Analyzing Cryptographic Protocols in a Reactive Framework

R.K. Shyamasundar

School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India
shyam@tcs.tifr.res.in

Abstract. In this paper, we analyze the suitability of reactive frameworks for modelling and verification of cryptographic protocols. Our study shows that cryptographic protocols can be modelled easily and naturally including the communication feature of the Internet wherein a point-to-point communication could be interpreted as broadcast mechanism due to the underlying routing and LAN architectures. The reactive framework provides an effective modelling of attacks/intruders as well as the capturing of the security properties such as secrecy and authenticity as *observers.* The *observer-based* approach of synchronous reactive frameworks aids in the modelling of properties incrementally and the use of the simulate-compile-verify cycle of the synchronous programming environment. The anomalies that could arise due to possible concurrent runs of agents can be detected. For illustration purposes, we use the TMN protocol. We will also argue that the reactive frameworks also provide a basis for specifying cryptographic protocols.

1 Introduction

Public key encryptions are commonly used for secure communications over public computer networks. Development of e-commerce over the Internet has lead to a wide spectrum of secure protocols, called cryptographic protocols, that use cryptographic primitives for transactions. In these protocols, it is very important to ensure *security* as interactions take place over public networks that are basically insecure. In such protocols, it is very crucial to ensure:

- Messages meant of an agent cannot be read/accessed by others (*secrecy*).
- Guarantee genuineness of the sender of the message. (*authenticity*)

On any insecure network, there will be active attacks (attackers themselves are bona-fide users of the network!) to get information of other bona-fide agents by eavesdropping/impersonating other valid agents. Establishing secrecy and authenticity have two issues:

1. The possibility of breaking the encrypted message, and
2. Ensuring secrecy and authenticity of messages.

From the perspective of cryptographic protocol verification, it is the second aspect one is concerned with assuming that the encryption cannot be broken without the underlying secret key. The problem can be stated as follows:

> *Assuming the non-crackability of cryptographic primitives, establish that the protocol maintains secrecy and authenticity of transactions.*

In the literature, techniques of establishing the correctness of classical distributed protocols have been adapted for establishing the secrecy/authenticity of the cryptographic protocols. Approaches can be broadly categorized as follows: (1) State Machine Models [11, 14], (2) π-Calculus Based Models [1], (3) Methods Based on Belief Logics [5], and (4) Theorem Prover based Methods [8, 3, 16, 4, 19, 9]. [17] provides an annotated bibliography.

In the area provably correct systems, there have been quite considerable efforts in the specification and verification of reactive systems [7]. The family of synchronous languages [2] has been well studied for the design and synthesis of provably correct reactive systems. Synchronous languages also support a powerful programming environment including verification. In this paper, we show that the techniques of modelling and verifying reactive systems in synchronous languages such as ESTEREL can be effectively used for verifying the correctness of cryptographic protocols. As ESTEREL has a precise formal semantics and powerful tool based techniques for verification, it is possible to model the protocol and the attacker cleanly and establish the security properties using its' simulate-compile-verify programming environment. The environment also permits us to find if flaw exists if concurrent runs of the protocol are allowed for the principals. Our reactive approach seems to satisfy some of the goals set by approaches of defining simples specification languages exclusively for cryptographic protocols [10]. Furthermore, the availability of *preemption* in ESTEREL allows the modelling of complex e-commerce protocols that use preemptive features at various points. Throughtout our presentation, we try to convey the underlying principles and intuition for the sake of simplicity and brevity.

2 Analysis of Cryptographic Protocols

Analysis of the cryptographic protocols is usually based on

1. *The model of the intruder*: The most widely used classical model of the intruder is that described in Dolev and Yao [6] where:
   (a) Intruder can obtain/decompose any message passing via the network.
   (b) Intruder can remember/insert messages using the data seen.
   (c) As the intruder is a legitimate user of the network, and can initiate a conversation with any user.
   (d) Given the key, the intruder can encrypt/decrypt a message.
   (e) Intruder *cannot* get partial information, guess the key or perform statistical tests, and
   (f) Encrypted message can neither be altered nor read without the key.

2. *Characteristics of Protocols*: The main characteristics are:
   (a) Roles of principals are finite.
   (b) Computational steps of each principal are finite.
   (c) All the messages are bounded in size.
   (d) Nonces (new messages) can be created.
   (e) Except for the Nonces all other entities are constants.

Informally speaking, the protocol is said to be correct under the above general model of the intruder and the characteristics of the protocol, if it is not possible for an intruder, say C, to get the key of some principal, say A, fraudulently while A is trying to transact with another principal, say B.

2.1 Issues in the Modelling of the Protocol and the Intruder

Protocols in general are operational descriptions of actions and reactions (responses). In general, they follow the principle that for every set (or sequence) of actions a finite set of reactions is expected. The general characteristics that needs to be catered/adhered to are:

1. Possible actions/roles that can be taken by the principals; while the roles of the intruder may be nondeterministic fashion, the reactions are deterministic.
2. Properties of communication medium and the Cryptographic primitives.
3. The operational structure of the protocol.
4. Protocol properties other than that are related to security.
5. Capabilities of the Attacker: eavesdropping over the network, sending fraudulent/forged messages, interception of messages etc. These capabilities are quite general and are not dependent on the protocol.
6. The general principles adhered to by the *Server* (like never re-issuing session keys or agents their Nonces).
7. Thus, the attacker is also a valid principal in the system and the intruder could be doing his genuine role as well as that of an attacker concurrently.
8. Security properties such as secrecy, authenticity as intended.

3 Modelling Cryptographic Protocols in Esterel

In this section, we discuss modelling cryptographic protocols using Esterel and use use the TMN protocol [18] studied in detail in [12] for illustrations. First, we shall consider the basic protocol to illustrate the modelling and subsequently, we shall discuss possible attacks on the protocol and the improvements needed to make the protocol *secure*.

Note 1. For illustrative purpose, we shall be using Esterel to model cryptographic protocols with a view to model and verify relative to various types of intruders. Note that the cryptographic protocols do require the manipulation of cryptographic primitives. However, in synchronous languages the general programming language features are usually borrowed from those existing in the host language such as C or java in which they are embedded. In our illustrations of modelling in ESTEREL, we abuse the usual notation of procedure calls and parameter transmissions for the sake of simplicity and succinctness. As the notation is fairly standard, no confusion should result. Such an interface for ESTEREL for cryptographic protocol analysis purposes is being built.

3.1 Basic TMN Protocol

TMN protocol [18] consists of three players: an *initiator* A, a *responder* B, and a *server* S who mediates between them for the generation of session keys. The protocol employs two methods of encryption:
a. Standard encryption: The function denoted E when applied to a message m produces an encrypted text that can be decrypted with a secret key. Further, the encrypted message can be decrypted only by the server. A typical example of such a method is the RSA encryption.
b. Vernam Encryption: This is written $V(k_1, k_2)$ and gives the bit-wise exclusive-or. It is be noted that $V(k_1, V(k_1, k_2)) = k_2$ and $V(k_2, V(k_1, k_2)) = k_1$.
Protocol: The protocol in the notation of [12] is given below:

`Message 1:` $A \rightarrow S$: $A.S.B.E(K_a)$
`Message 2:` $S \rightarrow B$: $S.B.A$
`Message 3:` $B \rightarrow S$: $B.S.A.E(K_b)$
`Message 4:` $S \rightarrow A$: $S.A.B.V(K_a,\ K_b)$

The simple model of the protocol corresponding to the above description is shown in Figure 1. We have used a bus rather than explicit channels as that resembles the model of the Internet structure (which is essentially a store forward network). The numbers over the messages shown in complete lines in Figure 1, correspond to the message numbers given above. The dotted lines with numbers with an apostrophe show the dual of picking up the message (corresponding to receipt).

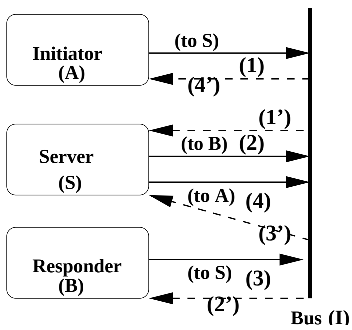

Fig. 1. Structure of the TMN Protocol

Note 2. We use $\mathrm{E}(K_a)$ and $\mathrm{V}(K_a, K_b)$ with the arguments explicitly even though only the respective values are seen rather than the components; explicit syntax helps us to keep track of different modules and parameter bindings.

3.2 Modelling the Basic TMN Protocol in Esterel

The model of communication used in ESTEREL is the broadcast mechanism; that is, whoever is ready to see /listen to a message can see/will get it at that point. Such a feature becomes handy in the modelling of the Internet which is a essentially a Store-Forward packet network (where message packets at any router can be seen by everyone who are on the same bus or access to it). This feature avoids the need of artificial channels for modelling communication as required in models based on CSP or CCS. Further, the orthogonality of concurrency and communication in ESTEREL makes it possible to model concurrent roles simultaneously. Specification of principals in ESTEREL taking the role of an initiator, responder and server respectively are shown in Figures 2, 3, and 4 respectively. The specification has been parameterized with respect to the behaviour of the principal. The labels on the statements are used for purposes of reference only. The overall TMN protocol with the above three agents is shown in Figure 5.

Note 3. The message structure is treated as a string from which the components can be extracted[1] by some function say, *Extract* (not shown).

Discussion: The three principals A,B and S are part of the TMN protocol. Each agent is doing some other computation other than that of participating in the protocol as required for transacting with other agents. Note that this a nonterminating behaviour. In the following, we shall informally discuss the actions performed by A as an initiator, B as a responder and S as a server.
Initiator: Labels $\ell_{i_0}, ...\ell_{i_3}$ correspond to actions with the interpretation: (i) await for some signal to initiate a session with a responder, (ii) make known to the server the intention to have a session with a responder with the underlying structure for the message packet, (iii) wait for some finite time for the server (which is indicated by the signal `tick`, and (iv) receive the response from the server as and when it sends the session key. Since, the underlying model assumes that the principals know how to extract the key given a priori known message structure, it follows easily that the initiator (in this case agent A) will get the key of the responder with whom it wanted to have a transaction (in this case agent B) using the simplification rules given for Vernam encryption earlier.
Responder: Labels $\ell_{r_0}, ...\ell_{r_3}$ correspond to actions with the interpretation: (i) await for request from the server for a response, (ii) waiting for some finite time before answering, and (iii) sending the requested information to the server.
Server: Labels $\ell_{s_0}, ...\ell_{s_3}$ correspond to actions having the interpretation: (i) awaiting for an initiator, (ii) waiting for sometime before sending the request to the responder, (iii) sending the request to the responder, (iv) awaiting a

[1] It must be noted, that in ESTEREL all such aspects are done as per the host language which can be C, Java etc.

```
module Initiator;
   input start; I_s: some_string_type;
   output I_a : some_string_type;
   var K_a: key_bit_type;
       A: initiator_type; B: responder_type; S: server_type;
   loop
       ℓ_{i_0}: await start;
       ℓ_{i_1}: emit I_a(A,S,B,E(K_a));
                   %Structure:(sender.via,destination,encrypted_msg)%
       ℓ_{i_2}: await tick;
       ℓ_{i_3}: await I_s(S,A,B,V(K_a,K_b))
success:
   endloop
end module
```

Fig. 2. Model of Initiator

```
module Responder;
   input I_s: some_string_type;
   output I_b : some_string_type;
   var K_b: key_type;
       A: initiator_type; B: responder_type; S: server_type;
   loop   ℓ_{r_0}: await immediate I_s(S,B,A);
          ℓ_{r_1}: await tick;
          ℓ_{r_2}: emit I_b(B,S,A,E(K_b);
   endloop
end module
```

Fig. 3. Model of Responder

```
module Server:
   input I_a, I_b: some_string_type;
   output I_s: some_string_type;
   var A: initiator_type; B: responder_type; S: server_type;
       key_initiator, key_receiver: key_type;
   loop ℓ_{s_0}: await immediate I_a(A,S,B,E(K_a);
        A:= first(I_a(A,S,B,E(K_a)); B:= third(I_a(A,S,B,E(K_a));
        key_initiator:= getkey (I_a(A,S,B,E(K_a));
        ℓ_{s_1}: await tick;
        ℓ_{s_2}: emit I_s(S,B,A);
        ℓ_{s_3}: await I_b(B,S,A,E(K_B));
        key_receiver:=getkey(I_r(B,S,A,E(K_b));
        ℓ_{s_4}: await tick;
        ℓ_{s_5}: emit I_s(S,A,B, V(K_a,K_b))
   endloop
end module
```

Fig. 4. Model of Server

```
module TMN;
  input start;
  output I_a, I_b, I_s: some_string_type;
  run initiator [I_a/I_a, I_s/I_s, K_a/K_a, A/A,B/B,S/S]
||
  run responder [I_b/I_b, I_s/I_s, K_b/K_b,A/A,B/B,S/S]
||
  run server;
end module
```

Fig. 5. Model of the TMN Protocol with Three Principals

response from the server - allowing it sometime for responding, (v) again awaiting before sending the information to the initiator and (vi) sending the required key information in the agreed structure to the initiator.

3.3 Characteristic Features of the Model

Capturing the roles and capabilities of the agents:

The model clearly articulates, the roles of agents, message packets, possible concurrent roles etc. To be more specific:

- The capabilities of the principals and the protocol followed by them are explicit.
- In the protocol, we have shown A to be performing the role of an initiator, B the role of a responder and S the role of a server. From the parameterized specifications, it is easy to see how the role of the principal can be changed and also several (finite) roles can be conceived concurrently.
- Further, I_a, I_b, I_s can be seen as packets that can be seen by the principals while being alert in the scope. This is one of the ways of abstracting the model of the public network with routers.

Verifying the proof obligations for message passing

- An initiator will get the key sent by the responder via the server given that responder and server react in finite time.
- Ignoring functional rewritings of the equation, the question corresponds to: $at(\ell_1) \leadsto after(\ell_{i_3})$
  or in terms of messages I_a(A,S,B,E(K_a)) $\leadsto$ I_s(S,A,B,V(K_a, K_b))
  where $at(p)$ $after(p)$ are the usual control predicates that indicate whether the control point is currently "at p" or has crossed "p" respectively.
- The inferences follow naturally from the automata constructed using the `Xeve` tool[2] for the above program and showing that the constructed automata is symbolically bisimilar to the automata corresponding to the trace $[I_a(A, S, B, E(K_a)) \quad I_s(S, A, B, V(K_a, K_b))]^*$ where "*" denotes the usual Kleene-star operation denoting finite a priori unbounded repetitions of the trace. Other implicit relations of the actions of the principals A,B and S can also be captured by similar trace structures.

3.4 Analyzing Security properties

In this section, we shall discuss the analysis of security properties.

First, we generalize the model with several principals as shown in Figure 6 having m initiators and n responders. Note that the correspondence of initiators and responders need not be one-to-one physically.

[2] A proof tool based on symbolic bisimulation available for synchronous frameworks.

```
module TMN-many;
  input start: boolean;
  output I_{a_1}, ··· , I_{a_m}, I_{b_1}, ··· , I_{b_n}, I_s: some_string_type;
  run initiator [I_{a_1}/I_a, I_s/I_s, K_{a_1}/K_a, A_1/A, B_{(1)}/B,S/S]
                ...
|| run initiator [I_{a_n}/I_a, I_s/I_s, K_{a_n}/K_a, A_n/A, B_{(m)}/B,S/S]
|| run responder [I_{b_1}/I_b, I_s/I_s, K_{b_1}/K_b, A_{(1)}/A, B_1/B,S/S]
|| ...
|| run responder [I_{b_m}/I_b, I_s/I_s, K_{b_m}/K_b,A_{(n)}/A,B_m/B, S/S]
|| run server;
```

Fig. 6. Model of the TMN Protocol with Several Principals

Security properties via Traces: Let A_i be the initiator and let B_j be the corresponding responder; we assume A_i is not the same as B_j. We say that *the protocol is secure under the assumption of non-crackability of the cryptographic primitive if the key, K_{b_j} sent by B_j to A_i via the server occurs only in the form $V(K_{a_i}, K_{b_j})$.* That is, for the latest request from A_i to B_j, there is no other message of the form $V(K_{a_r}, K_{b_j})$ for some $r \neq i$ message in the system. Now, let us see whether the protocol discussed above satisfies this property or a counterexample or an attack can be found. In the module structure shown in Figure 6, it can be seen that the module corresponding to the structure of the initiator or the responder for that instantiation can output only on that named output signal. That means, in the structure shown it is not possible to break security for the following reasons:

- Esterel is a lexically scoped typed language. Hence, unless the access to signals is explicitly given or derived under the scoping rules, the modules will not be able to access the signals. In the context shown, we have shown I_a being accessible only by A and the server, and I_b being accessible only by B and the server. In view of the strong typing, no intruder (which in this case has to be other than A or B) will be able send signals mimicking as other agents. In other words, the typing rules ensure that accessing messages not meant for it are out of bound for others.
- Considered in another way, the typing rules ensure the *authenticity* of the messages (i.e., the messages should have been sent by such and such a principal) that was perhaps implicit in the original protocol.

To model attacks wherein the intruders can see messages, create new messages etc., we need to derive a weaker structure than that shown in Figure 6. This is done by permitting each principal to access and broadcast on signals that are not necessarily meant for it. This is achieved by placing signals corresponding to the various initiators and responders in the lexical scope of all initiators and responders (or all valid principals). In this model, one cannot guarantee authenticity as will be seen later.

Note 4. It must be noted that ESTEREL permits concurrent incarnations of a module. For this reason, one could define explicit rules of *combination* when two signals are emitted concurrently.

4 Modelling Intruders, Attacks, and Analysis

The main capabilities of the intruder are:

1. Obtain any message, store them, and reinsert them at some point.
2. Decompose the plain text into different components, modify the components (no breaking is allowed) and insert the messages in the system.

Intruders following the above rules can be obtained by looking at the messages on the bus. Consider the modelling of an intruder corresponding to the messages emitted by the initiator shown in Figure 2. It may be noted that the initiator shown in Figure 2 emits only one message I_a(A,S,B,E(K_a)). As the server is unique, the messages constructed will have the server name unaltered as well as its position unchanged. The possible messages that can be generated or emitted with the capabilities as mentioned above on the message that is normally emitted by an initiator is given by (here, principal C is used to denote an intruder):

I_a(A,S,B,E(K_a)) -- corresponds to emitting a message stored earlier;
I_a(A,S,B,E(K_c)) -- Alter the key to that of agent C (intruder)
I_a(A,S,C,E(K_a)) -- Responder is altered to C.
I_a(A,S,C,E(K_c)) -- Responder as well as the key is changed to that of C
I_a(C,S,B,E(K_c)) -- C becomes initiator

The other possible messages will not play any role; in fact, the third message shown above also does not play any useful role as the intruder will not be able to derive any useful information as it is its' (intruder) own key that is being passed. Interception (in a sense blocking!) is modelled by an "abort" statement and a signal "intercept" that is a local input signal to the intruder. Thus, when I_a and *intercept* are simultaneous, I_a would be intercepted. We have not shown the possibility of it being stored just for simplicity. An intruder, C, that alters the messages read from A as above, is given in Figure 7. In the figure, the signals, "zero, one, two, three, and four" are input signals that have been used to model input nondeterminacy corresponding to the possibilities of emitting messages mentioned above.

On similar lines, one can construct intruders corresponding to the possible messages that can be seen *emitted* by the responder and server. *A parallel composition of all such intruders will be the actual intruder for the system*:

$||_i \; crude - init \; ||_j \; crude - respond \; ||_k \; crude - server$

will be the general intruder where i, j, and k denote the quantification over all messages that can be emitted by the initiator, responder and server respectively.

In the following, we shall consider some structured intruders that would illustrate the various attacks on the basic protocol described previously. It must be noted that the behaviours of these structured intruders will be contained in the behaviour of the general intruder constructed as above. These aspects will be

```
module Crude-init;
  input zero, one, two, three, four, intercept;
  loop
    abort
     await I_a(C,S,B,E(K_c));
     await immediate
      case zero do I_a(A,S,B,E(K_a));
      case one do emit I_a(A,S,B,E(K_c));
      case two do emit I_a(A,S,C,E(K_a)) ;
      case three do emit I_a(A,S,C,E(K_c));
      case four do emit I_a(C,S,B,E(K_c));
   when intercept;
  endloop
end module Crude-init
```

Fig. 7. Crude Intruder of Initiator

discussed in the full paper. As concurrent incarnations of modules are possible, we can model finite concurrent roles.

Since the intruders are assumed to be valid principals of the system, without loss of any generality, we can assume that messages sent will have the well-defined syntactic structure. The attackers that mimic the roles of initiator, responder or the server of sessions can be obtained by decomposing the possible messages that could possibly be emitted and performing operations as detailed above. We shall illustrate the possibilities of attacks seen from the initiator module, the initiating agent being called A:

1. replacing the key of the initiator by its' own key which is realized by:

$$C_A \equiv initiator[I_a/I_a, I_s/I_s, K_c/K_a, A/A, B/B, S/S]$$

2. creating a new message as if A has initiated. This is given by,

$$C_A \equiv initiator[I_a/I_a, I_s/I_s, K_c/K_a, A/A, B/B, S/S]$$

   The difference between (2) and (1) is that in (1) the message was in response to an initiation from the actual initiator A whereas in (2) a fraudulent message was created by the intruder C.

The module shown in Figure 8 combines the above two scenarios where input signals *one*, *two* represent signals that make the intruder decide the choice (this is where input indeterminacy comes into the picture). Actually the case corresponding to, *two*, is as good as C starting as an initiator. The other possibilities have been ignored as they don't seem to be structurally feasible. We have placed reactions pairwise (through the await-case statement) as the intruder once decides on a role will take the corresponding reaction.

On similar lines, the general intruder models can be obtained for playing the roles of responder and server. In short, breaking the message components and

```
module Intruder-init;
   input one, two;
         I_s: some_string_type;
   output I_a: some_string_type;
   var K_a: key_bit_type;
       A: initiator_type;
       B: responder_type;
       S: server_type;
   loop
       await
           case one  do emit I_a(A,S,B,E(K_c)); %change key%
           case two  do emit I_a(C,S,B,E(K_c)); % change A%
       end await
       await tick; % assume one is sustained till the nest reaction
                      of the await-case statement %
       await immediate
            case one do await immediate  I_s(S,A,B,V(K_c,K_b));
            case two do await immediate I_s(S,A,B,V(K_c,K_b))
       endawait;
   endloop
end module
```

Fig. 8. Model of General Intruder

plain substitution makes it possible to perform attacks like (i) eavesdropping, (ii) mimicking an initiator, (iii) mimicking a responder, (iv) and mimicking as a server for the un-encrypted part (or plain text) etc. With concurrent roles being possible, attacks such as (a) concurrently mimicking as initiator and responder and (b) attack through concurrent sessions are possible. These are modelled using the parallel operator that permits concurrent instantiations of modules. The possibility of the initiator, A, starting two concurrent sessions will be discussed later. The work in [12] elucidates the various attacks on the TMN protocol using the CSP model checker elaborately. In the following section, we discuss these attacks and show how the attacks can be derived as instances of the general model discussed above.

Note 5. The interception of a message classically corresponds to somebody blocking (and picking) the message from reaching the actual destination. In the following, we have allowed the intruder from copying the packet and allowing it to reach the destination as pure interception does not lead to new possible attacks (note we have already modelled interception earlier). It must be noted that interception can be trivially modeled through a preemptive message. Interception will become relevant relative to questions like whether the initiator as well as the intruder are able to get the key etc.

4.1 Attacks in the Protocols

In this section, we shall illustrate the various attacks described in [12] as instances of the model discussed so far.

Intruder Mimicking Initiator: Operationally, the intruder mimicking the initiator can be described as follows:

- The intruder picks up the request of the initiator, say A, to some responder, say B.
- Sends the modified request (to denote as if it has come from initiator A). This is accomplished by replacing in the message digest its' own key in place of the key of the initiator.

As the intruder is also a valid principal in the system, the behaviour of an agent mimicking as A can be described by the following module:

$$C_A \equiv initiator[I_a/I_a, I_s/I_s, K_c/K_a, A/A, B/B, S/S]$$

The verification for this property is done by constructing an observer for repeated occurrences of the trace "I_a(A,S,B,E(K_a)) I_s(S,A,B,V(K_c, K_b))" as shown below:

```
module observer;
input I_a,    I_s: some_string_type;
output alarm: alarm_type;
    loop
      await immediate I_a(A,S,B,E(K_a));
      await I_s(S,A,B,V(K_c,K_b));
      present ?I_s = (A,S,B,E(K_a))
                then emit alarm
      else skip
      end present;
    end loop
end module observer
```

The observer when run along with the protocol emits the signal `alarm` depicting the underlying attack. Now, let us analyze further additional questions about security we can ask about the system having A, B, S and C_A as the principals (i.e. the system shown in Figure 5 along with C_A).

1. Does the intruder get k_b? That is, does C_A get the message "I_s(S,A,B, V(K_c, K_b))" in response to I_a(A,S,B,E(K_c))? The deduction of getting the key after this message is obtained, follows since the intruder is also a valid principal and hence, knows the underlying simplification rules.

   From the `Xeve` *(also by simulation through* `Xes`*), the answer is YES.*
2. Is it possible that the principal A also could have initiated the request and got the key concurrently?

   The answer is YES on the same lines as (1) above.

3. Is it possible that the principal A as well as the intruder C_A can get the keys concurrently?

   The answer is YES since it is possible to have multiple emissions of the same signal for which a proper *combination function*[2] can be defined in the program. In other words, both A and C_A will see two messages out of which only one can be decoded by each using the simplifications laws.

4. The affirmative answer to the above question leads to the question: Does the initiator detect that there could be an intruder? or Does the intruder smell that the actual principal could have been there and hence, it could have detected the possible existence of an intruder?

   The answer is YES again on the same lines of argument (3) given above.

Scenarios corresponding to above cases are: (i) A not actually initiating but initiated by C_A (ii) A and C_A initiating concurrently, (iii) only A initiating and (iv) neither A nor C_A initiating.

Intruder Mimicking Responder: Here, the intruder picks up the response of the server to the responder and sends it as if it was the responder itself except that it replaces its' key for the key of the responder. The corresponding module is: $C_B \equiv \mathit{responder}[I_b/I_b, I_s/I_s, K_c/K_b, A/A, B/B, S/S]$. Similar questions as in section 4.1 can be answered on the same way.

Concurrent Initiator- and Responder-Intruders: Here, the initiator initiates a run with itself mimicking as A and wanting to respond with B; when A indeed wants to have a session with B then the intruder responds as B. This could be treated as multiple roles for an intruder. The model corresponds to running the basic protocol shown in Fig. 5 in parallel with C_A and C_B as above. On the same lines as above, it can be shown that the principals could be deceived by the intruders. Assuming in the Internet, the packets are not destroyable except by those to whom it is meant for, the following question again can be answered affirmatively:

> Is it possible that the initiator as well as the intruder would have got hold of the keys?

5 Modelling and Analysis of Modified TMN Protocol

The first modification of the TMN protocol to overcome the attacks discussed so far as discussed in [12] is as follows: the *initiator* and the *responder* share an explicit secret with the server and use them when sending messages to the server. Thus, the server matches the secret sent by the principal with its identity (again a secret known only to the Server and the agent itself). The modified protocol with the above mentioned changes [12] is shown below:

Message 1: $A \rightarrow S$: $A.S.B.E(s_a, K_a)$
Message 2: $S \rightarrow B$: $S.B.A$
Message 3: $B \rightarrow S$: $B.S.A.E(s_b, K_b)$
Message 4: $S \rightarrow A$: $S.A.B.V(K_a,\ K_b)$

The module structure shown in Figure 5 will remain the same except for the message structure and the additional secret the agents share with the server. The new model of the initiator, responder and the server are shown in Figures 9, 10, and 11 respectively with the modified TMN system shown in Figure 12.

Running the three modules shown in Figure 12 along with the model of the intruder shown in Figure 13, one arrives at the conclusion that that it is indeed the case the intruder can capture the secret key. On the same lines, one can show that it is also the case that it is quite possible that the initiator and the responder could get the clue that there is an intruder.

The attack as found in [12] (attacks 5.1 & 5.3) follows exactly as above; the difference from 5.1 and 5.3 as envisaged in [12] is that in 5.1 the message from the initiator has been intercepted and hence, it is not responding further whereas in 5.3 the initiator is continuing to respond. It is of interest to note that in the reactive framework, both of them are answered at one shot (in other words, one scenario comes for free).

The attack described in section 5.2 of [12] can be realized using the intruder

$C_B \equiv$ initiator-I$[s_a, K_a/s_a, K_a,\ I_s/I_s,$ A/A,C/B,S/S$]$

In this case, in the system consisting of A, B, S, and the intruder, C_B, the server, S, is responding to both the messages received from the initiator and the intruder. Thus, both A and C_B get the key K_a. This just corresponds to detecting the two messages (in fact, this is the question asked in section 4.1). It may be noted that the intruder is doing little more than than what it is supposed to do as it is assuming that the message is removed completely. (in fact, message 4' in attack 5.2 [12] will come by itself as the server is going to respond for the messages any way).

Attack of section 5.4 of [12] can be obtained using the same intruder as given above; the difference is that the initiator also gets the proper key.

5.1 Further Strengthened TMN Protocol

The strengthened TMN for the attacks [12] wherein the identity of the agent with which communication is requested is encrypted is given below;

Message 1: $A \rightarrow S$: $A.S.E(B,\ s_a, K_a)$
Message 2: $S \rightarrow B$: $S.B.A$
Message 3: $B \rightarrow S$: $B.S.E(A, s_b, K_b)$
Message 4: $S \rightarrow A$: $S.A.B.V(K_a,\ K_b)$

ESTEREL model for the above protocol follows from the models shown earlier. The required proof obligation is: $I_a(A, S, E(B, s_a, K_a)) \leadsto I_s(S, A, B, V(K_a, K_b))$. This easily follows in the ESTEREL model.

Now, let us see whether some intruder can indeed try to get a key of a principal fraudulently. Our first task is to construct the intruder. By the rules of the intruder model, messages 1 and 3 cannot be changed meaningfully as encryption

```
module Initiator-I;
   input start; $I_s$: some_string_type;
   output $I_a$: some_string_type;
   var $K_a$: key_bit_type; $s_a$: sec_identity_type;
       A: initiator_type; B: responder_type; S: server_type;
   loop $\ell_{i_0}$: await start;
        $\ell_{i_1}$: emit $I_a$(A,S,B,E($s_a, K_a$));
           %(sender.via,destination,encrypted_msg and secret_share)%
       $\ell_{i_2}$: await tick;
       $\ell_{i_3}$: await $I_s$(S,A,B,V($K_a, K_b$))
   success: endloop
end module
```

Fig. 9. Modified Initiator

```
module Responder-I;
   input $I_s$: some_string_type;
   output $I_b$: some_string_type;
   var $K_b$: key_type; $s_b$: sec_identity_type;
       A: initiator_type; B: responder_type; S: server_type;
   loop $\ell_{r_0}$: await immediate $I_s$(S,B,A);
        $\ell_{r_1}$: await tick;
        $\ell_{r_2}$: emit $I_b$(B,S,A,E($s_b$,  $K_b$);
   endloop
end module
```

Fig. 10. Modified Responder

```
module Server-I:
   input $I_a$, $I_b$: some_string_type;
   output $I_s$: some_string_type;
   var A: initiator_type; B: responder_type; S: server_type;
       key_initiator, key_receiver: key_type;
   loop $\ell_{s_0}$: await immediate $I_a$(A,S,B,E($s_a, K_a$);
      A:= first($I_a$(A,S,B,E($s_a, K_a$)); B:= third($I_a$(A,S,B,E($s_a, K_a$));
    key_initiator:= getkey ($I_a$(A,S,B,E($s_a, K_a$));
    $\ell_{s_1}$: await tick;
    $\ell_{s_2}$: emit $I_s$(S,B,A);
    $\ell_{s_3}$: await $I_b$(B,S,A,E($s_b, K_B$)); key_receiver:=getkey($I_r$(B,S,A,E($s_b, K_b$));
    $\ell_{s_4}$: await tick;
    $\ell_{s_5}$: emit $I_s$(S,A,B, V($K_a, K_b$))
endloop
end module
```

Fig. 11. Modified Server

```
module TMN-imp;
  input strat;
  output $I_a$, $I_b$, $I_s$: some_string_type;
  run initiator-I [$s_a, K_a/s_a, K_a$, $I_s/I_s$, A/A,B/B,S/S]
|| run responder-I [$s_b, K_b/s_b, K_b, I_s/I_s$, A/A,B/B,S/S]
|| run server-I
end module
```

Fig. 12. Modules of the Improved TMN Protocol

```
  run initiator-I[$s_c, K_c/s_a, K_a$, $I_s/I_s$, C/A,B/B,S/S] % mimics initiator
  || run responder-I[$s_c, K_c/s_b, K_b, I_s/I_s$, A/A,C/B,S/S] % mimics responder
```

Fig. 13. Model of the Intruder

is tied down to identities as well as secrets with the servers which are by definition noncrackable. The only other message that can be be modified or inserted are message 2 (the possibilities being $S.B.C$ or $S.C.B$) and message 4 (the possibilities are $S.C.B.V(K_a, K_b)$, $S.A.C.V(K_a, K_b)$). Thus, the intruder will be simpler than that obtained earlier due to the strengthened message transmission. A simple analysis shows that the altered possibilities of message 4 are of no use to the intruder as the key cannot be decrypted by anyone else other than A as per the simplifications rules. Since the traces are finite, the observer constructed to find whether there will be a message of the form $I_a(S.A.B.V(K_c, K_b))$ in response to a message $I_a(A, S, E(B, s_a, K_a))$ will be false – establishing the security of the system.

5.2 Authenticity

Authenticity as defined in [12] is given below:

1. If a responder B completes a run of the protocol, say with A, then it must be the case that A must have been previously been trying to run the protocol with B. Note that it is never the case that B can conclude that A must have received the message sent by it. Further there should be a 1-1 relationship between the runs of A and B.
2. If an initiator A completes a run of the protocol, say with B, then B has previously been trying to run the protocol with B, then B has previously been trying to run the protocol with A; there should be a 1-1 relationship between the runs of A and B. Also, the two agents should agree on the value of the key established.

The trace for a proper correct run of the initiator and responder is:
$I_a(A, S, E(B, s_a, K_a)),\ I_s(S, B, A),\ I_b(B, S, E(A, S_b, k_b)),\ I_s(S, A, B, V(K_a, K_b))$
Thus, if the system satisfies the property of authenticity, the above should be the only valid trace. However, the system will not only find the trace given above but also finds: $I_s(S, B, A),\ I_b(B, S, E(A, S_b, k_b)),\ I_s(S, A, B, V(K_a, K_b))$
Hence, we can conclude that it is not the case that the strengthened protocol satisfies the property of authenticity. The reason for the failure of authenticity is that "message 2" can be emitted by the intruder in response to which the server can respond. Note that the intruder is also capable of emitting the first message which may be an old stored one. The server does not check whether the message has been responded to or not. Now, let us ask some question that relates to concurrent runs that can be modelled easily in the reactive framework.

Question: What happens if A wants to have concurrent runs with the restriction that the response from A to B should keep track of the explicit session?

As the protocol does not satisfy the property of authenticity, the above property cannot be obviously satisfied. The counter example can be found easily by modelling the observer and showing that the traces can be interleaved. We can also model nonces and iterative authentication protocols. We will not going into these aspects for want of space.

6 Synchronous Languages as Specification Languages for Cryptographic Protocols

One of the works [10] on cryptographic protocols is concerned with the specification of languages that would allow a simple derivation of the obligations and commitments of the principals involved in the protocol. A careful examination of ESTEREL shows that it satisfies the main objectives highlighted in [10] Considering the underlying typing rules, it is possible to distinctly arrive at the capabilities and visibilities of signals and variables. Using such information, we can arrive at the various obligations/commitments of the modules. For instance, the following scenarios can be arrived at from Figure 5:

1. A has the role of *initiator*, B of *responder* and S the role of *server*.
2. A sends its request on I_a and awaits response on I_s.
3. B awaits for requests on I_s and responds on I_s.
4. S awaits for requests/responses on I_a, I_b and forwards them on I_s.
5. Keys K_a and k_b should not be revealed to any intruder.

Through such an interpretation, one can formulate required properties as postulates and commitments – leading a good understanding of obligations.

7 Conclusions

The reactive framework seems to be quite suitable for the analysis of cryptographic protocols for the following reasons:

1. The languages are well defined and the semantics is captured nicely in terms of rewrite systems - thus,capturing the inductive definitions succinctly.
2. Language supports input indeterminacy with deterministic reactions. The underlying broadcast communication aids in capturing the router/broadcast features naturally that are needed for the modelling.
3. It has good simulate-compile-verify environment and permits checking some of the security and authenticity properties to be verified by simulation using *observer* criteria.
4. Modularity of the language permits to check properties with concurrent roles and runs; also, it can be used as a specification language.
5. Verification can be done via symbolic bisimulation or via model checking.
6. The availability of preemption allows us to model complex protocols that allow withdrawing of requests.

Most of the security properties can be captured as safety properties. These could be specified in various logics which can be directly translated as observers as envisaged above automatically; we shall not discuss further due to lack of space. Further, our experience shows that in analyzing cryptographic protocols, counter-example generating tools will have a useful role. We are also extending the study to real-life complex protocols such as SET [13] with preemption.

Acknowledgments

The work was done under the project " Design and Implementation of Secure Systems for E-Commerce", supported from MIT, New Delhi. Part of the work was done while the author was visiting Max-Planck Institut für Informatik, Saarbrücken. The author thanks Harald Ganzinger for the encouragement and several suggestions.

References

1. M. Abadi, A. Gordon, *A Calculus for Cryptographic Protocols: The Spi Calculus*, 4th ACM Conf. on Computers and Communications Security, (1997) 36-47.
2. G. Berry and G. Gonthier, *The* ESTEREL *Synchronous Programming Language: Design, semantics, Implementation*, SCP, 19 (2):87-152, November 1992.
3. B. Blanchet, *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*, In 14th IEEE Computer Security Foundations Workshop (CSFW-14), Canada, 2001.
4. D. Bolignano, *An approach to the formal verification of cryptographic protocols*, 3rd ACM Conf. on Computer and Communications Security, pp. 106-118, 1996.
5. M. Burrows, M. Abadi, R. Needham, *A Logic of Authentication*, ACM Transactions on Computer Systems, 8(1), (1990) 18-36.
6. D. Dolev, A. Yao, *On the Security of Public Key Protocols*, IEEE Trans. on Information Theory, 29(2), (1983) 198-208.
7. D. Harel, A. Pnueli, *On the development of reactive systems: logics and models of concurrent systems*, Proc. NATO ASI Series, 477-498, Springer Verlag, 1985.
8. R. Kemmerer, *Analyzing encryption protocols using formal verification techniques*, IEEE J. on Selected Areas in Communications, 7(4), (1989) 448-457.
9. R. Kemmerer, C. Meadows and J. Millen, *Three Systems for Cryptographic Protocol Analysis*, J Cryptology (1994), 7:79-130.
10. J.G. Larrecq, *Clap, a simple language for cryptographic protocols*, INRIA, 2001.
11. G. Lowe, *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*, Proc. TACAS, LNCS 1055, 147-166, 1996, Springer Verlag.
12. G. Lowe, and B. Roscoe, *Using CSP to detect errors in the TMN protocol*, In IEEE Tr. on Software Engg., v. 23, 10, 1997.
13. Mastercard & VISA, *Secure Electronic Transaction Specification*, Books, 1-3, 1996.
14. J.C. Mitchell, M. Mitchell, U. Stern, *Automated Analysis of Cryptographic Protocols Using Murϕ*, Proc. IEEE Symp. on Security and Privacy, 1997, 141-151.
15. G. Pace, N. Halbwachs, and P. Raymond, *Counter-Example Generation in Symbolic Abstract Model Checking*, FMICS, 2001.
16. L.C. Paulson, *The Inductive approach to verifying cryptographic protocols*, J. Computer Security, 6, 1998, 85-128.
17. N. Raja and R.K. Shyamasundar, *A Cryptographic Protocol Analysis: an annotated bibliography*, Tutorial at Int. Conf. on Information Tech., Bhuvaneshwar, Dec. 2000.
18. M. Tatebayashi, N. Matsuzaki, D. Neuman, *Key distribution protocol for digital mobile communication systems*, Proc. CRYPTO '89, (90) 324-333, Springer Verlag.
19. C. Weidenbach, *Towards an automatic analysis of security protocols in first-order logic*, in H. Ganzinger, ed., 16th CADE-16, 1999, LNAI 1632, Springer, 378-382.