J. Lopez, J. J. Ortega, J. M. Troya, and J. L. Vivas, "High-level Specification of Security Systems", *IEEE Globecom 2003 - Communications Security Track*, pp. 1506-1510, 2003.
NICS Lab. Publications: https://www.nics.uma.es/publications

High-level Specification of Security Systems

Javier López, Juan J. Ortega, José M. Troya
Computer Science Department
University of Malaga
Malaga, Spain
{jlm, juanjose, troya} @ lcc.uma.es

José L. Vivas
Hewlett Packard Labs
Bristol, UK
jose-luis.vivas@hp.com

***Abstract*— In order to study security systems, we have developed a methodology for application of formal analysis techniques commonly used in communication protocols to the analysis of cryptographic protocols. In particular, we have extended the design and analysis phases of protocol design with security properties. Our proposal uses a specification notation based on HMSC/MSC, which can be automatically translated into a generic SDL specification.**

Keywords-Security system, specification, design, analysis, SDL language

I. INTRODUCTION

Nowadays it is widely accepted that critical systems have to be analysed formally to achieve the well-known benefits derived from the application of formal description methods [13]. These methods allow the description of system behaviour in a precise way and can be used to verify the specification. In particular, the design and analysis of security systems can greatly benefit from the use of formal methods, due to the critical nature of such systems.

A security protocol [14] is a general template describing a sequence of communications that makes use of cryptographic techniques to meet one or more particular security-related goals. In the present study we do not need to distinguish between cryptographic and security protocols, and therefore we regard them as equivalent. The international organization ITU-T has produced the Recommendation Series X.800 [6,7] with the aim of specifying the basic security services. Among these, those provided by the basic security mechanisms (cryptographic algorithms and secure protocols) are authentication [8], access control [9], data confidentiality [11], data integrity [12], and non-repudiation [10].

These services are commonly enforced using cryptographic protocols or similar mechanisms. It is worth noting that in order to specify a security critical system it is not necessary to know how the future system will be analysed, but it is certainly indispensable to identify the required security services.

In recent years, cryptographic protocol analysis research [16] has experienced an explosive growth, and numerous formalisms have been developed. However, in our opinion the results obtained from the analysis of cryptographic protocols are not directly applicable to the design of secure communication systems. Probably, one of the major reasons is the lack of a strong relationship between the analysis tools and the formal methods techniques commonly used in the specification and analysis of communication protocols. To bridge this gap is one of the major objectives of our approach.

We have developed a methodology for the specification of secure systems which allows us to verify that they are not vulnerable against both known and novel attacks [14,15]. Our approach includes the use of a requirement language called *Security Requirements Specification Language* (SRSL) to describe security protocols, which can be automatically translated into the *ITU-T Recommendation Specification and Description Language* (SDL) [3], a widely used formal notation well suited for protocol analysis. In addition, we have developed verification procedures and tools to check several security properties such as confidentiality, authentication, and non-repudiation of origin. In this approach we use a simple but powerful intruder process that is explicitly added to the specification of the system. As a result, the verification of the security properties guarantees the robustness of the protocol against the attacks of this kind of intruder. This is known as Dolev-Yao mechanism [2].

The structure of the rest of this document is as follows. In Section 2 we give an overview of our specification and analysis approach. Section 3 is dedicated to a detailed presentation the SRSL language, and Section 4 to the description of an application of SRSL to a real world case. Finally, in Section 5 we show some conclusions and give some suggestions for future work.

II. METHODOLOGY OVERVIEW

In our approach (summarized in Figure 1) the design and analysis of security protocols are carried out exactly as in traditional communication protocols, but including the security aspects.

First, the functional and security requirements of the system are captured in the usual informal way. The resulting informal specification, extended with the specification of behaviour associated with a variety of possible attacks, can then be described using our requirement language, SRSL.

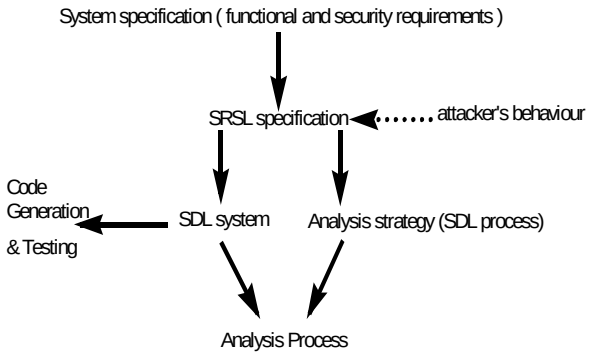

Figure 1. Overview of our approach

SRSL is an extension of *ITU-T Recommendations Message Sequence Chart* (MSC), and its expansion *High-level MSC* (HMSC) [4], augmented with textual tags. We make use of the HMSC/MSC text area to include these tags, which are used to identify the security characteristics of the data being transmitted, the intruder's possible actions, and the security analysis goals. In case the attacker's behaviour is not explicitly provided, we automatically generate a generic process that tries to examine all possible attacks.

In order to draw the graphical SRSL specifications, any standard MSC and HMSC editor can be used. In our case, we have used Telelogic's TAU, which also allows the automatic translation of the graphical MSC diagrams into a corresponding textual form. A translator program is then used to obtain the SDL system from the SRSL description (this program was written in C, using plain LEX and YACC tools). The resulting SDL system is composed of: (1) a package with the data types of the system for analysis; (2) a package with one process type for each protocol agent; and (3) a collection of process types ("observer" and "medium") for the analysis strategy.

In order to analyse the security properties, we evaluate the behaviour of the SDL system under different kinds of attacks (as specified by the medium processes defined in the analysis strategy). The observer process provided by the TAU Validator tool is used for these checks. Thus, we can check whether a specific state is reached, or whether a particular data is ever stored into the intruder's database knowledge. We also make use of the *assert* mechanism, which enables observer processes to generate reports during the state space exploration. These reports are maintained by the *Report Viewer*, and can be examined to identify security flaws.

In addition, the SDL system generated from the SRSL specification can be used to automatically generate C or C++ code, which can thus interact with exiting applications. In order to generate this code we need to replace the data types package with a corresponding package that defines the data types in ASN.1 or C. This prototype can also be used for testing, which is part of our future work.

III. THE SRSL LANGUAGE

SRSL is intended to be a high-level language for the specification of cryptographic protocols and secure systems. The requirements that have guided the design of SRSL are modularity (for the sake of reusability), suitability for expressing security notions, and to be easy to learn.

As a natural initial model for SRSL we selected the requirements language most widely used in telecommunications, namely MSC and its extension HMSC. With MSC we can specify elementary scenarios, and with the HMSC we can compose the latter to obtain more complex protocols. The version we have considered is the one previous to the MSC 2000 [5] release, but we plan to use the latter in a future version.

SRSL is divided into two main parts. The first one contains the definition of the protocol elements and the security analysis strategy. The second part describes the message exchange flow.

The first part is textual. The syntax of its main elements is shown in Figure 2. These elements can be grouped into different categories, and are listed below (language keywords in italics):

Entities: *Agent*, principal identification; Messages: *Data*, message text; *Random*, number created for freshness, also called nonce; *Timestamp*, actual time; *Sequence*, counter. Keys: *Public_key*, a pair of public and private keys; *Symmetric_key*, used for symmetric encryption; *Shared_key*, symmetric key shared by more than one entity; *Session_key*, a fresh symmetric key used to encrypt transmission.

The "*knowledge*" section contains the information needed to describe the initial knowledge of each party of the protocol.

The "*security_service*" section is split into the intruder's strategy section and the security property section. The first one defines a possible attack scenario. The second describes the security properties we want to be enforced by the protocol. We have used three different security statements: *Authenticated(A,B)*, stating that B is sure of the identity of A; *conf(X)*, stating that the data X cannot be deduced (also called confidentiality); and *NRO(A,X)*, or non-repudiation of origin, which provides an evidence that action X (the evidence) must have been performed by A. These statements have a formal description [8,9] which is used to analyse them.

The message exchange flow is described using the standard MSC and HMSC facilities. MSC references are used to achieve reusability. We have specified a set of standard protocols in SRSL that can be easily reused in different contexts, and combined them together to describe more complex protocols using their MSC references.

A message consists of an identification name (either a text string describing the meaning of the message, or a simple counter sequence), and message parameters defining the message data type format.

Some cryptographic operations can be applied to messages: *Concatenate* (“,”) for data composition; *Cipher* (“{<plaintext>}”<key>) to cipher data; *Hash* (“<hash-function>(<data>)”), the result of a one way algorithm; and *Sign* (“<plaintext>{hash(<plaintext>)} <Public_key>’ ”), to get a hash message signed with the signer’s private key. Further cryptographic functions can be defined if required.

```
Security_information ::= definition_section Security_service_section

Defintion_section ::= Definition var_definition  knowledge_section

var_definition ::= <varlist> : Agent ;
                 | <varlist> : Data ;
                 | <varlist> : Random ;
                 | <varlist> : Timestamp ;
                 | <varlist> : Sequence ;
                 | <varlist> : Public_key ;
                 | <varlist> : Symmetric_key ;
                 | <varlist> : Shared_key ;
                 | <varlist> : Session_key ;

Knowledge_section ::= Knowledge <listagent_id> : <varlistasig> ;

Security_service_section ::=  [intruder_strategy] security_property

intruder_strategy ::= Session instances [ <var>=<value> ] ;
                    | intruder_knowledge [ <initial_knowledge> ];
                    | intruder [ redirect | , impersonate | , eavesdrop ] ;
security_property ::= Security_service <security_service_list> ;

security_service_list ::= authenticated ( <agent> <agent> )
                        | conf ( <data> )
                        | NRO ( <agent> <data> )
```

Figure 2. BNF syntax of the protocol elements and the security analysis strategy

In addition, the MSC expressions constructed using the inline MSC operators alt, par, loop, opt and exc can also be used.

The keyword *alt* denotes alternative executions of several MSCs. Only one of the alternatives is applicable in an instantiation of the actual sequence. The *par* operator denotes the parallel execution of several MSCs. All events within the MSCs involved are executed, with the sole restriction that the event order within each MSC must be preserved. An MSC reference with a *loop* construct is used for iterations and can have several forms. The most general construct, *loop<n,m>*, where *n* and *m* are natural numbers, denotes iteration at least *n* and at most *m* times. The *opt* construct denotes a unary operator. It is interpreted in the same way as an *alt* operation where the second operand is an empty MSC. An MSC reference, where the text starts with *exc* followed by the name of an MSC, indicates that the MSC can be aborted at the position of the MSC reference symbol, and can continue instead with the referenced MSC.

IV. A Case Study: On-Line Contracting Process

We have applied our methodology to a system currently under development by one of the user’s partners in the CASENET project [1]. The company developing a virtual enterprise business scenario implementing on-line contracting processes by integration of Trusted Third Party services (TTPs), such as an existing electronic notary system, into a web-based multi-users services platform. The current on-line contracting process is rather complex and covers different tasks such as contract creation, negotiation, signing and final archiving.

We focus first on the contract signing process (contract signing management and notarisation process control). This procedure is part of the business-to-business scenario for setting up a virtual enterprise platform integrating technology components such as e-contracting, e-notary and role based authorization engine.

This section describes the existing electronic notary process within a current e-business scenario. The central core of this set-up is the MESA platform. MESA provides web-based user interfaces and role based control mechanisms for accessing functions made available by the TTPs.

The following diagram (figure 3) describes the contract signing process as it is implemented by an e-Notary reference application and used within the company scenario. A user accessing a web-based user interface provided by the MESA platform triggers the contract signing process within this business scenario manually:

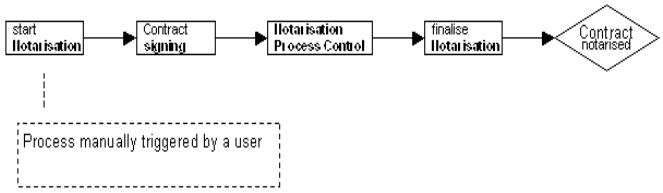

Figure 3. Contract signing process

In the sequel we describe the contract signing process, including both the security requirements and the relationships among the users, the MESA platform, and the e-notary service.

Our methodology has been used to examine this process in terms of communication security issues. The intended goals are to validate the model and evaluate both the current reference implementation and a proposed extension to an agent-based scenario for the reference implementation.

This implementation is being used within the current business scenario. However, the current client/server implementation, based on traditional public-key cryptographic technology, has inherent problems in terms of flexibility and scalability. While the reference scenario requires a certain infrastructure, being compliant to the European directives concerning digital signatures, to alternative public-key technologies and to certificate infrastructures, might be more suitable when adopting the e-notary process to other business

scenarios (giving different context of actors, content, legal requirements and liability issues).

In fact changing the context of a recent e-Notary deployment scenario and identification of implications in terms of security is the most interesting challenge.

We should note that what we have specified here is an already implemented system, i.e. a legacy system. Therefore, our task has been to describe the behaviour of the application in order to analyse and improve the current implementation. We started by emphasising for the developers the usefulness of specifying a system with the special aim of clarifying the different scenarios in order to understand them better and to avoid ambiguities.

The system is divided into three parts: the contract creation process, the signing process, and the notarisation process. These are represented in HMSC combining MSC references. Each MSC reference is described by a diagram in a lower abstraction level.

A representative part of the specification is the *create_contract* scenario (see Figure 4).

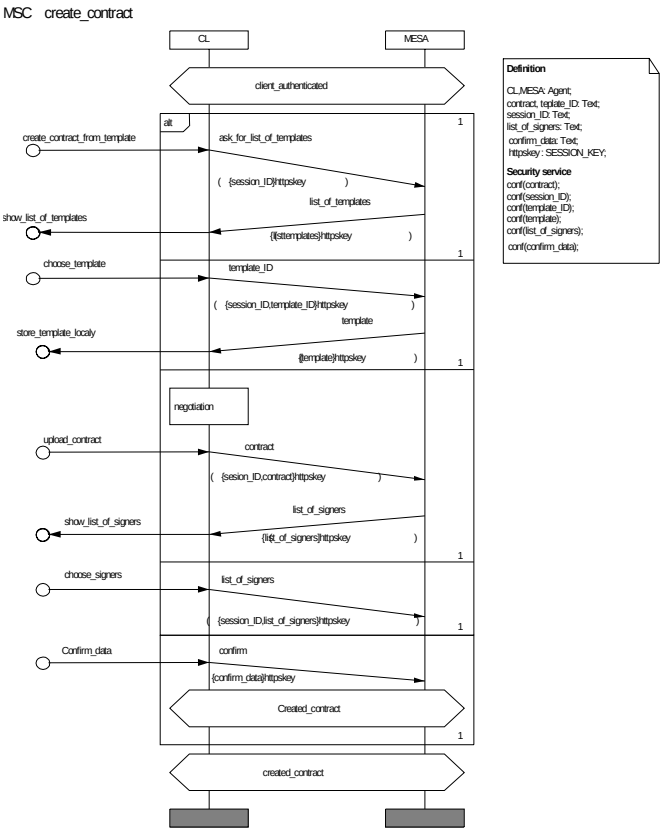

Figure 4. SRSL create_contract security scenario

The contract leader (*CL*) triggers the contract creation process. Previously, the contract leader and the *MESA* platform had to be authenticated, and a HTTPS session key exchanged. This is represented by the initial state "*client_authenticated*". The scenario is divided into four independent alternatives (*alt*-operator). In the third sub-scenario we use the task MSC operator to express the possibility of an external negotiation agreement, which is not part of our system. The fourth sub-scenario ends the process by accepting the uploaded contract and starting the next scenario in the state *created_contract*.

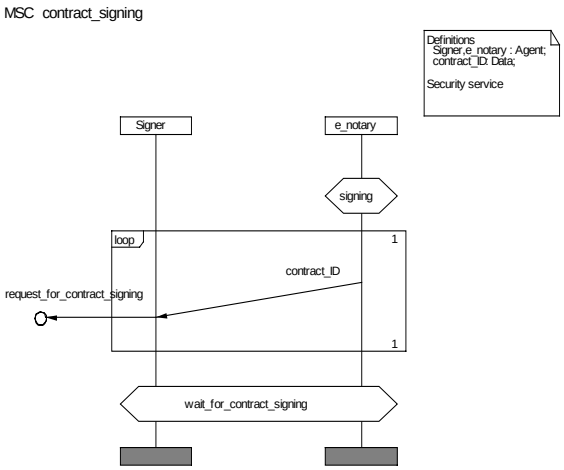

Figure 5. SRSL contract_signing security scenario

The developers in the company considered this methodology very useful for their purposes, especially with regard to the specification of the contract signing process (see figure 5). The notification was initially implemented by letting the E-notary service send an e-mail to each signer. This constitutes however an unreliable procedure with no security guarantees. When this fact had been drawn to their attention, the developers decided to modify the system in order to provide for security services, such as signing of the e-mail by the e-notary service to guarantee non-repudiation (see Figure 6).

The developers considered this approach easy to learn. They believed that it had been of great help for understanding the implementation, and for providing a method to improve the application with regard to the required security services and mechanisms.

V. CONCLUSIONS

We have presented a new specification method for describing and evaluating security protocols. Security protocols are specified in SRSL, which can then be translated into a working SDL system. Attacks are implemented by SDL processes that specify the intruder's behaviour and observer processes that check security properties. One of the benefits of our approach is that protocol specifications are described independently from the analysis procedures, so they can be re-used in other environments as well.

In order to illustrate the methodology, we have shown an application, consisting of an electronic notary process scenario, whose developers wanted to validate and improve.

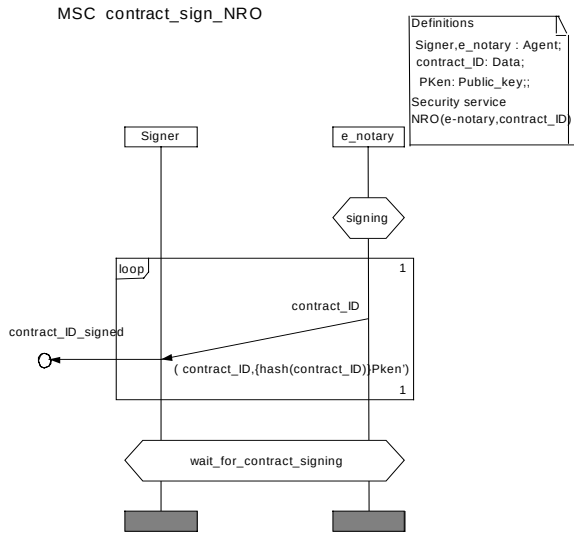

Figure 6. SRSL contract_signing security scenario with non-repudiation of origin requirement

Furthermore, we have described how this e-Notary process can be inserted into a different scenario, given different input parameters. In this way, we were able to offer a framework within which it became possible to define and to evaluate different deployment options for rolling out the security services. Finally, we note the fact that the solutions proposed were very well received by the developers, who considered them easy to learn and to apply.

Currently we are extending SRSL with the aim of enabling the specification of more complex protocols and the analysis of further properties. We are considering to use MSC-2000 features for specification purposes. Furthermore, we are developing a framework for defining protocol attacks in the Internet environment for purposes of testing.

REFERENCES

[1] CASENET V Framework Programme Research Project. http://www.casenet-eu.org/

[2] D. Dolev and A. Yao. *On the security of public key protocols*. IEEE Transactions on Information Theory, IT-29:198-208, 1983.

[3] ITU-T Recommendation Z.100 (11/99). *Specification and Description Language (SDL)*, Geneva, 1999.

[4] ITU-T, Recommendation Z.120. *Message Sequence Charts (MSC)*. Geneva, 1996.

[5] ITU-T, Recommendation Z.120 (11/99). *Message Sequence Charts (MSC)*. Geneva,1999.

[6] CCITT Recommendation X.800. *Security Architecture for Open Systems Interconnection for CCITT Applications*. 1991.

[7] ITU-T Recommendation X.810 (ISO/IEC 10181-1). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems – Overview*. 1995.

[8] ITU-T Recommendation X.811 (ISO/IEC 10181-2). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems – Authentication*. 1995.

[9] ITU-T Recommendation X.812 (ISO/IEC 10181-3). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems -- Access Control*. 1995.

[10] ITU-T Recommendation X.813 (ISO/IEC 10181-4). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems -- Non-Repudiation*. 1995.

[11] ITU-T Recommendation X.814 (ISO/IEC 10181-5). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems – Confidentiality*. 1995.

[12] ITU-T Recommendation X.815 (ISO/IEC 10181-6). *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems – Integrity*.1995.

[13] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, 1991.

[14] J. López, J. J. Ortega and J. M. Troya. *Protocol Engineering Applied to Formal Analysis of Security Systems*. Infrasec'02, LNCS 2437, Bristol, UK, October 2002.

[15] J. López, J.J. Ortega and J. M. Troya. *Verification of authentication protocols using SDL-Method*. Workshop of Information Security, Ciudad-Real- SPAIN, April 2002.

[16] C. Meadows. *Open issues in formal methods for cryptographic protocol analysis*. Proceedings of DISCEX 2000,pages 237-250. IEEE Comp. Society Press, 2000.

[17] A. Menezes, P.C. Van Oorschot, S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[18] P. Ryan and Scheneider. The Modelling and Analysis of Security Protocols: the CSP Approach. Addison-Wesley, 2001.