A framework for the development of protocols

Federico Crazzolara
NEC Europe, St. Augustin, Germany

Giuseppe Milicia
BRICS – University of Aarhus, Denmark *

Chi Spaces Technologies ltd.
{federico, milicia}@chispaces.com

Abstract

We present the χ-Spaces framework, a tool designed to support every step of a security protocol's life cycle. Its Integrated Development Environment (IDE) eases the task of protocol design, debugging and simulation.

1. Introduction

Security protocols describe a strategy that two or more parties can follow to obtain certain guarantees about each other by exchanging messages over an untrusted medium. For instance, parties may wish to ascertain each other's identity, exchange secret information, or even communicate without revealing their identity.

Experience shows that the design and implementation of security protocols is a complex and error-prone task. General purposes programming languages and environments provide little help. A protocol, specified in few lines, yields thousands of lines of code in languages such as Java and C++. As the semantics of the protocol become unclear, debugging can turn into a long and painful activity.

The χ-Spaces framework provides a *domain specific* programming environment designed for the development of security protocols. The χ-Spaces language is an implementation of SPL [1, 2]; protocols programmed with χ-Spaces are concise (most fit in less than one page) and enjoy rigorous semantics – a more traditional transitions-system semantics and a Petri-net semantics, both closely related [2]. We implemented the transition semantics, and use the net semantics to verify security properties of protocols. The framework has been shown to effectively aid the task of protocol development in several case studies [4, 5, 6, 7].

So far the χ-Spaces framework provided only *command line* utilities. Recently we unified the tools into a common *Integrated Development Environment* (IDE). The IDE includes advanced features which further aid the task of protocol design and implementation. Most notably, it is possible to graphically represent protocol runs, an invaluable aid in the prototyping and debugging stages of development.

*Centre of the Danish National Research Foundation.

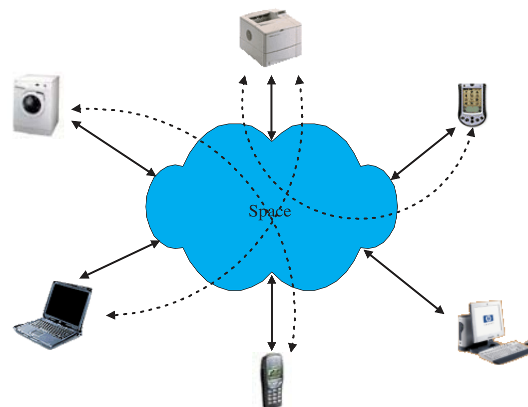

Figure 1. χ-Spaces Architecture

2. The χ-Spaces framework

The χ-Spaces framework is implemented in Java [8] and enjoys the portability typical of Java applications. At an abstract level, its architecture is sketched in Figure 1 and rests on the notion of network middleware or tuple space [9]. Our current χ-Spaces implementation uses an actual tuple space. Different implementations based on other communication mechanisms, are possible by providing a suitable *driver*.

Programming language. The core language is a process language with three main actions: *nonce* creation, *output* and *input*. Input is performed using pattern matching; given a pattern, the input operation will extract a *matching* message from the space. For example:

```
def Initiator(A,B) := {Key(A,B)}
      new(ab).out {(Key(ab),B)}Key(A,B);
    Responder(A,B) := {Key(A,B)}
      in [*C Key(A,B) > ($X,B)];
end
```

Proceedings of the Third International Conference on Application of Concurrency to System Design (ACSD'03)
0-7695-1887-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

This code implements the simple *ISO One-Pass Symmetric Key Unilateral Authentication* protocol [10]. Two roles, `Initiator` and `Responder` are defined. The keys initially known to a process must be declared – in this case the long-term key `Key(A,B)` is shared by both initiator and responder. The initiator creates a new key `Key(ab)` and sends it to the responder encrypted under the shared key. The responder attempts to input, decrypt and get hold of the new key. The expression `Key(ab)` containing a new value `ab` triggers key-generation, in this case of a symmetric key (a number of compiler directives are available to choose the underlying cryptographic algorithms for the protocol). χ-Spaces code is either *interpreted* or *compiled* to Java code. Further information on the χ-Spaces programming language and methodology can be found in [6].

Formal analysis of protocol implementations. The Petri-net semantics of the language supports formal proofs about the security of protocols [1, 2, 5, 6]. Due to its close relation to the implemented transition semantics (see [2]), proved properties are properties of protocol *implementations*. The simple, yet formal transition semantics of χ-Spaces describes the behavior of a system communicating via a tuple space. Our current implementation of the framework using tuple-spaces reflects faithfully what the χ-Spaces semantics formally describes.

3. The χ-Spaces IDE

The case studies presented in [4, 5, 6] are based on *command line* tools. Although the χ-Spaces methodology did smooth the protocol development process (protocols written in χ-Spaces are orders of magnitude smaller than their Java or C++ equivalents) we soon recognized ways to improve it. More specifically, typos in the patterns, debugging based on inserted `print` statements, and, generally, the absence of a graphical representation of protocol runs, became a constant nuisance. The χ-Spaces IDE provides:

Code insight. Protocol code is syntactically highlighted. The mouse hovering over a pattern causes matching messages to be highlighted.

Graphical representation. There is a close relation between the events of SPL-process runs and *Message Sequence Charts* (MSC) [3, 11]; protocol runs can be displayed as MSCs.

Debugging. Step wise execution of the protocols is possible. Every step can be represented graphically into a dynamically evolving MSC.

Simulation. Protocols can be *simulated*. The simulation can be *deep*, meaning that all the cryptographic operations are performed (this gives a rough benchmarking tool, useful in the early stages of protocol design), or *shallow* in which case cryptographic operations are treated formally and not performed. The simulation can be performed in different, even hostile *environments*. If a *spy* process is present, simulation stops when vulnerabilities are discovered in the simulated run. When an attack is found, it is graphically represented as a MSC.

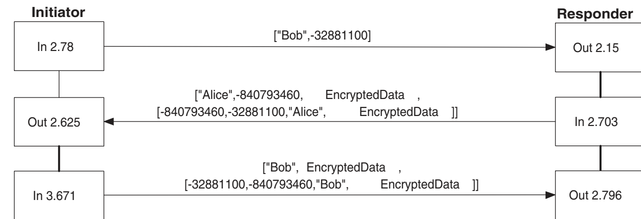

Figure 2. X.509 run

In Figure 2 we can see the MSC generated by the tool for a successful run of the X.509 authentication protocol [12].

4. Future Work

The correspondence between protocol code and the MSC representation of protocol runs drove a number of improvements to the χ-Spaces framework which aid in the task of protocol design and implementation. We plan to further exploit the relation between χ-Spaces processes and MSCs in devising a *visual programming language* where robust protocol code is automatically *synthesized* from a graphical representation of a number of expected runs.

References

[1] F. Crazzolara and G. Winskel, "Petri nets in Cryptographic protocols," in *6th FMPPTA Workshop*, 2001.
[2] F. Crazzolara and G. Winskel, "Event in security protocols," in *Eight ACM CCS Conference*, 2001.
[3] F. Crazzolara, "SPL and MSCs," manuscript, 2002.
[4] G. Milicia, "χ-Spaces: Programming Security Protocols," in *14th NWPT'02*, 2002.
[5] M. Cáccamo, F. Crazzolara, and G. Milicia, "The ISO 5-pass authentication in χ-Spaces," in *SAM Conference*, 2002.
[6] F. Crazzolara and G. Milicia, "Developing Security Protocols in χ-Spaces," in *7th NordSec Workshop*, 2002.
[7] F. Crazzolara and G. Milicia. Wireless authentication in χ-spaces. Technical Report RS-03-10, BRICS, February 2003.
[8] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification 2nd Ed.*. Sun microsystems, June 2000.
[9] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.
[10] J. Clark and J. Jacob, "A Survey of Authentication Protocol Literature: Version 1.0", 1997.
[11] E. Rudolph, P. Graubmann, and J. Grabowski, "Tutorial on message sequence charts," in *Computer Networks and ISDN Systems—SDL and MSC*, vol 28, 1996.
[12] ITU-T, "Recommendation X.509: The directory authentication framework," 1988.

0-7695-1887-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY