Proc. of 12th J. Cartier Workshop on “Formal Methods and their Applications: Telecommunications, VLSI and Real-Time Computerized Control System”, Montreal, Canada, 2-4 Oct. 96.

Specification and verification of a TTP protocol for the conditional access to services[1]

G. Leduc[2], O. Bonaventure, E. Koerner, L. Léonard, C. Pecheur, D. Zanetti
Université de Liège, Institut d’Electricité Montefiore, B 28, B-4000 Liège 1, Belgium
Phone: + 32 4 3662691 Fax: + 32 4 3662989 E-mail: leduc@montefiore.ulg.ac.be

Abstract

In this paper we use the formal language LOTOS to specify the Equicrypt protocol and verify its robustness to attacks by an intruder. We use the model-based CADP verification tools from the Eucalyptus toolbox to discover some successful attacks against this protocol.

1. Introduction

The Equicrypt protocol is a conditional access protocol under design in the European ACTS OKAPI project [GBM96]. It allows users to subscribe to multimedia services such as video on demand. Equicrypt is designed to be equitable, meaning that any user or service provider can potentially enter the system provided that it complies with this minimal protocol. This contrasts with proprietary systems which all use different conditional access protocols and thus oblige users to implement almost as many protocols as there are different service providers, which is a severe limitation.

After a brief description of the protocol and its modelling in the formal language LOTOS [ISO 8807, BoB87], we will describe the verification process. We state some desired safety properties and formalize them. Then we describe a generic intruder process and its modelling. All properties are fulfilled without the intruder, but some of them are falsified when the intruder is added. The diagnostic sequences can be used almost directly to exhibit the scenarios of possible attacks on the protocol. Two of them are presented.

2. Presentation of the Equicrypt protocol

The following is a short summary of the Equicrypt protocol [LBQ+96] under design in the OKAPI project [GBM96].

---

[1] This work has been partially supported by the Commission of the European Union under the ACTS AC051 project OKAPI.

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

2.1. Structure

The aim of the Equicrypt system is to control the access to multimedia services broadcast on a public channel (the main examples being cable or satellite TV programs or Video On Demand services). Scrambling is used to make the data usable by authorized users only, using a special decoding device. To avoid requiring different decoders for every accessed service (provider), a unique decoder uses a public-key cryptography protocol to subscribe to and decode different services. An independent entity known as the Trusted Third Party (TTP) acts as a registering authority trusted by both users and providers.

The Equicrypt system (fig. 1) thus involves three kinds of entities:

- the Set Top Units (STU) of users,
- the Service Providers (SPv),
- the Trusted Third Party (TTP).

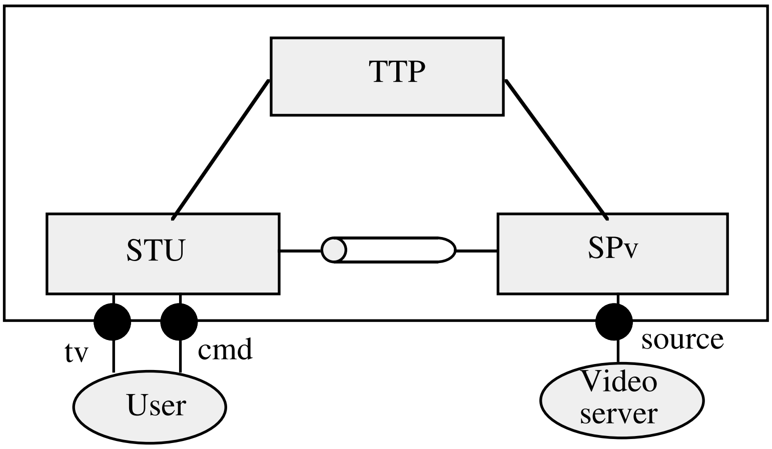

Fig. 1: The overall system

The communications between SPvs and STUs use an insecure (broadcast) channel, whereas their communications with the TTP use secure channels. The environment of this system is composed of video servers producing video images (through gate `source`) and users watching these images (through gate `tv`) and subscribing to services (through gate `cmd`). Every Set Top Box contains an Access Control Unit (ACU) which is basically a smart card that executes the security functions in the Set Top Box. In fact, it is the ACU that acts on the user's behalf in the Equicrypt system, and contains the critical security information.

2.2. Operations

Subscribing to services via TTPs occurs in three main phases:

- **ACU Certification:** a Certification Authority checks that the ACU performs sanely (e.g. will not trick the TTP or reveal concealed information) and assigns it a hard-wired certification number.
- **ACU registration:** an ACU gets registered by a TTP. It uses a cryptographic algorithm to prove the validity of its registration identifier w.r.t. its certificate without disclosing the latter.
- **Service subscription:** an ACU asks for access to a service. The SPv gets the ACU's registration information from the TTP. If this succeeds, the SPv sends to the ACU a key

allowing to de-scramble the desired service. Cryptography is used to prevent against eavesdroppers.

ACU certification is an administrative procedure that does not concern us. We shall specify certified ACUs, and describe ACU registration and service subscription, as well as the broadcasting of the service in itself.

2.3. Modelling of encryption operations

The TTP algorithms involve several encryption operations, for which we give an abstract view only. Each scheme uses peer encryption and decryption keys K_E and K_D and functions $E(\_, \_)$ and $D(\_, \_)$ such that $D(K_D, E(K_E, m)) = m$ for any message m. In the simple cases such as secret key cryptography, $K_E = K_D$ is the shared secret key. In public key cryptography, K_E is public and K_D is private for encryption (or vice-versa for authenticity).

2.4. Description of operations

All the messages have the following structure:

Number: Source $\rightarrow$ Destination: Message_id $\langle$parameters_list$\rangle$

Therefore we omit the source and destination in parameters_list.

We also use the more compact notation $\{m\}_K$ to denote the message m encrypted with the key K, that is $\{m\}_K = E(K, m)$.

Identification and keys:

During certification, a user's ACU that has the serial identification number A is assigned a certificate C_a correlated to (a hash value of) its identity A, in such a way that they operate as peer encryption/decryption keys, that is $D(A, \{m\}_{C_a}) = m$. Then the ACU generates peer keys K_a^p and K_a^s, respectively public and private (secret), and chooses an identification number A' which can be seen as an alias of A for privacy purposes. K_a^s will remain internal to the user's ACU.

A service provider B has peer public and private keys K_b^p and K_b^s and advertises K_b^p with proposed services. For each provided service S, it also keeps a key K_S.

Let N_{ta} be a nonce[1] (called challenge) generated by the TTP T to authenticate user A; and let N_{ab} be another nonce (called ticket) generated by user A to authenticate the service provider B

Registration:

1: $A \rightarrow T$: Register request $\langle A', K_a^p\rangle$
If T accepts the request, it generates and sends a random challenge N_{ta}.

2: $T \rightarrow A$: Register challenge $\langle\{N_{ta}\}_{K_a^p}\rangle$

[1] A nonce is a random number generated with the purpose of being used once (i.e. in at most one run of the protocol).

3:	$A \rightarrow T$:	Register response $\langle \{N_{ta}\}_{C_a} \rangle$ T checks $D(A, \{N_{ta}\}_{C_a}) = N_{ta}$. If so, A is registered with its public key K_a^p and its alias A', that is T stores the tuple $\langle A, A', K_a^p \rangle$ in its directory.
3':	$T \rightarrow A$:	Register ack

The real algorithm also involves a random number introduced by the ACU for preventing the TTP from guessing C_a. We ignore this in our model.

Subscription:

4:	$? \rightarrow B$:	Subscribe request $\langle A', S, \{N_{ab}\}_{K_b^p} \rangle$ B gets $N_{ab} = D(K_b^s, \{N_{ab}\}_{K_b^p})$. If this nonce N_{ab} has been used in a previous subscription, B ignores the request, otherwise the protocol continues.
5:	$B \rightarrow T$:	Check request $\langle A' \rangle$
6^+:	$T \rightarrow B$:	Check answer $\langle A'$, true, $K_a^p \rangle$
6^-:	$T \rightarrow B$:	Check answer $\langle A'$, false, $- \rangle$ A negative answer is provided to B if A is not registered or blacklisted. If the answer is positive, then B sends the message 7^+ below to A, otherwise B sends message 7^-.
7^+:	$B \rightarrow$ All:	Subscribe answer $\langle S$, true, $\{N_{ab}, K_S\}_{K_a^p} \rangle$ Only A can get $\langle N_{ab}, K_S \rangle = D(K_a^s, \{N_{ab}, K_S\}_{K_a^p})$. A then checks N_{ab} and, if correct, registers K_S and sends the acknowledgement message 8. Note that the ACU certification ensures that the service key K_S will remain concealed in the ACU and thus never be sent to other users.
7^-:	$B \rightarrow$ All:	Subscribe answer $\langle S$, false, $\{N_{ab}, -\}_{K_a^p} \rangle$
8:	$? \rightarrow B$:	Subscribe ack $\langle \{A', S\}_{K_a^s} \rangle$ It contains A's signature. B checks whether $D(K_a^p, \{A', S\}_{K_a^s}) = \langle A', S \rangle$ and keeps $\{A', S\}_{K_a^s}$ as a proof of delivery.

Broadcasting:

The broadcasting of the video service can then be a sequence of messages of the form:

repeat select a new control word CW;
send CW encrypted with K_S (message 9 below);
send a couple of images scrambled with CW (message 10 below);
until the service is over.

9:	$B \rightarrow$ All:	Send control word $\langle S, \{CW\}_{K_S} \rangle$ ACUs knowing K_S get $CW = D(K_S, \{CW\}_{K_S})$ and keep it in the (decoder of the) hosting set top box.

10: $B \rightarrow$ All: Send image $\langle S, \{\text{image}\}_{CW}\rangle$
The (decoders of the) set top boxes knowing the CW descramble the image as $D(CW, \{\text{image}\}_{CW})$.

In practice, the control words have to be sent a bit more ahead of the scrambled images, because the decryption of $\{CW\}_{K_S}$ takes some time, but we do not consider this issue here.

3. Specification

3.1. Behaviour

The LOTOS specification models both the Equicrypt system and the environment that it interacts with as two processes `EquicryptSystem` and `Environment` (fig. 1). The `EquicryptSystem` is composed of four main processes, modelling its three main components (TTP, SPvs and STUs) and the common broadcast channel.

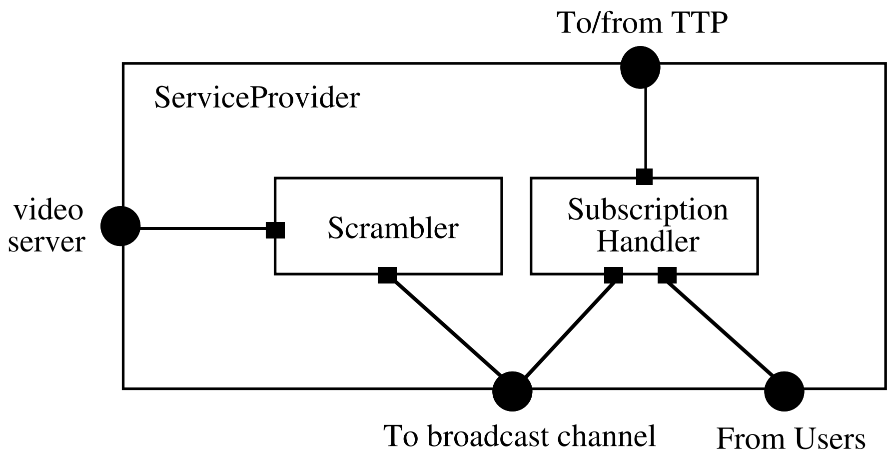

Fig. 2: The structure of the Service Provider

The service provider is split into a process that handles control words and scrambles images, and a process handling the subscriptions (fig. 2).

The set top unit contains a descrambler and the ACU, itself decomposed into a process handling registration, one handling subscription to new services and one decoding control words for subscribed services (fig. 3). New service keys are passed through gate `skey`, new subscriptions are notified to the descrambler via `nsvc` and `dcw` is used by the descrambler to ask for decoding of newly received control words.

Finally, the trusted third party is split into two processes dealing resp. with users and providers (fig. 5). Information about registered users is passed through gate `tup` for consultation by providers.

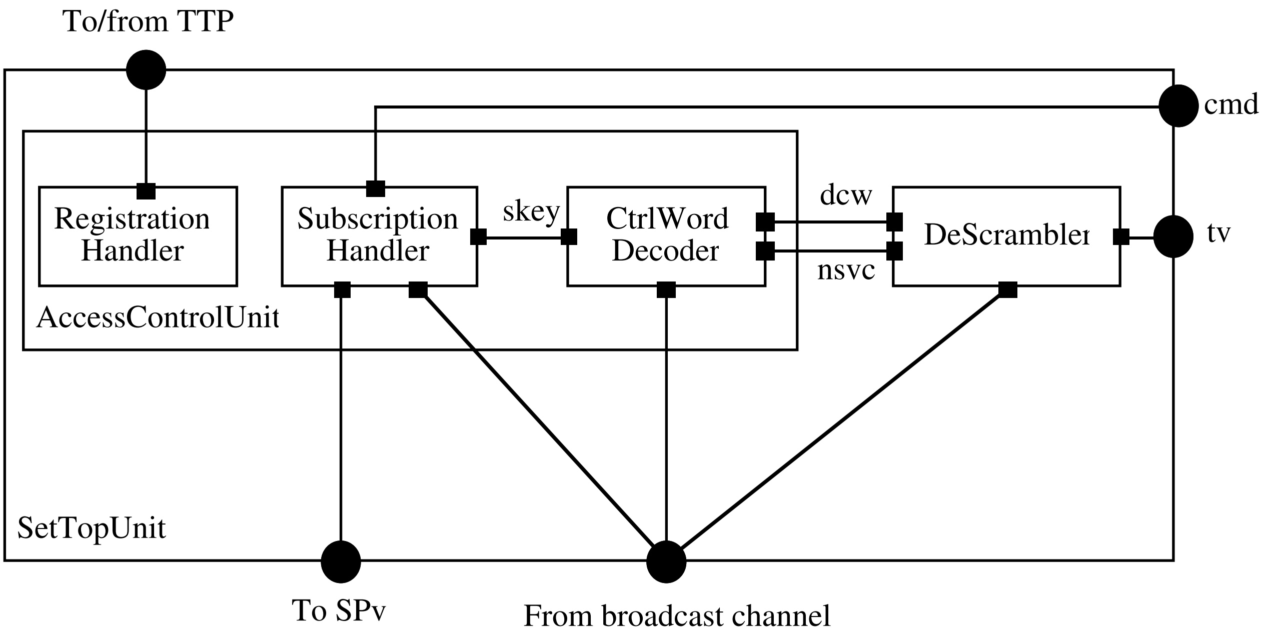

Fig. 3: The structure of the Set Top Unit of the User

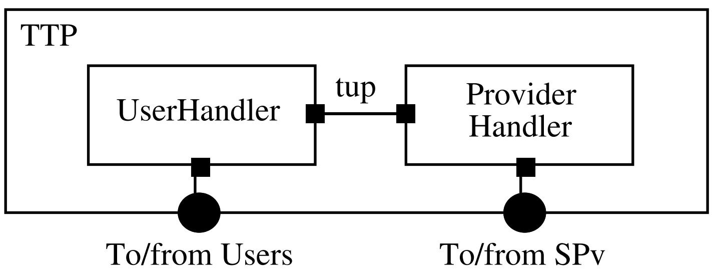

Fig. 4: The structure of the TTP

3.2. Data types

This specification has been written using data type language extensions, as offered by the APERO tools [Pec96] included in the Eucalyptus toolbox [Gar96]. The original text has to be processed by the APERO translator to get a valid LOTOS specification. This provides for a smaller and more readable specification and for some level of immunity w.r.t. underlying processing tools. Some types have been written from scratch, though, hence it was necessary to take tool restrictions explicitly into account.

The data types are classified in 4 broad categories:
- *Base values*: identifiers, keys, transmitted data, etc., described as explicit enumerations.
- *Encryption and decryption*: functions for each sort of encrypted value. These are modelled as abstract operations that are the reverse of each other. This allowed us not to model the actual algorithms such as RSA [RSA78]. Decryption with a bad key is handled explicitly and produces a distinguished value `xyzJunk` for each sort `xyz`:

```
sorts Msg, EMsg, EK, DK
opns  Match : EK, DK -> Bool   (* Define a binary predicate Match pairing those keys *)
      E : EK, Msg -> EMsg      (* Define operations E and D for encryption/decryption *)
      D : DK, EMsg -> Msg      (*  of some sort Msg with those keys *)
eqns forall m : Msg, ek : EK, dk : DK
ofsort  Msg
      Match(ek, dk) => D(dk, E(ek, m)) = m ;
      Not Match(ek, dk) => D(dk, E(ek, m)) = msgJunk;
```

- *Interaction primitives*: one APERO `recordtype` for each primitive. Gate multiplexing is used to model several similar communication channels as a single gate.
- *Tables*: needed for storing registered information in several processes, and defined using the APERO `tabletype` extension.

3.3. Preliminary assessment of the specification

To gain confidence into the specification, it has been simulated with the XEludo tool [STS94] from the Eucalyptus toolbox in step-by-step non-symbolic execution mode. All the entities were present, in particular two service providers and two set top units. The broadcast channel was modelled as an unbounded queue. Several early bugs were detected. The simulation was judged satisfactory, when it was possible to subscribe the two users simultaneously to both services and the images were descrambled correctly.

4. Safety properties to be verified

- P_1: Authentication of the user by the TTP during registration: when the TTP registers a user A, this user A must have started a registration procedure with the TTP.
- P_2: Authentication of the TTP by the user during registration: when the user A believes it is registered by the TTP, the TTP must have started a registration procedure with this user A.
- P_3: Authentication of the service provider by the user during subscription: when the user A believes it has successfully subscribed to a service provider B, the service provider B must have started a subscription procedure with A' (A' being the alias of A).
- P_4: Authentication of the user by the service provider during subscription: when the service provider B accepts the subscription of A', the user A (that has alias A') must have started a subscription procedure with B.
- P_5: Authentication of the TTP by the service provider: when the service provider B accepts the subscription of A', the TTP must have given guarantees regarding the registration of user A (that has alias A').
- P_6: When a user successfully subscribes, it must have successfully registered.

For properties P_1 to P_5, the dual properties P_1' to P_5' should also hold. We give P_1' as an example:

P_1': When the TTP decides *not* to register a user A, this user A must have started a registration procedure with the TTP.

Finally, we have no authentication property of the service provider B by the TTP, because there is no need for such an authentication, since the key requested by B is public.

5. Verification of the protocol

5.1. Model of an intruder

We want to model a 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 her own public key, and store them;
- Introduce fake messages in the system (a fake message can be an old message which is replayed or a new message built up from components of old messages, including components that she cannot decrypt).

The intruder behaves in such a way that neither the receiver of a fake message, nor the sender of an intercepted message can notice the intrusion. In fact the intruder merely replaces the channel linking users to providers in the model.

In this paper, the intruder will only act on the insecure channel between the users and the providers, because the proposed Equicrypt protocol relies heavily on the hypothesis that the communication channels with the TTP are secure. A more advanced design that does not rely on such assumptions will be verified at a later stage with an intruder that can act on all communication channels.

The intruder is parameterized with some initial knowledge, which should be realistic and give her enough power to act as a user when she communicates with a service provider, and act as a provider when she communicates with a user. In particular, we do *not* exclude the following two cases: (1) the intruder is a certified user, or (2) is (known to users as) a service provider. However, if the intruder is a certified user, we still consider that her ACU behaves as a certified ACU. For example, the ACU cannot disseminate a stored service key to other users. Therefore the initial knowledge of the intruder is as follows:

I, I':	The identification and alias of the intruder.
K_i^p, K_i^s:	The public and private (secret) keys of I.
N_{ia}, N_{ib}:	Nonces that I can use in fake messages sent to A and B.
C_i:	The certificate of I.
K_a^p, K_b^p:	The public keys of user A and service provider B.

During a protocol run, the intruder can increase her knowledge base. This is modelled by additional parameters such as sets of (parts of) encrypted messages that she has been unable to decrypt, and sets of (parts of) plaintext messages that she has been able to decrypt (or were sent in plaintext).

By having a single nonce to talk to A and a single nonce to talk to B, our intruder can only take part in a single run of the protocol. Therefore, any form of attack that relies on several attempts cannot be exhibited by this intruder.

Furthermore, we assume that our intruder cannot break the public key cryptosystem by getting the message in clear from the encrypted message and the public encryption key, or forging a signed message from the message in clear and the public decryption key. In our model, this would mean for example guessing m from $\{m\}_K$ and K. Note that LOTOS easily provides processes that transgress this rule, and thus break any cryptosystem:

```
process GuessMsg (emsg:EMsg, ek:EK) : exit(Msg) :=
      choice msg:Msg [] [E(ek, msg) = emsg] -> exit(msg)
endproc
```

When building, for verification purposes, an intruder process that tries and breaks the access control mechanisms, care must be taken to avoid these kinds of unrealistic behaviours. In practice one can interpret them as trying all possible encodings until the correct one is found, which is precisely the computationally unfeasible operation on which security relies.

5.2. Formalizing the properties

To model the properties presented in section 4, it is necessary to observe some states of all the entities (user, provider and TTP). To achieve this, we will add special events to be executed in those states. Here are some of them used in the subscription procedure. Others are used in the registration procedure:

`U_start_sub!A!B!S`	The user *A* starts a subscription procedure with the service provider *B* requesting service *S*. This event is executed by the user *A* just before it sends message 4 to *B*.
`U_sub!A!B!S`	The user *A* believes it has successfully subscribed to service *S* provided by *B*. This event is executed by the user *A* just after it has received a positive message 7 from *B* with the expected nonce.
`U_Rsub!A!B!S`	The user *A* believes it has not successfully subscribed to service *S* provided by *B*. This event is executed by the user *A* just after it has received a negative message 7 from *B*, or a positive message 7 with a wrong nonce.
`P_start_sub!A'!B!S`	The service provider *B* has started a subscription procedure with *A′* requesting service *S*. This event is executed by the service provider *B* just after it has received message 4 from *A′*.
`P_sub!A'!B!S`	The service provider *B* accepts the subscription of *A′* regarding service *S*. This event is executed by the service provider *B* just before it sends a positive message 7 to *A′*.
`P_Rsub!A'!B!S`	The service provider *B* refuses the subscription of *A′* regarding service *S*. This event is executed by the service provider *B* just before it sends a negative message 7 to *A′*.
`T_sub!A!T`	The TTP *T* gives registration guarantees for user *A*. This event is executed by the TTP *T* just before it sends a positive message 6'.
`T_Rsub!A!T`	The TTP *T* does not give registration guarantees for user *A*. This event is executed by the TTP *T* just before it sends a negative message 6'.

The properties can now be expressed solely in terms of these events. We give properties P_3 and P_4 as examples below:

P_3: `U_sub!A!B!S` must be preceded by `P_start_sub!A'!B!S`, where *A′* is the alias of user *A*.
P_4: `P_sub!A'!B!S` must be preceded by `U_start_sub!A!B!S`, where *A′* is the alias of user *A*.

5.3 The verification

We have used the CADP package [FGK+96] included in the Eucalyptus toolbox to carry out the verification. The first step consists of using the Cæsar tool to generate a graph called Labelled Transition System (LTS) from the LOTOS specification. To be able to generate a *finite-state* LTS of *reasonable size*, some simplifications were required, such as the addition of an environment, the bounded capacity of the broadcast channel and some reduction of nondeterminism. Several variants of this simplified specification with an intruder have been written. For example: a specification with two service providers offering distinct and identical services. Let us consider one such configuration and call it `spec`.

The second step consists of using the Aldebaran tool to minimize the resulting graph. The properties being all safety properties, the minimization was done modulo the safety equivalence [BFG+91]. The minimized LTS still has several thousands of states and transitions. It is called `spec.safety`.

Not all the observable actions are relevant to verify the properties. In particular, our properties only rely on some special actions that have been described in section 5.2. Every property was separately modelled as a reference LTS constructed with these special actions (e.g. the LTS modelling property P_i is called `Pi.safety`), and the `spec.safety` LTS was checked against such a property by verifying the safety preorder relation between `spec.safety` and `Pi.safety`, while hiding irrelevant actions. Formally, the safety preorder is the preorder that generates the safety equivalence, and is nothing else than the weak simulation preorder (i.e. a one-way weak bisimulation). Therefore, this verification ensures that the behaviour exhibited by the specification is allowed (i.e. can be simulated) by the safety property.

When a property was not verified, Aldebaran produced a diagnostic sequence. 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 (i.e. with more visible actions) that can clearly identify the scenario of the intruder's attack, we have encoded this abstract diagnostic sequence in a format suitable for input to the Exhibitor tool, which was then instructed to find the detailed sequence (allowed by the specification) matching the abstract one.

5.4 Results of the verification

The properties P_3 and P_4, namely the mutual authentication of the user and the provider during subscription are not verified in the presence of our intruder. The following two scenarios translated from those provided by Exhibitor will illustrate why these properties are falsified. The reason why the other properties are still fulfilled is because, at this stage, the intruder was not designed to interfere on the secure communication channels connecting the TTP to the users and service providers.

First scenario: The intruder does not need to be registered; she just needs to know the public keys of two service providers B and C. We consider a user A subscribing successfully to service S provided by server B. This subscription runs *implicitly* in parallel with the attack described below. Now suppose that another service provider C also offers this service S, and suppose an intruder is aware of that (for example, the intruder might be C itself, or C's accomplice). The story goes as follows: the intruder can copy the subscription request sent by A to B, namely the message:

4: ? $\rightarrow B$: Subscribe request $\langle A', S, \{N_{ab}\}_{K_b^p}\rangle$

Then, the intruder sends a fake message to service provider C, which is basically the above message where the nonce has been changed and encrypted with C's public key:

4: ? $\rightarrow$ C: Subscribe request $\langle A', S, \{N_{ic}\}_{K_c^p}\rangle$

C then gets N_{ic} and starts its subscription procedure with A', namely it executes `P_start_sub!A'!C!S`, and exchanges messages 5 and 6^+ with the TTP. This successful exchange leads C to commit on a subscription with A': C executes `P_sub!A'!C!S`. Then C sends message:

7: $C \rightarrow$ All: Subscribe answer $\langle S, \text{true}, \{N_{ic}, K_S\}_{K_a^p}\rangle$

I intercepts this message and sends a fake subscribe acknowledgement *containing A's signature* to C, which is a copy of the corresponding subscribe acknowledgement that A sent earlier to B, namely:

8: ? $\rightarrow$ C: Subscribe ack $\langle\{A', S\}_{K_a^s}\rangle$

Clearly property P_4 is not fulfilled, because event `P_sub!A'!C!S` is not preceded by the event `U_start_sub!A!C!S`.The consequence is that the service provider C can claim money to A for the subscription, because it possesses a signed subscribe acknowledgement from A for this service S. A way to make this attack impossible consists of encrypting the service identifier S in message 4. Note that the above scenario only makes sense if two service providers can provide the same service S; in other words, if S is a service type rather than a service identifier. Even if this hypothesis is not fulfilled, it would be wise to encrypt the service identifier S in message 4 (and possibly 7), because there are other attacks (not described here) that rely on its non protection.

Second scenario: The intruder does not need to be registered either, but needs to be (known to A as) a service provider. The story goes as follows: A starts a subscription procedure with the intruder known as a service provider, namely it executes `U_start_sub!A!I!S`. Then A sends the message:

4: ? $\rightarrow$ I: Subscribe request $\langle A', S, \{N_{ai}\}_{K_i^p}\rangle$

I decrypts N_{ai} and sends the fake message:

4: ? $\rightarrow$ B: Subscribe request $\langle A', S, \{N_{ai}\}_{K_b^p}\rangle$

B then gets N_{ai} and starts its subscription procedure with A': namely it executes `P_start_sub!A'!B!S`, and exchanges messages 5 and 6^+ with the TTP. This successful exchange leads B to commit on a subscription with A': B executes `P_sub!A'!B!S`. Then B sends message:

7: $B \rightarrow$ All: Subscribe answer $\langle S, \text{true}, \{N_{ai}, K_S\}_{K_a^p}\rangle$

I intercepts it, cannot decrypt it, but replays it to A:

7: $I \rightarrow$ All: Subscribe answer $\langle S, \text{true}, \{N_{ai}, K_S\}_{K_a^p}\rangle$

Finally A decrypts this messages, finds its nonce N_{ai} and therefore commits to subscription by executing `U_sub!A!I!S` and sends the subscription acknowledgement with its signature to I:

8: ? $\rightarrow$ I: Subscribe ack $\langle\{A', S\}_{K_a^s}\rangle$

Clearly the properties P_3 and P_4 are not fulfilled, because event `P_sub!A'!B!S` is not preceded by the event `U_start_sub!A!B!S`, and `U_sub!A!I!S` is not preceded by the event `P_start_sub!A'!I!S`. The consequence is that the intruder can claim the money without giving the service. On the other hand, *B* will not be able to do so. This attack resembles the classical Bucket-Brigade attack, but is more subtle, since it is only made possible when the intruder is (known to *A* as) a service provider. A way to make this attack impossible consists of adding the originator of message 7 in the encrypted part.

6. Conclusion and future work

From the above errors, we can conclude that the problems of the Equicrypt protocols are as follows:

1. When *B* sends the Subscribe answer, *B* only knows that A' is registered in the TTP (and thus certified), but is not sure that A' is the originator of the Subscribe request.

2. When *A* receives the Subscribe answer, one could think *A* has authenticated *B* by the use of the nonce N_{ab}, but this is not correct. This error is similar to an error found recently in the Needham-Schroeder authentication algorithm [Low95].

A revised design of the Equicrypt protocol will be proposed along the lines sketched in this paper. Also, the unrealistic assumption about the security of the channels used to communicate with the TTP will be removed and the protocol made robust to attacks on these channels by an extended intruder.

References

[BFG+91] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez and J. Sifakis. Safety for Branching Time Semantics. *In: 18th ICALP*, Berlin, July 1991. Springer-Verlag.

[BoB 87] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems* 14 (1) 25-59 (1987).

[FGK+96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier and M. Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. *In: R. Alur and T. Henzinger, eds, Proc. of the 8th Conference on Computer-Aided Verification* (New Brunswick, New Jersey, USA), Aug. 1996.

[Gar96] H. Garavel. An overview of the Eucalyptus Toolbox. *In: Proc. of the COST247 workshop* (Maribor, Slovenia), June 1996.

[GBM96] J. Guimaraes, J.-M. Boucqueau and B. Macq. OKAPI: a Kernel for Access Control to Multimedia Services based on Trusted Third Parties. *In: Proc. of ECMAST 96 – European Conference on Multimedia Applications, Services and Techniques* (Louvain-la-Neuve, Belgium), pp. 783-798, May 1996.

[ISO 8807] 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.

[LBQ+96] S. Lacroix, J.-M. Boucqueau, J.-J. Quisquater and B. Macq. Providing Equitable Conditional Access by Use of Trusted Third Parties. *In: Proc. of ECMAST 96 – European Conference on Multimedia Applications, Services and Techniques* (Louvain-la-Neuve, Belgium), pp. 763-782, May 1996.

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol, *Information Processing Letters*, 56:131-133, 1995.

[Pec 96] C. Pecheur. Improving the Specification of Data Types in LOTOS. *Doctoral dissertation, University of Liège, July 1996*. To appear.

[RSA78] R. Rivest, A. Shamir and L. Adleman. On a Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communication of the ACM*, vol. 21, pp. 120-126, Feb. 1978.

[STS94] B. Stepien, J. Tourrilhes and J. Sincennes. ELUDO: The University of Ottawa Toolkit. *Technical Report, University of Ottawa, 1994*. Obtainable by FTP at `lotos.csi.uottawa.ca`.