Revisiting the BAN-Modified Andrew Secure RPC Protocol

Alberto Gugel
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
alberto.gugel@port.ac.uk

Benjamin Aziz
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

Geoff Hamilton
School of Computing
Dublin City University
Dublin, Ireland
geoff.hamilton@computing.dcu.ie

Abstract

We have analysed the well-known BAN modified Andrew Secure RPC authentication protocol by means of the AVISPA Web tool considering all the available back-ends and with the basic configurations of sessions. The protocol has been found vulnerable to a replay/mutation attack based on homomorphism by one of the back-ends. In order to fix it, we integrated into the protocol a common solution, including a new addition to the original protocol and the solution proposed by Liu, Ma and Yang, who earlier found a man-in-the-middle attack by means of a different model checker instantiated with different session compositions. When we tested this solution in AVISPA, under both conditions, we discovered that AVISPA considers it safe, while it can be demonstrated that it suffers from the same mutation attack as in the original protocol.

1 Introduction

Protocol verification using formal methods tools is a rich area of research [1, 2, 3, 4, 5, 7, 8, 10, 14, 15] that has contributed a great deal to the understanding and development of security and safety properties and solutions in critical computing systems in recent years. This paper presents the results of the application of a well-known formal analysis tool, namely AVISPA (Automated Validation of Internet Security Protocols and Applications) to a famous authentication protocol, namely BAN-modified Andrew RPC. Although seemingly straightforward, the analysis sheds new light into the security of the protocol and the behaviour of the tool. Infamously, Gavin Lowe found an attack [10] on the Needham-Schroeder protocol 18 years after the protocol was originally published in [13]. This goes to show how notoriously error-prone security protocols are and that it is never late to revisit any such protocol.

The AVISPA tool[1] has been developed to enable the automatic validation of security protocols. The tool is available both as a downloadable standalone package (running on UNIX platforms) and as a web application and, for the purpose of our experiments, we employed the latter. This choice does not affect the correctness of the results eventually obtained because both the offered solutions use the same formal language (HLPSL) to specify the protocol and the same logics to verify it. More precisely, through HLPSL each agent is modelled as a finite state machine capable of sending and receiving messages over an (unsafe) channel, triggering in this way state transitions. All the reachable combinations of states have to be explored in order to establish if the protocol is safe or not [11] and AVISPA performs this search adopting four different approaches correspondent to the four back-ends: OFMC, Cl-AtSe, SATMC and TA4SP [17]. We shall not delve in this paper into the theoretical workings behind these backends and focus on an applied approach to the AVISPA toolkit.

[1] www.avispa-project.org

The present paper is structured as follows: after a review of the related work in Section 2, we will focus on the BAN modified Andrew Secure RPC protocol analysing its steps and its authentication properties in Section 3. Then in Section 3.1, we discuss the attack we found against it and the fixing we devised, showing that the AVISPA tool considers it safe. Subsequently in Section 3.2, we will discuss the reason why this result is not correct and, finally, we will analyse the authentication properties of the BAN modified Andrew Secure protocol in Section 4 in light of this attack, and conclude the paper with discussion of future work 5.

2 Related Work

The area of protocol security analysis is rich in its literature, therefore we only discuss here the most relevant literature. The Andrew Secure RPC protocol, as shown in Figure 1, was implemented around 1986 at Carnegie Mellon University with the partnership of IBM as part of the Andrew distributed system[2] whose aim was to provide the students with a file sharing environment across the University.

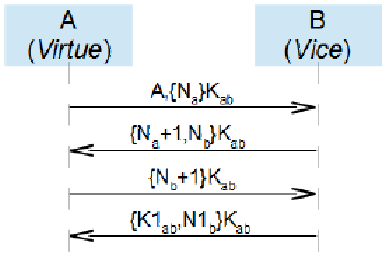

Figure 1: The Andrew RPC Protocol

In particular, the protocol was intended to ensure mutual authentication and secret exchange of fresh keys to securely execute remote procedure calls between client (called Virtue) and server (called Vice) components of the system. The author, Mahadev Satyanarayanan, then described it in [16], after it was already amended by Burrows, Abadi and Needham (BAN) in [6] in order to protect it from replay attacks by bounding the fourth message to the session in which the protocol initiator (Virtue) acts by means of the nonce it produced at the beginning, as shown in Figure 2.

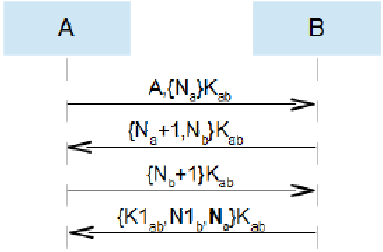

Figure 2: The BAN-modified Andrew RPC Protocol

The handshake described in the BAN modified Andrew Secure protocol is made of four steps. First, A, playing as initiator, starts the sequence providing B, acting as responder, with its identity and the nonce N_a (on which it will be performing the responder's authentication) encrypted with the previously

[2] http://www.cmu.edu/corporate/news/2007/features/andrew/index.shtml

shared key K_{ab}:

$$A \rightarrow B : A, \{N_a\}K_{ab}$$

Then, B responds with a fully encrypted message containing the successor of N_a and another nonce, N_b, on which it will be performing A's authentication.

$$B \rightarrow A : \{N_a + 1, N_b\}K_{ab}$$

When A receives it, it compares N_a and $N_a + 1$ and, if coherent, authenticates B because it is the only agent that could have decrypted the second part of the first message and gained the knowledge of N_a. With the third message, A simply replies to B with the nonce it received in the previous message increased by 1 so that B, on the other side, can authenticate A comparing N_b and $N_b + 1$.

$$A \rightarrow B : \{N_b + 1\}K_{ab}$$

Finally, B sends to A the new shared key $K1_{ab}$, a new nonce $N1_b$ (for further communications) and N_a (that represents BAN addition to the original Andrew Secure RPC).

$$B \rightarrow A : \{K1_{ab}, N1_b, N_a\}K_{ab}$$

The goals of this protocol are to guarantee a mutual authentication between A and B. Based on this and on cryptography, this further guarantees a secret exchange of K_{1ab} and N_{1b}. More formally, the goals are:

- (G.1) Secrecy of K_{1ab}
- (G.2) Secrecy of N_{1b}
- (G.3) Authentication of B by A on N_a
- (G.4) Authentication of A by B on N_b

The level of authentication the protocol can guarantee is the message agreement when A authenticates B, and the weak agreement when B authenticates A, as it can be proved by positioning Commit and Running events [15, 12] along the protocol run, as shown in Figure 3.

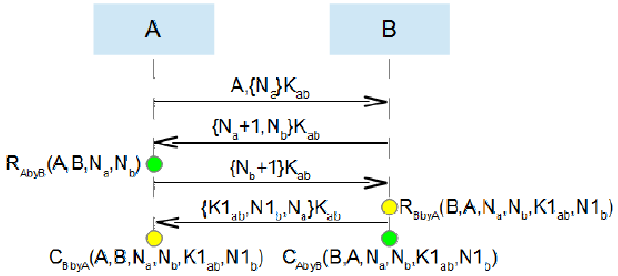

Figure 3: Commit and Running events in the BAN-modified Andrew RPC Protocol

In fact, in the first case, when A commits:

- B is *alive* because it has sent the last message (otherwise A would not have committed)
- both A and B have already compared their respective nonces in the preceding two steps, thus achieving some level of certainty on the other party's identity (*weak agreement*)
- both A and B know the content of the fourth message (*message agreement*)

On the other hand, when B commits:

- A is *alive* because C_{AbyB} cannot be reached without passing through R_{AbyB}
- both B and A knows about each other thanks to the nonces comparison, reaching *weak agreement*
- B holds more pieces of information than A, so the protocol cannot guarantee *message agreement*

In [9], Liu, Ma and Yang (LMY) found a man-in-the-middle attack to the version amended by BAN initializing their SAT-based tool with two sessions in which A and B exchange their roles: in the first A acts as initiator and B as responder, while in the second B acts as initiator and A as responder, as shown in Figure 4.

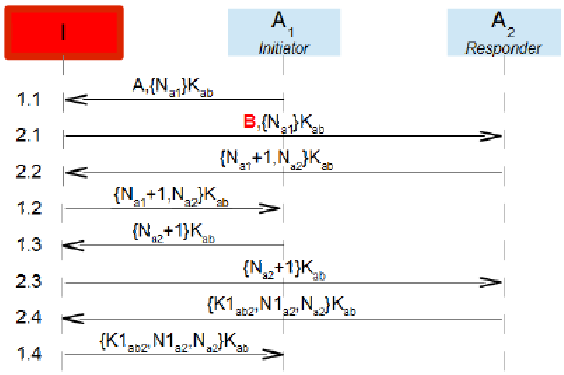

Figure 4: Man in the middle attack with A and B exchanging their roles

In this situation, the intruder is able to hide B's identity substituting B to A in Message 2.1 (this piece of information is in clear), i.e. the first of session number 2, so that both the instances of A (A1 and A2) think of being talking with an instance of B (respectively B1 and B2). As a consequence, at the end, A thinks of having authenticated B in both the sessions when, in truth, it authenticated itself two times. Moreover, A agrees in both the sessions on the same session key (K_{ab2}) that was created by itself and not by B. It is similar to the situation where the intruder builds his own session with A1 and A2, without any of them being aware of that.

The basic reason behind this attack is that the initiator never receives a direct indication on the responder's identity. Therefore, LMY suggested adding the responder's identity to the second message of the protocol, as shown in Figure 5 below.

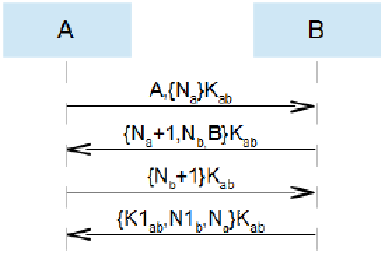

Figure 5: Amended version proposed by Liu, Ma and Yang

This way, Message 2.2 would have looked like:

$$\{N_{a1}+1, N_{a2}, A\}K_{ab} \tag{1}$$

Thus, A in session 1 would have known that it was actually talking with itself, rather than with B.

3 The BAN modified Andrew Secure RPC protocol in AVISPA

A full HLPSL protocol specification was developed (see Appendix A). In this section, we will focus on the main modelling issue faced: only one of the AVISPA back-ends (OFMC) natively supports replay attacks detection. To overcome this shortcoming and following what the AVISPA Team suggest [18], we modelled two identical parallel sessions A-B, in addition to the ones in which the intruder is specifically involved and "authorized" to deal with the honest agents (later in this paper we will refer to it as the "basic configuration of sessions" or "configuration 1"):

`session(a,b,kab,succ)`$\wedge$ `session(a,b,kab,succ)`$\wedge$
`session(a,i,kai,succ)`$\wedge$ `session(i,b,kib,succ)`

where `succ` represents the function used to compute $N_a + 1$ and $N_b + 1$, and `kab`, `kai` and `kib` represent the pre-shared keys that the agents (intruder included) must know before starting the handshake described in the protocol. However, this is not sufficient and, as again suggested by the AVISPA team [2], we need to add a further goal: the strong authentication on $K1_{ab}$ of one agent by the other (who authenticates who is not important). This way, the key is created in the current session and cannot be replayed from another session without failing to achieve the goal. As a consequence, the protocol goals specified in AVISPA are the followings:

- (G.1) Secrecy of $K1_{ab}$
- (G.2) Secrecy of $N1_b$
- (G.3) Authentication of B by A on N_a
- (G.4) Authentication of A by B on N_b
- (G.5) Authentication of B by A on $K1_{ab}$

For the rest, the protocol has been translated in HLPSL following the original specification literally.

3.1 A New Attack

We submitted our HLPSL protocol specification to all the four back-ends available in AVISPA and one of them (CL-AtSe) found it unsafe. The result of the performed test is summarized in Table 1.

Table 1: BAN modified Andrew Secure RPC attack summary

	G1	G2	G3	G4	G5
Summary	SAFE	SAFE	SAFE	SAFE	UNSAFE
Analysed	-	-	-	-	422 States
Reachable	-	-	-	-	119 States
Translation	-	-	-	-	0.02 Seconds
Computation	-	-	-	-	0.00 Seconds

Note that, for the purpose of this paper, we are not interested in discussing how many states the model checker has checked and how long did it take to compute the search of a state that corresponds to an attack, but these data were reported for completeness. We concentrate, instead, on the fact that the protocol failed to meet Goal G.5, the one added in order to detect replay attack. The full attack trace is shown in the Appendix E, where the notation A_x or B_y indicates agents A and B respectively in session x

and y. It describes a replay/mutation attack based on homomorphism in which the intruder stays between the two A-B sessions we set up and plays a passive role for almost all the two protocol runs: it basically intercepts all the messages and forwards them to the right receiver without applying any changes. In particular, it intercepts and remembers messages 1.4 and 2.4:

$$(1.4)\ \ B_1 \rightarrow A_1 :\ \{K1_{ab1}, N1_{b1}, N_{a1}\}K_{ab}$$
$$(2.4)\ \ B_2 \rightarrow A_2 :\ \{K1_{ab2}, N1_{b2}, N_{a2}\}K_{ab}$$

Note that both 1.4 and 2.4 are encrypted with K_{ab}. Finally, the intruder takes an action: combining 1.4 and 2.4, it creates a new message (2.4(I)) and sends it to A_2 in place of message 2.4. More precisely, the new message contains the key and the nonce created for further communications in session 1, i.e. the ones B_1 created for A_1, and the nonce created at the beginning by A_2 for B_2:

$$(2.4(I))I \rightarrow A_2 : \{K1_{ab1}, N1_{b1}, N_{a2}\}K_{ab}$$

When A_2 receives it, if the protocol implementation does not foreseen a mechanism to check if keys and nonces where previously used, A_2 will accept the message, authenticate B_2 on N_{a2} and, finally, it will be induced to use $K1_{ab1}$ and $N1_{b1}$ with B_2 that, however, will be using $K1_{ab2}$ and $N1_{b2}$.

This attack is possible under the assumption that the system is implemented using a homomorphic encryption scheme with respect to concatenation. ECB (Electronic Code Book) is an example of this kind of encryption and works if the keys' and nonces' length is a multiple of a fixed length block (e.g. 64 bit). With ECB, both the messages can be seen as the juxtaposition of the encryption (made through K_{ab}) of their components:

$$(1.4)\ \ \{K1_{ab1}, N1_{b1}, N_{a1}\}K_{ab} = \{K1_{ab1}\}K_{ab}, \{N1_{b1}\}K_{ab}, \{N_{a1}\}K_{ab}$$
$$(2.4)\ \ \{K1_{ab2}, N1_{b2}, N_{a2}\}K_{ab} = \{K1_{ab2}\}K_{ab}, \{N1_{b2}\}K_{ab}, \{N_{a2}\}K_{ab}$$

Once the intruder has decomposed the messages, it can compose a new message choosing the first two (encrypted) components of 1.4 and the last from 2.4 and concatenate them. This, for the properties of homomorphism, is equivalent to message 2.4(I).

$$(2.4(I))\ \ \{K1_{ab1}\}K_{ab}, \{N1_{b1}\}K_{ab}, \{N_{a2}\}K_{ab} = \{K1_{ab1}, N1_{b1}, N_{a2}\}K_{ab}$$

3.2 A False Positive

To fix the protocol and protect it from the attack described above, we thought at first that an effective solution would be to add N_b to the fourth message, as shown in Figure 6. The idea behind this approach

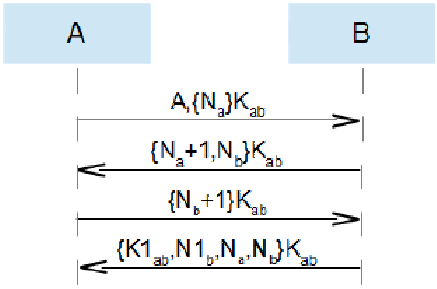

Figure 6: Amendment Version 0

was that N_a and N_b form a sort of session identifier that can be recognized by A and then used to "validate"

the session itself. We will refer to this version as "version 0" of our amendment. We added this piece of information into the HLPSL specification (for the case of two parallel A-B sessions, see Appendix B) and, when tested in AVISPA with the basic configuration of session defined early in this paper, it returned a completely safe result. When, instead, we tested our version 0 in the starting condition used by LMY in their experiments (configuration 2: A-B, B-A, A-Intruder, Intruder-B), AVISPA found it was vulnerable to the same attack LMY found on the original protocol (see Figure 4): A fails to authenticate B on N_a as in fact it authenticates itself to itself in the session in which it plays as responder.

Therefore, we decided to integrate the fixing proposed by LMY in our version 1 amendment: in the second message of the protocol, B has to communicate its identity, as shown in Figure 7. At this point,

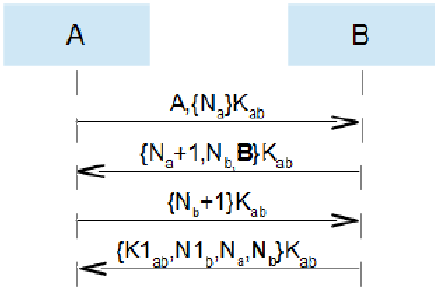

Figure 7: Amendment Version 1

when tested in both configurations 1 and 2 (i.e. for the case of two parallel A-B sessions (Appendix C) and the case of A-B/B-A sessions (Appendix D) with only authentication of B by A), the protocol was found to be robust. A summary of the tests and their results are shown in Table 2.

Table 2: Test results summary

Amendment version	Session configuration	OFMC	CL-AtSe	SATMC	TA4SP
0	1	SAFE	SAFE	SAFE	SAFE
0	2	UNSAFE (G.3)	UNSAFE (G.3)	UNSAFE (G.3)	SAFE
1	1	SAFE	SAFE	SAFE	SAFE
1	2	SAFE	SAFE	SAFE	SAFE

4 Discussion

Despite the apparently incontrovertible result given by AVISPA, it can be demonstrated that the homomorphic attack is still possible. In fact, adding N_b to the fourth message (and B to the second) still allows the intruder to use homomorphism to decompose message 1.4 and 2.4:

$$(1.4)\ \{K1_{ab1}, N1_{b1}, N_{a1}, N_{b1}\}K_{ab} = \{K1_{ab1}\}K_{ab}, \{N1_{b1}\}K_{ab}, \{N_{a1}\}K_{ab}, \{N_{b1}\}K_{ab}$$

$$(2.4)\ \{K1_{ab2}, N1_{b2}, N_{a2}, N_{b2}\}K_{ab} = \{K1_{ab2}\}K_{ab}, \{N1_{b2}\}K_{ab}, \{N_{a2}\}K_{ab}, \{N_{b2}\}K_{ab}$$

Then, as before, it can compose a new message choosing the (encrypted) new key and nonce from session 1 and the (encrypted) nonces exchanged during session 2; this, for the properties of homomorphism, is

equivalent to creating an encrypted message starting from the considered components in clear:

$$(2.4(I)) \quad \{K1_{ab1}\}K_{ab}, \{N1_{b1}\}K_{ab}, \{N_{a2}\}K_{ab}, \{N_{b2}\}K_{ab} = \{K1_{ab1}, N1_{b1}, N_{a2}, N_{b2}\}K_{ab}$$

In light of this attack, the protocol can only guarantee the weak agreement to the participating agents. It loses the message agreement property while A authenticates B because the attack demonstrates that A and B do not always agree on the data they respectively hold. This can also be seen through the Commit and Running events, as shown in Figure 8, an extract of the attack trace.

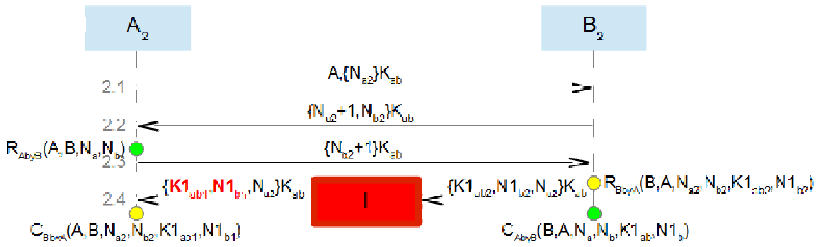

Figure 8: Commit and Running events when the attack occurs

When A authenticates B (yellow dots), the protocol guarantees, first of all, the aliveness of B because there is no way to arrive to C_{BbyA} without passing through R_{BbyA}; secondly, it guarantees weak agreement because A and B have already mutually authenticated in messages 2.2 and 2.3; finally, it cannot guarantee the message agreement because of the attack we described so far. The other authentication direction is not affected by the attack and, thus, preserves its weak agreement as previously shown.

5 Conclusion and Future Work

We have analysed the BAN modified Andrew Secure RPC authentication protocol by means of AVISPA with two identical parallel sessions (A-B,A-B) and the protocol has been found vulnerable to a replay/-mutation attack based on homomorphism. The fixing devised, further integrated with the enhancement proposed by LMY, seemed to be effective because AVISPA considered it safe, but we finally demonstrate that, in theory, the attack is still possible. This leads us to three main conclusions.

First, adopting homomorphic encryption schemes, especially if with respect to concatenation (like ECB), is not compatible with implementing the BAN-modified Andrew Secure RPC protocol to establish authentication and secure fresh key exchange in a system. More generally, any protocol should be checked against the effects of homomorphic cryptography. Second, AVISPA is an easy-to-use tool as it permits newbies to quickly analyse security protocols and work on their reasoning, however false positives are possible as with any automated tool. Therefore, any protocol modifications need to be carefully and critically analysed to ensure they do not introduce false positives or false negatives. Finally, generally speaking, it is always a good practice to test protocols in more than one initial set-up condition in order to consider the various possible permutations of the protocol instances.

For future work, we plan to suggest a new version of the protocol and to apply the AVISPA tool to other, well-known protocols. We also plan to study the false positive and false negative ratios of the tool, in order to provide better idea of its accuracy.

References

[1] Martín Abadi and Bruno Blanchet. Computer-assisted verification of a protocol for certified email. In Radhia Cousot, editor, *Proc. of the 10th International Symposium in Static Analysis (SAS 2003)*, volume 2694 of *Lecture Notes in Computer Science*, pages 316–335, San Diego, California, U.S.A., June 2003. Springer-Verlag.

[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. In David Schmidt, editor, *Programming Languages and Systems: Proc. of the 13th European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354, Barcelona, Spain, March 2004. Springer-Verlag.

[3] Alessandro Armando and Luca Compagna. Satmc: A sat-based model checker for security protocols. In *Logics in Artificial Intelligence, 9th European Conference, (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, pages 730–733. Springer-Verlag, 2004.

[4] Benjamin Aziz and Geoff Hamilton. Verifying a delegation protocol for grid systems. *Future Generation Computer Systems: The International Journal of Grid Computing and eScience*, 27(5):476–485, 2011.

[5] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. Yapa: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.*, 14(1):4, 2013.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1989.

[7] Vincent Cheval and Bruno Blanchet. Proving More Observational Equivalences with ProVerif. In David Basin and John Mitchel, editors, *The Second International Conference on Principles of Security and Trust (POST 2013)*, volume 7796 of *Lecture Notes in Computer Science*, pages 226–246. Springer-Verlag, 2013.

[8] Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.

[9] W. Liu, W. Ma, and Y. Yang. A New Attack on the BAN Modified Andrew Secure RPC Protocol. In *Proc. of the Second International Conference on Network Security, Wireless Communications and Trusted Computing (Volume 2)*, pages 219–222. IEEE Computer Society, 2010.

[10] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Proc. of the Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, Passau, Germany, March 1996. Springer-Verlag.

[11] M. Mannan. Mini Project 1, Cryptoprotocol Specification and Verification in the AVISPA Tool. Unpublished internal document.

[12] Matteo Maffei. *Dynamic Typing for Security Protocols*. PhD thesis, 2006.

[13] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[14] Laurence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, January 1998.

[15] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley Professional, December 2000.

[16] M. Satyanarayanan. Integrating Security in a Large Distributed System. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[17] The AVISPA Team. AVISPA v1.1 User Manual, 2006.

[18] The AVISPA Team. HLPSL Tutorial, A Beginner's Guide to Modelling and Ana-lysing Internet Security Protocols, 2006.

A BAN-modified Andrew Secure RPC protocol HLPSL specification (two parallel A-B sessions)

```
role alice (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by A def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 0
transition
1. State = 0 /\ RCV(start) =|>
State' := 2 /\ Na' := new() /\ SND(A.{Na'}_Kab)
2. State = 2 /\ RCV({Succ(Na).Nb'}_Kab) =|>
State' := 4 /\ SND({Succ(Nb')}_Kab) /\ witness(A,B,bob_alice_nb,Nb')
3. State = 4 /\ RCV({K1ab'.N1b'.Na}_Kab) =|>
State' := 6 /\ request(A,B,alice_bob_na,Na) /\
request(A,B,alice_bob_k1ab,K1ab')
end role
role bob (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by B def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 1
transition
1. State = 1 /\ RCV(A.{Na'}_Kab) =|>
State' := 3 /\ Nb' := new() /\ SND({Succ(Na').Nb'}_Kab)
2. State = 3 /\ RCV({Succ(Nb)}_Kab) =|>
State' := 5 /\ K1ab' := new() /\ N1b' := new() /\
SND({K1ab'.N1b'.Na}_Kab) /\
secret(K1ab',k1ab,{A,B}) /\ secret(N1b',n1b,{A,B}) /\
request(B,A,bob_alice_nb,Nb) /\ witness(B,A,alice_bob_na,Na) /\
witness(B,A,alice_bob_k1ab,K1ab')
end role
role session (A,B: agent,Kab: symmetric_key,Succ: hash_func
) def=
local
SND_A,RCV_A,SND_B,RCV_B: channel(dy)
composition
alice(A,B,SND_A,RCV_A,Kab,Succ) /\ bob(A,B,SND_B,RCV_B,Kab,Succ)
end role
role environment() def=
const
a,b: agent,
kab: symmetric_key,
succ: hash_func,
alice_bob_na,bob_alice_nb,k1ab,n1b: protocol_id

intruder_knowledge = {a,b,kai,kib,succ}

composition
session(a,b,kab,succ) /\ session(a,b,kab,succ) /\
session(a,i,kai,succ) /\ session(i,b,kib,succ)

end role
goal
secrecy_of k1ab, n1b
authentication_on alice_bob_na, bob_alice_nb, alice_bob_k1ab
end goal
environment()
```

B First amended version of BAN-modified Andrew Secure RPC protocol HLPSL specification (version 0, two parallel A-B sessions)

```
role alice (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by A def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 0
transition
1. State = 0 /\ RCV(start) =|>
State' := 2 /\ Na' := new() /\ SND(A.{Na'}_Kab)
2. State = 2 /\ RCV({Succ(Na).Nb'}_Kab) =|>
State' := 4 /\ SND({Succ(Nb')}_Kab) /\ witness(A,B,bob_alice_nb,Nb')
3. State = 4 /\ RCV({K1ab'.N1b'.Na.Nb}_Kab) =|>
State' := 6 /\ request(A,B,alice_bob_na,Na) /\
request(A,B,alice_bob_k1ab,K1ab')
end role
role bob (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by B def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 1
transition
1. State = 1 /\ RCV(A.{Na'}_Kab) =|>
State' := 3 /\ Nb' := new() /\ SND({Succ(Na').Nb'}_Kab)
2. State = 3 /\ RCV({Succ(Nb)}_Kab) =|>
State' := 5 /\ K1ab' := new() /\ N1b' := new() /\
SND({K1ab'.N1b'.Na.Nb}_Kab) /\
secret(K1ab',k1ab,{A,B}) /\ secret(N1b',n1b,{A,B}) /\
request(B,A,bob_alice_nb,Nb) /\ witness(B,A,alice_bob_na,Na) /\
witness(B,A,alice_bob_k1ab,K1ab')
end role
role session (A,B: agent, Kab: symmetric_key, Succ: hash_func
) def=
local
SND_A,RCV_A,SND_B,RCV_B: channel(dy)
composition
alice(A,B,SND_A,RCV_A,Kab,Succ) /\ bob(A,B,SND_B,RCV_B,Kab,Succ)
end role
role environment() def=
const
a,b: agent,
kab: symmetric_key,
succ: hash_func,
alice_bob_na,bob_alice_nb,k1ab,n1b: protocol_id

intruder_knowledge = {a,b,kai,kib,succ}

composition
session(a,b,kab,succ) /\ session(a,b,kab,succ) /\
session(a,i,kai,succ) /\ session(i,b,kib,succ)

end role
goal
secrecy_of k1ab, n1b
authentication_on alice_bob_na, bob_alice_nb, alice_bob_k1ab
end goal
environment()
```

C Second amended version of BAN-modified Andrew Secure RPC protocol HLPSL specification (version 1, two parallel A-B sessions)

```
role alice (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by A def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 0
transition
1. State = 0 /\ RCV(start) =|>
State' := 2 /\ Na' := new() /\ SND(A.{Na'}_Kab)
2. State = 2 /\ RCV({Succ(Na).Nb'.B}_Kab) =|>
State' := 4 /\ SND({Succ(Nb')}_Kab) /\ witness(A,B,bob_alice_nb,Nb')
3. State = 4 /\ RCV({K1ab'.N1b'.Na.Nb}_Kab) =|>
State' := 6 /\ request(A,B,alice_bob_na,Na) /\
request(A,B,alice_bob_k1ab,K1ab')
end role
role bob (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by B def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 1
transition
1. State = 1 /\ RCV(A.{Na'}_Kab) =|>
State' := 3 /\ Nb' := new() /\ SND({Succ(Na').Nb'.B}_Kab)
2. State = 3 /\ RCV({Succ(Nb)}_Kab) =|>
State' := 5 /\ K1ab' := new() /\ N1b' := new() /\
SND({K1ab'.N1b'.Na.Nb}_Kab) /\
secret(K1ab',k1ab,{A,B}) /\ secret(N1b',n1b,{A,B}) /\
request(B,A,bob_alice_nb,Nb) /\ witness(B,A,alice_bob_na,Na) /\
witness(B,A,alice_bob_k1ab,K1ab')
end role
role session (A,B: agent, Kab: symmetric_key, Succ: hash_func
) def=
local
SND_A,RCV_A,SND_B,RCV_B: channel(dy)
composition
alice(A,B,SND_A,RCV_A,Kab,Succ) /\ bob(A,B,SND_B,RCV_B,Kab,Succ)
end role
role environment() def=
const
a,b: agent,
kab: symmetric_key,
succ: hash_func,
alice_bob_na,bob_alice_nb,k1ab,n1b: protocol_id

intruder_knowledge = {a,b,kai,kib,succ}

composition
session(a,b,kab,succ) /\ session(a,b,kab,succ) /\
session(a,i,kai,succ) /\ session(i,b,kib,succ)

end role
goal
secrecy_of k1ab, n1b
authentication_on alice_bob_na, bob_alice_nb, alice_bob_k1ab
end goal
environment()
```

D Second amended version of BAN-modified Andrew Secure RPC protocol HLPSL specification (version 1, A-B/B-A sessions and only authentication of B by A (as per LMY)

```
role alice (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func) played_by A def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 0
transition
1. State = 0 /\ RCV(start) =|>
State' := 2 /\ Na' := new() /\ SND(A.{Na'}_Kab)
2. State = 2 /\ RCV({Succ(Na).Nb'.B}_Kab) =|>
State' := 4 /\ SND({Succ(Nb')}_Kab) /\ witness(A,B,bob_alice_nb,Nb')
3. State = 4 /\ RCV({K1ab'.N1b'.Na.Nb}_Kab) =|>
State' := 6 /\ request(A,B,alice_bob_na,Na) /\
request(A,B,alice_bob_k1ab,K1ab')
end role
role bob (A,B: agent, SND,RCV: channel(dy), Kab: symmetric_key, Succ: hash_func
) played_by B def=
local
Na,Nb,N1b: text,
K1ab: symmetric_key,
State: nat
init
State := 1
transition
1. State = 1 /\ RCV(A.{Na'}_Kab) =|>
State' := 3 /\ Nb' := new() /\ SND({Succ(Na').Nb'.B}_Kab)
2. State = 3 /\ RCV({Succ(Nb)}_Kab) =|>
State' := 5 /\ K1ab' := new() /\ N1b' := new() /\
SND({K1ab'.N1b'.Na.Nb}_Kab) /\
secret(K1ab',k1ab,{A,B}) /\ secret(N1b',n1b,{A,B}) /\
request(B,A,bob_alice_nb,Nb) /\ witness(B,A,alice_bob_na,Na) /\
witness(B,A,alice_bob_k1ab,K1ab')
end role
role session (A,B: agent, Kab: symmetric_key, Succ: hash_func
) def=
local
SND_A,RCV_A,SND_B,RCV_B: channel(dy)
composition
alice(A,B,SND_A,RCV_A,Kab,Succ) /\ bob(A,B,SND_B,RCV_B,Kab,Succ)
end role
role environment() def=
const
a,b: agent,
kab: symmetric_key,
succ: hash_func,
alice_bob_na,bob_alice_nb,k1ab,n1b: protocol_id

intruder_knowledge = {a,b,kai,kib,succ}

composition
session(a,b,kab,succ) /\ session(b,a,kab,succ) /\ session(a,i,kai,succ) /\ session(i,b,kib,succ)

end role
goal
secrecy_of k1ab, n1b
authentication_on alice_bob_na, alice_bob_k1ab
end goal
environment()
```

E Attack trace found through the AVISPA Web tool to the BAN-modified Andrew Secure RPC protocol modelling the following sessions: A-B, A-B, A-I, I-B