A Computer Aided Design of a Secure Registration Protocol

F. Germeau, G. Leduc
Research Unit in Networking, Université de Liège
{germeau,leduc}@montefiore.ulg.ac.be

Abstract
We use the formal language LOTOS to specify a registration protocol between a user and a Trusted Third Party, that requires mutual authentication. We explain how a model-based verification method can be used to verify its robustness to attacks by an intruder. This method is also used to find a simpler protocol that remains secure.

Keywords
Security, Authentication, Registration protocol, Guillou-Quisquater, Trusted third party, Formal verification, LOTOS specification

1 INTRODUCTION

With the development of the Internet and especially with the birth of electronic commerce, the security of communications between computers becomes a crucial point. All these new applications require reliable protocols able to perform secure transactions. The environment of these operations is very hostile because no transmission channel can be considered safe. Formal descriptions and verifications can be used to obtain the assurance that a protocol cannot be threatened by an intruder.

In this paper, we will show that it is possible to make a formal verification of a security protocol. We can certify that an intruder cannot break a protocol with different kinds of attacks. We will also show how the verification process is able to give useful information to correct the protocol if necessary. The verification technique we have developed is based on the LOTOS (Bolognesi *et al.* 1987)(ISO8807 1989) language and the CADP package (Fernandez *et al.* 1996) included in the Eucalyptus toolbox (Garavel 1996).

We use a model-based approach that, until recently, was not felt adequate to tackle the verification of security protocols (Leduc *et al.* 1996)(Lowe 1996).

We will illustrate the method on a registration protocol. The Equicrypt protocol (Lacroix *et al.* 1996) is a conditional access protocol under design in the European ACTS OKAPI project (Guimaraes *et al.* 1996). It allows a user to subscribe to multimedia services such as video on demand. Equicrypt is an open protocol where the user must first register with a Trusted Third Party (TTP) using a challenge-response exchange. After a successful registration, this third party issues a public-key certificate which allows the user to subscribe to a service with different service providers. The subscription part has been studied in (Leduc *et al.* 1996) and some possible attacks have been reported. In this paper, we will focus on the design and verification of the registration protocol which must provide the authentication of the user by the TTP and authentication of the TTP by the user. The protocol is also used to transmit the user's public key to the TTP.

The paper is organized as follow. The section 2 describes the registration protocol that we want to verify and possibly correct. In section 3 we present the formal specification of the protocol written in LOTOS and the section 4 is dedicated to the properties we want to verify. The verification itself is explained in section 5 and concludes this paper.

2 THE REGISTRATION PROTOCOL

2.1 Notation

The protocol involves several cryptographic operations, for which we give an abstract view only. Each scheme uses peer encryption and decryption keys K_E and K_D and functions $E(\_,\_)$ and $D(\_,\_)$ such that $D(K_D, E(K_E, m)) = m$ for any message m. In public key cryptography, the encryption key is the public key and the decryption key is the private key for ciphering operations. For signature operations, the encryption key is the private key and the decryption key is the public key. We also use the more compact notation $\{m\}K_E$ to denote the message m encrypted with the key K_E. That is $\{m\}K_E = E(K_E, m)$.

K_A^P denotes the public key of the user A and K_A^S the private secret key of the user A. The expression $\{m\}K_E$ where K_E is a public key represents the message m encrypted with the key K_E. The same expression where K_E is a private key represents both the message m in clear and a hash of the message m encrypted with the key K_E.

We widely use the concept of nonce (i.e. a number used only once). A nonce is a random number that must be used during only one instance of the protocol. This prevents an intruder from replaying outdated messages and is an abstract model of the pair "time stamps, random number".

All the messages have the following structure:

$Number : Source \rightarrow Destination : Message\ Id\ < Message\ Fields >$

2.2 Principles

The following is a presentation of the Equicrypt system and its registration protocol. The aim of the Equicrypt system is to control the access to multimedia services proposed by service providers. To avoid requiring different access systems for every service provider, a unique decoder uses a public-key cryptography protocol to subscribe to and decode different services. An independent entity known as the Trusted Third Party (TTP) acts as a registering authority trusted by both users and providers. The registration protocol must achieve the mutual authentication of the user and the TTP. The TTP must be sure that the claimed identity of the user is the right one and the user must be sure that it registers with the right TTP. The TTP must also receive the good user's public-key to issue a public-key certificate similar to X.509 certificates (ITU-T X.509 1993). This kind of certificate is the user's public key signed with the TTP's private key.

The authentication of the user by the TTP uses the Guillou-Quisquater zero-knowledge identification scheme (Guillou *et al.* 1988). When the user buys his decoder, he receives secret personal credentials derived from its real-life identity. These credentials will help the user to prove who he is. The goal of the Guillou-Quisquater (GQ) algorithm is to convince the TTP that the user has valid credentials without revealing them. The authentication of the TTP by the user uses a challenge based on a nonce similar to the 3-way authentication protocol (Schneier 1996). When the user receives his credentials, he also receives the TTP's public-key that will allow him to perform the required checks on the messages and to authenticate the TTP.

The transmission of the user's public key is the third purpose of the registration protocol. The TTP must be sure that the received public key is really the user's one. He must make a link between the user's identity and his public key. An improved version of GQ algorithm proposed in Lacroix *et al.* (1996) can be used to check this.

2.3 The Guillou-Quisquater identification scheme

The cryptographic details of the GQ algorithm are beyond the scope of this paper but the principles will be exposed. Basically, the credentials the user receives are mathematically related to its identity. Let the user act as the prover P and the TTP act as the verifier V in the following protocol.

$1 : P \to V : \mathit{Request}\ < ID, K_P^P, T(K_P^P, r) >$
$2 : V \to P : \mathit{Challenge}\ < d >$
$3 : P \to V : \mathit{Response}\ < t(r, d, B) >$

The prover generates a random number r and computes a function T of this number and of his public key. He sends the verifier his identity ID, his public key K_P^P and the result of the function T. As a response, the verifier sends back another random number d. Then the prover computes a function t with the two random numbers r and d and his credentials B and sends it to the verifier. When he receives the response, the verifier can check that the credentials used to compute t correspond to the identity claimed in the first message, thanks to the existing mathematical relationship between ID and B. The user's credentials B must be kept secret so that the only one who could have computed a right function t is the real user. Thus the TTP has obviously received a fresh response from the right user and has authenticated him.

In message 1, the user's public key has also been scrambled (by the function T) with the random number r. When the verifier received message 3, he gains the mathematical ability to check that the public key received in message 1 is also the one used to compute T. Although the public key is transmitted in clear in message 1 and is thus known to an intruder, this intruder cannot forge a fake message 1 with another public key. This is because he does not know the random number r used again in message 3 and so he cannot generate a valid function $T(K_P^P, r)$.

2.4 Abstract model of the Guillou-Quisquater algorithm

In fact, the GQ algorithm can be seen as a general encryption/decryption scheme. This will be very useful for our formal description. We can consider the user's identity together with its public key as a public decryption key and the credentials as a corresponding secret encryption key. Then, the GQ algorithm looks like an authentication scheme based on a nonce and works as follows. The prover sends his decryption key and receives back the random number d from the verifier. The random number d acts as the nonce. Then he encrypts it with his encryption key. The verifier can check that the nonce he sent has a good signature.

This scheme resists to the “man-in-the-middle” attack because the decryption key is mathematically linked to the prover's identity : the identity itself being a part of the decryption key. When this authentication scheme is used with the classical public key cryptography, not the GQ algorithm, the verifier must receive the prover's public key in another way by a secure channel.

The real algorithm also involves the random number r. As said previously, its main purpose is to scramble the user's public key in the function T. If the intruder generates such a fake function in the first message, the credentials computation performed by the verifier when he receives the third message will fail. We will obtain the same result if the intruder changes the user's public key. This behaviour is exactly transposed in our model because both the user's identity and its public key are used to check the credentials. The second purpose of r is to prevent the TTP from guessing B. Our specification does not take these cryptographic attacks into account. Thus we do not need to consider the random number r and we can ignore it in our model. To avoid confusion, we use the special notation $F(B, d)$ to express the encryption of the nonce d with the credentials B. This will help the reader to keep the modelling in mind.

2.5 Protocol description

The complete registration protocol is as follows. The protocol comprises the authentification of the user by the TTP with the GQ algorithm. We have added the authentication of the TTP by the user with a challenge based on a nonce. Finally, we have added a fourth message to carry the registration result and we use the abstraction of the GQ identification scheme depicted above. This first version of the protocol has a flaw. We will see in section 5 that the formal verification has revealed it and has given information to correct the protocol and to produce new versions.

(a) Initial knowledge of the user

- An identity : $UserID$.
- A pair of public/private keys : K_U^P and K_U^S.
- Credentials : B.
- The public parameters of the GQ algorithm.
- The public key of the TTP : K_{TTP}^P.

(b) Initial knowledge of the TTP

- A pair of public/private keys : K_{TTP}^P and K_{TTP}^S.
- The public parameters of the GQ algorithm.

(c) Message exchanges

The user generates a random nonce n and sends the message 1.

$1 : User \rightarrow TTP : Register\ Request\ < UserID, K_U^P, \{n\}K_{TTP}^P >$

When the TTP receives message 1, he decrypts the nonce n and signs it, generates a random number d and sends them to the user. The TTP can handle several registrations at a time. So he maintains an internal table with one entry for each user who has a registration in progress and he records the tuple $< UserID, K_U^P, n, d >$.

$2 : TTP \rightarrow User : Register\ Challenge\ < d, \{n\}K_{TTP}^S >$

When the user receives message 2, he checks the signature. If the signature is correct, he performs the GQ calculation and sends the result to the TTP.

$3 : User \rightarrow TTP : Register\ Response\ < F(B, d) >$

When the TTP receives message 3, he checks the GQ authentication using this message and the data found in his internal table. Then, he sends a response according to the result. The response is signed and includes both the user's identity and the nonce n. The user's entry in the internal table is deleted. If the response is positive, the TTP registers the tuple $< UserID, K_U^P >$.

$4^+ : TTP \rightarrow User : Register\ Ack\ < \{Yes, UserID, n\}K_{TTP}^S >$
$4^- : TTP \rightarrow User : Register\ Ack\ < \{No, UserID, n\}K_{TTP}^S >$

Now that we have presented the registration protocol, we will continue with its specification, its verification.

3 FORMAL SPECIFICATION

The formal specification has been written in LOTOS which is a standardized description language suitable for the description of distributed systems.

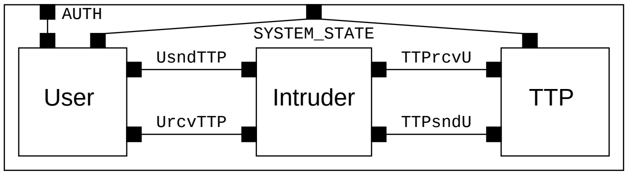

Figure 1 Structure of the LOTOS specification

3.1 Behaviour

The LOTOS specification models both the authentication system and the environment. The authentication system is composed of the user, the TTP and the intruder. Figure 1 shows the general structure of the processes and their interaction points.

The communication channel between the user and the TTP is replaced by the intruder. He intercepts all messages and transmits them or not, with or without modification. We give more details about the intruder in section 3.3. Gates `UsndTTP` and `UrcvTTP` are used by the user for its communication in both directions. The TTP uses the gates `TTPsndU` and `TTPrcvU`.

The environment is responsible for the management of specific events. Firstly, he plays the role of the real user who asks his decoder to register with an interaction at the gate `AUTH`. Secondly, he receives messages that give information about the internal state of the user and about the internal state of the TTP. These messages will help us to perform the formal verification. In this paper, we call them the special events. We have defined six of them received through the gate `SYSTEM_STATE` :

$1 : User \rightarrow Environment : USER\_START\_REG \ \ < UserID >$

This message notifies the environment that the user whose identity is $UserID$ has received the order to register. The user generates this message before sending a valid registration request to the TTP. In our specification, the user and the TTP always behave correctly.

$2 : TTP \rightarrow Environment : TTP\_START\_REG \ \ < UserID >$

With this message, the TTP informs the environment that he has received a valid registration request from the user who claims that his identity is $UserID$.

$3 : TTP \rightarrow Environment : TTP\_REG\_SUCCEEDED \ \ < UserID, K >$

When the TTP sends this message, this means that he has successfully registered the user $UserID$ with the public key K. This message occurs when the TTP owns a valid response to his GQ verification. He will then send a message 4^{+}.

$4 : TTP \rightarrow Environment : TTP\_REG\_FAILED \ \ < UserID, K >$

This message corresponds to the previous one but when the GQ verification has failed. The TTP will send a message 4^{-}.

$5 : User \rightarrow Environment : USER\_REG\_SUCCEEDED \ \ < UserID >$

The user informs the environment he has received a valid successful registration acknowledgement from the TTP.

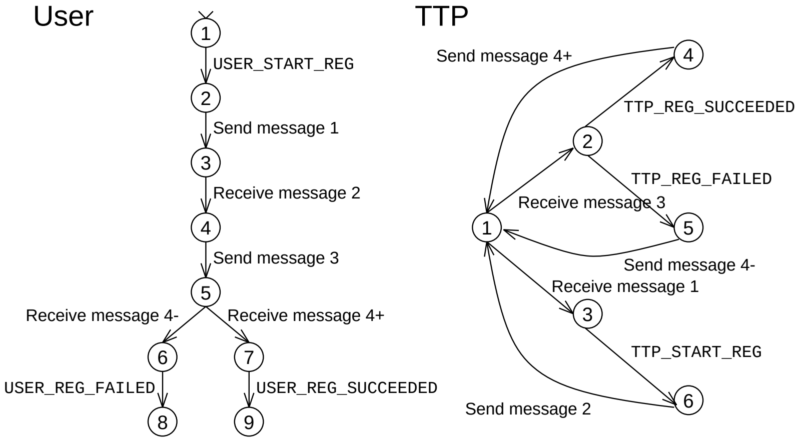

Figure 2 Behaviour of the user and the TTP

$6 : User \rightarrow Environment : USER\_REG\_FAILED \ < UserID >$

The user informs the environment that he has received a valid refused registration acknowledgement from the TTP. That is, the user has received a message 4^- where the TTP's signature is valid but his response is negative.

Finally, the third task of the environment is to receive error messages. The user and the TTP perform several checks when they receive a message. If one of these checks fails, a message indicating the reason of the error is generated. It is very important to understand the difference between the two kinds of interruptions a registration can encounter. The registration can fail because the TTP has decided that the user does not own good credentials. That is what we will call a failure. The other cases are errors. An error occurs when the registration protocol stops due to a badly formed message : wrong signature, wrong nonce, ... We obviously focus on failures because we want to defeat the intruder when he generates good messages. An intruder can always create errors by sending garbage in the transmission channel.

Figure 2 sketches the main behaviours of the user and the TTP. Each transition is labelled with the transmission of a message, the reception of a message or the generation of a special event. Error cases and data manipulation are not shown for simplicity.

3.2 Data types

This specification has been written using data type language extensions, as offered by the APERO tools (Pecheur 1996) included in the Eucalyptus toolbox. The original text has to be processed by the APERO translator to get a valid LOTOS specification. This provides for a smaller and more readable specification.

The abstract data types are composed of :

- Base values : identifiers, keys, credentials described as explicit enumerations.
- Cryptographic functions : Encryption and decryption are modelled as abstract operations that are the reverse of each other. If a decryption is performed with a bad key, the result is not the encrypted message but a special junk value.

```
type EncryptedMessage is Message, PublicKey, PrivateKey
sorts EncryptedMessage
opns
 E (*! constructor *) : PublicKey, Message -> EncryptedMessage
 D : PrivateKey, EncryptedMessage -> Message
eqns
forall msg : Message,
       pubkey : PublicKey
       prvkey : PrivateKey
ofsort Message
  Match(pubkey,prvkey) => D(prvkey,E(pubkey,msg))=msg;
  not(Match(pubkey,prvkey)) => D(prvkey,E(pubkey,msg))=Message_Junk;
endtype
```

- Set of values : They are specially used to model the knowledge of the intruder. For example, to form a message, the intruder will pick a value in each of his sets non determinatically.
- Tables : Needed for storing information about registrations. The TTP can manage several registrations simultaneously so he must store the values received in the messages to make the authentication.

3.3 The Intruder

The intruder replaces the channel between the user and the provider. We want him to mimic any attack a real-world intruder can realize. Thus our intruder must be able to :

- Eavesdrop on and/or intercept any message exchanged among the entities.
- Decrypt parts of messages that are encrypted with his own public key and store them.
- Introduce fake messages in the system. A fake message is an old message replayed or a new one built up from components of old messages including components he was unable to decrypt.

The LOTOS process that models the intruder is always ready to interact at the four gates `UsndTTP`, `TTPsndU`, `UrcvTTP` and `TTPrcvU`. When the user, respectively the TTP, sends a message to the gate `UsndTTP`, respectively `TTPsndU`, the intruder catches the message and tries to decrypt its encrypted parts. Then he stores each part of the message in separate sets of values. These sets constitute the intruder's knowledge base that increases each time a message is received. When the user, respectively the TTP, expects a message on the gate `UrcvTTP`, respectively `TTPrcvU`, the intruder builds a new message with values stored in his sets. With this method, the intruder tries every message it can create.

The intruder is parameterized with some initial knowledge which gives him a certain amount of power. This power includes the capabilities to act as a user with the real TTP and to act as a TTP with the real user. Thus the intruder owns a valid identity, valid credentials and a valid pair of public/private keys. To give the intruder the capability of generating nonces, his initial knowledge also contains nonces that are distinct from those used by the entities. The system we modelled only includes one real user and one real TTP. With his knowledge, the intruder can be seen as a second user and a second TTP. So, our specification incorporates the case where a second valid user tries to cheat and the case where a second valid TTP tries to catch the registration.

The initial knowledge of the intruder is as follows:

- An identity: $IntruderID$.
- The identity of the user: $UserID$.
- A pair of public/private keys: K_I^P et K_I^S.
- Valid credentials: B_I.
- The public parameters of the GQ algorithm.
- The public key of the user K_U^P and the public key of the TTP K_{TTP}^P.
- Nonces.

We assume that our intruder cannot break the public key cryptosystem. That is, he cannot get a message in clear from an encrypted message and he cannot forge a signature without the private key. Note that LOTOS easily provides processes that transgress this rule. Care must be taken to avoid these kinds of unrealistic behaviours. A more detailed description of the intruder can be found in Germeau *et al.* (1997)

3.4 Labelled Transition System

To gain confidence into the specification, it has been simulated with the XSimulator tool from the Eucalyptus toolbox in step-by-step execution mode. This allows us to get a LOTOS specification which is likely to behave correctly without the intruder. Then we have used the CADP package to carry out the verification. The first step consists of using the Caesar tool to generate from the LOTOS specification a graph called Labelled Transition System (LTS). To be able to generate a finite-state LTS of reasonable size, some limitations were required. The exponential growth of states we meet forces us to limit the user to only one registration and the TTP to only two registrations. This has no effect on the generality of our result because the intruder is still able to perform a registration aside the user's one.

The size of the resulting graph greatly depends on the version of the protocol we study. The generated LTS of the protocol presented previously was composed of 487446 states and 2944856 transitions. But the corrected version that will be used in section 5.2 raises to 973684 states and 7578109 transitions. All the computations were performed on a Sun Ultra-2 workstation running Solaris 2.5.1 with 2 CPUs and 832 Mb of RAM. The CPU time required for the generation went up to six hours.

The second step in the process consists of using the Aldebaran tool to minimize the resulting graph. The minimization is always done modulo the strong bisimulation equivalence that preserves all the properties of the graph. This phase is generally carried out in less than fifteen minutes of CPU time. The reduction factor obtained is very important. The minimized LTS of the first protocol is made of 3968 states and 37161 transitions. This clearly shows that our biggest problem is the generation of the brute LTS with the Caesar tool.

As we will see in the next section, all the properties we want to verify are safety properties. Thus the minimization could have been improved modulo the safety equivalence which preserves all the properties expressible in Branching time Safety Logic (Bouajjani *et al.* 1991). This was not mandatory because the graphs were already small enough to make the verification.

4 SAFETY PROPERTIES TO BE VERIFIED

Our goal is to verify that the user always correctly authenticates the TTP, that the TTP always correctly authenticates the user and that the TTP receives the right user's public key. We are going to reach it with the combination of the following safety properties.

- P1: When the TTP successfully registers the user, the user must have started a registration with the TTP before.
- P2: When the TTP successfully registers the user, it must have started a registration with this user before.
- P3: When the TTP refuses to register the user, it must have started a registration with this user before. This refusal is what we called a failure.
- P4: The verdict given by the TTP (i.e. registered or failed) must always be correct and consistent with the acknowledgement received by the user. This property will be further explained below.
- P5: The TTP always registers the user with its real public key.

Each of these properties can be expressed with the special events managed by the environment. For instance, property P1 is translated to "All `TTP_REG_SUCCEEDED` with a particular user identifier must be preceded by a `USER_START_REG` with the same user identifier". This kind of condition can be easily written in the language of our verification tools as a reference graph composed of 3 states and 3 transitions.

If we consider the user whose identity is `USERID_A` and whose public key is `USERPKEY_A`, the graph is as follows:

```
des(0,3,3)
(0, "SYSTEM_STATE !USER_START_REG !USERID_A",1)
(1, "SYSTEM_STATE !TTP_REG_SUCCEEDED !USERID_A !USERPKEY_A", 2)
(2, "SYSTEM_STATE !TTP_REG_SUCCEEDED !USERID_A !USERPKEY_A", 2)
```

This is a small graph that requires a `USER_START_REG` event before any `TTP_REG_SUCCEEDED` event. Property P1 will be verified if the LTS of our system where events other than these two have been turned into internal events is related to this LTS by the safety preorder (Bouajjani *et al.* 1991). Informally,

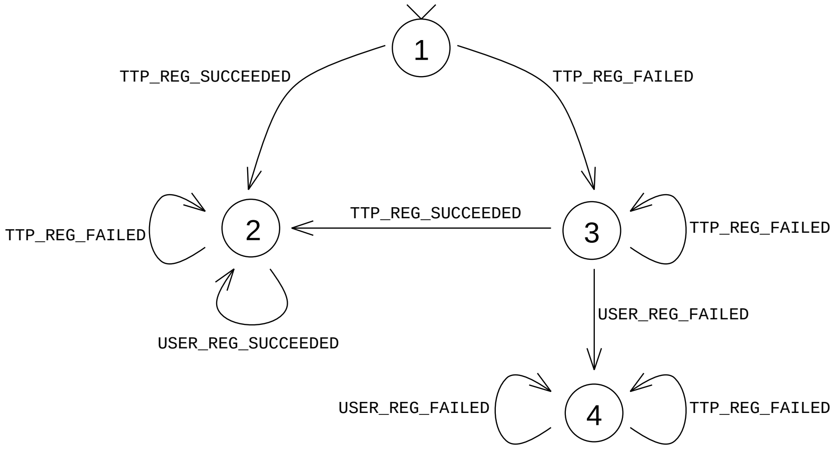

Figure 3 Labelled transition system modelling property P4

the LTS of a system is related to the LTS of a safety property by the safety preorder if and only if the behaviour of the system is allowed by the property. The comparison of two graphs modulo a particular relation is performed by the Aldebaran tool.

Property P4 can be best expressed by the graph shown on figure 3. It shows the temporal orderings that we authorize among the `TTP_REG_SUCCEEDED`, `TTP_REG_FAILED`, `USER_REG_SUCCEEDED` and `USER_REG_FAILED` events. In particular, a `USER_REG_SUCCEEDED` must always be preceded by one `TTP_REG_SUCCEEDED` because, when the user learns that he has successfully registered, the TTP must have successfully registered him. A `USER_REG_FAILED` must always be preceded by at least one `TTP_REG_FAILED` and no `TTP_REG_SUCCEEDED` because, when the user learns that his registration failed, the TTP must have refused to register him at least once and the TTP must not have registered that user successfully. A `USER_REG_FAILED` must never follow a `TTP_REG_SUCCEEDED`.

Properties P1 and P4 achieve the mutual authentication of the user and the TTP. The authentication of the user by the TTP is considered successful only if the TTP registers the user when the user wants to be registered. Thus we need to be sure that the user has started a registration with the TTP when the TTP registers the user. This is provided by property P1. We also need to be sure that the intruder is unable to perform a new registration of the user. Hence, property P4 allows only one successful registration. The authentication of the TTP by the user is considered successful if the user receives the right response from the TTP. This is guaranteed by property P4.

Properties P2 and P3 ensure that the TTP has really started a registration with the user when he gives a verdict. We need this check because the TTP can manage several registrations simultaneously. Finally, property P5 ensures that the user is always registered with its own public key (and not e.g. the intruder's one). To do so, the `TTP_REG_SUCCEEDED` event has two parameters: the user's identity and its public key. We must verify that these two fields always match for every `TTP_REG_SUCCEEDED` event in the LTS of our system.

5 VERIFICATION OF THE PROTOCOL

This section is the core of our study. We will show how the registration protocol can be certified using the Eucalyptus toolbox.

5.1 A flaw

When checking our properties, Aldebaran discovered that property P4 was not satisfied. We use the Exhibitor tool of the CADP package to produce a diagnostic sequence of 19 steps that exhibits one scenario that leads to the undesirable state. This sequence of transitions comprises an event `USER_REG_FAILED` before an event `TTP_REG_SUCCEEDED`. Thus the TTP successfully registers the user after the user has learned that his registration failed. This clearly does not fulfil property P4.

The diagnostic sequence is the following :

```
<initial state>
 1: "AUTH !USERID_A"
 2: "SYSTEM_STATE !USER_START_REG !USERID_A"
 3: "USNDTTP !USERID_A !USERPKEY_A !E (TTPPKEY, NONCE_A)"
 4: "TTPRCVU !USERID_A !USERPKEY_A !E (TTPPKEY, NONCE_A)"
 5: "SYSTEM_STATE !TTP_START_REG !USERID_A"
 6: "TTPSNDU !RANDOM1_TTP !S (TTPSKEY, NONCE_A)"
 7: "TTPRCVU !USERID_A !S (CERT_I, RANDOM1_TTP)"
 8: "SYSTEM_STATE !TTP_REG_FAILED !USERID_A !USERPKEY_A"
 9: "TTPSNDU !S (TTPSKEY, NO, NONCE_A, USERID_A)"
10: "TTPRCVU !USERID_A !USERPKEY_A !E (TTPPKEY, NONCE_A)"
11: "SYSTEM_STATE !TTP_START_REG !USERID_A"
12: "TTPSNDU !RANDOM2_TTP !S (TTPSKEY, NONCE_A)"
13: "URCVTTP !RANDOM2_TTP !S (TTPSKEY, NONCE_A)"
14: "USNDTTP !USERID_A !S (CERT_A, RANDOM2_TTP)"
15: "URCVTTP !S (TTPSKEY, NO, NONCE_A, USERID_A)"
16: "SYSTEM_STATE !USER_REG_FAILED !USERID_A"
17: "TTPRCVU !USERID_A !S (CERT_A, RANDOM2_TTP)"
18: "SYSTEM_STATE !TTP_REG_SUCCEEDED !USERID_A !USERPKEY_A"
<goal state>
19: "TTPSNDU !S (TTPSKEY, YES, NONCE_A, USERID_A)"
```

At line 1, the environment asks for a registration of user A. The user's decoder receives the order and begins the registration with a `USER_START_REG` event. It sends a register request message to the TTP at step 3 (see section 2.5).

$User \rightarrow Intruder : Register\ Request\ < A, K_A^P, \{N_A\}K_{TTP}^P >$

The intruder intercepts the message and replays it without alteration to the TTP at line 4.

$Intruder \rightarrow TTP : Register\ Request\ < A, K_A^P, \{N_A\}K_{TTP}^P >$

When the TTP receives this message, he starts the registration and sends back a message 2 with a random number R_1 at step 6.

$TTP \rightarrow Intruder : Register\ Challenge\ < R_1, \{N_A\}K_{TTP}^S >$

The intruder learns the random number required by the GQ verification when he receives this message. He immediately generates a fake response: that is line 7.

$Intruder \rightarrow TTP$: *Register Response* $< F(B_I, R_1) >$

Obviously, the GQ verification fails because the intruder does not own the user's credentials. The TTP declares a failed authentication and sends a negative response.

$TTP \rightarrow Intruder$: *Register Ack* $< \{No, A, N_A\}K^S_{TTP} >$

At this point, the TTP knows that he has refused the user A's registration but this user is still waiting for a response to his registration request. The intruder goes on with the attack by replaying the register request at line 10. The TTP starts a second registration of the user A and sends back a new challenge with a random number R_2 different from the previous one. The intruder still intercepts the message but this time he forwards it to the user (steps 12 and 13).

$Intruder \rightarrow TTP$: *Register Request* $< A, K^P_A, \{N_A\}K^P_{TTP} >$
$TTP \rightarrow Intruder$: *Register Challenge* $< R_2, \{N_A\}K^S_{TTP} >$
$Intruder \rightarrow User$: *Register Challenge* $< R_2, \{N_A\}K^S_{TTP} >$

The user receives the so long awaited response and answers to it.

$User \rightarrow Intruder$: *Register Response* $< F(B_A, R_2) >$

The intruder immediately replies by replaying the previous negative register acknowledgement message recorded at stage 9.

$Intruder \rightarrow User$: *Register Ack* $< \{No, A, N_A\}K^S_{TTP} >$

This acknowledgement is considered valid by the user though it does not belong to the right registration. The user closes by declaring a failed registration with the event `USER_REG_FAILED` at step 16. Meanwhile, the intruder forwards the user's response to the TTP.

$User \rightarrow Intruder$: *Register Response* $< F(B_A, R_2) >$

This response is valid, so the TTP successfully registers the user and sends a positive response.

$TTP \rightarrow Intruder$: *Register Ack* $< \{Yes, A, N_A\}K^S_{TTP} >$

Both the user and the TTP have finished their exchange but they have not the same view of the registration.

For this attack to succeed, the intruder does not even need valid credentials. It only needs to create a fake response to the first registration to obtain a negative acknowledgement from the TTP. When he owns it, he replays the user's request and inserts the negative response in the exchange at the right place. Hopefully, this attack does not allow the intruder to authenticate himself as the user. So the TTP still authenticates correctly the user. But the authentication of the TTP by the user failed. The intruder can obtain a denial of service by performing this attack systematically.

The strength of our technique is that the analysis of the sequence immediately brings us the reason of the failure. The acknowledgement of the TTP is too general because it can be considered valid in two distinct registrations.

5.2 A corrected version

A way to prevent the attack is to add to the acknowledgement a unique identifier of the registration. The random number used in the GQ verification is the right candidate. This number is meant to be different at each registration. Its integration into the signature of the fourth message will allow the user to check its freshness. Here is the corrected version of our registration protocol:

$1 : User \rightarrow TTP : Register\ Request\ \ < UserID, K_U^P, \{n\}K_{TTP}^P >$
$2 : TTP \rightarrow User : Register\ Challenge\ \ < d, \{n\}K_{TTP}^S >$
$3 : User \rightarrow TTP : Register\ Response\ \ < F(B,d) >$
$4^+ : TTP \rightarrow User : Register\ Ack\ \ < \{Yes, UserID, n, d\}K_{TTP}^S >$
$4^- : TTP \rightarrow User : Register\ Ack\ \ < \{No, UserID, n, d\}K_{TTP}^S >$

Aldebaran states that all our properties are fulfilled with this version. Hence, the mutual authentication and the transmission of the public key succeed despite the attempts of the intruder. We conclude that this is a secure registration protocol provided that the cryptographic computations cannot be broken.

5.3 The simplest protocol

Section 5.2 demonstrates that the signature of the registration acknowledgement message is very important. It can certainly not be removed as it performs the authentication of the whole registration. We have found that the addition of the random number d in the signature of the fourth message makes the nonce n useless. It was used at first for the user to authenticate the TTP but the TTP's signature of the acknowledgement is sufficient to perform this authentication. The authentication of d with a signature in the registration challenge message is not anymore mandatory. These two simplifications lead to a very simple protocol with only one signature:

$1 : User \rightarrow TTP : Register\ Request\ \ < UserID, K_U^P >$
$2 : TTP \rightarrow User : Register\ Challenge\ \ < d >$
$3 : User \rightarrow TTP : Register\ Response\ \ < F(B,d) >$
$4^+ : TTP \rightarrow User : Register\ Ack\ \ < \{Yes, UserID, d\}K_{TTP}^S >$
$4^- : TTP \rightarrow User : Register\ Ack\ \ < \{No, UserID, d\}K_{TTP}^S >$

All the five properties are satisfied. This version is as robust as the previous one from the point of view of the mutual authentication. Obviously, the intruder can more easily disturb the registration. The only difference is that the intruder's actions will be discovered later in the protocol. Formally, there exists a safety preorder between the corrected version of the protocol and this simplified version regarding the six special events only. Hence the former satisfies all safety properties verified by the latter.

6 CONCLUSION

This paper presents a formal description of a security protocol. We have chosen a protocol that achieves the registration of a user to a trusted third party. We have shown how complex cryptographic operations can be abstracted away from mathematical details and specified by abstract data types. Our model of the Guillou-Quisquater algorithm is particularly simple while still capturing the essence of it.

We have shown how intrusions can be taken into account by adding an intruder process. Our model of this intruder is very simple and powerful. He can mimic very easily all reasonable real-world attacks, that is all non cryptographic and non repetitive attacks.

We have shown how to model the security properties, and in particular authentication properties as simple safety properties that can be checked automatically. The verification is based on the safety preorder which should hold between the system and the property.

Finally, we have shown on a concrete protocol how helpful formal description techniques and model-checkers can be to design security protocols. Many subtle attacks were indeed found (such as those provided in this paper) during the design that could probably not have been discovered, at least so early, by a human-being.

The computer aided design aspect of this work has been pushed further in Germeau *et al.* (1997) where we have made an improvement of the protocol. We show how to give the entities the ability to know exactly why a registration does not complete. We want to make a distinction between registration failures due to intruder's actions or due to a genuine user with bad credentials. A new version of the protocol have been designed with the verification tools to meet this additional requirement.

The results of the verification are obviously based on our set of safety properties and on some assumptions on our model. In particular, we do not prove formally the correctness of our abstract finite model with respect a more realistic model composed of more users and more TTPs. To strengthen our verification, it would be interesting to add such a proof, as in Lowe (1996), but our case-study is more complex. Another possible approach, proposed recently in Bolignano (1997), is based on an abstraction function and automates the computation of a correct abstract model. Finally, we do not prove any sort of completeness of our set of safety properties. Methods to automate the definition of security properties would be desirable. Some work in this direction is proposed in Abadi *et al.* (1997).

REFERENCES

Abadi, M. and Gordon, A.D. (1997) *A Calculus for Cryptographic Protocols The Spi Calculus*, Proceedings of the 4th ACM Conference on Computer and Communications Security.

Bolignano, D. (1997) *Towards a Mechanization of Cryptographic Protocol Verification*, Proceedings of CAV 97, LNCS 1254, Springer-Verlag.

Bolognesi, T. and Brinksma E. (1987) *Introduction to the ISO Specification Language LOTOS*, Computer Networks and ISDN Systems 14.

Bouajjani, A. Fernandez, J.C. Graf, S. Rodriguez, C. and Sifakis, J. (1991) *Safety for Branching Time Semantics*, 18th ICALP, Springer-Verlag.

Fernandez, J.C. Garavel, H. Kerbat, A. Mateescu, R. Mounier, L. and Sighireanu, M. (1996) *CAESAR/ALDEBARAN Development Package: A Protocol Validation and Verification Toolbox*, Proceedings of the 8th Conference on Computer-Aided Verification, Alur & Henzinger Eds.

Garabel, H. (1996) *An overview of the Eucalyptus Toolbox*, Proceedings of COST247 workshop.

Germeau, F. and Leduc, G. (1997) *Model-based Design and Verification of Security Protocols using LOTOS*, Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols.

Guillou, L. and Quiquater, J.J. (1988) *A Practical Zero-knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proceedings of Eurocrypt 88, Springer-Verlag.

Guimaraes, J. Boucqueau, J.M. Macq, B. (1996) *OKAPI: a Kernel for Access Control to Multimedia Services based on Trusted Third Parties*, Proceedings of ECMAST 96, pp. 783-798.

ISO (1989) *LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, Information Processing Systems - Open Systems Interconnection: IS 8807.

ITU-T (1993) *The Directory : Authentication Framework*, Information Technology - Open Systems Interconnection: ITU-T Recommendation X.509.

Lacroix, S. Boucqueau, J.M. Quisquater, J.J. and Macq, B. (1996) *Providing Equitable Conditional Access by Use of Trusted Third Parties*, Proceedings of ECMAST 96, pp. 763-782.

Leduc, G. Bonaventure, O. Koerner, E. Léonard, L. Pecheur, C. and Zanetti, D. (1996) *Specification and Verification of a TTP Protocol for the Conditional Access to Services.*, Proceedings of 12th J. Cartier Workshop on Formal Methods and their Applications: Telecommunications, VLSI and Real-Time Computerized Control System, Canada.

Lowe, G. (1996) *Breaking and Fixing the Needham-Schroeder Public-Key Authentication Protocol using FDR*, T. Margaria and B. Steffen Eds., Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1055, Springer-Verlag.

Pecheur, C. (1996) *Improving the Specification of Data Types in LOTOS*, Doctoral dissertation, University of Liège.

Schneier, B. (1996) *Applied Cryptography*, Second Edition, J. Wiley & Sons.

7 BIOGRAPHY

François Germeau has joined the Research Unit in Networking in 1996 and is studying the conception and verification of security protocols with formal description techniques.

Guy Leduc is professor at the University of Liège, his main research field is on formal languages and methods applicable to the software engineering of computer networks and distributed systems.

This work has been partially supported by the Commission of the European Union (DG XIII) under the ACTS AC051 project OKAPI: "Open Kernel for Access to Protected Interoperable Interactive Services".