOTwoPassUni
export * .

shrK  : type.
randomNo: type.                          randomNo  <: msg.
enc : shrK -> msg -> msg.
text: type.                  text <: msg.

net : msg -> state.

initiator:
forall B : princ.
{

                  exists LA : randomNo. -> state.
  ISO2initRule1:
  forall text1 : text.
  empty
  => exists rB : randomNo.
    net(rB & text1), LA rB.

 ISO2initRule3:
  forall A:princ.
  forall KAB : shrK.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
    net(text3 & enc KAB (rB & B & text2)), LA rB
  => empty.
}

reciever:
forall A : princ.
{
  ISOinitRule2:
  forall B : princ.
```

```
  forall KAB : shrK.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
   net(rB & text1)
  => net(text3 & enc KAB (rB & B & text2)).
}

)
```

ISO Symmetric Key Two-Pass Mutual Authentication Protocol

Purpose

In this protocol each principal establishes that the other is operational with the help of a shared secret key. It consists of two independent uses of one-pass authentication protocol.

Informal Specification

1. A -> B: text2, E(Kab:[Ta|Na],B,text1)
2. B -> A: text2, E(Kab:[Tb|Nb],A,text3)

MSR Specification

```
--- Author: rishav
--- Date: 29 May 2007
--- ISO Symmetric Key Two-Pass Mutual Authentication Protocol
---
---     A -> B: Text2, E(Kab:[Ta|Na],B,text1)
---       B -> A: Text4, E(Kab:[Tb|Nb],A,text3)

(
module ISOTwoPassMut
export * .

shrK  : type.                              pubK <: msg.
nonce : type.                            nonce  <: msg.
```

```
enc : shrK -> msg -> msg.
text: type.                    text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
  ISO3initRule1:
  forall B : princ.
  forall KAB : shrK.
  forall text2 : text.
  forall text1 : text.
    empty
  => exists nA : nonce.
     net (text2 & enc KAB (nA & B & text1)).
}

---they are independent
recicever:
forall B : princ.
{
  ISO3initRule2:
  forall A : princ.
  forall KAB : shrK.
  forall text3 : text.
  forall text4 : text.
   empty
  => exists nB : nonce.
     net (text4 & enc KAB (nB & A & text3)).
}

)
```

ISO Symmetric Key Three-Pass Mutual Authentication Protocol

Purpose

In this protocol random numbers are used to help authentication. B checks for the presence of B and Rb in the message 2 and A checks for Ra and Rb in the message.

Informal Specification

1. B -> A: Rb,Text1
2. A -> B: Text3, E(Kab:Ra,Rb,B,text2)
3. B -> A: Text5, E(Kab:Ra,Rb,text4)

MSR Specification

```
--- Author: rishav
--- Date: 29 May 2007
--- ISO Symmetric Key Three-Pass Mutual Authentication Protocol
---  B -> A: Rb,Text1
---  A -> B: Text3, E(Kab:Ra,Rb,B,text2)
---  B -> A: Text5, E(Kab:Ra,Rb,text4)
---
(reset)
(
module ISOThreePassMutual
export * .

key : type.
shrk : princ -> princ -> type.          shrk A B <: key.
randomNo: type.                           randomNo  <: msg.

enc : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall B : princ.
{
  ISO4initRule1:
    forall text1 : text.
    empty
  => exists Rb : randomNo.
    net(Rb & text1).

 ISO4initRule3:
  forall A : princ.
  forall Rb : randomNo.
```

```
  forall Ra : randomNo.
  forall Kab : shrk A B.
  forall text3 : text.
  forall text2 : text.
  forall text4 : text.
  forall text5 : text.
     net(text3 & enc Kab (Rb & Ra & B & text2))
  => net(text5 & enc Kab (Rb & Ra & text4)).
}


reciever:
forall A : princ.
{
  ISO4initRule2:
  forall B : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
  forall Ka-1 : pvtk.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
   net(Rb & text1)
  =>exists Ra : randomNo.
    net(text3 & enc Kab (Rb & Ra & B & text2)).

  ISO4initRule4:
  forall B : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
  forall text4 : text.
  forall text5 : text.
    net(text5 & enc Kab (Rb & Ra & text4))
  => empty.
}
)
```

Using Non-reversible function

Purpose

The protocol allows the principal to generate a new session key K. B checks the value of f(Rb) and then sends f(Ra) encrypted with K sent by A which A checks if

it is correct or not.

Informal Specification

1. B -> A: B,Rb
2. A -> B: A, E(Kab:f(Rb),Ra,A,K)
3. B -> A: B, E(K:f(Ra))

MSR Specification

```
--- Author: rishav
--- Date: 29,30 May 2007
--- Using Non-reversible function
---
---     B -> A: B,Rb
---       A -> B: A, E(Kab:f(Rb),Ra,A,K)
---       B -> A: B, E(K:f(Ra))
(reset)
(
module nonRevFunc
export * .

shrK  : type.                                shrK <: msg.
randomNo: type.                              randomNo  <: msg.
enc : shrK -> msg -> msg.
f : randomNo -> msg.

net : msg -> state.

initiator:
forall B : princ.
{
                  exists LA : randomNo -> princ -> state.
  revFuncRule1:
  forall A : princ.
   empty
  => exists rB : randomNo.
    net(rB & B), LA rB B.


 revFuncRule3:
```

```
  forall A : princ.
  forall rB : randomNo.
  forall rA : randomNo.
  forall K : shrK.
 net(A & enc KAB (f rB & rA & A & K)), LA rB B
  => net(B & enc K (f rA)).
}

reciever:
forall A : princ.
{
                  exists LB : randomNo -> princ -> state.
  revFuncRule2:
  forall B : princ.
  forall rB : randomNo.
   net(rB & B)
  => exists K : shrK.
     exists rA : randomNo.
     net(A & enc KAB (f rB & rA & A & K)), LB rA A.


 revFuncRule4:
  forall B : princ.
  forall rB : randomNo.
  forall rA : randomNo.
  net(B & enc K (f rA)), LB rA A
  => empty.
}


)
```

Findings

After declaring the function f, while encrypting the random number with a key, MSR implementation demanded a set of parentheses.

Andrew Secure RPC Protocol

Purpose

This protocol is intended to distribute a new session key between two principals A and B. The final message contains N'B which can be used in future messages as a handshake number.

Informal Specification

1. A -> B: A,E(Kab:Na)
2. B -> A: E(Kab: Na+1, Nb)
3. A -> B: E(Kab: Na+1)
4. B -> A: E(Kab: K'ab, N'b)

MSR Specification

```
--- Author: rishav
--- Date: 1 June 2007
---Andrew Secure RPC Protocol
---          A -> B: A,E(Kab:Na)
---          B -> A: E(Kab: Na+1, Nb)
---          A -> B: E(Kab: Na+1)
---          B -> A: E(Kab: K'ab, N'b)
(reset)
(
module AndrewSecureRPC
export * .

key : type.
stk : princ -> princ -> type.          stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.          ltk A B <: key.
nonce : type.                           nonce  <: msg.

enc : key -> msg -> msg.
inc : nonce -> nonce.
```

```
net : msg -> state.

initiator:
forall A : princ.
{
      exists LA1 : nonce -> {B : princ} ltk A B -> state.
      exists LA2 : {B : princ} ltk A B -> state.

RPCRule1:
  forall B : princ.
  forall KAB : ltk A B.
    empty
  => exists nA : nonce.
    net(A & enc KAB (nA)),
     LA1 nA B KAB.

RPCRule3:
  forall B : princ.
  forall nB : nonce.
  forall nA : nonce.
  forall KAB : ltk A B.
  net(enc KAB ((inc nA) & nB)),
---  LA1(nA, B, KAB)
     LA1 nA B KAB
  => net(enc KAB (inc nB)), LA2 B KAB.

RPCRule5:
  forall B : princ.
  forall n'B : nonce.
  forall KAB : ltk A B.
  forall K'AB : stk A B.
  net(enc KAB (K'AB & n'B)),
      LA2 B KAB
  => empty.

}


reciever:
forall B : princ.
{
      exists LB1 : nonce -> {A : princ} ltk A B -> state.

RPCRule2:
  forall A : princ.
  forall nA : nonce.
  forall nB : nonce.
```

```
  forall KAB : ltk A B.
  forall K'AB : stk A B.
    net(A & enc KAB (nA))
  => exists nB : nonce.
     net(enc KAB ((inc nA) & nB)),
     LB1 nB A KAB.

RPCRule4:
  forall A : princ.
  forall nB : nonce.
  forall KAB : ltk A B.
  net(enc KAB (inc NB)),
     LB1 NB A KAB
  => exists n'B : nonce.
     exists K'AB : stk A B.
     net(enc KAB (K'AB & n'B)).

}

)
```

Findings

Easy implementation of two sets of keys - long term and short term.
Reconstruction works very well here.

Denning Sacco Protocol

Purpose

Denning and Sacco suggested the use of time stamps to remove the freshness flaw in Needham Schroeder protocol. It cut shorts the size of Needham Schroeder protocol too.

Informal Specification

1. A -> S: A,B
2. S -> A: E(Kas: B, Kab, T, E(Kbs: A, Kab, T))

3. A -> B: E(Kbs: A, Kab, T)

MSR Specification

```
--- Author: rishav
--- Date: 10 June 2007
--- Denning Sacco Protocol
---      A -> S: A,B
---      S -> A: E(Kas: B, Kab, T, E(Kbs: A, Kab, T))
---      A -> B: E(Kbs: A, Kab, T)
---
(reset)
(
module denningSacco
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
time : type.                            time  <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{

DSRule1:
   empty =>  net(A & B).

DSRule3:
   forall KAS : ltk A S.
   forall KAB : stk A B.
   forall KBS : ltk B S.
   net(enc KAS (B & KAB & T & enc KBS (A & KAB & T)))
 => net(enc KBS (A & KAB & T)).
}

receiver:
forall B : princ.
```

```
{

DSRule4:
   forall KAB : stk A B.
   forall KBS : ltk B S.
   net(enc KBS (A & KAB & T))
 => empty.
}

server:
forall S : server.
{
DSRule2:
   forall KAS : ltk A S.
   forall KBS : ltk B S.
   net(A & B)
 => exists KAB : stk A B.
    net(enc KAS (B & KAB & T & enc KBS (A & KAB & T))).

}

)
```

Otway-Rees Protocol

Purpose

The nonce M identifies the session number.Kas and Kbs are symmetric keys whose values are initially known only by A and S, respectively B and S. Kab is a fresh symmetric key generated by S in message 3 and distributed to B, directly in message 3, and to A, indirectly, when B forwards blindly {Na, Kab}Kas to A in message 4.

Informal Specification

1. A -> B : M,A,B,E(Kas: Na,M,A,B)
2. B -> S : M,A,B,E(Kas: Na,M,A,B),E(Kbs: Nb,M,A,B)

3. S -> B : M,E(Kas: Kab,Na),E(Kbs: Kab,Nb)
4. B -> A : M,E(Kas: Kab,Na)

MSR Specification

```
--- Author: rishav
--- Date: 13 June 2007
--- Otway-Rees Protocol
---     A -> B : M,A,B,E(Kas: Na,M,A,B)
---     B -> S : M,A,B,E(Kas: Na,M,A,B),E(Kbs: Nb,M,A,B)
---     S -> B : M,E(Kas: Kab,Na),E(Kbs: Kab,Nb)
---     B -> A : M,E(Kas: Kab,Na)
(reset)
(
module otwayrees
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce  <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
                           exists LA : nonce -> nonce -> {S :
princ} ltk A S -> state.
ORRule1:
 forall Kas : ltk A S.
  empty =>
  exists M : nonce.
  exists Na : nonce.
  net(M & A & B & enc Kas (Na & M & A & B)), LA M Na S Kas.

ORRule5:
   forall Kas : ltk A S.
   forall Kab : stk A B.
```

```
   forall M : nonce.
   forall Na : nonce.
   net(M & enc Kas (Na & Kab)), LA M Na S Kas
     => empty.
--- when the L wasnt declared, Na declaration with forall wasnt
necessary...but after that..reconstruction was not possible.

}


receiver:
forall B : princ.
{
                            exists LB : nonce -> nonce -> {S :
princ} ltk B S -> state.
ORRule2:
   forall Kab : stk A B.
   forall Kbs : ltk B S.
   forall Kas : ltk A S.
   forall M : nonce.
---   forall Na : nonce.
    net(M & A & B & enc Kas (Na & M & A & B))
 => exists Nb : nonce.
    net(M & A & B & enc Kas (Na & M & A & B) & enc Kbs (Nb & M &
A & B)), LB M Nb S Kbs.

ORRule4:
   forall Kas : ltk A S.
   forall Kab : stk A B.
   forall Kbs : ltk B S.
   forall M : nonce.
---   forall Na : nonce.
  forall Nb : nonce.
   net(M & enc Kas (Na & Kab) & enc Kbs (Nb & Kab)), LB M Nb S
Kbs
 => net(M & enc Kas (Na & Kab)).

}


server:
forall S : server.
{
ORRule3:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall M : nonce.
---   forall Na : nonce.
```

```
   forall Nb : nonce.
   net(M & A & B & enc Kas (Na & M & A & B) & enc Kbs (Nb & M & A
& B))
 => exists Kab : stk A B.
    net(M & enc Kas (Na & Kab) & enc Kbs (Nb & Kab)).

}

)
```

Findings

Reconstruction not possible to large extent, thus the variables needed to be defined using forall in most cases.

Wide Mouthed Frog Protocol

Purpose

In this protocol A generates the session key. S checks if the timestamp is timely then it forwards the key to B with its own timestamp. B checks if the timestamp is later than any other message it has received form S.

Informal Specification

1. A -> S : A, E(Kas: Ta, B, Kab)
2. S -> B : E(Kbs: Ts, A, Kab)

More Commonly Used Representation:
Tx - Timestamp generated by X

MSR Specification

```
--- Author: rishav
--- Date: 16 June 2007
--- Wide Mouthed Frog Protocol
---     A -> S : A, E(Kas: Ta, B, Kab)
```

```
---     S -> B : E(Kbs: Ts, A, Kab)

(reset)
(
module WMFprotocol
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
time : type.                            time <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{

WMFRule1:
--- forall B : princ.
--- forall Kas : ltk A S.
--- forall Ta : time.
  empty =>
  exists Kab : stk A B.
   net(A & enc Kas (Ta & B & Kab)).

}


receiver:
forall B : princ.
{

WMFRule3:
---   forall Kab : stk A B.
  forall Kbs : ltk B S.
  forall Ts : time.
    net(enc Kbs (Ts & A & Kab))
 => empty.
}


server:
forall S : server.
```

```
{
WMFRule2:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Ts : time.
   net(A & enc Kas (Ta & B & Kab))
 => net(enc Kbs (Ts & A & Kab)).
}

)
```

Findings

Reconstruction possible in most places. But in some places MSR does not allow reconstruction when it is possible.

Yahalom

Purpose

The server generates the shared key Kab and sends it to A directly and B indirectly.

Informal Specification

1. A -> B : A,Na
2. B -> S : B,E(Kbs : A, Na, Nb)
3. S -> A : E(Kas : B, Kab, Na, Nb), E(Kbs: A, Kab)
4. A -> B : E(Kbs: A, Kab), E(Kab:Nb)

MSR Specification

```
--- Author: rishav
--- Date: 17 June 2007
---Yahalom
---     A -> B : A,Na
---     B -> S : B,E(Kbs : A, Na, Nb)
---     S -> A : E(Kas : B, Kab, Na, Nb), E(Kbs: A, Kab)
---     A -> B : E(Kbs: A, Kab), E(Kab:Nb)

(reset)
(
module yahalom
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
                  exists LA : nonce -> state.

YRule1:
  empty
  => exists Na : nonce.
   net(A & Na), LA Na.

YRule4:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(enc Kas (B & Kab & Na & Nb) & enc Kbs (A & Kab)), LA Na
   => net(enc Kbs (A & Kab) & enc Kab Nb).
}
```

```
receiver:
forall B : princ.
{

                     exists LB : nonce -> nonce -> {S : server}
ltk B S -> state.

YRule2:
  forall A : princ.
  forall S : server.
  forall Kbs : ltk B S.
  forall Na : nonce.
    net(A & Na)
 => exists Nb : nonce.
   net(B & enc Kbs (A & Na & Nb)), LB Na Nb S Kbs.

YRule5:
   forall S : server.
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(enc Kbs (A & Kab) & enc Kab Nb), LB Na Nb S Kbs
   => empty.

}


server:
forall S : server.
{
YRule3:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(B & enc Kbs (A & Na & Nb))
     => exists Kab : stk A B.
       net(enc Kas (B & Kab & Na & Nb) & enc Kbs (A & Kab)).
}

)
```

Carlsen's Secret Key initiative protocol

Purpose

The server generates Kab and passes it on to A and B.

Informal Specification

1. A -> B : A,Na
2. B -> S : A,Na,B,Nb
3. S -> B : E(Kbs: Kab, Nb, A), E(Kas : B, Kab, Na, B)
4. B -> A : E(Kas : B, Kab, Na, B), E(Kab : Na), N'b
5. A -> B : E(Kab : N'b)

MSR Specification

```
--- Author: rishav
--- Date: 18 June 2007
---Carlsen's Secret Key initiative protocol
---     A -> B : A,Na
---     B -> S : A,Na,B,Nb
---     S -> B : E(Kbs: Kab, Nb, A), E(Kas : B, Kab, Na, B)
---     B -> A : E(Kas : B, Kab, Na, B), E(Kab : Na), N'b
---     A -> B : E(Kab : N'b)

(reset)
(
module carlsen
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
```

```
ltk : princ -> princ -> type.          ltk A B <: key.
nonce : type.                           nonce <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
                exists LA : nonce -> state.
CRule1:
  empty
  => exists Na : nonce.
   net(A & Na), LA Na.

CRule5:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Na : nonce.
   forall N'b : nonce.
    net(enc Kas (B & Kab & Na) & enc Kab Na & N'b), LA Na
   => net(enc Kab N'b).
}


receiver:
forall B : princ.
{
                  exists LB1 : princ -> nonce -> nonce -> state.
                  exists LB2 : nonce -> state.
CRule2:
  forall A : princ.
  forall Kbs : ltk B S.
  forall Na : nonce.
    net(A & Na)
 => exists Nb : nonce.
   net(B & Nb & A & Na), LB1 A Na Nb.

CRule4:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
     net(enc Kas (B & Kab & Na) & enc Kbs (A & Kab & Nb)), LB1 A
Na Nb
  => exists N'b : nonce.
```

```
   net(enc Kas (B & Kab & Na) & enc Kab Na & N'b), LB2 N'b.


CRule6:
   forall Kab : stk A B.
   forall N'b : nonce.
    net(enc Kab N'b), LB2 N'b
   => empty.

}


server:
forall S : server.
{
CRule3:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(B & Nb & A & Na )
     => exists Kab : stk A B.
        net(enc Kas (B & Kab & Na) & enc Kbs (A & Kab & Nb)).
}

)
```

ISO Five-Pass Authentication protocol

Purpose

The server creates the shared key Kab and sends it to A and B. Meanwhile, B encrypts the random number with this shared key and sends it to A. A uses Kas to get Kab which then A uses to encrypt other text to be sent to B.

Informal Specification

1. A -> B : Ra, Text1
2. B -> S : R'b, Ra, A, Text2

3. S -> B : Text5, E(Kbs: Kab, R'b, A, Text4), E(Kas : B, Kab, Ra, Text3)
4. B -> A : Text7, E(Kas : B, Kab, Ra, Text3), E(Kab : Ra, Rb, Text6)
5. A -> B : Text9, E(Kab : Ra, Rb, Text8)

MSR Specification

```
--- Author: rishav
--- Date: 19 June 2007
--- ISO Five-Pass Authentication protocol
---     A -> B : Ra, Text1
---     B -> S : R'b, Ra, A, Text2
---     S -> B : Text5, E(Kbs: Kab, R'b, A, Text4), E(Kas : B,
Kab, Ra, Text3)
---     B -> A : Text7, E(Kas : B, Kab, Ra, Text3), E(Kab : Ra,
Rb, Text6)
---     A -> B : Text9, E(Kab : Ra, Rb, Text8)

(reset)
(
module ISO5pass
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
---assuming random nos to be nonce
nonce : type.                            nonce <: msg.
text: type.                   text <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
                exists LA : nonce -> state.
IRule1:
  forall Text1 : text.
  empty
```

```
  => exists Ra : nonce.
   net(Text1 & Ra), LA Ra.

IRule5:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Text7 : text.
   forall Text3 : text.
   forall Text6 : text.
   forall Text9 : text.
   forall Text8 : text.
   forall Ra : nonce.
   forall Rb : nonce.
    net(Text7 & enc Kas (B & Kab & Ra & Text3) & enc Kab (Ra & Rb
& Text6)), LA Ra
   => net(Text9 & enc Kab (Ra & Rb & Text8)).
}


receiver:
forall B : princ.
{
                  exists LB1 : princ -> nonce -> nonce -> state.
                  exists LB2 : nonce -> nonce -> state.
IRule2:
   forall A : princ.
   forall Text1 : text.
   forall Text2 : text.
   forall Ra : nonce.
    net(Text1 & Ra)
 => exists R'b : nonce.
   net(Text2 & R'b & A & Ra), LB1 A Ra Rb.

IRule4:
   forall Text7 : text.
   forall Text3 : text.
   forall Text6 : text.
   forall Text4 : text.
   forall Text5 : text.
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Ra : nonce.
   forall R'b : nonce.
     net(Text5 & enc Kas (B & Kab & Ra & Text3) & enc Kbs (A &
Kab & R'b & Text4)), LB1 A Ra Rb
  => exists Rb : nonce.
```

```
   net(Text7 & enc Kas (B & Kab & Ra & Text3) & enc Kab (Ra & Rb
& Text6)), LB2 Ra Rb.


IRule6:
   forall Text9 : text.
   forall Text8 : text.
   forall Kab : stk A B.
   forall Rb : nonce.
   forall Ra : nonce.
    net(Text9 & enc Kab (Ra & Rb & Text8)), LB2 Ra Rb
   => empty.

}


server:
forall S : server.
{
IRule3:
   forall Text3 : text.
   forall Text4 : text.
   forall Text2 : text.
   forall Text5 : text.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Ra : nonce.
   forall R'b : nonce.
    net(Text2 & R'b & A & Ra)
     => exists Kab : stk A B.
        net(Text5 & enc Kas (B & Kab & Ra & Text3) & enc Kbs (A &
Kab & R'b & Text4)).
}

)
```

Findings

It was found that the purpose of reconstruction was not met due to Ls because they require the exact type to be specified as it can't be derived.

Woo and Lam authentication (Pi_f) protocol

Purpose

Kas is a long term symmetric key shared by A and S. Initially, A only knows Kas and the name of B, B only knows Kbs and S knows all shared keys, i.e. S knows the ``function'' shared.

Informal Specification

1. A -> B : A
2. B -> A : Nb
3. A -> B : E(Kas : B, Nb, A)
4. B -> S : E(Kbs : A, B, Nb, E(Kab : A, B, Nb)
5. S -> B : E(Kbs : A, B, Nb)

MSR Specification

```
--- Author: rishav
--- Date: 19 June 2007
--- Woo and Lam authentication (Pif) protocol
---     A -> B : A
---     B -> A : Nb
---     A -> B : E(Kas : B, Nb, A)
---     B -> S : E(Kbs : A, B, Nb, E(Kab : A, B, Nb)
---     S -> B : E(Kbs : A, B, Nb)

(reset)
(
module WLPf
```

```
export * .

server : type.                          server <: princ.
key : type.
---stk : princ -> princ -> type.          stk A B <: msg.  stk A
B <: key.
ltk : princ -> princ -> type.          ltk A B <: key.
nonce : type.                           nonce <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
--- Couldnt find out a way to have L without parameters
               exists LA : state.
PfRule1:
  empty
  => net A, LA.

PfRule3:
   forall Kas : ltk A S.
   forall Nb : nonce.
    net Nb, LA
   => net(enc Kas (A & B & Nb)).
}

receiver:
forall B : princ.
{
                  exists LB1 : princ -> nonce -> state.
                  exists LB2 : princ -> nonce -> state.
PfRule2:
    forall A : princ.
    net A
 => exists Nb : nonce.
   net Nb, LB1 A Nb.

PfRule4:
   forall A : princ.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Nb : nonce.
     net(enc Kas (A & B & Nb)), LB1 A Nb
  => net(enc Kbs (A & B & Nb & enc Kas (A & B & Nb))), LB2 A Nb.
```

```
PfRule6:
   forall A : princ.
   forall Kbs : ltk B S.
   forall Nb : nonce.
    net(enc Kbs (A & Nb & B)), LB2 A Nb
   => empty.

}


server:
forall S : server.
{
PfRule5:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Nb : nonce.
    net(enc Kbs (A & B & Nb & enc Kas (A & B & Nb)))
     => net(enc Kbs (A & Nb & B)).
}

)
```

Amended Needham Schroeder protocol

Purpose

An improved version of the original Needham Schroeder protocol.

Informal Specification

1. A -> B : A
2. B -> A : E(Kbs: A, Nb0)
3. A -> S : A, B, Na, E(Kbs: A, Nb0)
4. S -> A : E(Kas : A, Na, Kab, E(Kbs : A, Kab, Nb0))

5. A -> B : E(Kbs : A, Kab, Nb0)
6. B -> A : E(Kab : Nb)
7. A -> B : E(Kab : Nb-1)

MSR Specification

```
--- Author: rishav
--- Date: 23 June 2007
--- Amended Needham Schroeder protocol
---     A -> B : A
---     B -> A : E(Kbs: A, Nb0)
---     A -> S : A, B, Na, E(Kbs: A, Nb0)
---     S -> A : E(Kas : A, Na, Kab, E(Kbs : A, Kab, Nb0))
---     A -> B : E(Kbs : A, Kab, Nb0)
---     B -> A : E(Kab : Nb)
---     A -> B : E(Kab : Nb-1)

(reset)
(
module ANS
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce <: msg.

enc : key -> msg -> msg.
--- to decrement the nonce by one creating another nonce
dec : nonce -> nonce.

net : msg -> state.

initiator:
forall A : princ.
{
                    exists LA : state.
                    exists LA1 : nonce -> {B : princ} {S :
server} ltk B S -> ltk A S -> state.
                    exists LA2 : {B : princ} stk A B -> state.
```

```
ANSRule1:
  empty
  => net A, LA.

ANSRule3:
   forall B : princ.
   forall S : server.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Nb0 : nonce.
    net (enc Kbs (A & Nb0)), LA
   => exists Na : nonce.
      net(A & B & Na & enc Kbs (A & Nb0)), LA1 Nb0 B S Kbs Kas.

ANSRule5:
   forall B : princ.
   forall S : server.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Kab : stk A B.
   forall Nb0 : nonce.
     net(enc Kas (Na & B & Kab & enc Kbs (A & Nb0 & Kab))), LA1
Nb0 B S Kbs Kas
   => net(enc Kbs (A & Nb0 & Kab)), LA2 B Kab.

ANSRule7:
   forall B : princ.
   forall Kab : stk A B.
   forall Nb : nonce.
    net(enc Kab Nb), LA2 B Kab
   => net(enc Kab (dec Nb)).
}

receiver:
forall B : princ.
{
                  exists LB1 : princ -> nonce -> {S : server} ltk
B S -> state.
                  exists LB2 : nonce -> {A : princ} stk A B ->
state.
ANSRule2:
    forall A : princ.
    forall S : server.
    forall Kbs : ltk B S.
    net A
 => exists Nb0 : nonce.
   net (enc Kbs (A & Nb0)), LB1 A Nb0 S Kbs.
```

```
ANSRule6:
   forall A : princ.
   forall S : server.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Kab : stk A B.
   forall Nb0 : nonce.
     net(enc Kbs (A & Nb0 & Kab)), LB1 A Nb0 S Kbs
  => exists Nb : nonce.
     net(enc Kab Nb), LB2 Nb A Kab.

ANSRule8:
   forall A : princ.
   forall Kab : stk A B.
   forall Nb : nonce.
    net(enc Kab (dec Nb)), LB2 Nb A Kab
   => empty.

}


server:
---for S : server. if S is declared as global.
forall S : server.
{
ANSRule4:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Nb0 : nonce.
    net(A & B & Na & enc Kbs (A & Nb0))
     => exists Kab : stk A B.
        net(enc Kas (Na & B & Kab & enc Kbs (A & Nb0 & Kab))).
}


)
```

Findings

It was easily implemented but with a weird local state predicate problems. Initially it gave unsolvable constraints error. But later after doing a few changes and coming back to the same state, it worked. This was performed both by Dr Iliano and me. There can be 2 possibilities. One, We must have overlooked the error before and somehow the error was rectified unknowingly. Two, MSR2 has bugs which need to be de-bugged.

Needham-Schroeder Signature Protocol

Purpose

A has a one way function that creates a digest CS of the message and sends it to S using shared key Kas. S sends back the digest and A to A using its own prrivate key Kss. A then passes this on to B who then sends it back to S. S checks if the digest is equal.

Informal Specification

1. A -> S : A, E(Kas : CS)
2. S -> A : E(Kss : A, CS)
3. A -> B : Message, E(Kss : A, CS)
4. B -> S : B, E(Kss : A, CS)
5. S -> B : E(Kbs : A, CS)

MSR Specification

```
--- Author: rishav
--- Date: 6 August 2007
--- Needham-Schroeder Signature Protocol
---     A -> S : A, E(Kas : CS)
---     S -> A : E(Kss : A, CS)
---     A -> B : Message, E(Kss : A, CS)
---     B -> S : B, E(Kss : A, CS)
---     S -> B : E(Kbs : A, CS)

(reset)
(
```

```
module NSS
export * .

server : type.                          server <: princ.
key : type.
ltk : princ -> princ -> type.           ltk A B <: key.
ltks : princ -> type.                 ltks S <: key.
digest : type.                          digest <: msg.
dig : msg -> digest.

enc : key -> msg -> msg.

net : msg -> state.


initiator:
forall A : princ.
{
                  exists LA : msg -> state.
NSSRule1:
  forall Kas : ltk A S.
  forall mess : msg.
  empty
  => net(A & enc Kas (dig mess)), LA mess.

NSSRule3:
   forall Kss : ltks S.
   forall mess : msg.
   net(enc Kss (A & dig mess))
   => net(mess & enc Kss (A & dig mess)), LA mess.
}

receiver:
forall B : princ.
{
                  exists LB : msg -> princ -> state.
NSSRule4:
   forall A : princ.
   forall Kss : ltks S.
   forall mess : msg.
     net(mess & enc Kss (A & dig mess))
  => net(B & enc Kss (A & dig mess)), LB mess A.


NSSRule6:
   forall A : princ.
   forall Kbs : ltk B S.
   forall mess : msg.
```

```
    net(enc Kbs (A & dig mess)), LB mess A
   => empty.

}


server:
forall S : server.
{
                  exists LB : msg -> {S : server} ltks S -> state.
NSSRule2:
   forall S : server.
   forall Kas : ltk A S.
   forall Kss : ltks S.
   forall mess : msg.
   net(A & enc Kas (dig mess))
     => net(enc Kss (A & dig mess)), LB mess S Kss.

NSSRule5:
   forall S : server.
   forall Kss : ltks S.
   forall Kbs : ltk B S.
   forall mess : msg.
     net(B & enc Kss (A & dig mess)), LB mess S Kss
  => net(enc Kbs (A & dig mess)).


}


)
```

Kerberos version 5 Protocol

Purpose

(taken from SPORE)
C is a client,
S is a a server (C wants to communicate with S),
U is a user on behalf of which A and S communicate,
G is a ticket granting server,
A is a key distribution center (trusted server).

The keys Kag and Kgs are long term symmetric key whose values are supposed to be known initially only by, A and G, respectively G and S.
L1 and L2 are lifetimes, N1 and N2 are nonces. Tstart, Texpire, T'start, T;expire are time stamps which define the interval of validity of the ticket in which they are contained.
U is a user on behalf of whom the client C communicates. In particular, C initially knows the value of the key Ku.
The key Kcg is freshly generated by A for communication between C and G, and in transmitted to C in message 2, encrypted by Ku, and indirectly to G, in the ticket {U, C, G, Kcg, Tstart, Texpire}Kag which C transmits blindly to G in message 3.
The authentificator {C, T1}Kcg is used by G to check timeliness of the ticket.
The key Kcs is freshly generated by G for communication between C and G, and in transmitted to C in message 4, encrypted by Kcg, and indirectly to S, in the ticket {U, C, S, Kcs, T'start, T'expire}Kgs which C transmits blindly to S in message 5.

Informal Specification

1. C -> A : U, G, L1, N1
2. A -> C : U, Tcg, E(Ku: G, Kcg, Tstart, Texpire, N1)
3. C -> G : S, L2, N2, Tcg, Acg
4. G -> C : U, Tcs, E(Kcg: S, Kcs, T'start, T'expire, N2)
5. C -> S : Tcs, Acs
6. S -> C : E(Kcs : T')

MSR Specification

```
--- Author: rishav
--- Date: 6 August 2007
--- Kerberos version 5 Protocol
---     C -> A : U, G, L1, N1
---     A -> C : U, Tcg, E(Ku: G, Kcg, Tstart, Texpire, N1)
---     C -> G : S, L2, N2, Tcg, Acg
```

```
---     G -> C : U, Tcs, E(Kcg: S, Kcs, T'start, T'expire, N2)
---     C -> S : Tcs, Acs
---     S -> C : E(Kcs : T')


(reset)
(
module KERB
export * .

client : type.                        client <: princ.
server : type.                           server <: princ.
TGS : type.                               TGS <: server.
KDC : type.                               KDC <: server.
key : type.
ltk : princ -> princ -> type.             ltk A B <: key.
stk : princ -> princ -> type.             stk A B <: key.      stk A
B <: msg.
ltks : princ -> type.                  ltks S <: key.       ltks S
<: msg.
enc : key -> msg -> msg.
time : type.                              time <: msg.
nonce : type.                             nonce <: msg.
net : msg -> state.
clock : time -> state.

TGT : princ -> time -> time -> {C : client} {G : TGS} stk C G ->
{A : KDC} {G : TGS} ltk A G -> state.
TGT' : princ -> time -> time -> {C : client} {S : server} stk C S
-> state.


initiatorClient:
forall C : client.
{
                  exists LC : princ -> TGS -> nonce -> state.
KRule1:
  forall U : princ.
  forall G : TGS.
  empty
  => exists L1 : time.
     exists N1 : nonce.
     net(U & G & L1 & N1), LC U G N1.

KRule3:
  forall A : KDC.
  forall U : princ.
  forall G : TGS.
```

```
  forall Kcg : stk C G.
  forall Ku : ltks U.
  forall Tcg : msg.
  forall N1 : nonce.
  forall Tstart : time.
  forall Texpire : time.
  forall Kag : ltk A G.
   net(U & Tcg & enc Ku (G & Kcg & Tstart & Texpire & N1)), LC U
G N1
  => TGT U Tstart Texpire C G Kcg A G Kag.

}


keyDistributionCentre:
forall A : KDC.
{

KRule2:
  forall U : princ.
  forall G : TGS.
  forall Tstart : time.
  forall Texpire : time.
  forall Ku : ltks U.
  forall N1 : nonce.
    net(U & G & L1 & N1)
   => exists Kcg : stk C G.
      net(U & enc Kag (U & C & G & Kcg & Tstart & Texpire) & enc
Ku (G & Kcg & Tstart & Texpire & N1)).

}

initiatorClient2:
forall C : client.
{
                  exists LC2 : KDC -> princ -> server -> {G : TGS}
stk C G -> nonce -> state.
KRule4:
  forall A : KDC.
  forall U : princ.
  forall G : TGS.
  forall S : server.
  forall Kcg : stk C G.
  forall Kag : ltk A G.
  forall Tcg : msg.
  forall Tstart : time.
  forall T : time.
  forall Texpire : time.
```

```
  TGT U Tstart Texpire C G Kcg A G Kag
  => exists L2 : time.
     exists N2 : nonce.
     net(S & Tcg & enc Kcg (C & T) & L2 & N2), LC2 A U S G Kcg
N2.

KRUle6:
  forall A : KDC.
  forall U : princ.
  forall S : server.
--- forall G needed for LC2 or else gives unsolved constraints
error
  forall G : TGS.
  forall Kcg : stk C G.
  forall Kcs : stk C S.
  forall T'start : time.
  forall T'expire : time.
  forall N2 : nonce.
    net(U & enc Kcg (U & C & G & S & Kcs & T'start & T'expire) &
enc Kcg (S & Kcs & T'start & T'expire & N2)), LC2 A U S G Kcg N2
  => TGT' U T'start T'expire C S Kcs.
}


ticketGrantingServer:
forall G : TGS.
{
---needs everything to be declared>>> weird as reconstruction is
supposed to work..for eg time.
KRule5:
  forall C : princ.
  forall A : KDC.
  forall U : princ.
  forall S : server.
  forall Kcg : stk C G.
  forall T : time.
  forall Tstart : time.
  forall Texpire : time.
  forall T'start : time.
  forall T'expire : time.
  forall L2 : time.
  forall N2 : nonce.
    net(S & enc Kag (U & C & G & Kcg & Tstart & Texpire) & enc
Kcg (C & T) & L2 & N2)
   => exists Kcs : stk C S.
      net(U & enc Kcg (U & C & G & S & Kcs & T'start & T'expire)
& enc Kcg (S & Kcs & T'start & T'expire & N2)) if clock T.
```

```
}

initiatorClient3:
forall C : client.
{
                  exists C3 : {S : server} stk C S -> time ->
state.
KRule7:
  forall U : princ.
  forall G : TGS.
  forall S : server.
  forall Kcg : stk C G.
  forall T'start : time.
  forall T' : time.
  forall Kcs : stk C S.
  forall T'expire : time.
  TGT' U T'start T'expire C S Kcs
  => net(enc Kcg (U & C & G & S & Kcs & T'start & T'expire) & enc
Kcs (C & T')), C3 S Kcs T' if clock T'.

KRUle8:
  forall S : server.
  forall Kcs : stk C S.
  forall T' : time.
    net(enc Kcs T'), C3 S Kcs T'
  => empty.
}

server:
forall S : server.
{

KRule9:

  forall U : princ.
  forall G : TGS.
  forall S : server.
  forall Kcg : stk C G.
  forall T'start : time.
  forall T' : time.
  forall Kcs : stk C S.
  forall T'expire : time.
  forall C : client.
   net(enc Kcg (U & C & G & S & Kcs & T'start & T'expire) & enc
Kcs (C & T'))
  => net(enc Kcs T').
}
)
```

Findings

A very good example of how MSR handles repetition. It took time to only understand the concept. But once cleared, representing kerberos didn't take much time.

Neuman Stubblebine

Purpose

It involves bringing about the exchange of some ticket and the second is a protocol for multiple authentications.

Informal Specification

1. A -> B : A,Na
2. B -> S : B,E(Kbs : A, Na, tb), Nb
3. S -> A : E(Kas : B, Kab, Na, tb), E(Kbs : A, Kab, tb), Nb
4. A -> B : E(Kbs : A, Kab, tb), E(Kab : Nb)
5. A -> B : N'a, E(Kbs : A, Kab, tb)
6. B -> A : N'B, E(Kab : N'a)
7. A -> B : E(Kab : N'b)

More Commonly Used Representations
tb - timestamp

MSR Specification

```
--- Author: rishav
--- Date: 17 August 2007
--- Neuman Stubblebine
---     A -> B : A,Na
---     B -> S : B,E(Kbs : A, Na, tb), Nb
---     S -> A : E(Kas : B, Kab, Na, tb), E(Kbs : A, Kab, tb), Nb
---     A -> B : E(Kbs : A, Kab, tb), E(Kab : Nb)
---     A -> B : N'a, E(Kbs : A, Kab, tb)
---     B -> A : N'B, E(Kab : N'a)
---     A -> B : E(Kab : N'b)

(reset)
(
module neuman
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce <: msg.
time : type.                             time <: msg.

enc : key -> msg -> msg.

net : msg -> state.

GL : time -> {A : princ} {B : princ} stk A B -> state.

initiator:
forall A : princ.
{
                  exists L: nonce -> state.
YRule1:
  empty
  => exists Na : nonce.
   net(A & Na), L Na.

YRule4:
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
```

```
    net(enc Kas (B & Kab & Na & Tb) & enc Kbs (A & Kab & Tb) &
Nb), L Na
   => net(enc Kbs (A & Kab & Tb) & enc Kab Nb).
}


receiver:
forall B : princ.
{
                  exists L: nonce -> nonce -> {B : princ} {S :
server} ltk B S -> state.
YRule2:
  forall A : princ.
  forall S : server.
  forall Kbs : ltk B S.
  forall Na : nonce.
    net(A & Na)
 => exists Nb : nonce.
   net(B & enc Kbs (A & Na & Tb) & Nb), L Na Nb B S Kbs.

YRule5:
   forall A : princ.
   forall S : server.
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
   forall Tb : time.
    net(enc Kbs (A & Kab & Tb) & enc Kab Nb), L Na Nb B S Kbs
   => GL Tb A B Kab.

}


server:
forall S : server.
{
YRule3:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(B & enc Kbs (A & Na & Tb) & Nb)
     => exists Kab : stk A B.
       net(enc Kas (B & Kab & Na & Tb) & enc Kbs (A & Kab & Tb) &
Nb).
}
```

```
initiator2:
forall A : princ.
{
                  exists L: nonce -> {A : princ} {B : princ} stk A
B -> state.
YRule5:
  forall B : princ.
  forall Tb : time.
  forall Kab : stk A B.
  forall Kbs : ltk B S.
  GL Tb A B Kab
  => exists N'a : nonce.
   net(N'a & enc Kbs (A & Kab & Tb)), L N'a A B Kab.

YRule7:
  forall B : princ.
  forall Kab : stk A B.
  forall N'a : nonce.
   net(N'b & enc Kab (N'a)), L N'a A B Kab
  => net(enc Kab (N'b)).

}


receiver2:
forall B : princ.
{
                  exists L: nonce -> {A : princ} {B : princ} stk A
B -> state.
YRule6:
  forall A : princ.
  forall Kab : stk A B.
  forall Kbs : ltk B S.
  forall N'a : nonce.
  forall Tb : time.
   net(N'a & enc Kbs (A & Kab & Tb))
 => exists N'b : nonce.
   net(N'b & enc Kab (N'a)), L N'b A B Kab.

YRule8:
   forall N'b : nonce.
   forall Kab : stk A B.
   forall N'b : nonce.
   forall Tb : time.
    net(enc Kab (N'b)), L N'b A B Kab
   => empty.
```

```
}

)
```

Findings

This is another example of repeatation, although on a smaller scale than kerberos.

Kehne Langendorfer Schoenwalder

Purpose

Another repeated authentication protocol

Informal Specification

1. A -> B : Na, A
2. B -> S : Na, Nb, A, B
3. S -> B : E(Kbs: Kab, Nb, A), E(Kas : B, Kab, Na)
4. B -> A : E(Kas : B, Kab, Na), E(Kbb : tb, A, Kab), Nc, E(Kab:Na))
5. A -> B : E(Kab : Nc)

MSR Specification

```
--- Author: rishav
--- Date: 9 September 2007
--- Kehne Langendorfer Schoenwalder
---     A -> B : Na, A
---     B -> S : Na, Nb, A, B
---     S -> B : E(Kbs: Kab, Nb, A), E(Kas : B, Kab, Na)
---     B -> A : E(Kas : B, Kab, Na), E(Kbb : tb, A, Kab), Nc,
```

```
E(Kab:Na))
---     A -> B : E(Kab : Nc)

(reset)
(
module KLS
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
ltks : princ -> type.                   ltks B <: key.
nonce : type.                            nonce <: msg.
time : type.                            time <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
                exists LA : nonce -> state.
KLSRule1:
  empty
  => exists Na : nonce.
   net(A & Na), LA Na.

---tb is not recognised
KLSRule5:
   forall tb : time.
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Kbb : ltks B.
   forall Na : nonce.
   forall Nb : nonce.
    net(enc Kas (B & Kab & Na) & Nc & enc Kbb (tb & A & Kab) &
enc Kab Na), LA Na
   => net(enc Kab Nc).
}


receiver:
forall B : princ.
{
```

```
                exists LB1 : nonce -> nonce -> state.
                exists LB2 : nonce -> {A : princ} {B : princ}
stk A B -> state.
KLSRule2:
   forall A : princ.
   forall Na : nonce.
    net(A & Na)
 => exists Nb : nonce.
   net(Nb & A & Na & B), LB1 Na Nb.

KLSRule4:
   forall A : princ.
   forall tb : time.
   forall Kab : stk A B.
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Kbb : ltks B.
   forall Na : nonce.
   forall Nb : nonce.
     net(enc Kbs (A & Kab & Nb) & enc Kas (B & Kab & Na)), LB1 Na
Nb
  => exists Nc : nonce.
     net(enc Kas (B & Kab & Na) & Nc & enc Kbb (tb & A & Kab) &
enc Kab Na), LB2 Nc A B Kab.

KLSRule6:
   forall Kab : stk A B.
   forall Nc : nonce.
   net(enc Kab Nc), LB2 Nc A B Kab
   => empty.

}


server:
forall S : server.
{
KLSRule3:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
   forall Nb : nonce.
    net(Nb & A & Na & B)
     => exists Kab : stk A B.
        net(enc Kbs (A & Kab & Nb) & enc Kas (B & Kab & Na)).
}

)
```

Kao Chow Repeated Authentication Protocol

Purpose

It is not susceptible to the attacks on the Neuman Stubblebine

Informal Specification

1. A -> S : A,B,Na
2. S -> B : E(Kas: Na,Kab,A,B),E(Kbs: Na,Kab,A,B)
3. B -> A : E(Kas: Na,Kab,A,B),E(Kab: Na), Nb
4. A -> B : E(Kab: Nb)

MSR Specification

```
--- Author: rishav
--- Date: 10 September 2007
--- Kao Chow Repeated Authentication Protocol
---     A -> S : A,B,Na
---     S -> B : E(Kas: Na,Kab,A,B),E(Kbs: Na,Kab,A,B)
---     B -> A : E(Kas: Na,Kab,A,B),E(Kab: Na), Nb
---     A -> B : E(Kab: Nb)
(reset)
(
module KaoChow
export * .

server : type.                          server <: princ.
key : type.
stk : princ -> princ -> type.           stk A B <: msg.  stk A B
<: key.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce  <: msg.

enc : key -> msg -> msg.
```

```
net : msg -> state.

initiator:
forall A : princ.
{
                           exists LA : nonce -> princ -> state.
KCule1:
  forall B : princ.
  empty =>
   exists Na : nonce.
  net(A & B & Na), LA Na B.

KCRule4:
   forall B : princ.
   forall Kab : stk A B.
   forall Kbs : ltk B S.
   forall Kas : ltk A S.
   forall Na : nonce.
   forall Na : nonce.
    net(enc Kas (A & B & Na & Kab) & enc Kab Na & Nb), LA Na B
  => net(enc Kab Nb).

}


receiver:
forall B : princ.
{
                           exists LB : nonce -> {A : princ} {B :
princ} stk A B -> state.
KCRule3:
   forall A : princ.
   forall Kab : stk A B.
   forall Kbs : ltk B S.
   forall Kas : ltk A S.
   forall Na : nonce.
    net(enc Kas (A & B & Na & Kab) & enc Kbs (A & B & Na & Kab))
 => exists Nb : nonce.
    net(enc Kas (A & B & Na & Kab) & enc Kab Na & Nb), LB Nb A B
Kab.

KCRule5:
   forall A : princ.
   forall Kab : stk A B.
   forall Nb : nonce.
   net(enc Kab Nb), LB Nb A B Kab
 => empty.
```

```
}


server:
forall S : server.
{
KCRule2:
   forall Kas : ltk A S.
   forall Kbs : ltk B S.
   forall Na : nonce.
  net(A & B & Na)
 => exists Kab : stk A B.
    net(enc Kas (A & B & Na & Kab) & enc Kbs (A & B & Na & Kab)).

}

)
```

ISO Public Key One-Pass Unilateral Authentication Protocol

Purpose

Self-explanatory.

Informal Specification

1. A -> B: CertA,[Ta|Na],B,text2,E(Ka-1: [Ta|Na],B,text1)

MSR Specification

```
--- Author: rishav
--- Date: 10 September 2007
--- ISO Public Key One-Pass Unilateral Authentication Protocol
---
---       A -> B: CertA,[Ta|Na],B,text2,E(Ka-1: [Ta|Na],B,text1)
```

```
(reset)
(
module ISOPublic
export * .

key : type.
pvtk : princ -> type.          pvtk A <: key.
nonce : type.                          nonce  <: msg.
enc : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
  ISOinitRule1:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text2 : text.
  forall text1 : text.
    empty
  => exists Na : nonce.
     exists CertA : msg.
     net (CertA & B & Na & text2 & enc Ka-1 (Na & B & text1)).
}


receiver:
forall B : princ.
{
  ISORecRule2:
  forall A : princ.
  forall Ka-1 : pvtk A.
  forall text2 : text.
  forall text1 : text.
    net (CertA & B & Na & text2 & enc Ka-1 (Na & B & text1))
  => empty.
}

)
```

ISO Public Key Two-Pass Unilateral Authentication Protocol

Purpose

Self-explanatory.

Informal Specification

1. B -> A: Rb,Text1
2. A -> B: CertA, Ra, Rb, B, Text3, E(Ka-1:Ra,Rb,B,text2)

MSR Specification

```
--- Author: rishav
--- Date: 10 September 2007
--- ISO Public Key Two-Pass Unilateral Authentication Protocol
---
---     B -> A: Rb,Text1
---       A -> B: CertA, Ra, Rb, B, Text3, E(Ka-1:Ra,Rb,B,text2)
(reset)
(
module ISOTwoPassPublicUni
export * .

key : type.
pvtk : princ -> type.          pvtk A <: key.
randomNo: type.                          randomNo  <: msg.
enc : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall B : princ.
{
                  exists LB : {A : princ} pvtk A -> randomNo ->
text -> state.
```

```
  ISO2initRule1:
  forall Ka-1 : pvtk A.
  forall text1 : text.
  empty
  => exists rB : randomNo.
    net(rB & text1), LB A Ka-1 rB text1.

 ISO2initRule3:
  forall A : princ.
  forall Ka-1 : pvtk A.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
  forall rA : randomNo.
    net(CertA & rA & rB & text3 & enc Ka-1 (rB & rA & B &
text2)), LB A Ka-1 rB text1
  => empty.
}

reciever:
forall A : princ.
{
  ISOinitRule2:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
   net(rB & text1)
  => exists CertA : msg.
     exists rA : randomNo.
     net(CertA & rA & rB & text3 & enc Ka-1 (rB & rA & B &
text2)).
}

)
```

ISO Public Key Two-Pass Mutual Authentication Protocol

Purpose

Self-explanatory.

Informal Specification

1. A -> B: CertA, [Ta|Na], B, Text2, E(Ka-1:[Ta|Na],B,text1)
2. B -> A: CertA, [Ta|Na], A, Text4, E(Kb-1:[Tb|Nb],A,text3)

MSR Specification

```
--- Author: rishav
--- Date: 10 September 2007
--- ISO Public Key Two-Pass Unilateral Authentication Protocol
---
---     B -> A: Rb,Text1
---       A -> B: CertA, Ra, Rb, B, Text3, E(Ka-1:Ra,Rb,B,text2)
(reset)
(
module ISOTwoPassPublicUni
export * .

key : type.
pvtk : princ -> type.          pvtk A <: key.
randomNo: type.                          randomNo  <: msg.
enc : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall B : princ.
{
                  exists LB : {A : princ} pvtk A -> randomNo ->
text -> state.
```

```
  ISO2initRule1:
  forall Ka-1 : pvtk A.
  forall text1 : text.
  empty
  => exists rB : randomNo.
    net(rB & text1), LB A Ka-1 rB text1.

 ISO2initRule3:
  forall A : princ.
  forall Ka-1 : pvtk A.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
  forall rA : randomNo.
    net(CertA & rA & rB & text3 & enc Ka-1 (rB & rA & B &
text2)), LB A Ka-1 rB text1
  => empty.
}

reciever:
forall A : princ.
{
  ISOinitRule2:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall rB : randomNo.
   net(rB & text1)
  => exists CertA : msg.
     exists rA : randomNo.
     net(CertA & rA & rB & text3 & enc Ka-1 (rB & rA & B &
text2)).
}

)
```

ISO Public Key Three-Pass Mutual Authentication Protocol

Purpose

Self-explanatory

Informal Specification

1. B -> A: Rb,Text1
2. A -> B: CertA, Ra, Rb, B, Text3, E(Ka-1:Ra,Rb,B,text2)
3. B -> A: CertB, Rb, Ra, A, Text5, E(Kb-1:Rb,Ra,A,text4)

MSR Specification

```
--- Author: rishav
--- Date: 16 September 2007
--- ISO Three-Pass Mutual Authentication Protocol
---
---     B -> A: Rb,Text1
---       A -> B: CertA, Ra, Rb, B, Text3, E(Ka-1:Ra,Rb,B,text2)
---       B -> A: CertB, Rb, Ra, A, Text5, E(Kb-1:Rb,Ra,A,text4)
(reset)
(
module ISOPublicThreePassMutual
export * .

key : type.
skey : type.
pvtk : princ -> type.          pvtk A <: key.
randomNo: type.                          randomNo  <: msg.

enc : pvtk -> msg -> msg.
text: type.                  text <: msg.

net : msg -> state.
```

```
initiator:
forall B : princ.
{
            exists LB : randomNo -> state.

 ISO3initRule1:
    forall text1 : text.
    empty
  => exists Rb : randomNo.
    net(Rb & text1), LB Rb.

 ISO3initRule3:
  forall A : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
  forall Ka-1 : pvtk.
  forall text3 : text.
  forall text2 : text.
  forall text4 : text.
  forall text5 : text.
     net(CertA & Ra & Rb & B & text3 & enc Ka-1 (Rb & Ra & B &
text2)), LB Rb
  => net(CertB & Rb & Ra & A & text5 & enc Kb-1 (Rb & Ra & A &
text4)).
}


reciever:
forall A : princ.
{
                        exists LA : princ -> randomNo -> randomNo
-> state.
  ISO3initRule2:
  forall B : princ.
  forall Rb : randomNo.
  forall Ka-1 : pvtk.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
   net(Rb & text1)
  =>exists Ra : randomNo.
    net(CertA & Ra & Rb & B & text3 & enc Ka-1 (Rb & Ra & B &
text2)), LA B Ra Rb.

  ISO3initRule4:
  forall B : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
```

```
  forall text4 : text.
  forall text5 : text.
    net(CertB & Rb & Ra & A & text5 & enc Kb-1 (Rb & Ra & A &
text4)), LA B Ra Rb
  => empty.
}
)
```

ISO Public Key Two-Pass Parallel Mutual Authentication Protocol

Purpose

messages 1 and 2, 3 and 4 are sent parallely.

Informal Specification

1. A -> B: CertA, Ra, Text1
2. B -> A: Rb, Ra, A, Text6, E(Kb-1: Rb, Ra, A, Text5)
3. B -> A: CertB, Rb, Text2
4. A -> B: Ra, Rb, B, Text4, E(Kb-1: Rb, Ra, B, Text3)

MSR Specification

```
--- Author: rishav
--- Date: 30 September 2007
--- ISO Public Key Two-Pass Parallel Mutual Authentication
Protocol
---
---     A -> B: CertA, Ra, Text1
---     B -> A: Rb, Ra, A, Text6, E(Kb-1: Rb, Ra, A, Text5)
```

```
---     B -> A: CertB, Rb, Text2
---     A -> B: Ra, Rb, B, Text4, E(Kb-1: Rb, Ra, B, Text3)

(
module ISOTwoPassParr
export * .
key : type.
pvtk : princ -> type.          pvtk A <: key.
nonce : type.                          nonce  <: msg.
enc : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
                  exists LA : princ -> nonce -> pvtk A -> state.
  ISO2PinitRule1:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text1 : text.
    empty
  => exists Ra : nonce.
     net (CertA & Ra & text1), LA B Ra Ka-1.


  ISO2PinitRule3:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text5 : text.
  forall text6 : text.
    net (Rb & Ra & A & text6 & enc Kb-1 (Rb & Ra & A & text5)),
LA B Ra Ka-1
  => empty.
}

recicever:
forall B : princ.
{
  ISO3PinitRule2:
  forall A : princ.
  forall Kb-1 : pvtk B.
  forall text5 : text.
  forall text6 : text.
  net (CertA & Ra & text1)
  => exists Rb : nonce.
     net (Rb & Ra & A & text6 & enc Kb-1 (Rb & Ra & A & text5)).
```

```
}


initiator2:
forall B : princ.
{
                        exists LB : princ -> nonce -> pvtk B ->
state.
  ISO2PinitRule1:
  forall A : princ.
  forall Kb-1 : pvtk B.
  forall text2 : text.
    empty
  => exists Rb : nonce.
     net (CertB & Rb & text2), LB A Rb Kb-1.


 ISO2PinitRule3:
  forall A : princ.
  forall Kb-1 : pvtk B.
  forall text3 : text.
  forall text4 : text.
  forall Rb : nonce.
    net (Rb & Ra & B & text4 & enc Ka-1 (Rb & Ra & B & text3)),
LB A Rb Kb-1
  => empty.


}

recicever2:
forall A : princ.
{
  ISO3PinitRule2:
  forall B : princ.
  forall Ka-1 : pvtk A.
  forall text3 : text.
  forall text4 : text.
  forall text2 : text.
  net (CertB & Rb & text2)
  => exists Ra : nonce.
     net (Rb & Ra & A & text4 & enc Ka-1 (Rb & Ra & B & text3)).
}

)
```

Diffie Hellman Exchange

Purpose

In this protocol two numbers are publicly agreed by the communicating principals A and B. After the messages are exchangd, A computes k = Y^xmod N and B computes k = X^ymod N where X = G^xmod N and Y = G^ymod N. The result of the two calculations is the same and equal new session key.

Informal Specification

1. A -> B: G^xmod N
2. B -> A: G^ymod N

MSR Specification

```
--- Author: rishav
--- Date: 7 October 2007
--- Diffie Hellman Exchange
---
---     A -> B: X=G^xmod N
---     B -> A: Y=G^ymod N

(
module DHE
export * .
key : type.                                key <: msg.
random : type.                             random  <: msg.
enc : key -> msg -> msg.
expm : random -> random -> random -> msg.
DHpar: random -> random -> state.
--- calculation performed to get key K by both A and B
dh : msg -> random -> random -> key -> state.

net : msg -> state.

initiator:
forall A : princ.
{
```

```
                  exists LA1 : random -> random -> random ->
state.
                  exists LA2 : key -> state.
  DHERule1:
  forall G : random.
  forall N : random.
    empty
  => exists x : random.
     net (expm x G N), LA1 G N x if DHpar G N.


  DHERule3:
  forall Y : msg.
  forall G : random.
  forall N : random.
  forall x : random.
   net(Y), LA1 G N x
  => exists K : key.
     LA2 K if dh Y x N K.


}

receiver:
forall B : princ.
{
                  exists LB1 : random -> random -> random ->
state.
                  exists LB2 : key -> state.

  DHERule2:
  forall G : random.
  forall N : random.
    empty
  => exists y : random.
     net (expm y G N), LB1 G N y if DHpar G N.


  DHERule4:
  forall X : msg.
  forall G : random.
  forall N : random.
  forall y : random.
   net(X), LB1 G N y
  => exists K : key.
     LB2 K if dh X y N K.
```

}

)

Needham-Schroeder Public Key Protocol

Purpose

Same as Needham Schroeder protocol but with public keys.

Informal Specification

1. A -> S : A, B
2. S -> A : E(Ks-1 : Kb, B)
3. A -> B : E(Kb : Na, A)
4. B -> S : B, A
5. S -> B : E(Ks-1 : Ka, A)
6. B -> A : E(Ka: Na, Nb)
7. A -> B : E(Kb: Nb)

MSR Specification

```
--- Author: rishav
--- Date: 11 October 2007
--- Needham-Schroeder Public Key Protocol
---     A -> S : A, B
---     S -> A : E(Ks-1 : Kb, B)
---     A -> B : E(Kb : Na, A)
---     B -> S : B, A
---     S -> B : E(Ks-1 : Ka, A)
---     B -> A : E(Ka: Na, Nb)
```

```
---     A -> B : E(Kb: Nb)
(reset)
(
module NSP
export * .

server : type.                          server <: princ.
key : type.
ltk : princ -> type.          ltk A <: key.          ltk A <:
msg.
ltks : princ -> type.               ltks S <: key.
nonce : type.                         nonce <: msg.

enc : key -> msg -> msg.

net : msg -> state.


initiator:
forall A : princ.
{
                 exists LA : princ -> state.
                 exists LA2 : nonce -> {B : princ} ltk B ->
state.
NSPRule1:
  forall B : princ.
  empty
  => net(A & B), LA B.

NSPRule3:
   forall B : princ.
   forall Ks-1 : ltks S.
   forall Kb : ltk B.
   net(enc Ks-1 (B & Kb)), LA B
   => exists Na : nonce.
      net(enc Kb (A & Na)), LA2 Na B Kb.

NSPRule7:
    forall B : princ.
    forall Kb : ltk B.
    forall Ka : ltk A.
    forall Na : nonce.
    forall Nb : nonce.
      net(enc Ka (Nb & Na)), LA2 Na B Kb
    => net(enc Kb Nb).
}

receiver:
```

```
forall B : princ.
{
                  exists LB : princ -> nonce -> state.
                  exists LB2 : nonce -> state.
NSPRule4:
   forall A : princ.
   forall Na : nonce.
   forall Kb : ltk B.
     net(enc Kb (A & Na))
  => net(B & A), LB A Na.


NSPRule6:
   forall A : princ.
   forall Na : nonce.
   forall Ks-1 : ltks S.
   forall Ka : ltk A.
     net(enc Ks-1 (A & Ka)), LB A Na
   => exists Nb : nonce.
      net(enc Ka (Nb & Na)), LB2 Nb.


NSPRule8:
   forall Nb : nonce.
   forall Kb : ltk B.
    net(enc Kb Nb), LB2 Nb
  => empty.
}


server:
forall S : server.
{
                  exists LS : princ -> princ -> state.
NSPRule2:
   forall A : princ.
   forall B : princ.
   forall Ks-1 : ltks S.
   forall Kb : ltk B.
   net(A & B)
     => net(enc Ks-1 (B & Kb)), LS A B.

NSPRule5:
   forall A : princ.
   forall B : princ.
   forall Ks-1 : ltks S.
   forall Ka : ltk A.
     net(B & A), LS A B
```

```
  => net(enc Ks-1 (A & Ka)).

}

)
```

SPLICE/AS authentication Protocol

Purpose

We assume that initially, the client C and the server S only know their own public and private key, and that the authority AS knows everyone's public key. E(Kas-1: AS, C, N1, Ks) (in message 2) and E(Kas-1 : AS, S, N3, Kc) (in message 5) are certificates signed and distributed by the authority AS, for the respective public keys Ks and Kc. After a successfull run of the protocol, the value of N2 can be used by C and S as a symmetric key for secure communications.

Informal Specification

1. C -> AS : C, S, N1
2. AS -> C : AS, E(Kas-1 : AS, C, N1, Ks)
3. C -> S : C, S, E(Kc-1 : C, T, L E(Ks:N2))
4. S -> AS : S, C, N3
5. AS -> S : AS, E(Kas-1 : AS, S, N3, Kc)
6. S -> C : S, C, E(Kc: S, N2+1)

MSR Specification

```
--- Author: rishav
--- Date: 11 October 2007
```

```
--- SPLICE/AS authentication Protocol
---     C -> AS : C, S, N1
---     AS -> C : AS, E(Kas-1 : AS, C, N1, Ks)
---     C -> S : C, S, E(Kc-1 : C, T, L E(Ks:N2))
---     S -> AS : S, C, N3
---     AS -> S : AS, E(Kas-1 : AS, S, N3, Kc)
---     S -> C : S, C, E(Kc: S, N2+1)
(reset)
(
module SPLICE
export * .

server : type.                          server <: princ.
CA : type.                              CA <: princ.
key : type.
ltk : princ -> type.          ltk A <: key.          ltk A <:
msg.
ltks : princ -> type.               ltks S <: key.
nonce : type.                        nonce <: msg.

enc : key -> msg -> msg.
inc : nonce -> nonce.
net : msg -> state.


initiator:
forall C : princ.
{
                  exists LC : princ -> nonce -> state.
                  exists LC2 : princ -> nonce -> state.
SPRule1:
   forall S : server.
   empty
  => exists N1 : nonce.
     net(C & S & N1), LC S N1.

SPRule3:
   forall AS : CA.
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Ks : ltk S.
   forall Kc-1 : ltks C.
   forall Ks : ltk S.
   forall N1 : nonce.
   forall T : msg.
   forall L : msg.
   net(AS & enc Kas-1 (AS & C & N1 & Ks)), LC S N1
   => exists N2 : nonce.
```

```
      net(C & S & enc Kc-1 (C & T & L & enc Ks N2)), LC S N2.

SPRule7:
   forall S : server.
   forall N2 : nonce.
   forall Kc : ltk C.
    net(S & C & enc Kc (S & inc N2)), LC S N2
  => empty.


}

receiver:
forall AS : CA.
{
                  exists LAS : princ -> princ -> state.
SPRule2:
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Ks : ltk S.
   forall N1 : nonce.
     net(C & S & N1)
  => net(AS & enc Kas-1 (AS & C & N1 & Ks)), LAS S AS.


SPRule5:
   forall C : princ.
   forall S : server.
   forall N3 : nonce.
   forall Kas-1 : ltks AS.
   forall Kc : ltk C.
     net(S & C & N3), LAS S AS
   => net(AS & enc Kas-1 (AS & S & N3 & Kc)).

}


server:
forall S : server.
{
                  exists LS : princ -> princ -> nonce -> state.
SPRule4:
   forall C : princ.
   forall AS : CA.
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Kc-1 : ltks C.
   forall Ks : ltk S.
```

```
   forall N2 : nonce.
   forall T : msg.
   forall L : msg.
    net(C & S & enc Kc-1 (C & T & L & enc Ks N2))
   => exists N3 : nonce.
      net(S & C & N3), LS C AS N3.

SPRule6:
   forall AS : CA.
   forall C : princ.
   forall N3 : nonce.
   forall N2 : nonce.
   forall Kas-1 : ltks AS.
   forall Kc : ltk C.
     net(AS & enc Kas-1 (AS & S & N3 & Kc)), LS C AS N3
  => net(S & C & enc Kc (S & inc N2)).

}


)
```

Hwang and Chen's SPLICE/AS authentication Protocol

Purpose

The name of the owner of the public key is included in certificate to overcomes the flaws of SPLICE/AS.

Informal Specification

1. C -> AS : C, S, N1
2. AS -> C : AS, E(Kas-1 : AS, C, N1, S, Ks)
3. C -> S : C, S, E(Kc-1 : C, T, L E(Ks:N2))
4. S -> AS : S, C, N3
5. AS -> S : AS, E(Kas-1 : AS, S, N3, C, Kc)

6. S -> C : S, C, E(Kc: S, N2+1)

More Commonly Used Representations
C - Client
S - server
AS - certification authority
T - Timestamp
L - Lifetime

MSR Specification

```
--- Author: rishav
--- Date: 12 October 2007
--- Hwang and Chen's SPLICE/AS authentication Protocol
---     C -> AS : C, S, N1
---     AS -> C : AS, E(Kas-1 : AS, C, N1, S, Ks)
---     C -> S : C, S, E(Kc-1 : C, T, L E(Ks:N2))
---     S -> AS : S, C, N3
---     AS -> S : AS, E(Kas-1 : AS, S, N3, C, Kc)
---     S -> C : S, C, E(Kc: S, N2+1)
(reset)
(
module SPLICE
export * .

server : type.                          server <: princ.
CA : type.                              CA <: princ.
key : type.
ltk : princ -> type.          ltk A <: key.          ltk A <:
msg.
ltks : princ -> type.                ltks S <: key.
nonce : type.                          nonce <: msg.

enc : key -> msg -> msg.
inc : nonce -> nonce.
net : msg -> state.


initiator:
forall C : princ.
{
                  exists LC : princ -> nonce -> state.
                  exists LC2 : princ -> nonce -> state.
SPRule1:
   forall S : server.
```

```
   empty
  => exists N1 : nonce.
     net(C & S & N1), LC S N1.

SPRule3:
   forall AS : CA.
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Ks : ltk S.
   forall Kc-1 : ltks C.
   forall Ks : ltk S.
   forall N1 : nonce.
   forall T : msg.
   forall L : msg.
   net(AS & enc Kas-1 (AS & C & N1 & S & Ks)), LC S N1
   => exists N2 : nonce.
      net(C & S & enc Kc-1 (C & T & L & enc Ks N2)), LC S N2.

SPRule7:
   forall S : server.
   forall N2 : nonce.
   forall Kc : ltk C.
    net(S & C & enc Kc (S & inc N2)), LC S N2
  => empty.


}

receiver:
forall AS : CA.
{
                  exists LAS : princ -> princ -> state.
SPRule2:
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Ks : ltk S.
   forall N1 : nonce.
     net(C & S & N1)
  => net(AS & enc Kas-1 (AS & C & N1 & S & Ks)), LAS S AS.


SPRule5:
   forall C : princ.
   forall S : server.
   forall N3 : nonce.
   forall Kas-1 : ltks AS.
   forall Kc : ltk C.
     net(S & C & N3), LAS S AS
```

```
   => net(AS & enc Kas-1 (AS & S & N3 & C & Kc)).

}


server:
forall S : server.
{
                  exists LS : princ -> princ -> nonce -> state.
SPRule4:
   forall C : princ.
   forall AS : CA.
   forall S : server.
   forall Kas-1 : ltks AS.
   forall Kc-1 : ltks C.
   forall Ks : ltk S.
   forall N2 : nonce.
   forall T : msg.
   forall L : msg.
    net(C & S & enc Kc-1 (C & T & L & enc Ks N2))
   => exists N3 : nonce.
      net(S & C & N3), LS C AS N3.

SPRule6:
   forall AS : CA.
   forall C : princ.
   forall N3 : nonce.
   forall N2 : nonce.
   forall Kas-1 : ltks AS.
   forall Kc : ltk C.
     net(AS & enc Kas-1 (AS & S & N3 & C & Kc)), LS C AS N3
  => net(S & C & enc Kc (S & inc N2)).

}


)
```

CCITT X.509

Purpose

This is the classic description of three protocols, either message 1 or message 1 and 2 or all three.

Informal Specification

1. A -> B : A, E(Ka-1: Ta, Na, B, Xa, E(Kb : Ya))
2. B -> A : B, E(Kb-1: Tb, Nb, A, Na, Xb, E(Ka : Yb))
3. A -> B : A, E(Ka-1 : Nb)

More Commonly Used Representations
Xa,Xb,Ya,Yb - user data

MSR Specification

```
--- Author: rishav
--- Date: 13 October 2007
---  CCITT X.509
---     A -> B : A, E(Ka-1: Ta, Na, B, Xa, E(Kb : Ya))
---     B -> A : B, E(Kb-1: Tb, Nb, A, Na, Xb, E(Ka : Yb))
---     A -> B : A, E(Ka-1 : Nb)

(reset)
(
module CCITT
export * .

key : type.
ltk : princ -> type.          ltk A <: key.        ltk A <: msg.
ltks : princ -> type.         ltks A <: key.
nonce : type.                           nonce <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
               exists LA : nonce -> {A : princ} ltks A -> state.
CCRule1:
  forall B : princ.
  forall Ka-1 : ltks A.
  forall Kb : ltk B.
  forall Ta : msg.
  forall Xa : msg.
  forall Ya : msg.
```

```
  empty
  => exists Na : nonce.
     net (A & enc Ka-1 (Ta & Na & B & Xa & enc Kb Ya)), LA Na A
Ka-1.

CCRule3:
   forall B : princ.
   forall Nb : nonce.
   forall Na : nonce.
   forall Kb-1 : ltks B.
   forall Ka-1 : ltks A.
   forall Ka : ltk A.
   forall Tb : msg.
   forall Xb : msg.
   forall Yb : msg.
    net (B & enc Kb-1 (Tb & Nb & A & Na & Xb & enc Ka Yb)), LA Na
A Ka-1
   => net(enc Ka-1 Nb).
}

receiver:
forall B : princ.
{
                  exists LB : nonce -> {A : princ} ltks A ->
state.
CCRule2:
  forall A : princ.
  forall Na : nonce.
  forall Ka-1 : ltks A.
  forall Kb-1 : ltks B.
  forall Kb : ltk B.
  forall Ka : ltk A.
  forall Ta : msg.
  forall Xa : msg.
  forall Ya : msg.
  forall Tb : msg.
  forall Xb : msg.
  forall Yb : msg.
   net (A & enc Ka-1 (Ta & Na & B & Xa & enc Kb Ya))
 => exists Nb : nonce.
    net (B & enc Kb-1 (Tb & Nb & A & Na & Xb & enc Ka Yb)), LB Nb
A Ka-1.

CCRule4:
   forall A : princ.
   forall Ka-1 : ltks A.
   forall Nb : nonce.
     net(enc Ka-1 Nb), LB Nb A Ka-1
```

```
  => empty.

}

)
```

Shamir Rivest Adelman Three Pass protocol

Purpose

In this protocol participants share no secrets.

Informal Specification

1. A -> B: E(Ka : M)
2. B -> A: E(Kb : E(Ka : M))
3. A -> B: E(Kb : M)

MSR Specification

```
--- Author: rishav
--- Date: 12 October 2007
--- Shamir Rivest Adelman Three Pass protocol
---      A -> B: E(Ka : M)
---      B -> A: E(Kb : E(Ka : M))
---      A -> B: E(Kb : M)
---
(reset)
(
module SRA
export * .
key : type.
ltk : princ -> type.                    ltk A <: key.
nonce : type.                           nonce <: msg.
```

```
enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall A : princ.
{
            exists LA : nonce -> state.
SRule1:
  forall A : princ.
  forall Ka : ltk A.
  empty
  => exists M : nonce.
     net (enc Ka M), LA M.

SRule3:
  forall B : princ.
  forall M : nonce.
  forall Kb : ltk B.
  forall Ka : ltk A.
    net(enc Kb (enc Ka M)), LA M
   => net(enc Kb M).
}

receiver:
forall B : princ.
{
                  exists LB : nonce -> state.
SRule2:
  forall A : princ.
  forall M : nonce.
  forall Kb : ltk B.
  forall Ka : ltk A.
   net (enc Ka M)
 => net(enc Kb (enc Ka M)), LB M.

SRule4:
   forall Kb : ltk B.
   forall M : nonce.
     net(enc Kb M), LB M
  => empty.

}

)
```

Encrypted Key Exchange

Purpose

P is the password used as the symmetric key. Ka is randomly generated public key. R is randomly generated session key.

Informal Specification

1. A -> B: E(P : Ka)
2. B -> A: E(P : E(Ka : R))
3. A -> B: E(R : Na)
4. B -> A: E(R : Na, Nb)
5. A -> B: E(R : Nb)

MSR Specification

```
--- Author: rishav
--- Date: 14 October 2007
--- Encrypted Key Exchange
---         A -> B: E(P : Ka)
---         B -> A: E(P : E(Ka : R))
---         A -> B: E(R : Na)
---         B -> A: E(R : Na, Nb)
---         A -> B: E(R : Nb)
(reset)
(
module EKE
export * .

key : type.
stk : type.          stk <: msg.  stk <: key.
ltk : princ -> type.          ltk A <: msg.  ltk A <: key.
nonce : type.                    nonce  <: msg.

enc : key -> msg -> msg.
```

```
net : msg -> state.

initiator:
forall A : princ.
{
                  exists LA1 : stk -> {A : princ} ltk A -> state.
                  exists LA2 : nonce -> stk -> state.

EKERule1:
  forall P : stk.
    empty
  => exists Ka : ltk A.
    net(enc P (Ka)), LA1 P A Ka.

EKERule3:
  forall R : stk.
  forall Ka : ltk A.
  forall P : stk.
   net(enc P (enc Ka R)), LA1 P A Ka
  => exists Na : nonce.
     net(enc R Na), LA2 NA R.

EKERule5:
  forall Na : nonce.
  forall Nb : nonce.
  forall R : stk.
  net(enc R (Na & Nb)), LA2 NA R
  => net(enc R Nb).

}

reciever:
forall B : princ.
{

                  exists LB1 : stk -> state.
                  exists LB2 : nonce -> stk -> state.

EKERule2:
  forall Ka : ltk A.
  forall P : stk.
    net(enc P (Ka))
  => exists R : stk.
     net(enc P (enc Ka R)), LB1 R.

EKERule4:
  forall Na : nonce.
```

```
  forall R : stk.
  net(enc R Na), LB1 R
  => exists Nb : nonce.
     net(enc R (Na & Nb)), LB2 Nb R.

EKERule6:
   forall Nb : nonce.
   forall R : stk.
   net(enc R Nb), LB2 Nb R
  => empty.
}

)
```

Davis Swick Private Key Certificates

Purpose

T is a trusted translator. Kbt is shared key between B and T and Kat is between A and T.

Informal Specification

1. B -> A : E(Kbt : A, msg)
2. A -> T : E(Kbt : A, msg), B
3. T -> A : E(kat : msg, B)

MSR Specification

```
--- Author: rishav
--- Date: 13 October 2007
--- Davis Swick Private Key Certificates
---     B -> A : E(Kbt : A, msg)
---     A -> T : E(Kbt : A, msg), B
---     T -> A : E(kat : msg, B)
(reset)
(
module DSP
```

```
export * .

translator : type.                          translator <: princ.
key : type.
ltk : princ -> princ -> type.           ltk A B <: key.
nonce : type.                            nonce  <: msg.

enc : key -> msg -> msg.

net : msg -> state.

initiator:
forall B : princ.
{

DSPRule1:
 forall Kbt : ltk B T.
  empty =>
  exists mess : msg.
  net(enc Kbt (A & mess)).
}


receiver:
forall A : princ.
{
                           exists LA : msg -> state.
DSPRule2:
   forall mess : msg.
   forall Kbt : ltk B T.
    net(enc Kbt (A & mess))
 =>  net(enc Kbt (A & mess) & B), LA mess.

ORRule4:
   forall mess : msg.
   forall Kat : ltk A T.
   net(enc Kat (mess & B)), LA mess
 => empty.

}


server:
forall T : translator.
{
DSPRule3:
   forall mess : msg.
   forall Kat : ltk A T.
```

```
   net(enc Kbt (A & mess) & B)
 => net(enc Kat (mess & B)).

}

)
```

Gong Mutual Authentication Protocol

Purpose

It is a mutual authentication protocol of two principals with a trusted server, and exchange of a new symmetric key. Uses one-way functions and no encryption. f and g are one way publicly known functions. the principal A and B shares a secret Pa and Pb respectively, with the authentication server. xor is the xor function and equals is the equality function. Ha and Hb are handshake numbers In message 3 (k,ha,hb) = f(Ns, Na, B, Pa) is calculated by the server. K is the secret to be shared between A and B. B computes f(Ns, Nb, A , Pb) to retrieve (K,Ha,Hb) from the second item of the message. It also computes g(K,Ha,Hb,Pb) to check against the third item that tampering has not taken place. A computes f(Ns,Na,B,Pa) to get (K,Ha,Hb). If the value of Hb matches the one sent by B then A replies with message 5.

Informal Specification

1. A -> B : A,B,Na
2. B -> S : A, B, Na, Nb
3. S -> B : Ns, f(Ns,Nb,A,Pb) xor (K,Ha,Hb), g(K,Ha,Hb,Pb)
4. B -> A : Ns,Hb
5. A -> B : Ha

MSR Specification

```
--- Author: rishav
```

```
--- Date: 28 October 2007
--- Gong Mutual Authentication Protocol
---     A -> B : A,B,Na
---     B -> S : A, B, Na, Nb
---     S -> B : Ns, f(Ns,Nb,A,Pb) xor (K,Ha,Hb), g(K,Ha,Hb,Pb)
---     B -> A : Ns,Hb
---     A -> B : Ha

(reset)
(
module GMAP
export * .

server : type.                          server <: princ.
key : type.                             key <: msg.
nonce : type.                            nonce <: msg.
time : type.                            time <: msg.
xor : msg -> msg -> msg.
f : msg -> msg -> msg -> msg -> msg.
g : msg -> msg -> msg -> msg -> msg.
m : msg -> msg -> msg -> msg.
equal : msg -> msg -> msg.
net : msg -> state.
initiator:
forall A : princ.
{
                exists LA : nonce -> state.
GRule1:
  empty
  => exists Na : nonce.
   net(A & B & Na), LA Na.

GRule5:
  forall Na : nonce.
  forall Ns : nonce.
  forall Pa : msg.
  forall Hb : msg.
  forall Ha : msg.
  forall k : key.
    net(Ns & Hb)
   => net(Ha) , LA Na
   if (equal (f Ns Na B Pa)(k & Ha & Hb)).
}


receiver:
forall B : princ.
{
```

```
                exists LB1 : nonce -> nonce -> state.
                exists LB2 : msg -> state.
GRule2:
  forall A : princ.
  forall Na : nonce.
    net(A & B & Na)
 => exists Nb : nonce.
   net(Nb & A & Na & B), LB1 Na Nb.

GRule4:
   forall A : princ.
   forall Hb : msg.
   forall Ha : msg.
   forall Pb : msg.
   forall M : msg.
   forall N : msg.
   forall Ns : nonce.
   forall Na : nonce.
   forall Nb : nonce.
   forall k : key.
     net(Ns & M & N), LB1 Na Nb
  => net(Ns & Hb), LB2 Ha
  if (xor M (equal (f Ns Nb A Pb) ( k & Ha & Hb))) & (equal N (g
K Ha Hb Pb)).

GRule6:
   forall Ha : msg.
   net(Ha), LB2 Ha
   => empty.

}


server:
forall S : server.
{
GRule3:
   forall A : princ.
   forall B : princ.
   forall Hb : msg.
   forall Ha : msg.
   forall Pb : msg.
   forall Na : nonce.
   forall Nb : nonce.
   forall k : key.
    net(Nb & A & Na & B)
     => exists Ns : nonce.
        net(Ns & xor (f Ns Nb A Pb) T & (g k Ha Hb Pb))
```

```
     if equal (f Ns Na B Pa) T & equal T (k & Ha & Hb).
}

)
```

Bilateral Key Exchange with Public Key

Purpose

The two principals exchange their public key while encrypting the nonce of the other protocol to verify.

Informal Specification

1. B -> A: B, E(Ka: Nb,B)
2. A -> B: E(Kb: f(Nb),Na,A,K)
3. B -> A: E(Kb: f(Na))

MSR Specification

```
--- Author: Rishav Bhowmick
--- Date: 4 Oct 2007
--- Bilateral Key Exchange with Public Key
---
---     B -> A: B, E(Ka: Nb,B)
---       A -> B: E(Kb: f(Nb),Na,A,K)
---       B -> A: E(Kb: f(Na))
(reset)
(
module BKEPL
export * .

key : type.
skey : princ -> princ -> type.          skey A B <: key.
```

```
skey A B <: msg.
pubk : princ -> type.          pubk A <: key.
nonce: type.                           nonce  <: msg.

enc : key -> msg -> msg.
f : nonce -> msg.

net : msg -> state.


initiator:
forall B : princ.
{
                  exists LB : {A: princ} pubk A -> nonce -> state.
  BKRule1:
  forall A : princ.
  forall Ka : pubk A.
    empty
  => exists Nb : nonce.
    net(B & enc Ka (Nb & B)), LB A Ka Nb.

  BKRule3:
  forall A : princ.
  forall Nb : nonce.
  forall Na : nonce.
  forall Kb : pubk B.
  forall Ka : pubk A.
  forall K : skey A B.
     net(enc Kb ((f Nb) & Na & A & K)), LB A Ka Nb
  => net(enc K (f Na)).
}



reciever:
forall A : princ.
{

                  exists LA : {B : princ} skey A B -> nonce ->
nonce -> state.
  BKRule2:
  forall B : princ.
  forall Nb : nonce.
  forall Kb : pubk B.
  forall Ka : pubk A.
   net(B & enc Ka (Nb & B))
  => exists Na : nonce.
     exists K : skey A B.
```

```
    net(enc Kb ((f Nb) & Na & A & K)), LA B K Na Nb.

  BKRule4:
  forall B : princ.
  forall Nb : nonce.
  forall Na : nonce.
  forall K : skey A B.
    net(enc K (f Na)), LA B K Na Nb
  => empty.
}
)
```

ISO One-Pass Unilateral Authentication Protocol with CCFs (Cryptographic Check Functions)

Purpose

The keyed function fKab(X) returns hashed value for data X in a manner determined by the key Kab.

Informal Specification

1. A -> B: [Ta|Na],B,text2,fKab([Ta|Na],B,text1)

MSR Specification

```
--- Author: rishav
--- Date: 10 December 2007
--- ISO One-Pass Unilateral Authentication Protocol with CCFs
(Cryptographic Check Functions)
---
---       A -> B: [Ta|Na],B,text2,fKab([Ta|Na],B,text1)
(reset)
(
module ISOPublicCCf
```

```
export * .

key : type.
shrk : princ -> princ -> type.          shrk A B <: key.
nonce : type.                           nonce  <: msg.
f : key -> msg -> msg.
text: type.                  text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
  ISOinitRule1:
  forall B : princ.
  forall Kab : shrk A B.
  forall text2 : text.
  forall text1 : text.
    empty
  => exists Na : nonce.
     net (B & Na & text2 & f Kab (Na & B & text1)).
}


receiver:
forall B : princ.
{
  ISORecRule2:
  forall A : princ.
  forall Kab : shrk A B.
  forall text2 : text.
  forall text1 : text.
    net (B & Na & text2 & f Kab (Na & B & text1))
  => empty.
}

)
```

ISO Two-Pass Unilateral Authentication Protocol with CCFs (Cryptographic Check Functions)

Purpose

The keyed function fKab(X) returns hashed value for data X in a manner determined by the key Kab.

Informal Specification

1. B -> A: Rb,Text1
2. A -> B: Text3, fKAB(Rb,B,Text2)

MSR Specification

```
--- Author: rishav
--- Date: 10 December 2007
--- ISO Two-Pass Unilateral Authentication Protocol CCFs
---
---     B -> A: Rb,Text1
---       A -> B: Text3, fKAB(Rb,B,Text2)
(reset)
(
module ISOTUA
export * .

key : type.
shrk : princ -> princ -> type.          shrk A B <: key.
randomNo: type.                           randomNo  <: msg.
f : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall B : princ.
```

```
{
                  exists LB : randomNo -> state.
  ISO2initRule1:
  forall text1 : text.
  empty
  => exists Rb : randomNo.
    net(Rb & text1), LB Rb.

 ISO2initRule3:
  forall A : princ.
  forall KaB : shrk A B.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall Rb : randomNo.
    net(text3 & f Kab (Rb & B & text2)), LB Rb
  => empty.
}

reciever:
forall A : princ.
{
  ISOinitRule2:
  forall B : princ.
  forall Kab : shrk A B.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
  forall Rb : randomNo.
   net(Rb & text1)
  => net(text3 & f Kab (Rb & B & text2)).
}

)
```

ISO Two-Pass Mutual Authentication Protocol with CCFs (Cryptographic Check Functions)

Purpose

The keyed function fKab(X) returns hashed value for data X in a manner determined by the key Kab.

Informal Specification

1. A -> B: [Ta|Na], Text2, fKab([Ta|Na],B,text1)
2. B -> A: [Ta|Na], Text4, fKab([Tb|Nb],A,text3)

MSR Specification

```
--- Author: rishav
--- Date: 11 December 2007
--- ISO Two-Pass Mutual Authentication Protocol  CCFs
---
---     A -> B: [Ta|Na], Text2, fKab([Ta|Na],B,text1)
---       B -> A: [Ta|Na], Text4, fKab([Tb|Nb],A,text3)

(
module ISOTMA
export * .
key : type.
shrk : princ -> princ -> type.          shrk A B <: key.
nonce : type.                           nonce  <: msg.
f : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall A : princ.
{
```

```
  ISO2initRule1:
  forall B : princ.
  forall Kab : shrk A B.
  forall text2 : text.
  forall text1 : text.
    empty
  => exists Na : nonce.
     net (Na & text2 & f Kab (Na & B & text1)).
}

---they are independent
recicever:
forall B : princ.
{
  ISO2initRule2:
  forall A : princ.
  forall Kab : shrk A B.
  forall text3 : text.
  forall text4 : text.
   empty
  => exists Nb : nonce.
     net (Nb & text4 & f Kab (Nb & A & text3)).
}

)
```

ISO Three-Pass Mutual Authentication Protocol with CCFs (Cryptographic Check Functions)

Purpose

The keyed function fKab(X) returns hashed value for data X in a manner determined by the key Kab.

Informal Specification

1. B -> A: Rb,Text1
2. A -> B: Ra, Text3, fKab(Ra,Rb,B,text2)

3. B -> A: Text5, fKab(Rb,Ra,A,text4)

MSR Specification

```
--- Author: rishav
--- Date: 16 December 2007
--- ISO Three-Pass Mutual Authentication Protocol with CCFs
---
---     B -> A: Rb,Text1
---       A -> B: Ra, Text3, fKab(Ra,Rb,B,text2)
---       B -> A: Text5, fKab(Rb,Ra,A,text4)
(reset)
(
module ISOPublicThreePassMutual
export * .

key : type.
shrk : princ -> princ -> type.          shrk A B <: key.
randomNo: type.                           randomNo  <: msg.

f : key -> msg -> msg.
text: type.                   text <: msg.

net : msg -> state.

initiator:
forall B : princ.
{
            exists LB : randomNo -> state.

 ISO3initRule1:
    forall text1 : text.
    empty
  => exists Rb : randomNo.
    net(Rb & text1), LB Rb.

 ISO3initRule3:
  forall A : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
  forall Kab : shrk A B.
  forall text3 : text.
  forall text2 : text.
  forall text4 : text.
```

```
  forall text5 : text.
     net(Ra & text3 & f Kab (Ra & Rb & B & text2)), LB Rb
  => net(text5 & f Kab (Rb & Ra & text4)).
}


reciever:
forall A : princ.
{
                        exists LA : princ -> randomNo -> randomNo
-> state.
  ISO3initRule2:
  forall B : princ.
  forall Rb : randomNo.
  forall Kab : shrk A B.
  forall text3 : text.
  forall text2 : text.
  forall text1 : text.
   net(Rb & text1)
  =>exists Ra : randomNo.
   net(Ra & text3 & f Kab (Ra & Rb & B & text2)), LA B Ra Rb.

  ISO3initRule4:
  forall B : princ.
  forall Rb : randomNo.
  forall Ra : randomNo.
  forall text4 : text.
  forall text5 : text.
    net(text5 & f Kab (Rb & Ra & text4)), LA B Ra Rb
  => empty.
}
)
```