A formal security proof for the ZRTP Protocol

Riccardo Bresciani and Andrew Butterfield
FMG, Trinity College Dublin
bresciar@cs.tcd.ie, Andrew.Butterfield@cs.tcd.ie

Abstract

When some agents want to communicate through a media stream (for example voice or video), the Real Time Protocol *(RTP) is used. This protocol does not provide encryption, so it is necessary to use* Secure RTP *(SRTP) to secure the communication. In order for this to work, the agents need to agree on key material and ZRTP provides them with a procedure to perform this task: it is a key agreement protocol, which relies on a Diffie-Hellman exchange to generate SRTP session parameters, providing confidentiality and protecting against* Man-in-the-Middle *attacks even without a public key infrastructure or endpoint certificates. This is an analysis of the protocol performed with ProVerif, which tests security properties of ZRTP; in order to perform the analysis, the protocol has been modeled in the applied π-calculus*[1].

1. Introduction

In the last years research has strongly focused on proofs of security. The verification step to ensure that a computer program or a protocol have certain requested properties is a crucial one, and this task has to be done preferentially by formal reasoning, rather than by tests and simulations, as the latter approach is not as exhaustive as the formal one.

There are two possible approaches to protocol verification: the formal model and the computational model. In the first model, we are in a highly idealized setting, therefore this can be effectively implemented in fully-automated protocol verifiers. The second approach borrows ideas from complexity theory and requires much more human intervention in proofs, and it is being automated only in very recent times. [5]

These verification techniques allow us to uncover design faults that may remain hidden for years. There are a lot of examples that can be recalled on this topic, for example a successful application of verification in the formal model can be found in [8, 9]: the popular Needham-Schroeder protocol dates back to 1978, but it was just in 1995 that Gavin Lowe found that an attack on the protocol was possible and proposed a modification. To achieve this goal Gavin Lowe has used the tool FDR, which is a model checker for CSP.

Besides generic model checkers such as FDR, there are tools which have been conceived with communication protocols in mind. In the present paper we use Bruno Blanchet's ProVerif: if the original Needham-Schroeder protocol is analysed with this tool, this same security flaw can be uncovered and a trace of the attack given.

The purpose of the present paper is to present the results of a proof of security on the ZRTP protocol in the formal model. This protocol is being submitted as a RFC to the IETF by Philip Zimmermann, Alan Johnston and Jon Callas [11]: the purpose of this protocol is to provide key agreement and parameter negotiation to establish an SRTP session.

SRTP (*Secure RTP*) is a secure profile of RTP (*Real-time Transport Protocol*): it deals with security and privacy issues that are not built-in to RTP. Agents wishing to communicate by means of SRTP must agree on session keys and parameters in order to establish a secure session: these negotiations may be effectively made through ZRTP.

ZRTP bases the key agreement procedure on a Diffie-Hellman exchange and on cached secrets established in preceding sessions (if any): this creates a new shared secret, from which all key material can be derived by means of one-way functions.

In case no valid secret is found in the cache, the protocol is vulnerable to a *Man-in-the-Middle* attack. To ensure that this attack has not been performed, ZRTP provides a method to detect it: the agents have a *short authentication string* (SAS), which is a one-way function of the ZRTP session key and that can be verbally compared. If the SASs do not match an attack is taking place.

Keying material is destroyed at the end of each session, thus ZRTP offers perfect forward secrecy.

Some key continuity is kept by means of cached secrets:

[1]The present work has emanated from research conducted with the financial support of Science Foundation Ireland.
This work has been done using exclusively free software; presented images include modifications of media from the Tango Icon Library.

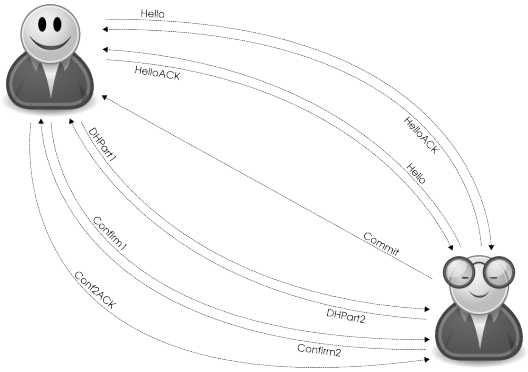

Figure 1. Key agreement call flow.

after each successful key agreement the cache is updated with a secret, which is a one-way function of the new secret generated through the key agreement procedure.

One key feature of ZRTP is that it does not rely on SIP signaling for key management, on any server or on any kind of public key infrastructure: only the endpoints have to interact for the key agreement to be performed.

ZRTP provides also protection against *Denial-of-Service* attacks, as it offers a way to detect and reject false ZRTP messages.

2. Protocol Description

In this section we will provide a brief description of the protocol, in order to show the way it works and how it enables two agents to agree on the key material and parameters needed for an SRTP session.

ZRTP has three possible working modes:

- the *Diffie-Hellman mode* is based on a Diffie-Hellman exchange: all SRTP keys are computed from the secret value computed by each party;
- the *Multistream mode* is usable only if there is already an active SRTP session between the endpoints: new SRTP keys for a new stream can be derived from a the preceding Diffie-Hellman exchange, avoiding the expensive computations of a new one;
- the *Preshared mode* does not rely on a Diffie-Hellman exchange, but on previously cached secrets only. This is secure as far as the secret cache is not corrupted. Indeed, even in this case, it mantains the perfect forward secrecy of the protocol, as keying material is deleted as soon as each session is terminated.

The present paper addresses the Diffie-Hellman mode only, as it is the setting where an attack can be performed: if no attack has succeeded in this session, the agents share reliable secrets, therefore all subsequent sessions in Multistream or Preshared mode that rely on them are safe, under the assumption that integrity of the secret cache is preserved.

During the run of the protocol two agents exchange messages: the *initiator* and the *responder*.

The key agreement and negotiation algorithm can be divided into 4 steps (see Figure 1):

- discovery — the agents exchange information about their identity and the supported session parameters;
- hash committment — the *initiator* starts the key agreement procedure;
- Diffie-Hellman exchange and key derivation — public keys are exchanged;
- confirmation — the endpoints acknowledge the successful key agreement.

After the discovery phase, when the endpoints have exchanged some basic information, the first step towards the negotiation of the key material is made with the hash commitment: besides containing all the session parameters that will have to be used and defining the roles in the protocol (it is symmetrical in the discovery phase, and the agent sending this message is the one who is willing to act as the initiator), it contains a value that commits the initiator not to change his Diffie-Hellman key pair.

In fact the initiator creates his keys prior to sending the hash commitment, but cannot reveal them immediately, as doing so could enable the other party (or an attacker) to choose maliciously his keys depending on the initiator's choice. The solution is to send a hash of the keys, concatenated to a hash of the message sent by the responder during the discovery phase: this second thing protects the protocol against *bid-down attacks*, that aim at making the agents rely on weaker algorithms, as an attacker may alter the information on supported session parameters. Finally the hash commitment prevents the agents from being able to influence deterministically the SAS: it is a function of the exchanged messages, so it can be influenced by an opportunistic choice of the agents' key — this cannot be done as the responder chooses his keys before knowing the initiator keys, and the initiator chooses his key before sending the hash commitment, that binds him to that choice. After the hash commitment the agents can perform the Diffie-Hellman exchange: they compute the Diffie-Hellman result by modular exponentiation of the other party's public key to the power of their own private key, thus computing a value that is known only to the two of them. Along with the key they send also a HMAC, keyed with a known string, for each retained secret that they already share: this allows them to distinguish matching secrets from non-matching ones (and discard them).

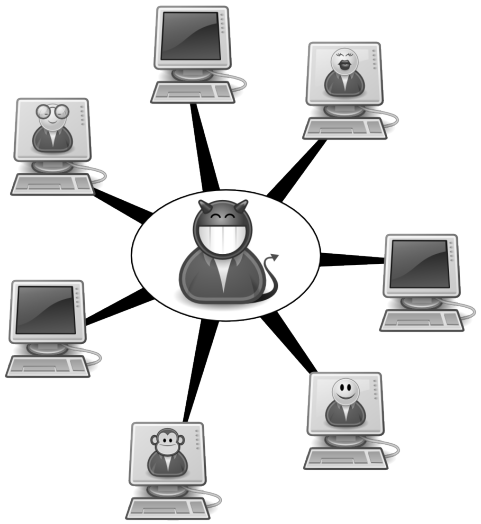

Figure 2. The Dolev-Yao model.

Both of these shared secrets will be concatenated with the hash of all the exchanged messages: the hash of this concatenation will be the new shared secret between the agents.

All the key material needed to establish a SRTP session will be derived from this shared secret by means of a variation of the HMAC function, keyed with different strings depending on the particular key to be generated. Also SAS is derived in this way. Once these operations have been completed, the endpoints exchange the confirmation messages to acknowledge that the key agreement procedure has been successful. They contain an encrypted block, which is encrypted using the newly generated keys: verifying the secrecy of this block will be the way to ensure that the protocol is safe.

The message exchange is protected with a chain of HMACs that cover each message: each HMAC is keyed with a value that is transmitted in clear only in the following message (thus it can be verified only in that moment). Moreover the values used as keys for the HMACs are generated from an 8-word nonce by subsequent hashing: for this reason they are referred to as *hash images*. The coherency of a hash image can be verified upon receipt of the following one. The original 8-word nonce is transmitted in the encrypted part of the confirmation messages.

3. Analysis

In the proposed analysis the protocol is modeled in the Applied π-calculus: the agents taking part in the protocol are expressed as two concurrent processes. The agents interact in a scenario, which is normally referred to as *Dolev-Yao model*, described in the following subsection.

3.1. The Dolev-Yao model

The Dolev-Yao model [7], schematised in Figure 2, assumes that:

- the net is under the intruder's control: messages can be intercepted and altered. New messages can be injected to the net;
- the cryptographic primitives are perfect;
- the protocol admits any number or participants and any number of parallel sessions;
- the protocol messages can be of any size.

The above formal model can be effectively captured by automatic protocol verifiers and it is much easier to be implemented than computational models: most automatic proofs on protocols have been done in this model.

In this model we can reason about an idealized version of the protocol, so we can abstract from the implementation issues: for example a flaw in an implementation of a protocol due to overflow will not be detected in the formal model, but a flaw due to misconception of the protocol will be found by a protocol verifier.

TERMS	M, N:==
Variables	x,y,z
Names	a,b,c,k,s
Constructor	$f(M_1, \ldots, M_n)$
PROCESSES	P, Q:==
Output	$\overline{M}\langle N \rangle.P$
Input	$M(x).P$
Destructor	$let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$
Conditional	$if\ M = N\ then\ P\ else\ Q$
Nil process	0
Parallel	$P \vert Q$
Replication	$!P$
Restriction	$(\nu a).P$

Figure 3. The syntax of the applied π-calculus.

Initiator =

νSVI. — generate secret value

νH0I. — generate hash image H0I

let $\{\texttt{H1I} = \mathcal{H}(\texttt{H0I})\}$ in — compute hash image H1I

let $\{\texttt{H2I} = \mathcal{H}(\texttt{H1I})\}$ in — compute hash image H2I

let $\{\texttt{H3I} = \mathcal{H}(\texttt{H2I})\}$ in — compute hash image H3I

hellor$((\texttt{HELLOSTRINGRI}, \texttt{H3RI}, \texttt{HMACHELLORI}))$. — wait for Hello message

$\overline{\text{helloackr}}\langle\texttt{helloack}\rangle$. — send HelloACK message

$\overline{\text{helloi}}\langle(\texttt{hellostringi}, \texttt{H3I}, \mathcal{H}_M((\texttt{hellostringi}, \texttt{H3I}), \texttt{H2I}))\rangle$. — send Hello message

helloacki(HELLOACKRI). — wait for HelloACK message

let $\{\texttt{PVI} = g^{\texttt{SVI}}\}$ in — compute public value

let $\{\texttt{SECRETSIDI} = \mathcal{H}_M(\texttt{initiator}, \texttt{secrets})\}$ in — compute IDs of the secrets

let $\{\texttt{HVI} = \mathcal{H}((\texttt{PVI}, \texttt{SECRETSIDI}, \texttt{H1I}, \texttt{HELLOSTRINGRI}, \texttt{H3RI}))\}$ in — compute hash commitment

$\overline{\text{commit}}\langle(\texttt{commitstring}, \texttt{HVI}, \texttt{H2I}, \mathcal{H}_M((\texttt{commitstring}, \texttt{HVI}, \texttt{H2I}), \texttt{H1I}))\rangle$. — send Commit message

dhpart1$((\texttt{PVRI}, \texttt{SECRETSIDRI}, \texttt{H1RI}, \texttt{HMACDHPART1RI}))$. — wait for DHPart1 message

if $\texttt{H3RI} = \mathcal{H}(\mathcal{H}(\texttt{H1RI}))$ then — check H3R

if $\texttt{HMACHELLORI} = \mathcal{H}_M((\texttt{HELLOSTRINGRI}, \texttt{H3RI}), \mathcal{H}(\texttt{H1RI}))$ then — check Hello HMAC

if $\texttt{SECRETSIDRI} = \mathcal{H}_M(\texttt{responder}, \texttt{secrets})$ then — check responder's IDs of the secrets

$\overline{\text{dhpart2}}\langle(\texttt{PVI}, \texttt{SECRETSIDI}, \texttt{H1I}, \mathcal{H}_M((\texttt{PVI}, \texttt{SECRETSIDI}, \texttt{H1I}), \texttt{H0I}))\rangle$. — send DHPart2 message

confirm1$((\texttt{CONFIRMRI}, \texttt{HMACSECRI}, \texttt{ENCH0RI}, \texttt{ENCSECRI}))$. — wait for Confirm1 message

let $\{\texttt{MHI} = \mathcal{H}((\texttt{HELLOSTRINGRI}, \texttt{H3RI}, \texttt{commitstring}, \texttt{HVI}, \texttt{H2I}, \texttt{PVRI}, \texttt{SECRETSIDRI}, \texttt{H1RI}, \texttt{PVI}, \texttt{SECRETSIDI}, \texttt{H1I}))\}$ in — compute the message hash

let $\{\texttt{S0I} = \mathcal{H}((\texttt{PVRI}^{\texttt{SVI}}, \texttt{secrets}, \texttt{MHI}))\}$ in — compute S0

let $\{\texttt{ZRTPKEYI} = \mathcal{K}(\texttt{S0I}, \texttt{zrtpi})\}$ in — compute ZRTP key

let $\{\texttt{ZRTPKEYRI} = \mathcal{K}(\texttt{S0I}, \texttt{zrtpr})\}$ in — compute responder's ZRTP key

let $\{\texttt{H0RI} = \mathcal{D}_{\texttt{ZRTPKEYRI}}(\texttt{ENCH0RI})\}$ in — decrypt H0R

let $\{\texttt{SECRI} = \mathcal{D}_{\texttt{ZRTPKEYRI}}(\texttt{ENCSECRI})\}$ in — decrypt responder's secret block

if $\texttt{H1RI} = \mathcal{H}(\texttt{H0RI})$ then — check H1R

if $\texttt{HMACDHPART1RI} = \mathcal{H}_M((\texttt{PVRI}, \texttt{SECRETSIDRI}, \texttt{H1RI}), \texttt{H0RI})$ then — check Confirm2 HMAC

if $\texttt{HMACSECRI} = \mathcal{H}_M((\texttt{H0RI}, \texttt{SECRI}), \texttt{ZRTPKEYRI})$ then — check integrity of encrypted part

$\overline{\text{confirm2}}\langle(\texttt{confirmi}, \mathcal{H}_M((\texttt{H0I}, \texttt{ZRTPKEYI}), \texttt{ZRTPKEYI}), \mathcal{E}_{\texttt{ZRTPKEYI}}(\texttt{H0I}), \mathcal{E}_{\texttt{ZRTPKEYI}}(\texttt{SECI}))\rangle$. — send Confirm2 message

conf2ack(CONF). — send Conf2ACK message

Figure 4. The "Initiator" process

3.2. Applied π-calculus and ProVerif

In order to reason about cryptographic protocols, Martín Abadi and Cédric Fournet have built the applied π-calculus [2] on top of Milner's π-calculus [10]: the main thing is that names are replaced by terms (the atomic values of the π-calculus are not enough to deal efficiently with the complexity of a cryptographic protocol). By using equational theories, it is possible to take fully advantage of this calculus to test security properties of communication protocols, as they allow to account for equational properties of the functions used in the protocols.

The syntax of the applied π-calculus is shown in Figure 3.

Once the protocol has been modeled in the Applied π-calculus (see Figures 4 and 5), the analysis can be performed with the tool ProVerif [1, 3] and will provide a formal proof of security for the model. ProVerif translates the Applied π-calculus process into a set of Horn clauses, that account for the initial knowledge of the attacker, for the inference rules he can apply to broaden his knowledge pool and for the messages that can be sent over the communication channels: by means of a resolution algorithm ProVerif will derive new clauses, which must not allow an attacker to compromise the protocol.

In the case of the present paper, the goal is proving that secrecy of data encrypted under the key, on which the agents have agreed by means of the protocol, is preserved and data is not disclosed to an attacker.

Responder =

νSVR. — generate secret value
νH0R. — generate hash image H0R
let $\{\texttt{H1R} = \mathcal{H}(\texttt{H0R})\}$ in — compute hash image H1R
let $\{\texttt{H2R} = \mathcal{H}(\texttt{H1R})\}$ in — compute hash image H2R
let $\{\texttt{H3R} = \mathcal{H}(\texttt{H2R})\}$ in — compute hash image H3R
$\overline{\text{hellor}}\langle(\texttt{hellostringr}, \texttt{H3R}, \mathcal{H}_M((\texttt{hellostringr}, \texttt{H3R}), \texttt{H2R}))\rangle$. — send Hello message
helloackr(HELLOACKIR). — wait for HelloACK message
helloi((HELLOSTRINGIR, H2IR, HMACHELLOIR)). — wait for Hello message
$\overline{\text{helloacki}}\langle\texttt{helloack}\rangle$. — send HelloACK message
commit((COMMITSTRINGIR, HVIR, H2IR, HMACCOMMITIR)). — wait for Commit message
if $\texttt{H3IR} = \mathcal{H}(\texttt{H2IR})$ then — check H3I
if $\texttt{HMACHELLOIR} = \mathcal{H}_M((\texttt{HELLOSTRINGIR}, \texttt{H3IR}), \texttt{H2IR})$ then — check Hello HMAC
let $\{\texttt{PVR} = g^{\text{SVR}}\}$ in — compute public value
let $\{\texttt{SECRETSIDR} = \mathcal{H}_M(\texttt{responder}, \texttt{secrets})\}$ in — compute IDs of the secrets
$\overline{\text{dhpart1}}\langle(\texttt{PVR}, \texttt{SECRETSIDR}, \texttt{H1R}, \mathcal{H}_M((\texttt{PVR}, \texttt{SECRETSIDR}, \texttt{H1R}), \texttt{H0R}))\rangle$. — send DHPart1 message
dhpart2((PVIR, SECRETSIDIR, H1IR, HMACDHPART2IR)). — wait for DHPart2 message
if $\texttt{H2IR} = \mathcal{H}(\texttt{H1IR})$ then — check H2I
if $\texttt{HMACCOMMITIR} = \mathcal{H}_M((\texttt{COMMITSTRINGIR}, \texttt{HVIR}, \texttt{H2IR}), \texttt{H1IR})$ then — check Commit HMAC
if $\texttt{SECRETSIDIR} = \mathcal{H}_M(\texttt{initiator}, \texttt{secrets})$ then — check initiator's IDs of the secrets
if $\texttt{HVIR} = \mathcal{H}((\texttt{PVIR}, \texttt{SECRETSIDIR}, \texttt{hellostringr}, \texttt{H3R}))$ then — check HVI
let $\{\texttt{MHR} = \mathcal{H}((\texttt{hellostringr}, \texttt{H3R}, \texttt{COMMITSTRINGIR}, \texttt{HVIR}, \texttt{H2IR}, \texttt{PVR}, \texttt{SECRETSIDR}, \texttt{H1R}, \texttt{PVIR}, \texttt{SECRETSIDIR}, \texttt{H1IR}))\}$ in — compute the message hash
let $\{\texttt{S0R} = \mathcal{H}((\texttt{PVIR}^{\text{SVR}}, \texttt{secrets}, \texttt{MHR}))\}$ in — compute S0
let $\{\texttt{ZRTPKEYR} = \mathcal{K}(\texttt{S0R}, \texttt{zrtpr})\}$ in — compute ZRTP key
$\overline{\text{confirm1}}\langle(\texttt{confirmr}, \mathcal{H}_M((\texttt{H0R}, \texttt{ZRTPKEYR}), \texttt{ZRTPKEYR}), \mathcal{E}_{\texttt{ZRTPKEYR}}(\texttt{H0R}), \mathcal{E}_{\texttt{ZRTPKEYR}}(\texttt{SECR}))\rangle$. — send Confirm1 message
confirm2((CONFIRMIR, HMACSECIR, ENCH0IR, ENCSECIR)). — wait for Confirm2 message
let $\{\texttt{ZRTPKEYIR} = \mathcal{K}(\texttt{S0R}, \texttt{zrtpi})\}$ in — compute initiator's ZRTP key
let $\{\texttt{H0IR} = \mathcal{D}_{\texttt{ZRTPKEYIR}}(\texttt{ENCH0IR})\}$ in — decrypt H0I
let $\{\texttt{SECIR} = \mathcal{D}_{\texttt{ZRTPKEYIR}}(\texttt{ENCSECIR})\}$ in — decrypt initiator's secret block
if $\texttt{H1IR} = \mathcal{H}(\texttt{H0IR})$ then — check H1I
if $\texttt{HMACDHPART2IR} = \mathcal{H}_M((\texttt{PVIR}, \texttt{SECRETSIDIR}, \texttt{H1IR}), \texttt{H0IR})$ then — check Confirm2 HMAC
if $\texttt{HMACSECIR} = \mathcal{H}_M((\texttt{H0IR}, \texttt{SECIR}), \texttt{ZRTPKEYIR})$ then — check integrity of encrypted part
$\overline{\text{conf2ack}}\langle\texttt{conf}\rangle$. — send Conf2ACK message

Figure 5. The "Responder" process

3.3. Protocol model and results

The protocol has been modeled in the following way:

- there is no mismatch in the secrets: the key agreement procedure can rely on this for key generation. This is ideally the typical run of the protocol, when SAS has been verified in the very first session between the agents and the secret cache has been correctly updated in each subsequent session;
- publicly known dummy constants have been used for what does not concern security;
- no negotiations is done during the discovery phase, thus the hash function is predefined and publicly known, as well as encryption algorithms, ZRTP version and so on.

We challenge the adversary to derive the terms that are sent encrypted under the negotiated key in the confirmation messages: if there is no way that an adversary can derive them by applying the rules, then the protocol is safe, as this means that the key agreement procedure has not been compromised and thus the key negotiated between the endpoints is a safe one.

Among the functions that will be used in the protocol there are one-way functions, such as the hashing function $\mathcal{H}$, the function to compute HMACs $\mathcal{H}_M$ and the key

derivation function $\mathcal{K}$: for these functions only a constructor is declared. The lack of the appropriate destructor makes it impossible to recover the argument passed to any of these functions.

The encryption functions are different, as a destructor is declared: when a message is encrypted under the key k via the function $\mathcal{E}$, it can be recovered by using the function $\mathcal{D}$, provided that the correct key k is passed to this function.

Finally the model is equipped with an equational theory that accounts for the commutative property of the exponential:

$$(g^x)^y = (g^y)^x$$

The messages are distinguished one from the other by having a different channel for each message type: channels are declared as free names and they belong to the initial knowledge of the attacker, *i.e.* any data flowing through these channels is knowable by the attacker. Since the beginning the attacker knows also any constant used in the protocol, such as the base of the exponentials or the constant strings. The terms to be sent under encryption are declared as *private* free names: this means that they do not belong to the initial knowledge pool of the attacker, *i.e.* there will be no Horn clause stating that the attacker knows those free names. ProVerif shows the protocol to be secure in a Dolev-Yao network, as the attacker cannot derive these terms: if the key agreement procedure can be performed, then we have the formal proof that an attacker cannot have compromised it and have broken into the session.

4. Conclusions

In the present paper the protocol run has been modeled as two concurrent processes that interact by exchanging messages, synchronizing on every message exchange.

The model does not bother with all the negotiation procedure of the discovery phase, as this is unessential to prove the security of the protocol: according to the Dolev-Yao model, the cryptographic functions are idealized, so every algorithm is just as strong as any other; moreover the chosen algorithms are publicly known, as they are sent in clear in the `Commit` message.

The analysis performed on the protocol has formally proven that ZRTP is a safe key agreement protocol: two endpoints that use it to agree on a key can be sure that their communications are secured against any attack. For this to happen it is crucial that there are some pre-shared secrets: if this is not the case, ProVerif shows that a *Man-in-the-Middle attack* is possible. This is the reason why one needs to use SAS to ensure that this attack has not been performed on the first session between the two agents: in this session a reliable shared secret will be created, and therefore all the subsequent sessions will be secured.

It must be noted that this is true under the assumption that SAS provides an effective way to detect the presence of an attacker. [6]

More in general, the present paper highlights the benefits of using the applied π-calculus and ProVerif to reason about cryptographic protocols: the model of the protocol accounts for all the peculiarities of a typical run of the ZRTP protocol and therefore provides a good support for reasoning about ZRTP, in view of future modifications and improvements.

5. References

[1] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, Jan. 2005.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115, New York, NY, USA, 2001. ACM.

[3] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop*, pages 86–100, 2001.

[4] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society.

[5] B. Blanchet. *Vérification automatique de protocoles cryptographiques : modèle formel et modèle calculatoire.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, 2008.

[6] R. Bresciani. The ZRTP protocol — Analysis on the Diffie-Hellman mode. (TCD-CS-2009-13), June 2009.

[7] D. Dolev and A. C. Yao. On the security of public-key protocols. *IEEE Transaction on Information Theory*, 2(29):198–208, March 1983.

[8] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[9] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166, London, UK, 1996. Springer-Verlag.

[10] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[11] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Secure RTP. March 2009.