Model-based Design and Verification of Security Protocols using LOTOS

F. Germeau, G. Leduc[1]

Research Unit in Networking (RUN)
Institut d'électricité Montefiore B.28, Université de Liège, B-4000 Liège, Belgium
{germeau,leduc}@montefiore.ulg.ac.be

Abstract

We explain how the formal language LOTOS can be used to specify security protocols and cryptgraphic operations. We describe how to model security properties as safety properties and how a model-based verification method can be used to verify the robustness of a protocol against attacks of an intruder. We illustrate our technique on a concrete registration protocol. We find a simpler protocol that remains secure, and a more sophisticated protocol that allows a better distinction between intruder's attacks and ordinary errors.

1 Introduction

Formal description techniques gain increased consideration due to significant advances and results recently obtained. A lot of computer systems achieve mission-critical tasks and thus require an absolute proof that they are working without any errors. Such a proof can be deduced with formal verifications. The ever growing power of computers and the increasing knowledge of verification techniques allow one to perform validations on real problems. With the development of the Internet and specially with the birth of the 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.

Special modal logics have been designed to verify authentication protocols. The most well-known such logic is the BAN logic [BAN90], but some others have been proposed to overcome some of its limitations. Such logics have been used successfully to verify several protocols, but have not proved very effective in some other circumstances. Another approach consists of using general purpose formal methods usually applied to more conventional protocols. They are supported by verification tools, such as theorem provers or model-checkers, which makes it possible to automate the proof process. Approaches based on theorem proving applicable to a large class of protocols and to general authentication properties have been proposed [Mea92][Bol96].

Until recently among the different formal methods, the model-checking approach was not felt adequate to tackle the verification of security protocols. Recent results prove the contrary, this approach can in fact be very efficient to achieve a real computer aided design of security protocols. To our knowledge, its first application to the verification of security protocols was achieved in [Low96] where the Needham-Schroeder protocol [Sch96] was specified in CSP [Hoa85] and model-checked by the FDR tool. Independently of this work, we specified the Equicrypt protocol

[1]Maître de recherches (Senior Research Associate) F.N.R.S. (National Fund for Scientific Research, Belgium)

[LBQ96] in LOTOS and used the Eucalyptus toolbox [Gar96] to verify it [LBK96][GL97]. The present paper will focus on the method we used to model and verify a security protocol using LOTOS.

The paper is organized as follow. In section 2, we will show that the LOTOS language is a very good performer to handle the specification of security protocols. With its flexibility, a wide range of cryptographic operations can be modelled. We will describe the establishment of security properties and the associated verification process in section 3. The verification is quite automatic and allows one to certify that an intruder cannot break a cryptographic protocol with different kinds of attacks. An application of our method on a concrete protocol will be presented in section 4. We will also point out that it is possible to tune a protocol in order to obtain new properties and improve its behaviour.

2 LOTOS specification

The formal specification of a security protocol is written in LOTOS [BB87][ISO89] which is a standardized language suitable for the description of distributed systems. It is made up of two components:

- A process algebra, moslty inspired by CCS [Mil89] and CSP [Hoa85], with a structured operational semantics. It describes the behaviour of processes and their interactions. LOTOS has a rich set of operators (multiway synchronization and abstraction like in CSP, disabling, ...), and an explicit internal action like in CCS.
- An abstract datatype language. ACT ONE [EM85], with an initial semantics. A type is defined by its signature (sorts + operation on the sorts) and by equations to give a meaning to the operations.

The revision of the LOTOS standard is under study in ISO/IEC since 1991. The design of this enhanced version called E-LOTOS is based on the feedback obtained from practical applications of LOTOS and will certainly improve its expressive power.

A LOTOS specification is composed of two differents parts. The first one is dedicated to the description of the abstract data types and the cryptographic operations in particular. The second part describes the behaviour of the different entities involved in the protocol. We will firstly deal with this description

2.1 Behaviour

Every security protocol involves several entities called principals. A principal can be any object that plays a role in the evolution of the protocol. Example of principals are users, hosts or processes. When we address the verification of the security of the protocol, we must make some assumptions on the behaviour of the principals. Thus principals are qualified trusted or not. A trusted principal will always react according to the expected behaviour. A non-trusted principal can try and break the protocol with an unexpected behaviour though is considered genuine by the other entities.

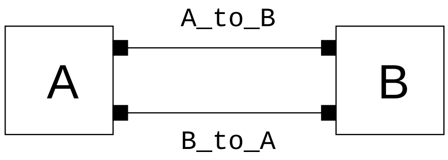

Figure 1 : Principals without intruder

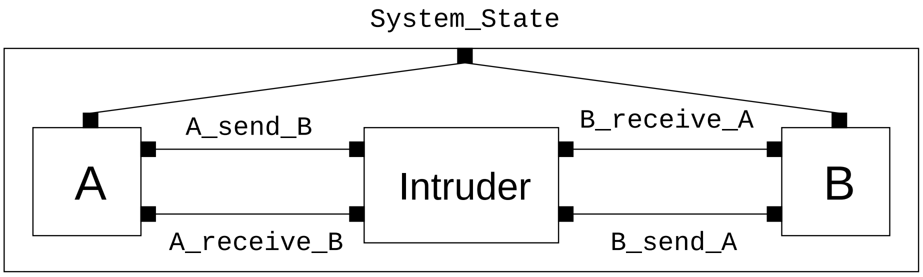

Figure 2 : Principals with intruder and environment

Principals are linked together with communication channels to exhange messages. These communication channels are generally considered insecure, that is an intruder can act passively or actively on the transferred information. He can eavesdrop on messages, intercept them, replay old ones, or create new ones. The goal followed by the intruder ranges from a simple denial of service to the access to prohibited rights.

The behaviour section of a LOTOS specification is composed of several processes which interact with each other through interaction points called gates. Each principal involved in the protocol is modelled by a process that describes its exact behaviour. LOTOS allows the synchronisation of two or more processes via interactions that can occur at each gate. A one way communication channel between two principals is modelled by the synchronisation of the transmission gate of one principal with the reception gate of the other principal. A second synchronisation handles the other way of the communication channel.

For instance, figure 1 depicts a system with two principals where the communication channel is modelled by the synchronisation of the gate `A_to_B` of principal A with the gate `A_to_B` of principal B and with the synchronisation of the gate `B_to_A` of principal A with the according gate of principal B.

To introduce the intruder that will try to threat the protocol we replace the simple communication channels by one central process that will act as the intruder. Thus the intruder can intercept all messages and transmit them or not, with or without modification. We will enter into the details of the intruder's behaviour in section 2.3. Back to our example, the principals are not interacting directly with each others but indirectly through the intruder process (figure 2). The intruder is the only principal considered untrusted. All other principals are trusted. We model cases where a principal is not trusted by giving enough power to the intruder to act as a genuine principal.

Finally, we use an environment to monitor the progress of the protocol. When a principal reaches a sensitive point, he informs the environment by sending him a message through the

`System_State` gate. These messages are called special events and will be developed further in section 3.2. They will be of a great help to perform the formal verification. The environment is also responsible for the reception of error messages. Figure 2 presents the complete structure of a typical LOTOS specification that models a security protocol between two principals.

Each process that represents a principal is parameterized with an initial knowledge. This knowledge includes identifiers, keys or whatever information a principal must know before runs of the prococol can occur. As we will see later, the knowledge is the core of the intruder's modelling.

2.2 Abstract data types

2.2.1 Principles

The specification of the behaviour only describes the exchange of messages. It does not consider the data transferred by these messages. Abstract data types define the elements that are handled by the behavioural part. They define which kind of data are used by the protocol but also which operations are allowed on these data. Only the defined operations are permitted. With this restriction, complex cryptographic operations can be abstracted away from mathematical details. We will see that only a simple description of their characteristics is needed.

With LOTOS, abstract data types are written in ACT ONE. Each LOTOS variable can only have values of a particular sort defined during the declaration. A LOTOS type is a module composed of one or several sorts, operations and equations. A sort is the name given to a set of values that belong to the same domain. Specific operations are defined on the values of each sort and the semantics of these operations is provided by specific equations. This structure allows for a great flexibility in the handling of data in LOTOS.

A lot of mechanisms exist in modern cryptography [Sch96], but only a few of them are actually used in security protocols. We do not intend to make an exhaustive translation of cryptographic operations in ACT ONE. We just want to show the level of abstraction provided by LOTOS and the relative simplicity in the definition. Thus we will focus on two examples that represent the most widely used operations : encryption and signature in public-key cryptography. More subtle and complex cryptographic operations can be modelled. In section 4 we present a registration protocol we have specified that uses a zero-knowledge identification scheme.

ACT ONE is not only used to define the data transferred in messages, it is also used to define the internal database of information of each principal. For instance, a registration principal need to manage a registration database that will also be defined in ACT ONE as a table of records with multiple fields. This application is quite common and will not be developed further in this paper.

Definition of abstract data types can rapidly become very cumbersome to design. Thus our specification are written using data type language extensions, as offered by the APERO tool [Pec96] included in the Eucalyptus toolbox. The original text has to be preprocessed by the APERO translator to get a valid LOTOS specification. This provides for a smaller and more readable specification and for some level of immunity w.r.t. underlying processing tools. However, some types were written from scratch, hence, it was necessary to take tools restrictions explicilty into account. The other parts of the toolset will be explained in section 3.3.

2.2.2 Public-key encryption and signature

The following ACT ONE definition models the public-key encryption operation. It does not rely on any particular implementation (e.g. RSA) nor on any particular mathematical concept. For simplicity, we assume the previous definition of the public and the private keys as base values and the existence of an operation `match(public key, private key)` that returns true if the public key corresponds to the private key.

```
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
```

The encryption function `E` and the decryption function `D` are defined as abstract operations that are the reverse of each other. Decryption with a bad key is handled explicitly and produces a distinguished value `Message_Junk` without any meaning. Once encrypted, the only way to access the message is through the decryption function called with the right private key. Obviously, no operation allows to read the private key.

The signature operation is defined in the same way with a verification function `V` that returns true if the signature is correct (i.e. the verification is performed with the right public key). We consider that a signed message is the message in clear and an encrypted hash of it. Thus our model needs the `GetMessage` operation to access the message without any key.

```
type SignedMessage is Message, PublicKey, PrivateKey
sorts SignedMessage
opns
 S (*! constructor *) : PrivateKey, Message -> SignedMessage
 V : PublicKey, SignedMessage -> Message
 GetMessage : SignedMessage -> Message
eqns
forall msg : Message,
       pubkey : PublicKey
       prvkey : PrivateKey
ofsort Message
  V(pubkey,S(prvkey,msg)) = Match(pubkey,prvkey);
  GetMessage(S(prvkey,msg)) = msg;
endtype
```

We assume with these definitions that no one can break the public key cryptosystem by getting the message in clear from the encryted message and the public key, or forging a signed message from the message in clear and the public key. Note that LOTOS easily provides processes that transgress this rule, and thus break any cryptosystem. Great care must be taken to avoid this kind of unrealistic behaviours.

2.3 The intruder

2.3.1 Model

We want to model an intruder as a process that can mimic any attack a real-world intruder can perform. Thus our intruder process shall 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 the intruder was unable to decrypt.

The intruder merely replaces communication channels linking principals involved in the protocol. He behaves in such a way that neither the receiver of a fake message, nor the sender of an intercepted message can notice the intrusion.

The LOTOS process that models the intruder manages a knowledge base. Each time the intruder catches a message, he 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. The intruder tries to collect as much information as he can with the intercepted messages. His behaviour is simple and repetitive. He does not deduce anything from his knowledge base. He just stores information for future use.

When one of the trusted principals is ready to receive a message, the intruder analyze his knowledge base to determine the messages he can create. He builds them with values stored in his sets. As he tries every combination of these values, the intruder tries to send every message he can create with his knowledge.

The intruder is parameterized with some initial knowledge which gives him a certain amount of power. Remember that all principals except the intruder are considered trusted. Thus as we want to cover cases where regular principals are untrusted, the intruder must be able to act as these principals. So his initial knowledge must comprise enough information to allow this behaviour. For instance, in a protocol where a user must register with a trusted authority. The intruder must be able to act as a valid user from the point of view of the trusted autority. But he must also be able to act as a valid trusted autority from the point of view of the user. This example will be explained in more details with the example of section 4.

The key point is the power given to the intruder. Security protocols are based on some assumptions provided by the mathematical background of cryptographic operations. As we want to be realistic, our intruder will not be powerful enough to break a cryptosystem. As LOTOS provides processes that trangress this rule, it would be easy to define an intruder that tries a brute force attack to guess a private key or a random number. The intruder's behaviour is thus deliberately limited in this respect.

2.3.2 Specification of the intruder

The following LOTOS code describes a 3-way exchange between two principals. Its purpose is to show the intruder's interactions with trusted principals, and thus data types are simplified.

Principal `A` interacts through gates `A_Send_B` and `A_Receive_B` and principal `B` uses gates `B_Send_A` and `B_Receive_A`. The intruder is synchronized with each gate. His behaviour is a loop composed of six possible actions: three actions to receive the three messages and three actions to send them. Each time a message is received, it is inserted in the intruder's knowledge. For clarity of this example, the `Insert` function hides all the analysis of the message. The `choice` operator commands the generation of all the possible distinct actions where the message sent is in the intruder's knowledge. The structure of the intruder is quite simple and thus can be guaranteed error free.

```
behaviour

Principal_A[A_Send_B,A_Receive_B](Initial_Knowledge_of_A)
|[A_Send_B,A_Receive_B]|
Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Initial_Knowledge_of_I)
|[B_Send_A,B_Receive_A]|
Principal_B[B_Send_A,B_Receive_A](Initial_Knowledge_of_B)

where

process Principal_A[A_Send_B,A_Receive_B](Knowledge_of_A :Knowledge) : noexit :=
A_Send_B!Message_1;
A_Receive_B?Message_2 :Type_2;
A_Send_B!Message_3;
stop
endproc

process Principal_B[A_Send_B,A_Receive_B](Knowledge_of_B :Knowledge) : noexit :=
B_Receive_A?Message_1 :Type_1;
B_Send_A!Message_2;
B_Receive_A?Message_3 :Type_3;
stop
endproc

process Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A]
                (Knowledge_of_I :Knowledge) : noexit :=
(A_Send_B?Message_1 : Type_1;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Insert(Message_1,Knowledge_of_I))
)
[]
(B_Send_A?Message_2 : Type_2;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Insert(Message_2,Knowledge_of_I))
)
[]
(A_Send_B?Message_3 : Type_3;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Insert(Message_3,Knowledge_of_I))
)
[]
(choice Message_1 : Type_1 [] [Message_1 is_in Knowledge_of_I] ->
 B_Receive_A!Message_1;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Knowledge_of_I)
)
[]
(choice Message_2 : Type_2 [] [Message_2 is_in Knowledge_of_I] ->
 A_Receive_B!Message_2;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Knowledge_of_I)
)
```

```
[]
(choice Message_3 : Type_3 [] [Message_3 is_in Knowledge_of_I] ->
 B_Receive_A!Message_3;
 Intruder[A_Send_B,A_Receive_B,B_Send_A,B_Receive_A](Knowledge_of_I)
)
endproc
```

2.4 Finite model

The LOTOS specification will be translated into a labelled transition system (a graph) where the nodes are the state of the LOTOS specification and the transitions are labelled by the LOTOS actions. This labelled transition system (LTS) must comprise all the possible executions of the protocol. But this graph must be kept finite to be generated.

Some elements like random numbers or time stamps are specific to one run of the protocol leading to an infinite number of possible messages. This infinity must be controlled by giving some well chosen properties to these specific elements. Trusted principals will use one specific element for each run they perform, so we give them a limited set of elements that will be used during their executions. We also give the intruder one element but which is different from those of the trusted principals. When the intruder will use his element in a particular message, this will, in fact, model all the possible messages where the specific elements is not the one expected by a trusted principal.

Messages with a specific element can be split into messages with a correct specific element and messages with a wrong specific element. The first ones are limited in number but not the second ones. With our abstraction we can keep our model finite because a simple message is enough to represent all the incorrect ones.

In some protocols, we may need to consider an infinity or a big number of possible principals. As the LTS must be kept finite and also of reasonable size to be managed, the number of trusted principals must sometimes be drastically limited. The intruder's knowledge allows him to act as any other principal in a trusted or untrusted way, but it may not be sufficient to cover all cases.

There exist other ways to prevent the exponential growth of states. An infinite loop in the behaviour of a principal can be replaced by a limited number of runs adjusted to still cover all the possible intruder's attacks. Multiple configurations with multiple specifications can be planned to address several independant aspects of the protocol.

Great care must be taken with the restrictions imposed by the assumptions we are forced to make. We do not prove formally the correctness of our abstract finite model with respect to these assumptions. It would be interesting to consider such proof on the case studies we derive from this method. Some work in this direction is proposed in [Low96] where the verification with a limited number of principals is generalized or in [Bol97] where an abstraction function automates the computation of a correct abstract model. The main difficulty comes from the complexity of the protocols we want to verify.

Now that we have presented the complete specification, we will enter deeply into the verification process.

3 Validation process

3.1 Properties to verify

Most security properties rely on the fact that the intruder does not know some secret information or is not able to construct the expected message. They can be characterize as safety properties. Informally, safety properties are properties stating "nothing bad will happen". Authentication, access control, confidentiality, integrity and non-repudiation are safety properties. Each of these security services require that a particular situation cannot occur.

The only liveness property is the non-denial of service, which current cryptographic protocols do not guarantee. Intuitively, liveness properties are properties stating "something good will happen". A denial of service happens if an intruder succeeds to get a protocol stuck or make it fail. Thus when a denial of service arises, the liveness property stating that the protocol will succeed is not satisfied.

In order to provide these security services, protocols implement particular mechanisms. The LOTOS specification of trusted principals apply them while the intruder process tries to defeat them. A way to verify the robustness against intruder's attacks during the execution of the specification is needed. Thus a formal translation of the properties to be achieved by security services is required in order to perform the verification.

3.2 Formalizing the properties

During message exchanges of security protocols, critical points are reached where certain security services are assured. The reception of a well-formed message can trigger a principal into a state where he trusts some facts. This behaviour needs to be formalized. We must translate the human idea that the required security service is satisfied into a precise definition of principals state.

In order to determine these critical points in the specification, we introduce some special events. Each time a critical point is reached by a trusted principal, he informs the environment by sending a specific message that gives information about its internal state. The environment of the LOTOS specification is responsible for the reception of these messages. By executing a special event, a principal declares that he is confident in a fact.

If we consider an authentication protocol between two principals. The prover must be authenticated by the verifier. There are two critical points in this protocol. The first one is when the prover starts his authentication and the second one is when the verifier is sure of the prover's identity. Thus we introduce two special events `PROVER_START_AUTHENTICATION` and `VERIFIER_AUTHENTICATION_SUCCESSFUL`. A common property required is that "the prover must have started an authentication with the verifier before the verifier successfully authenticates the prover". Otherwise, an intruder has been able to be authenticated with the prover's identity. This property will be captured by our special events regardless of the particular authentication mechanisms used. We just state that "No `VERIFIER_AUTHENTICATION_SUCCESSFUL` event must occur before a `PROVER_START_AUTHENTICATION`".

This technique can be applied to a wide range of security properties. Some refinements are sometimes needed to define more precisely critical points. A special event can be expressed with parameters that determine the context where the associated critical point occurs. A parameter

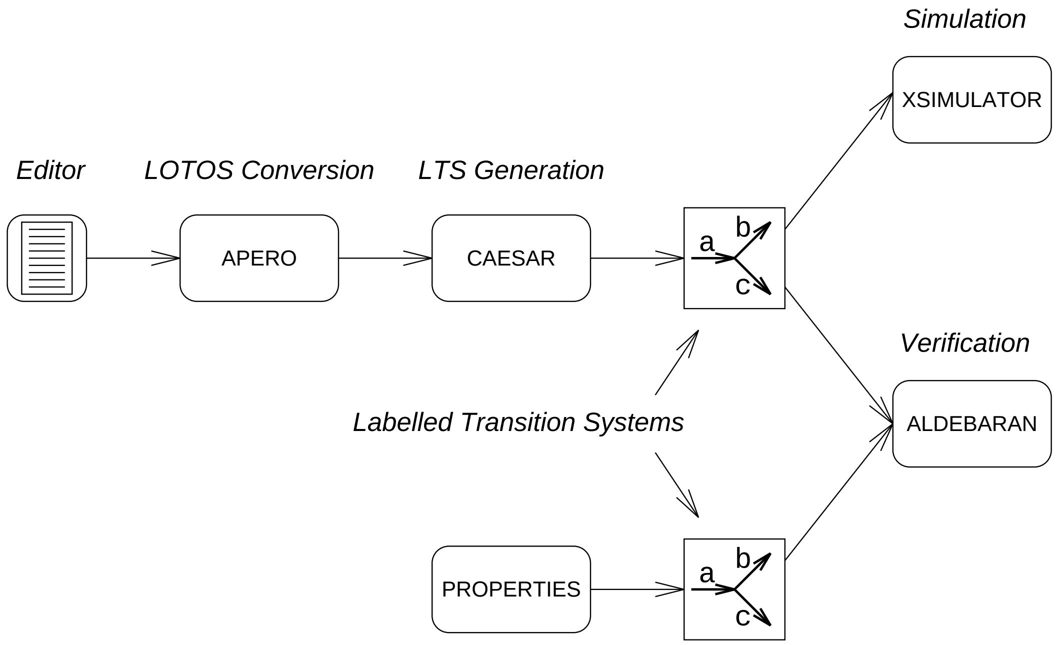

Figure 3 : The Eucalyptus toolbox

can be a principal's identity, an authentication token, a particular key or any other relevant data.

This method allows one to abstract away from all the details of security mechanisms. We can only focus on the security services achieved. One of the major difficulties is to gain the assurance that the critical points are well defined and the security properties are translated correctly into properties on special events. So we need to find the right abstraction level between the simplicity of the global view of the security services and the complexity of the underlying protocols. Method to automate the process would be desirable. Some researchs in this direction can be found in [AG97].

3.3 The verification toolbox

When the LOTOS specification is written and the properties are formalized, we can perform the verification itself. We use the CADP package [FGK96] included in the Eucalyptus toolbox to carry out the verification of the protocol. As figure 3 shows, the LOTOS specification with datatype language extensions is converted into ISO LOTOS with the APERO tool. The next step consists of applying the Caesar tool to generate a graph called Labelled Transition System (LTS) from the LOTOS specification. This graph contains all the possible execution sequences of the protocol studied. Section 2.4 has addressed the feasibility of the generation. To gain confidence into the specification, it is simulated with the XSimulator in step-by-step execution mode.

The Aldebaran tool is the last stage of the processing. It performs the comparison of two labelled transition systems. The verification requires the comparison of the LTS of the protocol as created by the Caesar tool with the graphs of our properties. Thus a final step in the formalization is needed. The properties based on special events must appear like a finite-state graph. The

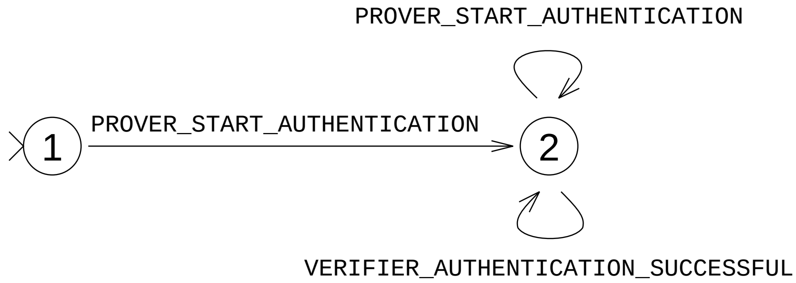

Figure 4 : LTS of the authentication property

process can be automated using the Caesar tool : each property is modelled as a reference LTS generated from a simple LOTOS process containing special events only.

The property discussed in section 3.2 can be specified in LOTOS as follows. The corresponding LTS generated by the Caesar tool is shown in figure 4.

```
behaviour

  System_State! PROVER_START_AUTHENTICATION;Property[System_State]

where

process Property[System_State] : noexit
:=
System_State! PROVER_START_AUTHENTICATION;Property[System_State]
[]
System_State! VERIFIER_AUTHENTICATION_SUCCESSFUL;Property[System_State]
endproc
```

3.4 The verification

Before any comparison between LTS, they must be minimized to speed up the computations. The Aldebaran tool can minimize a LTS modulo a particular equivalence. The first minimization is always done modulo the strong bisimulation equivalence, which preserves all the properties of the graph.

Consider a $LTS = \langle S, A, T, s_0 \rangle$ where S is the set of states, A the alphabet of actions (with i denoting the internal action), T the set of transitions and s_0 the initial state.

A relation $R \subseteq S \times S$ is a strong bisimulation iff :

If $\langle P, Q \rangle \in R$ then, $\forall\ a \in A$,
whenever $P \xrightarrow{a} P'$ then $\exists\ Q' :\ Q \xrightarrow{a} Q'$ and $\langle P', Q' \rangle \in R$;
whenever $Q \xrightarrow{a} Q'$ then $\exists\ P' :\ P \xrightarrow{a} P'$ and $\langle P', Q' \rangle \in R$

Two LTS $Sys_1 = \langle S_1, A, T_1, s_{01} \rangle$ and $Sys_2 = \langle S_2, A, T_2, s_{02} \rangle$ are related modulo the strong bisimulation denoted $Sys_1 \sim Sys_2$, iff there exists a strong bisimulation relation $R \subseteq S_1 \times S_2$ such that $\langle s_{01}, s_{02} \rangle \in R$.

Our security properties being all safety properties, the minimization can be further improved modulo the safety equivalence [BFG91], which preserves all the properties expressible in Branching time Safety Logic (BSL).

Not all the observable actions are relevant to verify the properties. In particular, our properties only rely on special events, so that other actions can be hidden. The minimized LTS of our protocol can be checked against the LTS of a property be verifying the safety preorder relation [BFG91] between them. Formally, the safety preorder ($\leq_s$) is the preorder that generates the safety equivalence ($\sim_s$), and is nothing else than the weak simulation preorder.

Consider again a $LTS = <S, A, T, s_0>$ and let's define $L = A - \{i\}$, a relation $R \subseteq S \times S$ is a weak simulation iff:

If $<P, Q> \in R$ then, $\forall\ a \in L$,
if $P \overset{i^*a}{\Rightarrow} P'$, then $\exists\ Q' :\ Q \overset{i^*a}{\Rightarrow} Q'$ and $<P', Q'> \in R$

A LTS $Sys_1 = <S_1, A, T_1, s_{01}>$ can be simulated by $Sys_2 = <S_2, A, T_2, s_{02}>$, denoted $Sys_1 \leq_s Sys_2$, iff there exists a weak simulation relation $R \subseteq S_1 \times S_2$ such that $<s_{01}, s_{02}> \in R$. Two LTS Sys_1 and Sys_2 are safety equivalent iff $Sys_1 \leq_s Sys_2$ and $Sys_2 \leq_s Sys_1$.

Informally, "behaviour $\leq_s$ property" means that the behaviour (exhibited by the protocol) is allowed (i.e. can be simulated) by the (safety) property.

When a property is not verified, meaning that Aldebaran has not found a safety preorder between the LTS of protocol and the LTS of the property, it produces a diagnostic sequence of actions. However, this sequence is usually of little help as such, because it only refers to the few non hidden actions that were kept for their relevance to express the properties. We call it the abtract diagnostic sequence.

To circumvent this difficulty and get a detailed sequence with all actions visible, we have to encode this abstract diagnostic sequence in a format suitable for input to the Exhibitor tool. This tool is then instructed to find the shortest detailed sequence allowed by the specification and matching the abstract one. We are then able to clearly identify the scenario that leads to the undesirable state where the property is not verified. Intruder's attacks can then be pointed out very easily. The complete diagnostic sequence shows the order of actions performed by the intruder and how he was able to acquire enough knowledge to succeed.

The verification process of the properties is then complete. If one or more of them are not satisfied, our method gives diagnostics of a greatful help to redesign the protocol.

4 An example of verification

To illustrate our method, this section presents an example of verification. We have chosen the registration part of the Equicrypt protocol, a conditional access protocol under design in the European ACTS OKAPI project [GBM96]. It allows a user to subscribe to multimedia services such as video on demand. The user must first register with a Trusted Third Party (TTP) using a challenge-response exchange. After a successful registration, this TTP issues a public-key certificate which allows the user to subscribe to a service offered by a service provider.

We concentrate on the verification of the registration protocol. This paper only presents an overview of the process. Complete details can be found in [GL97] for more interested readers.

4.1 The registration protocol

The registration protocol involves a user who wants to access a multimedia service and a TTP that acts as a notary. The mutual authentication of the user and the TTP must be achieved by the protocol. The TTP must be sure that the claimed identity of the user is the right one and the user must be sure that he registers with the right TTP. The TTP must also receive the good user's public-key during the procotol to issue a corresponding public-key certificate needed for the subcription phase.

The authentication of the user by the TTP uses the Guillou-Quisquater (GQ) zero-knowledge identification scheme [GQ88]. Before registering, the user has received secret personal credentials derived from its real-life identity. These credentials will help him to prove who he is to the TTP but without revealing them. The authentication of the TTP by the user uses a challenge based on a nonce (i.e. a number used once). The user has also received the TTP's public-key to perform the required checks on the messages and to authenticate the TTP. The transmission of the user's public-key to the TTP is possible with an improved version of the GQ algorithm [LBQ96]. The registration protocol presented in this paper is, in fact, an enhanced version of the original one found in [LBQ96].

The GQ identification scheme is based on complex mathematical relations derived from the user's identity, the user's public-key and the secret credentials. It uses a random number issued by the TTP to challenge the user and a second random number issued by the user to scramble the public-key and protect the credentials. To specify the algorithm, we have designed an abstract model which is particulary simple while still capturing the essence of it. The key point of the authentication are the secret credentials. If we consider them as a secret encryption key and the user's identity together with its public key as a corresponding public decryption key, the GQ algorithm looks like an authentication scheme based on a nonce and works as follows. The user sends his public decryption key to the TTP and receives back a nonce as a challenge. Then he returns to the TTP the nonce encrypted by his encryption key. The TTP can then check that the nonce has been encrypted as expected. This scheme resists to the "man-in-the-middle" attack because the decryption key is mathematically linked to the user's identity.

In the remaining of this paper, we will present all the messages with the following structure:

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

A couple (K_A^S, K_A^P) will denote the pair of private/public keys of the principal A. Encryption of $data$ will be written $\{data\}K_A^P$ while signature will be written $\{data\}K_A^S$. $F(B,d)$ will represent the special encryption of the GQ model where B are the credentials.

The protocol works as follows:

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 $< UserID, K_U^P >$

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

4.2 Procotol specification

Using the framework presented in previous sections, we have specified the protocol in LOTOS. Abstract data types were designed for all the cryptographic operations involved including the abstract model of the GQ algorithm. The user and the TTP are two trusted principals and the intruder is the untrusted one. The user always tries to perform a valid registration. The intruder's initial knowledge is adjusted to allow him to act as a second untrusted user and simultaneously as a second untrusted TTP. It includes :

- An identity : $IntruderID$
- Valid credentials : B_I
- A pair of private/public keys : K_I^S et K_I^P
- The public key of the user K_U^P and the public key of the TTP K_{TTP}^P
- The identity of the user : $UserID$
- Nonces and random numbers different from those of trusted principals.

After the step-by-step simulation stage, the labelled transition system (LTS) of the protocol has been generated. It is composed of 487446 states and 2944856 transitions and has required one hour of computation on a SUN Ultra-2 workstation running Solaris 2.5.1 with 2 CPUs and 832 Mb of RAM. The reduction factor of the minimization modulo the strong bisimulation was very important. The minimized LTS of the protocol is made of 3968 states and 37161 transitions. The reduction modulo the safety equivalence was not mandatory because the graph was small enough to carry out the verification.

4.3 Formalizing the properties

Among the five safety properties we have verified, we only present one of a particular interest. This property is necessary (but not sufficient) to the authentication of the TTP by the user and we will see later that the current protocol does not satisfy it.

- P4 : The verdict of the registration given by the TTP (i.e. registered or failed) must always be correct and consistent with the acknowledgement received by the user.

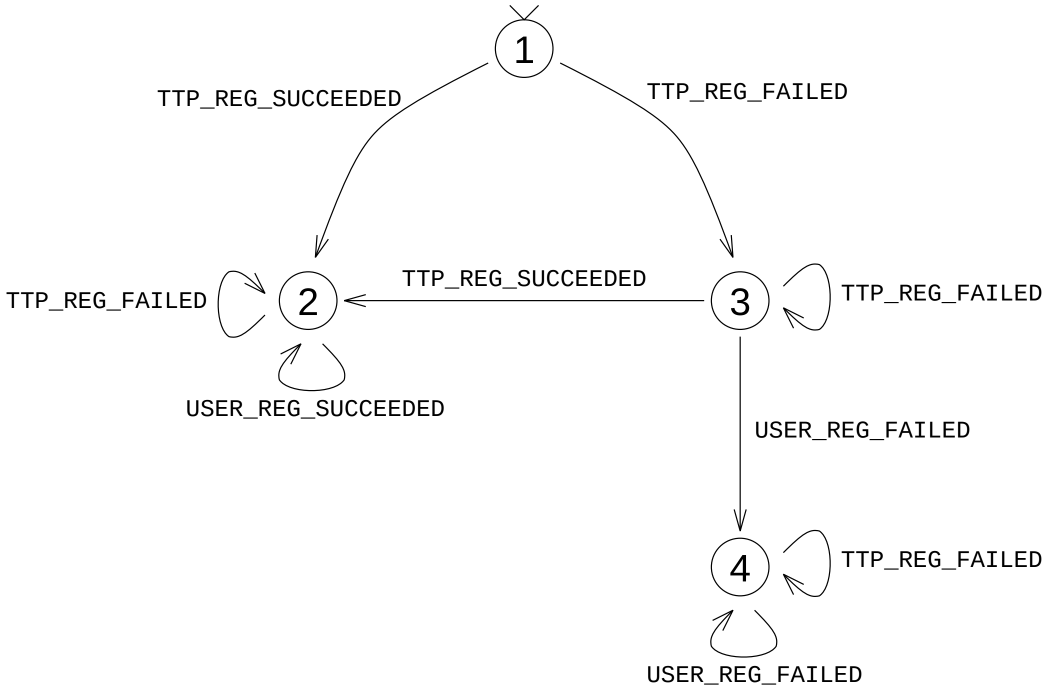

Figure 5 : Labelled transition system modelling property P4

Four special events are required to formalize this property. Two events are related to the verdict given by the TTP and two other events to the verdict received by the user. A critical point is reached when the TTP decides whether or not the registration is successful. This decision depends on the correctness of message 3. Before the TTP sends his positive or negative acknowledgement, he generates a special event. The `TTP_REG_SUCCEEDED` event corresponds to the positive acknowledgement and the `TTP_REG_FAILED` event corresponds to the negative acknowledgement. When the user receives the TTP's response, he also reaches a critical point. Thus, he generates a `USER_REG_SUCCEEDED` event or a `USER_REG_FAILED` according to the response received.

Property P4 can be expressed by the graph shown on figure 5. 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`.

4.4 A flaw

Aldebaran has discovered that the property P4 was not satisfied. The behaviour of the registration protocol cannot be simulated by the graph of the property regarding the relevant special events. It has found a sequence where a `USER_REG_FAILED` occurs before a `TTP_REG_SUCCEEDED`. The TTP successfully registers the user after the user has learned that his registration failed.

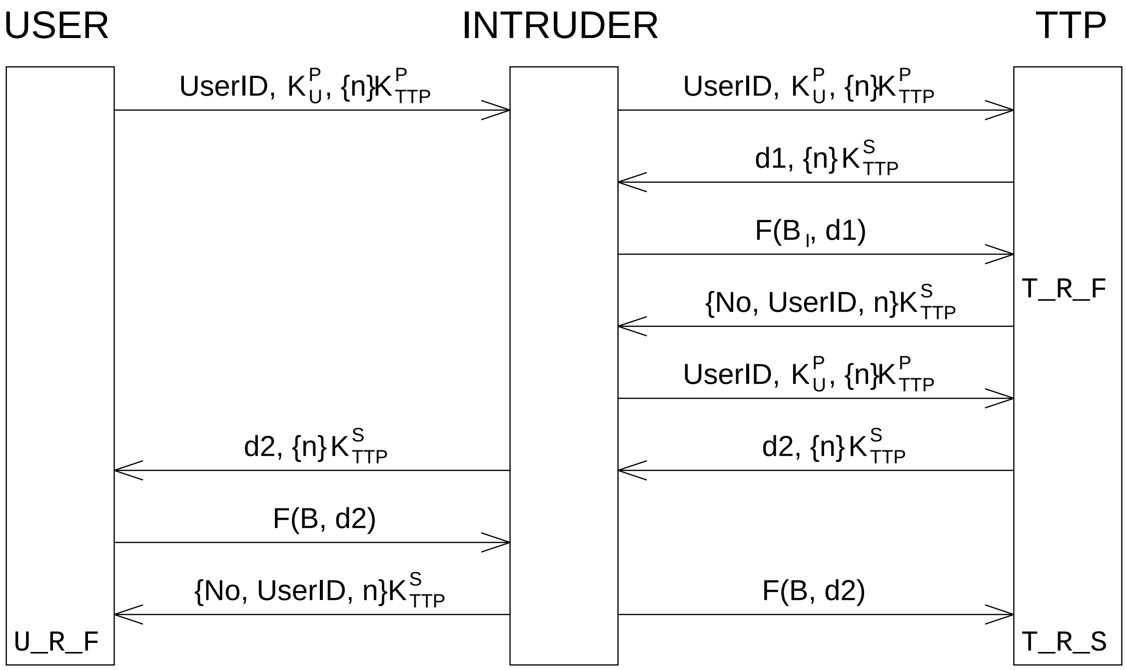

Figure 6 : Scenario of the intruder's attack

We use the Exhibitor tool to produce a diagnostic sequence that immediately shows us how the intruder has built his attack. The scenario is exhibited in figure 6.

When the intruder receives a registration request message from the user, he fowards it to the TTP and makes the first challenge fail with a fake response to obtain a negative acknowledgement from the TTP. Then the intruder follows on by replaying the registration request message previously recorded. Upon reception, the TTP starts a second registration with the user and sends a second challenge. This time, the intruder forwards the challenge to the user who is still waiting for his first challenge. The user replies with a valid message and waits for an acknowledgement. The intruder replays the negative one previously received. This acknowledgement is valid and thus the user declares that the registration failed. Meanwhile the intruder fowards the valid response of the user to the TTP who declares the registration successful. Both parties have finished their exchange but they do not have the same point of view of the situation.

For this attack to succeed, the intruder only needs to create a fake response to the first challenge. The strengh of our technique is that the analysis of the diagnostic 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.

4.5 Corrected protocol

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 :

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

Aldebaran states that all the properties, including P4, are fulfilled with this version. Hence, the mutual authentication and the transmission of the public key succeed despite the attempts of the intruder.

4.6 Enhancements of the protocol

This section deals with two improvements of the protocol. Firstly, we will try to obtain the simplest protocol. Encryptions and signatures were used to have the assurance that the intruder could not alter messages or parts of them. The formal description we made will help us to establish which cryptographic operations are really essential. Our guideline is to minimize cryptographic operations because public key cryptography has a very high computational cost.

Secondly, we will modify the protocol to help the entities to make the distinction between a failure and an error. When an entity receives a message, it performs several checks. If one of them fails, a message indicating the reason of the error is sent to the environment. 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. This separation between failures and errors helps to determine whether an intruder is disturbing the registration or not.

4.6.1 The simplest protocol

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. The TTP's signature of the acknowledgement is sufficient to perform this authentication. The user knows the TTP's public key so that he can verify that this message originates from the TTP. The random number d ensures that it belongs to the current registration and has not been replayed by the intruder. Thus, the user has the guarantee that he is talking to the TTP for the registration presently in progress.

Section 4.5 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. The message 4 is composed of the TTP's response, the user's identity and the random number d. So the authentication of d with a signature in the registration challenge message is not necessary. Only the final check of the acknowledgement is mandatory.

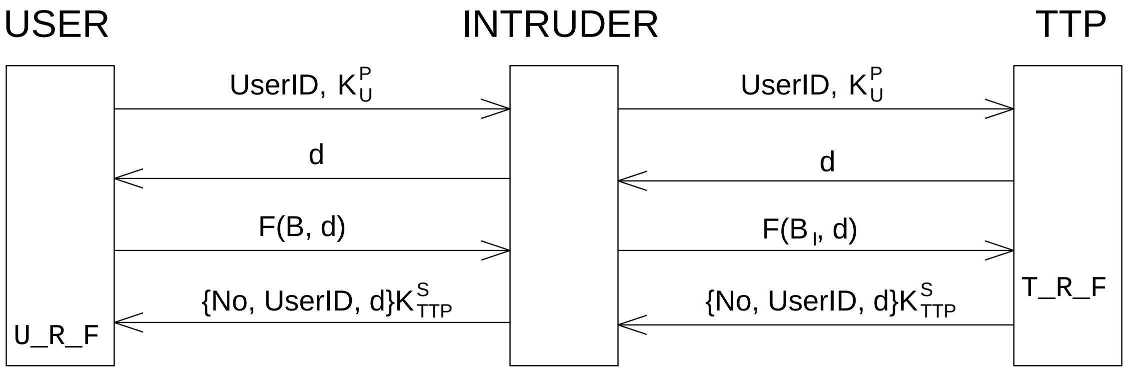

Figure 7 : A failure of the user generated by the intruder

These two simplifications lead to a very simple protocol with only one signature :

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

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. Regarding the special events only, a safety preorder exists between the corrected version of the protocol and this simplified version. Hence, all safety properties, expressible on the special events, verified on the latter are verified on the former.

4.6.2 Distinction between failures and errors

With this second improvement, we want to give the entities the ability to know exactly why a registration does not complete. This additional requirement will introduce complexity in the protocol. The simplification described in 4.6.1 brought us in the opposite direction but now we can build our strategy on solid bases.

In our specification, the special events only happened when neither the user nor the TTP meet an erroneous message otherwise, they declare an error and stop the protocol. By adding cryptographic operations and checks to the processing of the messages, we reduce the cases where a special event can take place. The model of our user owns valid credentials and always performs a correct registration but he still declares failures with `USER_REG_FAILED` events. This behaviour can only result from intruder's actions and shows that the user cannot completely distinguish errors from failures. In other words, some errors are interpreted as failures. Normally, the user must never meet a failure with our assumptions, meaning that the user's registration must always finish with a positive answer from the TTP or an error. The figure 7 exhibits a scenario that leads to such a failure with the simplified version of the protocol.

The user starts his registration and the protocol progresses normally until the intruder replaces the register response message of the user with another one. This new message is wrong because the intruder does not own credentials the TTP is waiting for and thus, a failure is declared and the TTP sends a negative acknowledgement. The user also declares a failure upon its reception.

This scenario is not related to the authentication properties of safety we have previously verified. The TTP refuses to authenticate the user due to an intruder's action but is not authenticating the user incorrectly. The reason of the failure is linked to the integrity of the messages transmitted during the protocol. In this particular case, the register response message has been changed by the intruder.

To achieve the user's distinction of errors from failures, we will strengthen the requirements on the protocol and add a new property.

- P6 : The user must never learn that his registration has been refused by the TTP.

or expressed with special events :

- P6 : No `USER_REG_FAILED` event is allowed in the LTS of the system.

From the point of view of the TTP, he would also make a complete distinction between failures and errors if he never declares a failure of the user because the user always tries to perform a valid registration. All disturbing elements must come from the intruder and must lead to errors (or possibly to a `TTP_REG_FAILED` with the intruder's identity). We model this case with another new property called P7 that does not allow the TTP to refuse to register the user. Formally, no `TTP_REG_FAILED` event with the user's identity (`TTP_REG_FAILED !USERID_A`) is permitted in the LTS of the system.

We check for the presence of `USER_REG_FAILED` and `TTP_REG_FAILED !USERID_A` events using the Exhibitor tool. If the verification does not find any of these events, our new properties are satisfied. The simplest protocol cannot guarantee P6 nor P7 because the parameters used in the GQ algorithm are not checked before the GQ verification (see the previous scenario). So we propose a new solution with two new signatures.

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

The main difficulty to solve comes from the GQ verification. The protocol must provide a way to find why the GQ calculation is not correct. If the problem is due to the use of bad credentials, the TTP must declare a failure, otherwise he must declare an error.

The signature of the register challenge message allows the user to verify that the data transmitted in the first message were correctly received. This could not be achieved by signing the register request because the TTP does not know the user's public key yet. If and only if the user agrees with the register challenge, he generates a response $F(B,d)$, signed with his private key. When the TTP receives this third message, he can use the recently received public key to check the signature. If the signature is incorrect, the TTP declares an error. Otherwise, if the result of the GQ computation is correct, that means that the user has received a valid register challenge message and thus agrees with the public key used in this message. Hence the TTP owns the

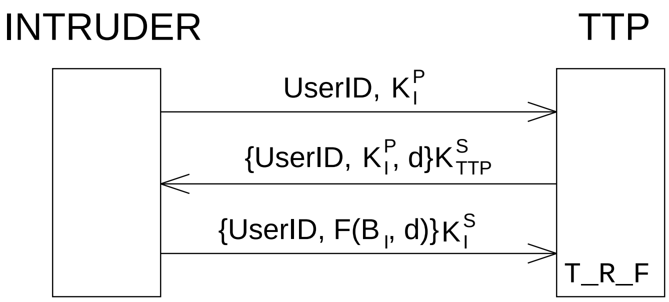

Figure 8 : A failure of the TTP generated by the intruder

real public key of the user. Both the GQ computation and the signature must be correct. One of them is not enough to make a good verification.

From the TTP's point of view, nothing distinguishes the received result of the function F from a random number before the GQ verification. So we have added the user's identity in the register response message to allow the TTP to check the user's signature.

With this version of the protocol the transmission of the user's public key does no longer need to be associated with the computations of the GQ verification. Our model of the GQ identification scheme states that the function F acts as a signature verified by the user's identity and the user's public key. In fact, this new version of the registration protocol can be used with a GQ algorithm in which B is only linked to user's identity and not to its public key. This is because the two new signatures in messages 2 and 3 allow the certification of the user's public key. This simplified GQ is in fact the original one [GQ88].

Property P6 is satisfied with this version. There is no possible `USER_REG_FAILED` event. All the intruder's actions are detected by the various checks involved in the cryptographic operations. Nevertheless, it was not possible to suppress all the `TTP_REG_FAILED` events. Property P7 is thus not satisfied. Indeed, the complete removal of these events would imply a kind of authentication before the authentication itself, and therefore constitutes an unreachable goal. The figure 8 will further clarify this. It exhibits a possible attack where the intruder replaces the user's public key with its own. Without knowing the right user's public key before the beginning of the registration, the TTP cannot detect the falsification.

5 Conclusion

This paper presents a formal verification process for security protocols using LOTOS. We have shown how to specify a protocol with the concept of trusted and untrusted principals. The flexibility of abstract data types allows the description of a wide range of cryptographic operations. We have shown the modelling of the classical public-key scheme but also a more complex one : the Guillou-Quisquater algorithm.

We have shown how intrusion can be taken into account by adding an intruder process replacing the communication channels. 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 explained the validation process and the formalization of security properties. They can be modelled as safety properties with the help of special events triggered when crucial states are reached. The verification is based on the safety preorder which should hold between the system and the property.

Our method is illustrated with a registration protocol where we have found a subtle flaw that could probably not have been discovered, at least so early, by a human-being. The verification is quite automatic and allows one to make efficient corrections and improvements. Some assumptions on the model were required and formal proofs of their correctness are an interesting future work. Nevertheless, this example demonstrates that LOTOS is suitable to verify complex cryptographic protocols that can enforce a wide variety of security properties.

6 Acknowledgements

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".

References

[AG97] M. Abadi and A.D. Gordon: *"A Calculus for Cryptographic Protocols The Spi Calculus"*, Proceedings of the 4th ACM Conference on Computer and Communication Security, 1997.

[Bol96] D. Bolignano: *"Formal Verification of Cryptographic Protocols"*, Proceedings of the 3rd ACM Conference on Computer and Communication Security, 1996.

[Bol97] D. Bolignano: *"Towards a Mechanization of Cryptographic Protocol Verification"*, Proceedings of CAV 97, LNCS 1254, Springer-Verlag, June 1997.

[BB87] T. Bolognesi and E. Brinksma: *"Introduction to the ISO Specification Language LOTOS"*, Computer Networks and ISDN Systems 14, pp. 25-59, 1987.

[BFG91] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez and J. Sifakis: *"Safety for Branching Time Semantics"*, 18th ICALP, Springer-Verlag, July 1991.

[BAN90] M. Burrow, M. Abadi and R. Needham: *"A Logic of Authentication"*, ACM Transactions on Computer Systems, 8, 1990.

[EM85] H. Ehrig and B. Mahr: *"Fundamentals of Algebraic Specification 1, Equations and Initial Semantics"*, In: W. Brauer, B. Rozenberg, A. Salomaa, Eds, EATCS, Monographs on Theorical Computer Science, Springer-Verlag, 1985.

[FGK96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier and M. Sighireanu: *"CAESAR/ALDEBARAN Development Package: A Protocol Validation and Verification Toolbox"*, Proceedings of the 8th Conference on Computer-Aided Verification, R. Alur & T. Henzinger Eds, August 1996.

[Gar96] H. Garavel: *"An Overview of the Eucalyptus Toolbox"*, Proceedings of COST247 workshop, June 1996.

[GL97] F. Germeau, G. Leduc : *"A Computer Aided Design of a Secure Registration Protocol"*, Proceedings of FORTE/PSTV 97, Chapman & Hall, 1997, to appear.

[GQ88] L. Guillou, J.-J. Quisquater : *"A Practical Zero-knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory"*, Proceedings of Eurocrypt 88, Springer-Verlag Eds, pp. 123-128, 1988.

[GBM96] J. Guimaraes, J.-M. Boucqueau and B. Macq : *"OKAPI: a Kernel for Access Control to Multimedia Services based on Trusted Third Parties"*, Proceedings of ECMAST 96, pp. 783-798, Louvain-la-Neuve, Belgium, May 1996.

[Hoa85] C.A.R. Hoare : *"Communicating Sequential Processes"*, Prentice-Hall International, 1985.

[ISO89] ISO/IEC. Information Processing Systems - Open Systems Interconnection : *"LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour"*, IS 8807, February 1989.

[LBQ96] S. Lacroix, J.-M. Boucqueau, J.-J. Quiquater and B. Macq : *"Providing Equitable Conditional Access by Use of Trusted Third Parties"*, Proceedings of ECMAST 96, pp. 763-782, Louvain-la-Neuve, Belgium, May 1996.

[LBK96] G. Leduc, O. Bonaventure, E. Koerner, L. Léonard, C. Pecheur, D. Zanetti : *"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, Montreal, Canada, October 1996.

[Low96] G. Lowe : *"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, 1996.

[Mea92] C. Meadows : *"Applying Formal Methods to the Analysis of a Key Management Protocol"*, Journal of Computer Security, 1992.

[Mil89] R. Milner : *"Communication and Concurrency"*, Prentice-Hall International, 1989.

[Pec96] C. Pecheur : *"Improving the Specification of Data Types in LOTOS"*, Doctoral dissertation, University of Liège, November 1996.

[Sch96] B. Schneier : *"Applied Cryptography"*, Second Edition, John Wiley & Sons Eds, 1996.