SIMULATION IN CRYPTOGRAPHIC PROTOCOL DESIGN AND ANALYSIS

Ning Su
Richard N. Zobel
Frantz O. Iwu
Department of Computer Science
University of Manchester
Oxford Road, Manchester, M13 9PL
United Kingdom
ningsu@lachine.co.uk, rzobel@cs.man.ac.uk, iwuo@cs.man.ac.uk

KEYWORDS
Agent-based simulation, cryptographic protocols, encryption, decryption, Java security features

ABSTRACT

Security and safety were and still are a major concern for distributed computing systems and similar networks of computer environments. A number of cryptographic protocols have been proposed to achieve security and safety of network communication. There is growing interest in using computer simulation to help with understanding, analysing and designing of dynamic complex real systems. This paper studies the Agent-based simulation system on modelling the cryptographic protocols. A general formula of Agent for the simulation of cryptographic protocols has been proposed, and the dynamical environment of the simulation, which includes encryption, decryption of messages, and communication between the Agents, has been achieved by using the Java technology. The simulation system provides an approach for the designer to analyse and verify the cryptographic protocol during the design process.

INTRODUCTION

There are two important aspects concerned in the security and safety of the distributed computing systems and similar networks of computers environment:

(a) Authentication of different computers and users in the network.

(b) Protection of messages passing among them and preventing illegal access of resources.

Data passing over networks is particularly vulnerable to attack. Encryption is necessary when transferring confidential information, such as bank details, ID and personal data, etc. In general, there are two encryption schemes: symmetric (secret-key) encryption and asymmetric (also called public-key) encryption [1]. In the symmetric encryption, the same key is used for encryption and for decryption, and therefore it must be distributed through a secure channel in the first place. In public encryption, there is a key pair consisting of a public key and a matching private key, in which one key is used for encryption and another for decryption. Asymmetric encryption (such as RSA and ECC) has advantages over symmetric encryption in the aspects of security and key management, but usually is much slower, requiring much more computation.

Simulation techniques have proven to be useful for designing real world systems. Modelling and simulation can provide a way of analysing, understanding and optimising the dynamic complexity of real systems. Simulation can also be used to verify reliability and correctness of system designs. The cryptographic protocol can be simulated to help for design, analysis and test before the real system is constructed and deployed.

Java is an object-oriented programming language with a comprehensive set of security and safety measures built in. The multithreading is also supported directly in Java. The simulation environment of cryptographic protocols presented in this paper has been achieved by using the Java technology.

CRPTOGRAPHIC PROTOCOLS

A cryptography system is built on many levels. Building upon the encryption algorithms are protocols. A number of cryptographic protocols have been proposed to achieve security and safety for network communications [2]. It is not sufficient to study the security of the underlying algorithms alone, as a weakness on a higher-level protocol can render the application insecure regardless of how good the underlying cryptographic algorithms are. Protocols involving shared-key cryptography use an authentication server which shares a key with each entity and typically generates new session keys for communication between the entities. Public-key protocols use a certification authority which is trusted to pass on the public keys of the entities. Entities in

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

cryptographic protocols are referred as principals and assumed to have unique identities.

The Otway-Rees Protocol

A share-key authentication protocol was proposed by Otway and Rees in 1987 [3]. The details of the Otway-Rees Protocol are as below:

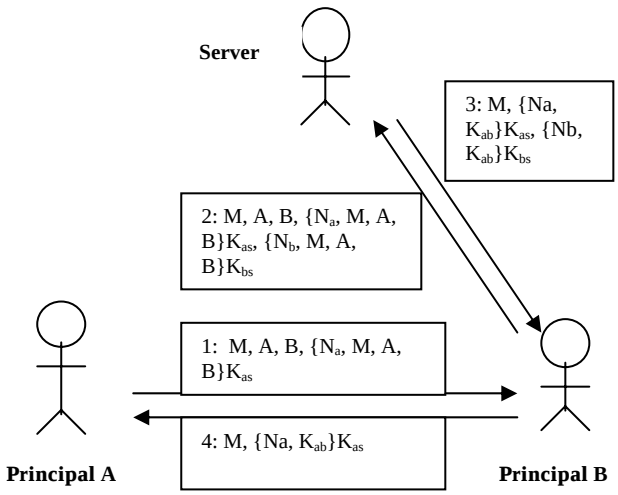

Figure 1 The Otway-Rees protocol

State 1: Principal A sends "M, A, B, {N_a, M, A, B}K_{as}" to principal B.
State 2: Principal B sends " M, A, B, {N_a, M, A, B}K_{as}, {N_b, M, A, B}K_{bs} " to Server.
State 3: Server sends " M, {N_a, K_{ab} }K_{as}, {N_b, K_{ab} }K_{bs} " to principal B.
State 4: Principal B forwards " M, {N_a, K_{ab} }K_{as} " to principal A.

A and B are unique identities of principals A and B.
M, N_a and N_b are specific statements generated by principals A and B.
K_{as} and K_{bs} are shared keys between server and principals A and B.
K_{ab} is a session key generated by server for communication between principals A and B.

In the protocol, principal A sends principal B some encrypted information encrypted with K_{as}, together with enough information for principal B to generate a similar encrypted message with K_{bs}. Principal B forwards both information to the server. The server decrypts the encrypted information and checks whether they match M, A and B. If they are matched, then the server generates session key K_{ab} and embeds it in two encrypted messages using shared key K_{as} and K_{bs} with appropriate statements. The two encrypted messages are sent back to principal B and then principal B forwards the appropriate part to principal A. Principals A and B decrypt the messages and verify the contents. If both satisfied, then principals A and B start to use session key K_{ab} to communicate between them.

A GENERAL FORMULA OF AGENT FOR SIMULATION OF CRYPTOGRAPHIC PROTOCOL

Intelligent agents are one of the most important developments in computer science to have emerged in the past decade [4]. Agent-based simulation can be used to model cryptographic protocols [5] [6]. A conceptual model of the simulation system needs to be introduced to describe how agents may communicate within a simulation environment and formalised using the Discrete Event Simulation (DEVS). DEVS can be used to describe the autonomous and dynamic behaviour of agents and their reaction for events [7]. A cryptographic protocol can be considered to include a set of agents and communication channels. Each agent is an autonomous and reactionary entity (principal) with the capability of performing a sequence of operations (events) on information. A channel is an abstraction of the communication facility. The agents interact each other according to some predefined rules to send, receive and process information via the communication channels. A general formula of agent characterised by a tuple is introduced as below:

$$\Sigma_{Agent} = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \gamma, M, ta)$$

in which,
$X = \{x_1, x_2, \ldots\ldots x_n\}$ *is a non empty set of input events.*
$S = \{s_1, s_2, \ldots\ldots s_n\}$ *is a non empty set of admissible sequence of states.*
$Y = \{y_1, y_2, \ldots\ldots y_n\}$ *is a non empty set of output events.*
δ_{int} : *An internal state transition function describing the behaviour of a finite state automaton.*
δ_{ext} : *An external state transition function describing reaction of the agent to external events.*
λ : *An output function, which maps the internal agent state to the output set.*
γ : *an input/output coupling relationship*
M : $\{m_1, m_2, \ldots\ldots m_n\}$ *is a non empty set of unique component references.*
ta : *Represents the time the agent stays in a particular state before transiting to the next sequential state.*

AGENT SIMULATION MODEL OF OTWAY-REES PROTOCOL

Using the agent definition, agent simulation model for the Otway-Rees protocols is presented. In the model, principal A, principal B and the Server are referred as Agent A, Agent B and Agent S. Figure 2 shows the input and output ports of Agent A, Agent B and Agent S in Otway-Rees Protocol respectively:

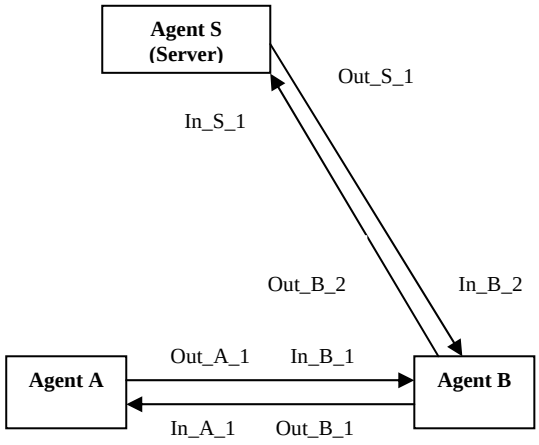

Figure 2: Simulation model of Otway-Rees Protocol

Agent A:

In this model, Agent A has an output port Out_A_1 and an input port In_A_1 for communication with Agent B. A formal description of agent A is shown as below:

$\Sigma_{AgentA} = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \gamma, M, ta)$

X = {In_A_1 }
S = {idle, send, receive, save, retrieve, process_message – verificate, generate, encrypt/decrypt}
Y = {Out_A_1 }.
δ_{int} *(idle) = (make_INI_REQUEST).*
δ_{int} *(cond = true) = (process_message).*
δ_{int} *(cond = true) = (send_message, send).*
δ_{ext} *= In_A_1 = (receive_message, receive).*
λ *(cond = true) = Out_A_1 = message.*
γ *= {(Out_A_1, In_B_1), (In_A_1, Out_B_1)}*
M = {Agent B}
ta = Time value.

The *cond* is a conditional variable, which indicates satisfaction of processing messages. The agent first makes an internal transition from *idle* to *process_message* state. When this occurs, a message is generated and then the state is transmitted to send state. The agent remains in this autonomous state until an external event occurs. Similarly, Agent B and Agent S can be developed trivially [8].

IMPLEMENTATION

The Java technology has been used to achieve the dynamical environment of the simulation. The virtual networking, which uses special address of **localhost,** has been used to establish the simulation environment [9]. In the DEVS models, each agent comprises various parts as shown in figure 3.

Agent name
Declaration of variables
Communication ports
Dynamic Events: Send and receive information; Save and retrieve information; Processing information – verificate, generate, encrypt, decrypt information, etc.

Figure 3 Basic structure of an agent composition

In the model, some agent needs to interact with more than one agent and others just communicate with one agent. The class **socket** is used to establish a communication channel for the agent without multi-communication capability:

Socket s = new Socket ("localhost", port number);

The multi-communication capability of agent is achieved by combining classes **Serversocket** and **Thread**:

ServerSocket ser = new ServerSocket (port number);

And a loop for the thread class:

```
While(!false)
{
Socket s =ser.accept();
Thread t= new ThreadedHandler (s);
t.start();
}
```

The class of Threadedhandler drives from class Thread and contains the communication loop with the other agent in its run method. Thus, for each new connection with a new agent (i.e. each new socket connection), a new thread will be launched to take care the communication and therefore the agent can communicate with more than one agent at the same time. Different types of information (text, binary data and serialized object) can be transferred by using different classes. In the simulation model, all information (plaintext or ciphertext) are converted to binary data and then transmitted [8].

In order to using the Java security features, packages **java.security** and **javax.crypto** should be imported into the Agent (Java program). If using RSA algorithm, a new provider ("Legion of Bouncy Castle") needs to be added in the security description file [10] (*jre/lib/security/java.security*):

Security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider

The following classes have been used for generating secret key:

```
KeyGenerator keygen = KeyGenerator.genInstance
                ("algorithmName");
SecureRandom random = new SecureRandom();
Keygen.init(random);
SecretKey key = keygen.generateKey();
```

In the simulation models, DES cipher algorithm is used, so

```
algorithmName = DES;
```

For generating asymmetric keys, following classes have been used:

```
KeyPairGenerator keysgen =
        KeyPairGenerator.genInstance
        ("algorithmName", "providerName");
SecureRandom random = new SecureRandom();
Keysgen.initialize(keysize, random);
KeyPair keys = keysgen.generateKeyPair();
Key  publickey = keys.getPublic();
Key privatekey = keys.getPrivate();
```

In the simulation models, RSA cipher algorithm is used and the provider is BC (Legion of Bouncy Castle), so

```
algorithmName = RSA;
providerName= BC;
```

The class *Cipher* is used for all encryption algorithms:

```
Cipher cipher =
        Cipher.getInstance("algorithmName",
        "providerName");
```

Details of the implementation of cryptosystem are presented in [8].

SIMULATION RESULTS

The computer used for the simulation is a PC (Pentium 2.0GHz, 256MB memory, Windows XP). Java™ 2 SDK, Standard Edition Version 1.4.0 has been installed in the PC.

As one window can only run one Agent, multiple windows should be opened simultaneously for the simulation of the protocols.

Symmetric key DES and asymmetric key RSA have been used throughout the simulation.

First, a simple simulation model of Otway-Rees Protocol was implemented to test the cryptosystem and communication between Agents. In the test, the plaintext of Agent A was encrypted using session key *Key_AB* that was generated by Agent S (Server) and then sent to Agent B. Agent B received the encrypted message and decrypted it using the same key. The screen output of the test is shown in Figure 4 and 5. The results show that the encryption, decryption and transfer of the message have been successfully carries out.

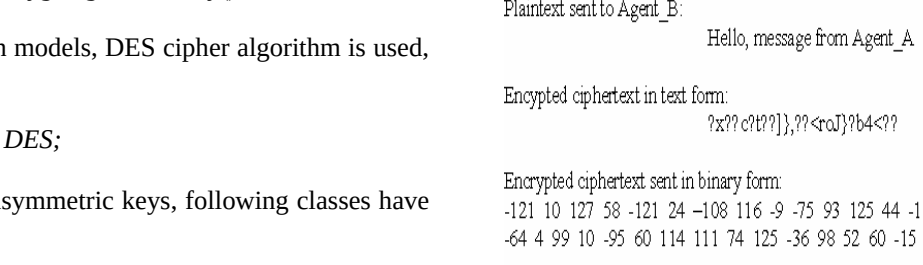

Figure 4 Agent A's message (plaintext and ciphertext) sent to Agent B

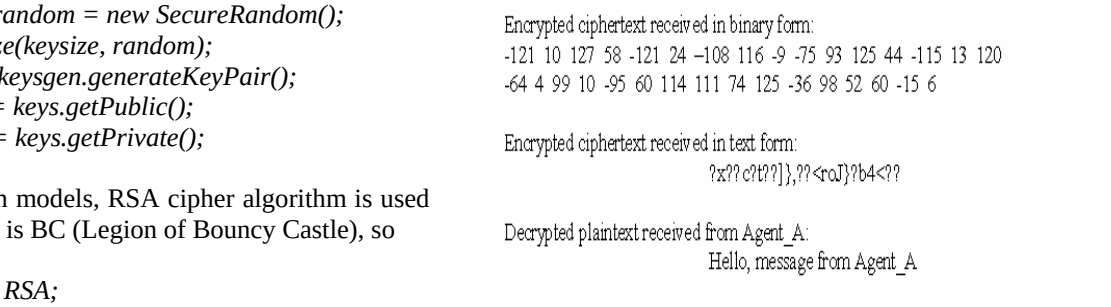

Figure 5 Agent A's message (ciphertext and plaintext) received by Agent B

The simulation models of four Cryptographic protocols (Otway-Rees, Needham-Schroeder, Kerberos and Digital Envelope protocols) have been developed and details of simulation results are presented in [8].

In the simulation of Otway-Rees Protocol, three windows have been opened simultaneously for Agent A, Agent B and Agent S respectively. The interaction between the Agents in the protocol is shown in Figures 6-8. The execution sequences are as following:

(1) Start Agent B to wait for communication.
(2) Start Agent A to make initial contact with Agent B. Agent A is then awaiting an input from keyboard after sending the message to Agent B. If **CON** is inputted, the state will be transmitted to the next state.
(3) Start Agent S to wait for communication with Agent B.
(4) Agent B contacts with Agent S.
(5) Agent S replies to Agent B's request.
(6) Agent B receives Agent S's message.
(7) Agent B verifies the message.
(8) Agent B passes the Agent S's message to Agent A.
(9) Agent A verifies the message.
(10) Secure communication is then established between Agent A and Agent B if satisfied.

Communication with Agent_B:

State 1: retrieve secret key key_AS for communication between Agent_A and Agent_S
State 2: Generate message to Agent_B: M, A, B, {Na, M, A, B}k_AS
State 3: Encrypt {Na, M, A, B} using Key_AS
State 4: Send the message to Agent_B

Enter END to exit, Enter CON to continue

CON
State 5: Received encrypted Agent_S's message via Agent_B
State 6: Decrypt the message using K_AS
State 7: Verify the received message

Verification satisfied

State 8: Unwrap the wrapped key_AB if satisfied
State 9: Decrypt the message from Agent_B using key_AB

Message received from Agent_B: Hello, message from Agent_B

State 10: Send an encrypted message to Agent_B using key_AB

Message sent to Agent_B: Hello, message from Agent_A

Figure 6 Agent A's execution in Otway-Rees Protocol

Communication with Agent_A:

State 1: Received Agent_A's message

Communication with Agent_S:

State 2: Retrieve secret key key_BS for communication between Agent_B and Agent_S
State 3: Generate Agent_B's message
State 4: Send the message (including Agent_A's message) to Agent_S
State 5: Received Agent_S's message
State 6: Verify the received message from Agent_S

Verification satisfied

State 7: Unwrapped session key: key_AB

Communication with Agent_A:

State 8: Generate Agent_B's message to Agent_A

Message sent to Agent_A: Hello, message from Agent_B

State 9: Encrypted Agent_B's message using key_AB
State 10: Send encrypted Agent_B's message and pass Agent_S's message to Agent_A

State 11: Received encrypted Agent_A's message
State 12: Decrypt Agent_A's message using key_AB

Message received fromt Agent_A: Hello, message from Agent_A

Figure 7 Agent B's execution in Otway-Rees Protocol

Communication with Agent_B:

State 1: Retrieve secret key Key_AS and Key_BS
State 2: Received Agent_B's message (including Agent_A's message)
State 3: Verify the received message

Verification satisfied

State 4: Generate message including session key key_AB for communication between Agent_A and Agent_B
State 5: Encrypt the message with key_AS and key_BS respectively
State 6: Send the message to Agent_B

Figure 8 Agent S's execution in Otway-Rees Protocol

APPLICATION OF THE SIMULATION

The simulation of cryptographic protocols could have various applications. For example, the efficiency of the protocols including encryption/decryption versus message length can be assessed. Attack activity can be also modelled to assess the security design of the protocols. An attack Agent has been developed to simulate the attack in Kerberos protocol, which retrieved the session key (*key_AB*) and then contacted Agent A using this key, as shown in Figure 9. In the simulation, an extra window was opened to run the attack Agent. Once Agent A received the communication encrypted by key_AB, the validation of the key has been checked as there is a lifetime for the key imposed by Agent S (Server) in Kerberos protocol. In the case, the attacker has used an out of date session key and therefore the verification has failed, as shown in Figure 10.

C:\MSc\Model>javac Agent_ATTACK_KP.java

C:\MSc\Model>java Agent_ATTACK_KP

Try to communicate with Agent_A using key_AB:

State 1: Retrieve key_AB:
State 2: Generate a message to Agent_A:
State 3: Encrypt the message using key_AB:
State 4: Send the message to Agent_A

Figure 9 Screen output of Agent Attack's execution in Kerberos Protocol

```
C:\MSc\Model>javac Agent_A_KP.java

C:\MSc\Model>java Agent_A_KP

Communication with Agent_S:

State 1:  Generate Agent_A's message to Agent_S

Message sent to Agent_S: A, B

State 2:  Send the cleartext message to Agent_S
State 3:  Received encrypted Agent_S's message
State 4:  Retrieve key_AS
State 5:  Decrypt the message using key_AS
State 6:  Verify the received messages: Lifetime L and B

Lifetime received from Agent_S, the key_AB is invalid after: 2002-12-07
15:06:36.154

Current time: 2002-12-07 15:04:57.638

Verification satisfied

Communication with Agent_B:

State 7:  Generate Agent_A's message to Agent_B: {A, Ta}Key_AB

Timestamp Ta: 2002-12-07 15:05:01.326

State 8:  Using key_AB to encrypt Agent_A's message
State 9:  Send the messages including Agent_S's to Agent_B

Message sent to Agent_B:  {Ts, L, key_AB, A}key_BS, {A, Ta}key_AB

State 10:  Received Agent_B's reply message
State 11:  Verification

Current time: 2002-12-07 15:05:02.622

Verification satisfied

Reply message received from Agent_B: 2002-12-07 15:05:01.357 + 1

State 12:  Received further message encrypted by key_AB
State 13:  Verification

Current time: 2002-12-07 15:07:03.06

Verification failed: the key is out-of-date!
```

Figure 10 Screen output of Agent A under attack in Kerberos Protocol

CONCLUSIONS

Dynamical environment of Agent-based cryptographic protocols simulation system has been achieved using Java technology. The Java's networking and multithreading features have been used to establish communications between Agents in the protocol. The Java's security technology has been applied to generate keys (secret key and public key schemes), encrypt and decrypt messages.

A general formalism of Agent for simulation of cryptographic protocols has been proposed and implemented. Otway-Rees protocol has been successfully modelled using the Agent-based simulation system.

It was suggested that the Agent-based cryptographic protocols simulation system could have various application. The simulation system could be used to analyse, verify and assess design of the protocols, including correctness, weakness, reliability, efficiency of protocols. The simulation system could be also used to model attack activity.

REFERENCES

[1] Http://www.ssh.com/support/cryptography/

[2] Michael Burrows, Martin Abadi and Roger Needham, A Logic of Authentication, SRC Research Report 39, February, 1990

[3] Otway, D. & Rees, O. 1987 Efficient and Timely Mutual Authentication. Operating Systems Review Vol. 21, No.1, pp. 8-10.

[4] Agent Technologies (http://www.insead.fr/CALT/Encyclopedia/ComputerSciences/Agents/)

[5] Frantz O. Iwu, PhD Thesis, Manchester University, 2003.

[6] Frantz O. Iwu and Richard N. Zobel, UK SIM Conference, Cambridge, April 2003.

[7] B. P. Zeigler, Multi-Facetted Modelling and Discrete Event Simulation. Academic Press, 1984.

[8] N. Su, Simulation In Cryptographic Protocol Design And Analysis, MSc Thesis, Department of Computer Science, University of Manchester, 2003.

[9] Java2 SDK, Standard Edition, Documentation Version 1.4.0

[10] Http://www.bouncycastle.org/index.html

NING SU obtained his BSc and MSc in Engineering Mechanics at Zhejiang University and Xian Jiaotong University in China respectively. He got his PhD in Civil Engineering at the University of Dundee and MSc in Computer Science at the University of Manchester in the UK. He is currently working for LaChine LinCom Limited and his interests include Java technology, distributed simulation and network security.

RICHARD ZOBEL graduated with BSc(Eng) (London) in 1962, and with PhD (Manchester) in 1970. He is a C.Eng, and Member of both BCS and IEE. He has 118 publications, including a book and a patent, and has supervised 100 postgraduate students. He retires fully from Manchester University Computer Science Department end October 2003 after 37 years service as Lecturer and Senior Lecturer. He is a Member of UKSim, and of SCS from whom he received an Outstanding Service Award in 2002. His interests are currently distributed simulation, distance learning and network security.

FRANTZ IWU is a research associate at the University of York. He obtained his MSc in Advanced Computer Science and his PhD from Manchester University, United Kingdom. His research interest includes distributed simulation systems, security for distributed simulation under commercial network protocols, application of HAZOP to computer systems, system safety analysis, software flow failures and fault propagation, systematic development of large software systems from reusable fragments using the B-Method. He is a member of the British Computer Society.