ELSEVIER

Computer Communications 23 (2000) 1089–1103

computer communications

www.elsevier.com/locate/comcom

Verification of security protocols using LOTOS-method and application

G. Leduc, F. Germeau

Research Unit in Networking (RUN), Institut Montefiore B28, University of Liège, B-4000 Liege, Belgium

Abstract

We explain how the formal language LOTOS can be used to specify security protocols and cryptographic operations. We describe how security properties can be modelled 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 an attack, correct the protocol, propose a simpler yet secure protocol, and finally a more sophisticated protocol that allows a better discrimination between intruder's attacks and classical protocol errors. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Security protocol; LOTOS; Protocol verification; Model-checking

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.

Our approach consists of using a generic formal language (LOTOS) and its associated verification methods and tools to verify security protocols. We explain how LOTOS can be used to specify security protocols and cryptographic operations, and show how security properties can be modelled as safety properties and checked automatically by a model-based verification tool. In our method a simple and powerful intruder process is explicitly added to the specification, so that the verification of the security properties guarantees the robustness of the protocol against attacks of such an intruder.

Our approach is similar to Refs. [24,25] where authentication protocols were specified in CSP [17] and checked by the FDR tool by verifying the trace inclusion relation between the system and the property. This tool and the one we have used are not classical model-checkers but rather equivalence or preorder checkers. Model-checkers (e.g. Refs. [26,30]) have also been used in similar ways.

The model-based methods are extremely powerful at finding subtle flaws in protocols, but are less adequate to prove correctness when no bug is found. This is because they are applied on simplified, though realistic, models of the systems. On the other hand, theorem provers [2,7,19,32] can provide such proofs and can also deal more easily with infinite-state systems. However, the proofs are usually less automated, and when no proof has been derived for a given property, it is not easy to know whether the property is wrong or whether the tool simply did not find it. In particular, an attack that falsifies the property is not provided automatically.

We illustrate our technique on a concrete registration protocol which is a part of the Equicrypt protocol [21] designed in the ACTS OKAPI project. We have already verified and corrected the subscription protocol [22,23] and the registration protocol [13,14] of Equicrypt. This paper extends our previous work in two ways: firstly, we present a more complete picture of our approach and secondly, we propose an enhanced design in two steps: we find a simpler registration protocol that remains secure, and a more sophisticated protocol that allows a better discrimination between intruder's attacks and classical protocol errors.

The paper is organized as follows: in Section 2, we will show that the LOTOS language is appropriate to handle the specification of security protocols at a high level of abstraction. 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

E-mail address: leduc@montefiore.ulg.ac.be (G. Leduc).

PII: S0140-3664(99)00239-X

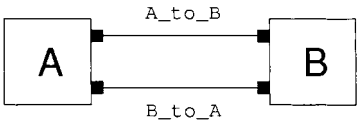

Fig. 1. Principals without intruder.

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. Finally, we compare our approach with related work.

2. LOTOS specification

In our approach the formal specification of a security protocol is written in LOTOS [4,18] which is a standardized language suitable for the description of distributed systems. It is made up of two components:

- A process algebra, mostly inspired by CCS [29] and CSP [17], 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 [10], 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.

A LOTOS specification is composed of two different 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 as 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 although it is considered genuine by the other entities.

Principals are linked together with communication channels to exchange 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 synchronization 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 synchronization of the transmission gate of one principal with the reception gate of the other principal. A second synchronization handles the other way of the communication channel.

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

To introduce the intruder that will try to threaten 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 other but indirectly through the intruder process (Fig. 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 it a message through

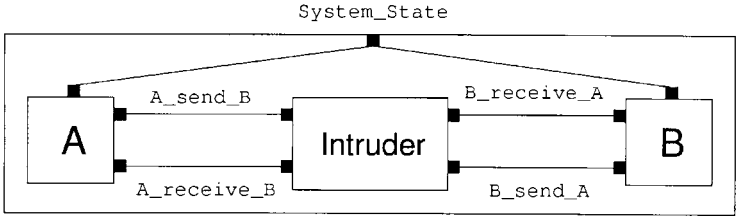

Fig. 2. Principals with intruder and environment.

the `System_State gate`. These messages are called security 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. Fig. 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 some initial knowledge. This knowledge includes identifiers, keys or whatever information a principal must know or generate locally before running the protocol. As we will see later, such a 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 [33], but only a few of them are actually used in security protocols. We do not intend to make an exhaustive translation of cryptographic operations into 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 that uses a zero-knowledge identification scheme.

ACT ONE is not only used to define the data transferred in messages, but also to define the internal database of information of each principal. For instance, a registration principal needs 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 specifications are written using data type language extensions, as offered by the APERO tool [31] 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 explicitly 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 that public and private keys are base values of some sorts and that a `match(PublicKey,PrivateKey)` operation exists 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.

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 composed of the message in clear and of an encrypted hash of it. Thus our model provides the `GetMessage` operation to access the message without any key. Of course, no operation allows the derivation of the private key.

```
type SignedMessage is
     Message, PublicKey, PrivateKey
sorts SignedMessage
opns
 S (*! constructor *):
    PrivateKey, Message
                        -> SignedMessage
 V: PublicKey, SignedMessage -> Boolean
 GetMessage: SignedMessage -> Message
eqns
forall msg: Message,
       pubkey: PublicKey
       prvkey: PrivateKey
ofsort Boolean
 V(pubkey,S(prvkey,msg)) =
                   Match(pubkey,prvkey);
ofsort Message
 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 encrypted message without having the private key, or forging a signed message from the message in clear without having the private key. Note that LOTOS easily provides processes that transgress this rule, and thus break any cryptosystem. For example, we can write a process that enumerates all possible messages, encrypts them and tests whether there is some matching between one of them and a given encrypted message. These kinds of unrealistic LOTOS behaviours should thus be avoided, because this would break any reasonable assumption about cryptography. Hopefully, these unrealistic processes are very special and easily avoidable. In particular, the tools will reject them because they would conceptually generate infinite-state (or very large) models.

2.3. The intruder

2.3.1. Model

We want to model an intruder as a process that can mimic attacks of a real-world intruder. 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; and
- 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, one per data sort. 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 from 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 analyzes 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 possible 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 authority. But he must also be able to act as a valid trusted authority from the point of view of the user. This example will be explained in more details in 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 transgress 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. Therefore 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 where each iteration is either a message reception or a message transmission. The structure of the intruder is thus very simple and not at all error prone. The body of the loop is merely an enumeration of a couple of possible message receptions, followed by a couple of message transmissions. When a message is received, it is segmented into all its fields, which are stored in separate sets. The encrypted fields that can be decrypted by one of the known keys are stored in clear. These actions are modelled by the `insert` operation in the specification

below. As regards message transmission, LOTOS provides the choice operator that automatically enumerates all the possible messages that can be built from a set of components. Here, this set is actually a multiset where each element is a set of message fields, and the selection is modelled by the is_in_knowledge predicate.

Although the structure of the LOTOS specification is such that only two principals are present, this does not mean that A and B know a priori that they are executing a run between them. This can only be known by executing a correct authentication protocol.

```
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

Model-based verification methods are inherently limited in the sense that they are powerful at finding bugs in protocols but fall short in proving full correctness. There are several reasons for that. Even though some research work is carried out to extend model-based methods to infinite-state systems, practical methods are presently limited to finite-state systems (of reasonable sizes), whereas most protocols are not, because either their data space is not, or their control structure is not (e.g. there may be an arbitrarily large number of protocol instances running in parallel). Therefore, these methods are only applicable if some abstraction is used that keeps the model finite-state and also of reasonable size. However, it is essential that these abstractions be error preserving, in the sense that an error found on the abstract model is a real error of the actual protocol. Clearly, the absence of error in the abstract model is no guarantee about the actual protocol. Knowing that, our objective is to capture as much as possible of the possible behaviours of the actual protocol.

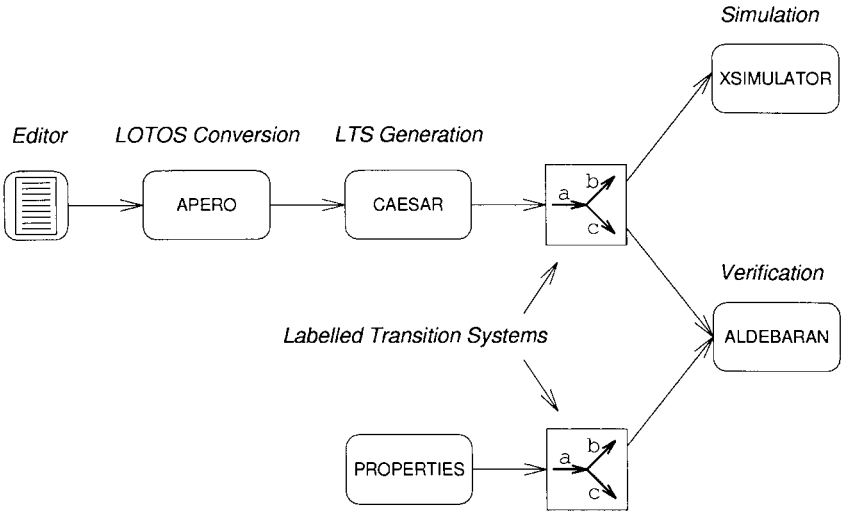

Fig. 3. The eucalyptus toolbox.

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

Although some message fields like random numbers or time stamps are specific to one run of the protocol, their number is potentially infinite. This infinity must be controlled by giving some well-chosen properties to these specific message fields. Trusted principals will typically use any but a single specific value in each run they perform, so we give them a limited set of values that will be used during their executions. We also give the intruder one such value but which is different from those of the trusted principals. When the intruder will use this value in a particular message field, this will, in fact, model all the possible messages not created by a trusted principal. This is an abstract interpretation which is often used when the protocol is independent from a piece of data, i.e. when the behaviours of the entities do not depend on the particular value used. This abstraction still allows the processes to check for equality and inequality of message fields. Moreover, this reduction is error preserving: the actual protocol can perform all the traces of the abstract model, because the latter merely reduces the possible values that can be used in the message fields.

The initial knowledge of principals is large enough to allow them to participate in several runs of the protocol, possibly in parallel. In addition, the intruder is given an initial knowledge that allows him to act as other trusted principals.

All in all, we believe we cover a large body of the possible behaviours of the actual system, but of course we are never sure we do not preclude a subtle attack which has been filtered out by our abstraction. As explained above, we do not aim at proving full correctness, but more modestly to prove very large parts of the real protocols.

Some researchers have complemented their model-based verification by additional proofs, e.g. Ref. [24] where the model-based verification based on two principals is further generalized by induction to an arbitrary number of principals. Another research direction is proposed in Ref. [3] where an abstraction function automates the computation of a correct abstract model. All these methods can push the limits of model-based verification further.

Now that we have presented the complete specification, we will detail the verification process.

3. Verification process

3.1. Properties to be verified

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 characterized 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 requires 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, by e.g. intercepting every message sent on the channel. 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 applies them while the intruder process tries to defeat them. A way to verify the robustness

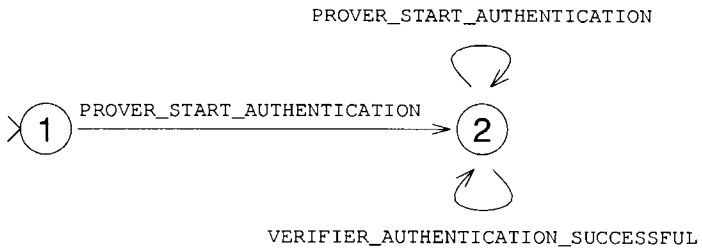

Fig. 4. LTS of the authentication property.

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, called security 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 the internal state of the principal. The environment of the LOTOS specification is responsible for receiving these messages. By executing a security event, a principal declares that he is confident of a fact.

Let us consider an authentication protocol between two principals where a prover must be authenticated by a 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 security events regardless of the particular authentication mechanisms used. We just state that "At least one `PROVER_START_AUTHENTICATION` event must have occurred before any `VERIFIER_AUTHENTICATION_SUCCESSFUL` event".

This technique can be applied to a wide range of security properties. In practice, the security events will have a finer structure to better identify the protocol run to which they refer. Parameters of security events can be a principal's identity, an authentication token, a particular key, nonce or any other data relevant to the properties we want to prove. So, the set of security events and their structure is linked to the set of properties.

This method allows one to abstract away from all the details of security mechanisms. We can only focus on the security services achieved. As a matter of fact, these events are some sort of service primitives exchanged with the environment. Some of them request security services, others indicate that a request has been issued by another principal, or confirm that a security service is completed. One of the difficulties is to gain the assurance that the security properties are translated correctly into properties on security events. But this is inherent to any verification approach: properties should be expressed in one way or another and we cannot guarantee that the properties are expressed correctly. This process could be made less error-prone by providing guidelines to express the most common properties. In Ref. [1] for example, an approach is proposed to model typical security properties in the framework of the Spi-calculus.

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 [11,12] included in the Eucalyptus toolbox to carry out the verification of the protocol. As Fig. 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 LTS from the LOTOS specification. This graph contains exactly the possible execution sequences of the studied protocol. Section 2.4 has addressed the feasibility of the generation. To gain confidence into the specification, it is first 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 LTS. 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 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 Fig. 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's is made, 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 (safety, liveness and fairness) 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's $Sys_1 = \langle S_1, A, T_1, s_{01} \rangle$ and $Sys_2 = \langle S_2, A, T_2, s_{02} \rangle$ are related modulo the strong bisimulation denoted $Sys_1 \Im Sys_2$, 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 simple safety properties obviously expressible in Branching time Safety Logic (BSL) [5], the minimization can be further improved modulo the safety equivalence (defined below), which preserves all the properties expressible in BSL.

Not all the observable actions are relevant to verify the properties. In particular, our properties only rely on security events, so that other actions can be hidden. The minimized LTS of our protocol can be checked against the LTS of a property by verifying the safety preorder relation [5] 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 $= \langle S, A, T, s_0 \rangle$ and let's define $L = A - \{i\}$, a relation $R \subseteq S \times S$ is a weak simulation if:

If $\langle P, Q \rangle \in R$ then, $\forall a \in L$,

if $P \stackrel{i^* a}{\Rightarrow} P'$, then $\exists\, Q' : Q \stackrel{i^* a}{\Rightarrow} Q'$ and $\langle P', Q' \rangle \in R$.

A LTS $Sys_1 = \langle S_1, A, T_1, s_{01} \rangle$ can be simulated by $Sys_2 = \langle S_2, A, T_2, s_{02} \rangle$, denoted $Sys_1 \leq_s Sys_2$, iff there exists a weak simulation relation $R \subseteq S_1 \times S_2$ such that $\langle s_{01}, s_{02} \rangle \in R$. Two LTS's 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 abstract 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 a detailed sequence allowed by the specification and matching the abstract one. This sequence always exists, but is not necessarily unique. This does not matter. It suffices to have one such trace as diagnostic to clearly identify the scenario that leads to the undesirable state where the property is not verified. The Exhibitor tool can even be used to find the shortest such trace, which helps understand the intruder's attack.

The verification process of the properties is then complete. If one or more of them are not satisfied, our method gives diagnostics of enormous help to the redesign of 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 [16]. 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. Readers interested in more details can refer to [14].

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 right user's public-key during the protocol to issue a corresponding public-key certificate needed for the subscription phase.

The authentication of the user by the TTP uses the Guillou–Quisquater (GQ) zero-knowledge identification scheme [15]. 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 [21]. The registration protocol presented in this paper is, in fact, an enhanced version of the original one found in Ref. [21].

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 particularly simple while still capturing the essence of it. The key point of the authentication is the secret credentials. If we consider them as a secret encryption key and the user's identity together with his 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 remainder of this paper, we will present all the messages with the following structure:

$$Nb : Source \rightarrow Destination : Message\ Id\langle Message\ Fields\rangle$$

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 is the credentials.

The protocol works as follows:

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

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

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 $\langle UserID, K_U^P, n, d\rangle$.

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

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.

$$3 : User \rightarrow TTP : Register\ \mathrm{Res}ponse\langle F(B, d)\rangle$$

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 (as an identifier of the registration run). If the response is positive, the TTP registers the tuple $\langle UserID, K_U^P\rangle$

$$4^+ : TTP \rightarrow User : Register\ Ack\langle\{Yes,\ UserID,\ n\}K_{TTP}^S\rangle$$

$$4^- : TTP \rightarrow User : Register\ Ack\langle No,\ UserID,\ n\rangle K_{TTP}^S\rangle$$

4.2. Protocol 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 and 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 LTS of the protocol has been generated. It is composed of 487 446 states and 2 944 856 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

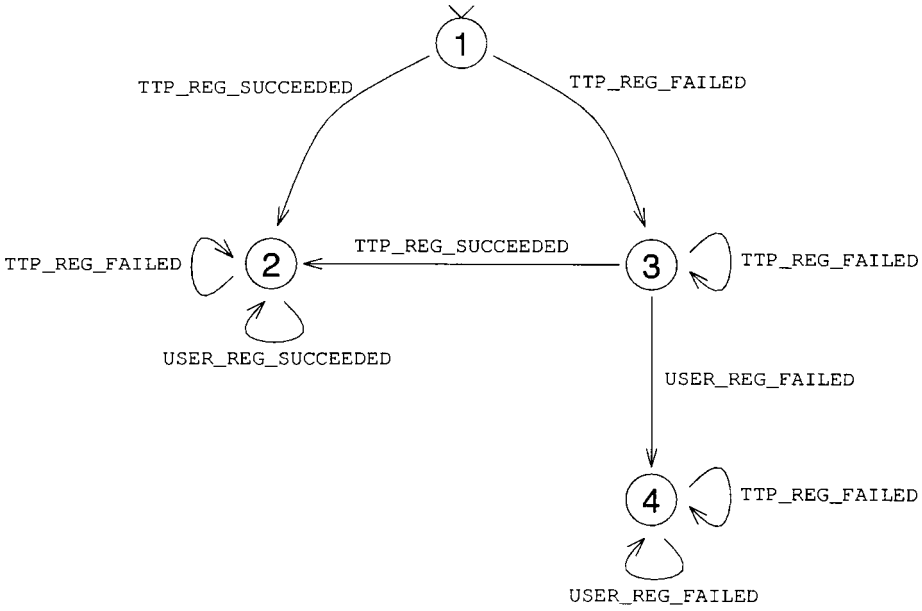

Fig. 5. Labelled transition system modelling property P4.

bisimulation was very important. The minimized LTS of the protocol is made of 3968 states and 37 161 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. More details can be found in Ref. [14]. This property is necessary (but not sufficient) to achieve 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.

Four security 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 sending his positive or negative acknowledgement, the TTP generates a security 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 Fig. 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 been 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`.

For clarity, Fig. 4 does not show the parameters of the security events, but it should be clear that this picture focuses on a single run of the protocol. The property should be true for every run. The fact that several `TTP_REG_FAILED` are allowed (refer to the loops) by the property means that we allow the intruder to try several fake registrations during the considered run. Such a graph models an upper bound on the possible behaviours which do not falsify the intended property. Our model is supposed to generate a subset of this graph, except if there is a security breach.

4.4. A flaw

Aldebaran has discovered that property P4 was not satisfied. The behaviour of the registration protocol cannot be simulated by the graph of the property regarding the relevant security 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. We use the Exhibitor tool to produce a diagnostic sequence that immediately shows us how the

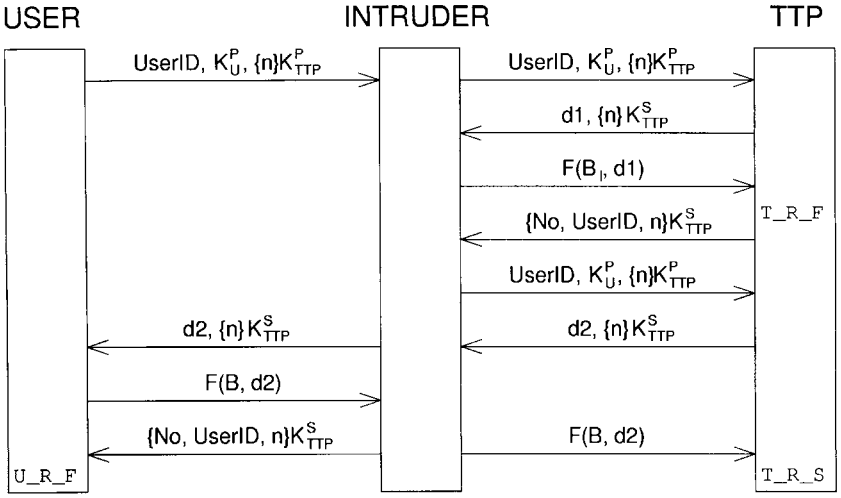

Fig. 6. Scenario of the intruder's attack.

intruder has built his attack. The scenario is exhibited in Fig. 6

When the intruder receives a registration request message from the user, he forwards 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 forwards 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 strength of our technique is that the analysis of the diagnostic sequence immediately brings us the reason of the failure. Despite the presence of the nonce *n*, 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:

$$1: \; User \rightarrow TTP \; : \; Register\ Request\langle UserID,\ K_U^P, \{n\}K_{TTP}^P\rangle$$

$$2: \; TTP \rightarrow User \; : \; Register\ Challenge\langle d, \{n\}K_{TTP}^S\rangle$$

$$3: \; User \rightarrow TTP \; : \; Register\ Response\langle F(B,d)\rangle$$

$$4^+: \; TTP \rightarrow User \; : \; Register\ Ack\langle\{Yes, UserID, n, d\}K_{TTP}^S\rangle$$

$$4^-: \; TTP \rightarrow User \; : \; Register\ Ack\langle\{No, UserID, n, d\}\rangle K_{TTP}^S\rangle$$

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 protocol entities to make the distinction between a normal registration failure due to bad credentials and a registration failure due to a protocol error (caused in fact by an intruder's interference). We call the former a failure and the latter 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

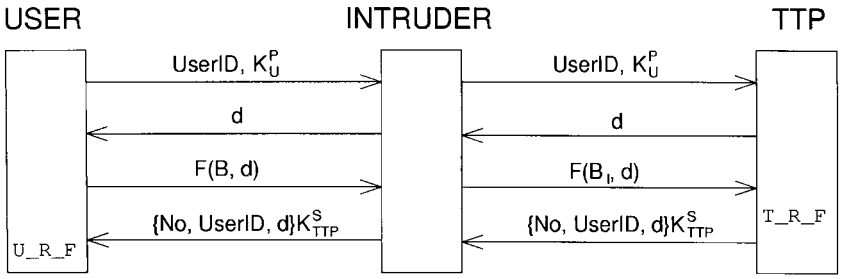

Fig. 7. A failure of the user generated by the intruder.

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. Therefore it seems unnecessary to authenticate the TTP already in the registration challenge (message 2). This suggests to remove the nonce *n* from the whole protocol.

These two simplifications lead to a very simple protocol with only one signature (and the GQ calculation):

1 : $User \rightarrow TTP$: $Register\ Request\langle UserID, K_U^P\rangle$

2 : $TTP \rightarrow User$: $Register\ Challenge\langle d\rangle$

3 : $User \rightarrow TTP$: $Register\ Response\langle F(B,d)\rangle$

4^+ : $TTP \rightarrow User$: $Register\ Ack\ \langle\{Yes, UserID, d\}K_{TTP}^S\rangle$

4^- : $TTP \rightarrow User$: $Register\ Ack\langle\{No, UserID, d\}K_{TTP}^S\rangle$

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 during the protocol run. Regarding the security events only, a safety preorder exists between the corrected version of the protocol and this simplified version. Hence, all safety properties, expressible with the security events and verified on the latter are necessarily 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, either because the user has used bad credentials or because of an intruder's attack. This additional requirement will introduce complexity in the protocol. The simplification described in 4.6.1 led us in the opposite direction, but now we can build our design strategy on solid bases. Before going further, we define an authentication failure as the occurrence of a `USER_REG_FAILED` event, and a protocol error as any other unsuccessful termination of the protocol due to any sort of invalid message reception (due to an intruder's interference). When the user or the TTP receive such an invalid message, they will just raise an error and stop the protocol. Ideally a `USER_REG_FAILED` event can only occur when the user owns invalid credentials.

In our model, the user owns valid credentials and always initiates a correct registration. So normally the user should never face a registration failure; which means that the user's registration must always terminate with a positive answer from the TTP or possibly with a protocol error. Nevertheless, the verification shows that `USER_REG_FAILED` events occur in some scenarios. This behaviour can only result from intruder's actions and shows that the user cannot completely distinguish protocol errors from authentication failures. In other words, some errors are interpreted as failures. Fig. 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 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 for the failure is related 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 between protocol errors and authentication 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.

The same reasoning is valid from the point of view of the TTP: he would make a complete distinction between failures and errors if he never declared a failure of the user, since 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 the registration of 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 or 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.

$1: \ User \rightarrow TTP \ : \ Register\ Request\langle UserID, K_U^P\rangle$

$2: \ TTP \rightarrow User$

$: \ Register\ Challenge\langle\{UserID, K_U^P, d\}K_{TTP}^S\rangle$

$3: \ User \rightarrow TTP$

$: \ Register\ Response\ \langle\{UserID, F(B,d)\}K_U^S\rangle$

$4^+: \ TTP \rightarrow User \ : \ Register\ Ack\langle\{Yes, UserID, d\}K_{TTP}^S\rangle$

$4^-: \ TTP \rightarrow User \ : \ Register\ Ack\langle\{No, UserID, d\}K_{TTP}^S\rangle$

The main difficulty to solve comes from the GQ verification.

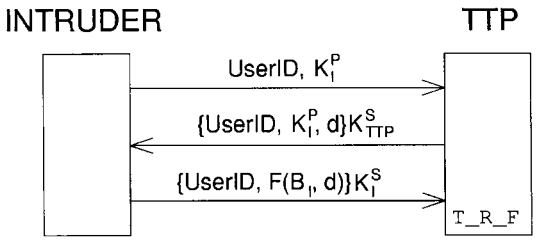

Fig. 8. A failure of the TTP generated by the intruder.

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, and 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 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 no longer needs 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 the 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 [15].

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. Fig. 8 will further clarify this. It exhibits a possible attack where the intruder replaces the user's public key with his own in the first message. Without knowing the right user's public key before the beginning of the registration (as one of the purposes of the registration is to transfer the public key), the TTP cannot detect the falsification. This means that from the TTP's viewpoint we have to accept that some intruder's interference will be indistinguishable from tentative registrations of users with invalid credentials.

5. Conclusion and related work

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. Our approach thus relies on classical formalisms and tools and contrasts with works that use a dedicated modal logic such as the BAN logic [6].

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 real-world non-cryptographic and non-repetitive attacks on the behaviour of the protocol. The idea of explicitly introducing an intruder was first proposed in Refs. [8,9] in another setting. This idea was then used in the Interrogator system [28], where the participants are modelled as communicating state-machines and the network is assumed to be under the control of an intruder, which can intercept messages, destroy or modify them, or pass them through unmodified. The NRL Protocol Analyser [20,27] is similar to the Interrogator, but the goal is here to prove the unreachability of some undesirable states. It can deal with infinite-state systems but the search is less automated than in the Interrogator. The difference between our approach and these methods is that we do not have to define some pathological target states to be searched for by the tool. We just give safety properties as reference graphs.

We have explained the validation process and the formalization of security properties as safety properties. These properties are similar to the correspondence properties, used in Ref. [34], which require that certain events can take place only if others have taken place previously. Basically, all properties that are expressible with security events can be checked with our approach. Our tool verifies that the safety preorder (i.e. the weak simulation) relation holds between the system and the property. We can check liveness properties is a similar way, but this is not very useful in practice because they are never satisfied as intruders can always intercept all messages in transit.

Our method is illustrated on a registration protocol. We have found a 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. However, as with any model-checking methods, we have had to simplify the model to keep it finite-state. There exist ways to extend the method to infinite-state systems. In a simpler case [24], an additional induction proof has been provided to extend the correctness guarantee to an arbitrary number of involved entities. Another possible approach, proposed in Ref. [3], is based on an abstraction function and automates the computation of a correct (finite) abstract model of the system.

Another approach which circumvents the problem of adding an explicit intruder process is proposed in Ref. [1] where the Spi-calculus is used to describe security protocols. The idea is to verify that the protocol specification placed in any Spi-calculus context is equivalent to the expected ideal behaviour (i.e. without intruder). Threads expressible in the Spi-calculus are thus implicitly considered among the possible contexts. However, this approach is not so easy to use in practice because the equivalence is sometimes too strong. For example, some intruder's actions may be such that the equivalence is not fulfilled, while the security of the system is not in danger, because the non-equivalence simply results from the falsification of an irrelevant property.

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

[1] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols—the spi calculus, in: Proceedings of the Fourth ACM Conference on Computer and Communication Security, 1997.

[2] D. Bolignano, Formal verification of cryptographic protocols, in: Proceedings of the Third ACM Conference on Computer and Communication Security, 1996.

[3] D. Bolignano, Towards a Mechanization of Cryptographic Protocol Verification, Proceedings of CAV 97, LNCS, vol. 1254, Springer, Berlin, 1997.

[4] T. Bolognesi, E. Brinksma, Introduction to the ISO specification language LOTOS, Computer Networks and ISDN Systems 14 (1987) 25–59.

[5] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez, J. Sifakis, Safety for branching time semantics, Proceedings of the 18th ICALP, LNCS, Springer, Berlin, 1991.

[6] M. Burrows, M. Abadi, R. Needham, A logic of authentication, ACM Transactions on Computer Systems 8 (1) (1990) 18–36.

[7] P. Chen, V. Gligor, On the formal specification and verification of a multiparty session protocol, in: Proceedings of the IEEE Symposium on Research in Security and Privacy, 1990.

[8] D. Dolev, S. Even, R. Karp, On the security of ping-pong protocols, Information and Control 55 (1982) 57–68.
[9] D. Dolev, A. Yao, On the security of public key protocols, IEEE Transactions on Information Theory 29 (2) (1983) 198–208.
[10] H. Ehrig, 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, Berlin, 1985.
[11] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, M. Sighireanu, in: R. Alur, T. Henzinger (Eds.), CAESAR/ALDEBARAN Development Package: a protocol validation and verification toolbox, Proceedings of the Eighth Conference on Computer-Aided Verification, LNCS, Springer, Berlin, 1996.
[12] H. Garavel, An overview of the Eucalyptus toolbox, in: Proceedings of COST247 workshop, June 1996.
[13] F. Germeau, G. Leduc, Model-based design and verification of security protocols using LOTOS, in: Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols, Rutgers University, Sept. 1997.
[14] F. Germeau, G. Leduc, A computer-aided design of a secure registration protocol, in: T. Mizuno, N. Shiratori, T. Higashino, A. Togashi (Eds.), Formal Description Techniques and Protocol Specification, Testing and Verification, Chapman & Hall, London, 1997, pp. 145–160.
[15] L. Guillou, J.-J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, Proceedings of Eurocrypt 88, Springer, Berlin, 1988 (pp. 123–128).
[16] J. Guimaraes, J.-M. Boucqueau, B. Macq, OKAPI: a kernel for access control to multimedia services based on trusted third parties, in: Proceedings of ECMAST 96, Louvain-la-Neuve, Belgium, May 1996, pp. 783–798.
[17] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[18] 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.
[19] R. Kemmerer, Using formal methods to analyse encryption protocols, IEEE Journal on Selected Areas in Communications 7 (4) (1989) 448–457.
[20] R. Kemmerer, C. Meadows, J. Millen, Three systems for cryptographic protocol analysis, Journal of Cryptology 7 (2) (1989) 14–18.
[21] S. Lacroix, J.-M. Boucqueau, J.-J. Quisquater, B. Macq, Providing equitable conditional access by use of trusted third parties, in: Proceedings of ECMAST 96, Louvain-la-Neuve, Belgium, May 1996, pp. 763–782.
[22] 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, in: Proceedings of 12th J. Cartier Workshop on Formal Methods and their Applications: Telecommunications, VLSI and Real-time Computerized Control System, Montreal, Canada, October 1996.
[23] G. Leduc, O. Bonaventure, L. Léonard, E. Koerner, C. Pecheur, Model-based verification of a security protocol for conditional access to services, Formal Methods in System Design 14 (2) (1999) 171–191.
[24] G. Lowe, Breaking and fixing the Needham-Schroeder public-key authentication protocol using FDR, in: T. Margaria, B. Steffen (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 1055, Springer, Berlin, 1996.
[25] G. Lowe, B. Roscoe, Using CSP to detect errors in the TMN protocol, IEEE Transactions on Software Engineering 23 (10) (1997) 659–669.
[26] W. Marrero, E. Clarke, S. Jha, A model checker for authentication protocols, in: Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols, Rutgers University, September 1997.
[27] C. Meadows, The NRL protocol analyser: an overview, Journal of Logic Programming 26 (8) (1996) 113–131.
[28] J. Millen, S. Clark, S. Freedman, The Interrogator: protocol security analysis, IEEE Transactions on Software Engineering SE-13 (2) (1987) 274–288.
[29] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[30] J. Mitchell, V. Shmatikov, U. Stern, Finite-state analysis of SSL 3.0 and related protocols, in: Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols, Rutgers University, September 1997.
[31] C. Pecheur, Improving the specification of data types in LOTOS (Doctoral dissertation, nr. 171, University of Liège, Nov. 1996).
[32] S. Schneider, Verifying authentication protocols in CSP, IEEE Transactions on Software Engineering 24 (9) (1998) 751–758.
[33] B. Schneier, Applied Cryptography, ed. 2, Wiley, New York, 1996.
[34] T. Woo, S. Lam, A semantic model for authentication protocols, in: Proceedings of IEEE Symposium on research in security and privacy, 1993.