# Verification of Security Protocols

**COMP 525: Formal verification**

Etienne Perot
April 23, 2012

## 1. Introduction

Security protocols are meant to ensure private and secure communication between two parties who can be assured that they are talking to each other; the entire purpose of the security protocol is to assure both parties about those guarantees. As such, if any of those guarantees turns out not to be fulfilled by the protocol, due to a flaw in the protocol's design or in the cryptographic primitives and operations that the protocol uses, that protocol is worthless. Thus, it is critical to thoroughly model check security protocols in order to be able to call them security protocols in the first place.

The Needham-Schroeder public-key protocol is a good example of a security protocol on which model checking reveals a flaw in the authentication guarantee that the protocol aims to provide. The flaw was found by Gavin Lowe and made a clear case for the importance of model checking security protocols.

In addition to the symbolic approach used in Lowe's paper, the (more complex) computational approach can also be used to model check security protocols, with different assumptions and a stronger guarantee that the protocol is indeed safe.

## 2. Summary of the papers

### 2.2 Gavin Lowe, 1996, "*Breaking and Fixing the Needham-Schroeder public-key protocol using FDR*" [4]

The paper presents the Needham-Schroeder public-key protocol, introduced in 1978 by Roger Needham and Michael Schroeder. Then, it models the protocol in CSP (Communicating Sequential Processes, a formal language to express interaction between multiple processes in concurrent systems), models the capabilities of a potential intruder on the network that could interfere with the protocol, models the specifications to check for authentication in both directions, and explains the trace revealed by FDR that shows that the authentication guarantee of the protocol doesn't hold. Finally, a fix to the protocol is proposed.

The Needham-Schroeder protocol works using asymmetric encryption only. It uses the following cryptographic primitives (terms):

- Identities ($I$) represent agents on the network.
- Nonces ($N_I$, where $I$ is an identity) are random numbers taken from a sufficiently large range of numbers such that the probability of two agents coming up with the same nonce is practically impossible. The nonces are generated using true randomness, using sources of entropy available only to the agent generating the nonce, such that it is practically impossible for an agent to deterministically obtain the same nonce as another.

- Asymmetric keys (tuple ($I_{pub}$, $I_{priv}$), where $I$ is the same letter as the identity of the agent the key belongs to). Each agent has one. Each part has certain capabilities, as defined in the operations below.

The primitives have the following operations defined on them:

- Concatenation ($t_1.t_2$) with terms $t_1$ and $t_2$: Two terms can be concatenated into a single term.
- Encryption: enc(t, $I_{pub}$) with term t, and $I_{pub}$ being the public key of agent I: Encrypts term t with the key $I_{pub}$.
- Decryption: dec(t, $I_{priv}$) with term t, and $I_{priv}$ being the private key of agent I: Decrypts term t with the key $I_{priv}$. This operation is only defined when t is a term resulting from the encryption of another term t' using key $I_{pub}$. The function then returns t'. We assert that this is the only possible method of obtaining the term t' out of enc(t', $I_{pub}$).

With these terms and operations defined, the paper describes a subset of the protocol: the actual authentication messages between protocol initiator A and responder B, with generated nonces $N_A$ and $N_B$ respectively.

1. Message 1: A → B : $A.B.enc(N_A.A, B_{pub})$
2. Message 2: B → A : $B.A.enc(N_A.N_B, A_{pub})$
3. Message 3: A → B : $A.B.enc(N_B, B_{pub})$

An initiator A who wishes to establish a session with responder B generates a nonce $N_A$. It encrypts that nonce and its identity A in a message addressed to B and encrypted with B's public key. B, having the private key $B_{priv}$, can decrypt that message and now knows the nonce $N_A$ and the fact that A is trying to talk to it. It responds by generating a nonce of its own $N_B$, and sends it along with $N_A$ to A in a message encrypted with A's public key $A_{pub}$. A can decrypt this using its private key $A_{priv}$, and will observe that the nonce received matches the original $N_A$ it had sent in the first message. Now A should be assured that it is talking to B, since only B could have decrypted the message that A had sent which contained $N_A$. Then, A encrypts B's nonce $N_B$ and sends it in a message encrypted with B's public key. B decrypts it and observes that the nonce matches the original $N_B$ it had sent in the second message. By the same logic, B should now be assured that it is talking to A, because only A could have decrypted the message that contained $N_B$.

The paper then models the protocol with CSP. Each of the three messages of the protocol is mapped to a CSP event, with appropriate variables in it to capture the information stored inside each message. Additionally, all agent state changes are modelled using CSP events; notably, the *Running* and *Commit* events are defined for both the initiator and responder, corresponding respectively to the fact that an agent either starts taking part in a run of the protocol with another agent (*Running*) or commits to a run of the protocol when it is done (*Commit*). The channels for communication (*comm*) and agent state changes (*user*, *session*) are also defined. Then, the paper describes the initiator process, using events from those three channels.

To model the intruder, each capability of the Dolev-Yao [3] adversary is modelled in the CSP system:

- **The ability to receive all packets (overhear)**
  This is modelled by defining four sets representing the intruder's knowledge: $M_1$, $M_2$, $M_3$, and N. The first three sets contain copies of the encrypted part of messages of type 1, 2, or 3 (respectively) as defined in the protocol subset. The set N contains all the nonces that the intruder is able to know.
  The paper describes (using CSP) how those four sets grow as each type of message is received. Whenever a message is encrypted using the intruder's public key, the nonces in that message are added to the set N, as the intruder is able to decrypt the message. If the message is encrypted using another key, the intruder cannot decrypt the encrypted part, but it can save this encrypted part in $M_1$, $M_2$, or $M_3$ according to the type of the message received.
- **The ability to block packets (intercept)**
  This is modelled by adding a new channel to the system, called *intercept*. A renaming process is applied to the initiator process such that all events that the initiator may send on the *comm* channel can now be sent on either the *comm* or the *intercept* channel, non-deterministically. The same is applied to the responder. However, neither the initiator or the responder receive events from the *intercept* channel. This represents the fact that events sent on the *intercept* channel are not received by the initiator or the responder.
- **The ability to send new packets (synthesize)**
  This is modelled by letting the intruder process send events on the *comm* channel such that the initiator or responder process would receive them. The intruder also has the ability to make a packet look like it was sent from another agent than itself.
  The domain of messages that the intruder is able to synthesize is restricted in order to keep the state space to a reasonable size. We will allow the intruder to send a packet from any agent to any other agent that has either an encrypted part that was captured by the intruder in $M_1$, $M_2$, or $M_3$, or any packet that can be constructed using any nonce that the intruder has learned (nonces in N), any identity, and any public key of any agent on the network. This means that the intruder can encrypt terms it knows, and send packets containing those encrypted terms.

Once we have defined our system as the interleaving of all the above processes, we can model check the protocol. The paper defines two specifications:

- **Authentication of the responder**: Can the initiator be assured that it really is talking to the responder?
  In our model, this corresponds to the fact that the event “the initiator A commits to a session with the responder B” should only happen **after** the event “the responder B takes part of a run of the protocol with initiator A”, with matching A and B in the two events. Indeed, if the event “the responder B takes part of a run of the protocol with initiator A” did not take place, then the initiator A should **not** commit to a session with B; if it does commit, then initiator has been tricked.
- **Authentication of the initiator**: Can the responder be assured that it really is talking to the initiator?
  By the same logic, this corresponds to the fact that the event “the responder B commits to a session with the initiator A” should only happen **after** the event “the initiator A takes part of a run of the protocol with responder B”, with matching A and B in the two events.

These two specifications match the guarantees that the Needham-Schroeder protocol aims to provide; since it is an authentication protocol, the above two specifications are the only aspects that are crucial to the protocol itself.

With this system and these two specifications, the FDR (Failures-Divergences Refinement) tool is run and claims that the system matches the first specification (Authentication of the responder) but not the second (Authentication of the initiator). The refinement checker gives an event trace which corresponds to the following steps:

1. The user controlling agent A initiates a run of the protocol with responder I.
2. A sends I a packet containing:

   Message 1: $A \rightarrow I : A.I.enc(N_A.A, I_{pub})$

   This causes I to add $N_A$ to its set of known nonces N.
3. I, pretending to be A, sends a packet to B containing:

   Message 1: $I \rightarrow B : A.B.enc(N_A.A, B_{pub})$
4. Due to the message B just received, B thinks it is taking part in a run of the protocol with initiator A.
5. B tries to send A a packet containing:

   Message 2: $B \rightarrow A : B.A.enc(N_A.N_B, A_{pub})$

   But I blocks the packet (*intercept*).

   I adds the encrypted part of the message ($enc(N_A.N_B, A_{pub})$) to $M_1$.
6. I sends a packet to A containing:

   Message 2: $I \rightarrow A : I.A.enc(N_A.N_B, A_{pub})$

   I is incapable of constructing the encrypted part of this message because it doesn't know $N_B$. However, it knows $enc(N_A.N_B, A_{pub})$ from the message it intercepted, so it can simply copy it.
7. A commits to the protocol with responder I, because the nonce it received from I matches the one it sent.
8. A sends a packet to I containing:

   Message 3: $A \rightarrow I : A.I.enc(N_B, I_{pub})$

   I is capable of decrypting this, and it adds $N_B$ to its set of known nonces N.
9. I, pretending to be A, sends a packet to B containing:

   Message 3: $I \rightarrow B : A.B.enc(N_B, B_{pub})$
10. B will observe that the nonces match and will commit to a protocol with initiator A.

In this manner, the intruder I is able to impersonate initiator A to the responder B, and can now send information that B will think is coming from A. The flaw in the protocol came from the fact that I could use A as an oracle to get $N_B$. Thus, the paper proposes a simple fix to prevent A from being used as an oracle: adding the identity of the responder in the second message.

Message 2: $B \rightarrow A : B.A.enc(N_A.N_B, A_{pub})$

becomes

Message 2: $B \rightarrow A : B.A.enc(N_A.N_B.B, A_{pub})$

Now, if the intruder took the same attack strategy as above, then A would reject the message 2 that it gets from I, because the identity inside that message would not match I (it would contain B's identity). This fixed protocol is commonly referred to as the Needham-Schroeder-Lowe protocol.

Lastly, the paper demonstrates that the fact that the fixed protocol is not vulnerable to attacks on a small-scale network of those 3 agents can be generalized to a network of arbitrary scale, by showing that any attack on this large-scale network would correspond to an attack a small-scale one; thus, if there really existed an attack on a large-scale network, we would have found it on a small-scale one.

## 2.3 **Véronique Cortier, 2009, "*Verification of Security Protocols*" [1]**

The paper presents two contrasting approaches to model checking security protocols: the symbolic approach, which is the one used by Lowe to model check the Needham-Schroeder protocol, and the computational approach, which provides stronger security guarantees. Then, it attempts to synthesize the two into a combined approach that benefits from the advantages of both.

First, we define similar primitives and operations on terms. Some new operations are added:

- Symmetric encryption: enc(t, k) and dec(t, k) respectively encrypt and decrypt a term t with the symmetric key k. A symmetric key can both encrypt and decrypt messages.
- Signature: sign(t, $K_{priv}$) returns a signed version of the term t with private key $K_{priv}$, that can be verified for authenticity by anyone with the public key $K_{pub}$.

The paper then describes a deduction system out of those operations; it shows what information (as terms) an intruder can deduce from some given information (also as terms). For example, if the intruder is given the term enc(x, y) and the key y, then the intruder can deduce x. This approach reduces the problem of verifying deducibility of a term, given a set of known terms, to a proof search.

Another symbolic approach is described; using a constraint system to determine whether an attack is possible or not. In this method, a constraint system is formed such that any solution to the constraint system corresponds to an attack on the property of the protocol that we want to check, such as the confidentiality of a certain piece of information like a nonce in the Needham-Schroeder protocol. This approach only works given a bounded number of runs of the protocol.

The last symbolic approach described uses satisfiability to verify properties. We define some sets of clauses:

- $C_{I0}$: A set of clauses representing what the intruder knows initially
- $C_{NS}$: A set of clauses expressing what information is contained in each message (thus information that can be intercepted by the intruder) in different configurations of the protocol:
  - A talking to B, and symmetrically
  - A talking to I, and I (pretending to be A) talking to B
  - A talking to B, and I (pretending to be A) talking to B
  - etc.
- $C_I$: A set of clauses representing what information can be deduced from other information, much like the deduction system of the first approach.

By taking the union of those sets of clauses, we can build clauses representing all the possibilities of an intruder on the protocol. Then, to verify certain properties, we can add a clause to this big set, such as ¬(intruder can deduce $N_B$). With the resulting set of clauses, checking for satisfiability is now analogous to finding whether the intruder can deduce $N_B$ or not.

The paper also presents a computational approach to verifying security protocols. In the computational approach, rather than relying on abstract primitives and operations and assuming that the guarantees behind those primitives and operations work is true, we work at the bit-string level with a real implementation of the protocol. We introduce a parameter η called the security parameter. It will be used to determine the strength of the cryptographic primitives such as nonces (larger η means larger domain of nonces) and keys (larger η means longer keys). The intruder is not modelled as a particular process or by a certain set of capabilities; instead, it is modelled by a probabilistic Turing machine

running in polynomial time with respect to η.

In order to verify properties under this model, we check whether there exists an intruder modelled this way such that, given the possibility of interacting with runs of the protocols, it has a non-negligible probability of being able to accomplish what we don't want it to be able to accomplish, such as learning a nonce. This provides a much stronger security guarantee than the symbolic approach, because this works with an existing implementation of cryptographic primitives and allows the intruder to be any possible (polynomial-time) algorithm; it doesn't make any assumptions about how strong the cryptographic primitives are, because the strength of the intruder is directly related to the strength of the cryptographic primitives. The drawback to this approach is the extreme complexity of the approach itself; as such, it is generally only usable for simple protocols.

Finally, the paper presents some recent research efforts aiming to combine the two approaches by showing an equivalence from one model to the other; this way, an attack on the computational model correspond to an attack on the protocol as expressed in the symbolic model, or an attack on the primitives and operations which the symbolic model takes for granted. However, current results need assumptions on such primitives in order to work in the first place.

# 3. Analysis

Gavin Lowe's paper marked the importance of thoroughly model checking security protocols. The Needham-Schroeder public-key protocol was thought to be secure for the 17 years following its publication. The attack and the reasoning are all described using the symbolic approach and were obtained as a trace in a CSP system. To obtain this trace, only a subset of the protocol was modelled: the communication between the two agents. While this subset is sufficient in order to find this particular attack, it doesn't model the entire Needham-Schroeder public-key protocol; it is missing the key-exchange phase which uses a common, trusted key server S. This simplification of the protocol would be of no importance if we assume that the agents' keys never change; however, this is not true in practice. Indeed, keys may sometimes be accidentally leaked or otherwise revealed to the network, which means that they could fall into a potential adversary's knowledge. In these cases, the key-exchange part of the Needham-Schroeder is vulnerable to a replay attack (acknowledged in the paper) that couldn't be detected if we omitted this part out of the protocol.

The attack goes like this, where A and C are initiators, B is a responder, I is an intruder, and S is the key server:

1. $A \rightarrow S : \quad A.B$
   A wishes to talk to B, and requests B's key from the key server S.
2. $S \rightarrow A : \quad sign(B_{pub}.B, S_{priv})$
   The key server responds by signing B's public key and B's identity and sending the signed message back to A.
   This message is saved by I.
3. A potentially long period of time goes by, during which B's private key gets leaked and is now in possession of I. B generates a new key and sends it to the key server, but other agents do not know about it yet.
4. $C \rightarrow S : \quad C.B$
   C wishes to talk to B, and requests B's key from the key server S.
   The intruder I blocks this message from reaching the key server.

5. `I → C :` $sign(B_{pub}.B, S_{priv})$
   The intruder replays the packet it had captured in step 2, sending it to C but pretending to be S. C now believes that $B_{pub}$ is B's public key, but this is false.
   From now on, any message that C sends to B will be encrypted using B's compromised key. All of those messages can therefore be blocked and decrypted by I, who can now impersonate B.

This attack [2] affects both the symmetric and the asymmetric variants of the Needham-Schroeder protocol, as the key-exchange part of both variants is similar. The attack can be fixed by introducing a nonce exchange with the key server in order to guarantee freshness of the key returned by the key server.

This shows how thorough one has to be in order to comprehensively model a protocol in order to model check it. Indeed, finding this attack using Gavin Lowe's CSP method would not be a simple matter of expanding the set of events to include the key server messages; we would also have to express the fact that keys can be compromised, and express how agents (processes) react to this event, what the intruder learns from it and how can it interfere with it, etc. This opens a rather large new set of problems: How do we safely generate a new key and get the key server to accept it to replace the old one? How do we prevent illegitimate users from doing the same and replacing the key with their own? How do agents know that they should ask the key server for B's public key again? What if the key is leaked in the middle of a run of the protocol? These problems are not addressed by the Needham-Schroeder protocol, nor are they what the Needham-Schroeder protocol is intended to provide, but the protocol inherently relies on the key-exchange process being safe and reliable. Therefore, these are additional security assumptions that we need to make in order to model check the Needham-Schroeder protocol itself. This is a problem of the symbolic approach; we have to make a lot of security assumptions, and it can sometimes be quite hard to list all the assumptions we did make because we may not even realize we are making them.

Another type of assumptions made while using symbolic approach is the set of capabilities of the intruder. The most common model is the Dolev-Yao intruder, who can be thought of as the messenger between two parties: it can read the contents of every message it carries, it can tamper with them, and it can send new (fake) messages on behalf of either party or on its own behalf. But we cannot be certain that these are the only possible types of attack. The intruder could be in possession of a very powerful computing machine and/or an unpublished algorithm capable of breaking the encryption used in the protocol's messages, or, on a protocol where nonces are generated pseudo-randomly, the intruder could have the ability to know an agent's random seed and thus obtain the same pseudo-random nonce.

The computational approach solves part of this problem. Instead of making security assumptions, the intruder is modelled by a Turing machine that runs in polynomial time with respect to the cryptographic scheme's security parameter; thus, instead of making an assumption on what the intruder is able to do, we make an assumption on the computational power of the intruder, and the existence (or lack, rather) of algorithms that can break the cryptographic primitives or operations in polynomial time. This is a stronger guarantee than in the symbolic approach, as it doesn't make any assumptions about how the intruder breaks a protocol. If the approach turns out to be wrong, and the intruder indeed does have a polynomial-time way to break the protocol, then this solution can usually be mapped back onto the model and reduced either to a flaw in the protocol itself, either to a polynomial-time solution to breaking one of the cryptographic primitives or operations that we assumed to be safe in the symbolic approach, which is usually a more significant discovery than a mere flaw in a protocol.

A model check on the Needham-Schroeder-Lowe protocol using the computational approach was made [5]; on top of proving computationally that the (fixed) protocol is secure, we can also determine how strong the encryption scheme needs to be.

The computational approach doesn't come without its own problems, however. The process is much more involved, computationally expensive, and thus only really doable for small protocols such as the Needham-Schroeder protocol.

# 4. Conclusion

Model checking security protocols is critical to ensure their usefulness. Multiple approaches to doing so exist.

The symbolic approach works at the level of abstract cryptographic primitives and operations, and thus requires a lot of assumptions about the inner workings of those cryptographic elements in order to hold any guarantee of security; however, it is easy to perform and can be automated. This is the approach that led to Gavin Lowe's discovery of the flaw in the Needham-Schroeder protocol.

The computational approach, on the other hand, works at the bit-string level and doesn't make assumptions about how the cryptographic elements work. Instead, it depends on the actual implementation of those cryptographic elements, and relies on their strength such that a potential adversary cannot break them in polynomial time. This approach is long and difficult, but provides stronger security guarantees.

The two models can be synthesized into a hybrid approach: if we are to show that any trace of a computational adversary can be mapped to a trace of a symbolic adversary, then it suffices to perform a symbolic model check on the protocol to guarantee its security.

# References


1. Véronique Cortier, 2009, *Verification of Security Protocols*, LORIA – CNRS, Nancy

2. Dorothy E. Denning and Giovanni Maria Sacco, 1981, *Timestamps in key distribution protocols*, Communications of the ACM, Volume 24 Issue 8 Pages 533-536

3. Danny Dolev and Andrew C. Yao, 1983, *On the security of public key protocols, IEEE transactions on Information Theory*, Volume IT-29 pages 198-208

4. Gavin Lowe, 1996, *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*, Oxford University Computing Laboratory, Oxford

5. Bogdan Warinschi, 2003, *A Computational Analysis of the Needham-Schroeder(-Lowe) Protocol*, Department of Computer Science and Engineering, University of California, San Diego