# Security Analysis of an E-commerce Solution

Yazan El-Hamwi

A Thesis

in

The Concordia institute

of

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

July 2008

*Your file Votre référence*
*ISBN: 978-0-494-42519-0*
*Our file Notre référence*
*ISBN: 978-0-494-42519-0*

# Abstract


**Security Analysis of an E-commerce Solution**

Yazan El-Hamwi

The escalation in the number of people with access to the Internet has fuelled the growth of e-commerce transactions. In order to stimulate this growth in e-commerce, the adoption of new business models will be required. In this thesis, we propose the idea of bringing the multi-level marketing business model into the e-commerce world. For e-commerce applications to take advantage of the business potential in this business model, some challenging security problems need to be resolved.

Our proposed protocol provides a method for fair exchange of valuable items between multiple-parties in accordance with the multi-level marketing business model. It also provides the required security services needed to increase the overall customers' trust in e-commerce, and hence increase the rate of committed online transactions. These security services include content assurance, confidentiality, fair exchange and non-repudiation.

The above security services are usually attained through the use of cryptography. For example, digital rights management systems deliver e-goods in an encrypted format. As these e-goods are decrypted before being presented to the end user, cryptographic keys may appear in the memory which leaves it vulnerable to memory disclosure attacks. In the second part of this thesis, we investigate a set of memory disclosure attacks which may compromise the confidentiality of cryptographic keys. We demonstrate that the threat of these attacks is real by

exposing the secret private keys of several cryptographic algorithms used by different cryptographic implementations of the Java Cryptographic Extension (JCE).

# Acknowledgement

I would like to express my deepest sense of gratitude to my supervisor, Dr. Amr Youssef, for his patient guidance, encouragement and advice throughout this study.

Special thanks to Dr. Willian Atwood and Dr. Mourad Debbabi for their help in the protocol verification part of this thesis.

I am also thankful to my lab-mates: Hamad Ben Saleeh , Saad Inshi, Mohamed Raslan, and Esam Elsheh for helping me throughout my research. Appreciation goes to my friend Hani Safadi at McGill University for his valuable opinions.

Finally, I take this opportunity to express my profound gratitude to my beloved parents, wife and brothers for their love and support during my studies at Concordia University. This thesis would not have been possible without their love.

# List of Contents

# List of Tables

# List of Figures

# List f Acronyms

| | |
|---|---|
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| CA | Content Assurance |
| CBC | Cipher-Block Chaining |
| CEGAR | Counter Example Guided Abstraction Refinement |
| CFB | Cipher feedback |
| CL | Client |
| COM | Commission |
| CSP | Cryptographic Service Provider |
| DES | Data Encryption Standard |
| DRM | Digital Rights Management |
| DST | Distributor |
| EP | Encrypted Product |
| EPA | Encrypted Product Assurance |

| | |
|---|---|
| FIFO | First-In First-Out |
| GUI | Graphical User Interface |
| HTTP | Hypertext Transfer Protocol |
| J2SE | Java 2 Platform, Standard Edition |
| JCA | Java cryptography Architecture |
| JCE | Java Cryptography Extension |
| LTL | Linear Temporal Logic |
| MER | Merchant |
| MLM | Multi-Level Marketing |
| MRMC | Markov Reward Model Checker |
| OFB | Output Feedback |
| OP | Order of Purchase |
| GR | Good Request |
| PROMELA | Process or Protocol Meta Language |
| RAM | Random Access Memory |
| REF | Referral |

| | |
|---|---|
| ROP | Referred Order of Purchase |
| RSA | Public Key Cryptographic algorithm described by Rivest, Shamir, and Adleman |
| SET | Secure Electronic Transaction protocol |
| SSL | Secure Socket Layer protocol |
| STTP | Semi Trusted Third Party |
| TLS | Transport Layer Security protocol |
| TO | Transaction Order |
| TTP | Trusted Third Party |

# Chapter 1

# Introduction

Electronic commerce, commonly known as e-commerce, consists of buying and selling products, information, and services over the Internet [1]. It provides customers with the flexibility of purchasing without leaving the houses with just few clicks. The web development and the increasing number of Internet users are contributing to the e-commerce business revolution. Purchasing any merchandize on the Internet has never been so popular. E-commerce not only includes Business-to-Consumer (B2C) e-commerce, but also includes Business-to-Business (B2B) e-commerce where the transactions are conducted between companies rather than between a company and a customer [1].

E-commerce transactions allow the purchase of valuable electronic goods (e-goods) from online websites such as e-bay (e-purchase). The required security services for an e-commerce protocol include confidentiality, fair exchange, non-repudiation, and content assurance [2].

Non-repudiation consists of two parts: non-repudiation of origin and non-repudiation of receipt. Non-repudiation of origin guarantees that data or product has originated from the claimed source, while the non-repudiation of receipt guarantees that the data has been received by the claimed entity. It protects parties against each other from falsely denying having sent or received an item during a transaction. While Fair exchange ensures that either all involved

parties receive expected items or no party receives any valuable item. It guarantees that a merchant and a customer involved in a transaction receive the payment and the e-good respectively. Content assurance security service assures the receiver of an item to be delivered the expected e-good item that was ordered.

In this thesis, we investigate some aspects of the multi-level marketing business model. One example of this model is the case when a special user, *distributor*, refers a merchandize provided by a merchant to a customer, and the customer goes on and buy that merchandize from the merchant and the *distributor* gets a commission for the referral. This is a typical transaction in the multi-level marketing model. We propose bringing this business model into the e-commerce world. Our first contribution in this thesis, is the proposal of a multi-party fair exchange protocol; a protocol that can be used as a building block to provide the ability to securely and fairly exchange e-goods among multiple parties in a Digital Rights management system according the multi-level marketing business model [3].

In order to provide confidentiality and other security services for e-goods and other components of a digital rights management systems, we use cryptography not only to secure communication but also applications. To provide strong protection, cryptography depends on the assumption that cryptographic keys are kept secret. This assumption is hard to assure, not because a cryptographic algorithm is not computationally strong enough, but because of the existence of side channel attacks that may reach used cryptographic keys while in use in memory. Such attacks make it relatively easy to compromise sensitive information such as cryptographic keys. In this thesis, we investigate a class of attacks, namely memory disclosure

attacks [4] which exploit memory disclosure vulnerabilities to compromise the confidentiality of cryptographic keys. We demonstrate that the threat is real by formulating an attack that exposes private and secret keys used by many cryptographic implementations of the Java Cryptographic Extension (JCE).

## 1.1 Aims and Objectives

One of our main objectives of this research work is to increase the user's confidence in e-commerce by attracting new customers to the e-commerce business through the facilitation of implementation of a new business model, namely the multi-level marketing model. Exchanging items between multiple parties in a fair way is an important building block in committing e-commerce transactions. In particular, we try to rise above the inherent unfairness when exchanging valuable items between parties on the Internet. The proposed protocol should be able to safeguard the involved parties form all types of fraud and misuse that disrupt e-commerce transactions. It should meet the security requirements of all parties and does not discriminate against any well behaving party.

Such a system would increase the overall customer`s trust in e-commerce and increase the rate of committed online transactions.

The proposed multi-party exchange protocol facilitates the exchange of products between parties which is a core operation in an e-commerce business. Exchanging items between multiple parties facilitates the bringing of multi-level marketing business model into e-commerce. This exchange can also empower digital rights systems by allowing a secure exchange of e-goods.

Our other emphasis, in this thesis, is the evaluation of the security services provided by cryptographic libraries which is heavily used in securing e-goods delivered by many e-commerce applications. These cryptographic libraries provide the security services needed for securing many communications, applications and systems. These libraries are successful in these services as long as it is successful in keeping sensitive information such as passwords and private keys secret. We explore *Memory disclosure attacks* against some of these popular cryptographic libraries. In Particular, we show that some popular implementations of the Java Cryptography Extension (JCE) have memory disclosure vulnerabilities. We describe the vulnerabilities, for many symmetric and public key algorithms. Then we discuss a set of counter measures to these vulnerabilities.

## 1.2 Achievements and Contributions

The presented research work in this thesis has led to the following contributions and achievements:

- The design of the multi-party fair exchange protocol, a protocol that facilitates the fair exchange of valuable items between multiple parties according the multi-level marketing business model. The exchange of items is done among a distributor, a client, and a merchant with the aid of a trusted third party. Our protocol provide the following features:

a. A fair exchange of items. That is, as the distributor provides a referral, she gets a commission, as the client pays for a product or service she gets it, and asthe merchant provides the product or service she gets paid. While the role of the trusted third party is to ensure either all parties get their expected items or none of them does. Our protocol provides *fairness* in the sense that either all parties get their expected items or none gets any valuable item [2].

b. Content assurance of the delivered items. That is each one of the three parties involved in the protocol should get her exact expected item. In a media distribution system, a customer who has bought a song or a book with a certain quality in a certain format, should get her song or book in the exact right specification that she has asked for and so does the merchant and distributor; they should get their exact amount of payment.

We provide informal and formal security analysis of the protocol. We achieve the formal verification analysis of the fairness property by modelling the protocol using PROMELA modelling language, then running the SPIN model checker to verify the fairness property described in Linear Temporal Logic against the PROMELA model of the protocol [18]. Verification of the security requirements and the protocol simulation provide assurance of the correctness of the protocol and its security properties to an acceptable level.

In order to secure a distributed system or application they have to run in a secure environment. A system is assumed to be secure only as long as it is successful to keep the

cryptographic keys secret that it uses safe. The security of entire system becomes vulnerable once its cryptographic keys or passwords are compromised.

We describe scenarios of locating cryptographic keys and passwords in memory by running multiple experiments on multiple cryptographic algorithms. Our main focus is to detect key signatures. These key signatures allow locating cryptographic keys used by the implementations of the Java Cryptography Extension (JCE) in the memory. We will present the key signature of both symmetric and public-key algorithms. Finally, we propose a set of countermeasure that makes exploiting the described vulnerabilities more difficult but unfortunately still possible.

## 1.3 Thesis Outline

The rest of the thesis is organised as follows:

- In Chapter 2, we describe the multi level marketing business model, then we move into presenting the design of the multi-party fair exchange protocol and sub protocols. We also describe how our multi-party fair exchange protocol can serve into an online service for selling products and services on the net and still adopt the multi-level marketing business model.
- In Chapter 3, we conduct formal analyses of the fairness property of our proposed multi-party fair exchange using the SPIN model checker.
- In Chapter 4, we present the vulnerability of locating and extracting cryptographic keys used by many applications and servers in the memory.

- Finally, our conclusions and suggestions for possible future research work are presented in Chapter 5.

# Chapter 2

# Multi-party Fair Exchange Protocol for Multi-level Marketing

## 2.1 Multi-Level Marketing

Multi-level marketing, also known as "network" marketing and "matrix" marketing, is a combination of direct marketing and franchising. It is a business model that adds the power of networking (sharing of resources, connection with people, and information) to the traditional marketing process.

Statistics show that large percent of the cost of any product or service you pick up is not in manufacturing but in distribution [11]. Multi-level marketing is a business model that is booming business all around the world; it is used in all states of the Unites States and here in Canada where it became a growing multi-billion dollar industry. The reputation of multi-level marketing has been affected by "pyramid" schemes and Ponzi scams which are illegal in Canada and most other countries [11].

The difference between traditional marketing and multi-level marketing is that in traditional marketing, the product reaches the consumers by a chain of middlemen who act as a link between the manufacturer and the consumer. Each middleman gets his profit from the difference between the price of selling and the price of buying. However, in Multi-level marketing, people signing up with the company as distributors receive commissions for the sales they made and whoever they recruit in the "downline".

### 2.1.1 Abusing Multi-Level Marketing

In the most illegal "pyramid" schemes, participants attempt to make money solely by recruiting new participants (clients) into the program. The Ponzi scheme is an illegal pyramid scheme named after Charles Ponzi, who took advantage of thousands in England, who invested their money in a postage stamp speculation scheme back in 1920s [11].

Usually legal pyramid schemes are associated with the sales of a product or service, and this could help in recognizing the legality of some multi-level marketing schemes. Another theme that distinguishes illegal "pyramid" schemes is the promise of a very high return in exchange for doing nothing but to handing over your money, and encouraging others to do the same. This is caused by the difficulty in providing the promised profits in pyramid schemes.

## 2.2 The Multi-Party Fair Exchange Protocol

The purpose of this protocol is to allow the three parties to perform an online transaction where each one of them is sure that the exchange of items, and money, is done in a fair and secure way. The three parties involved in this transaction are the distributor, client and merchant in the following described way.

First, the distributor refers a product or service to a client in exchange for a commission on the transaction that will take place between the client and the merchant. Then, the client buys the product or the service from the merchant. The client presents a referred order of purchase to the merchant and she expects to get the product or service in return. As, the merchant wants to get paid in exchange for selling a product or providing a service to the client, the merchant understands that she will pay a Commission to the distributor who referred the client.

## 2.3 Multi-Level Marketing and The Multi-Party Fair Exchange Protocol

The multi-party exchange protocol can be the core protocol of an online implementation of multi-level marketing for selling products and services on the net. Multi-level marketing transactions take place between three parties: the distributor, the clients and the merchant, where the distributor recommends a service or a product to the client who can obtain the product or service online from the merchant. Once the client does so, the three parties get their expected items from the transaction committed with the aid of a Trusted Third Party (TTP) .This is exactly what our protocol is trying to accomplish by trying to make sure that the distributor gets his/her

commission, the client gets his/her product or service, and the merchant get paid for what has sold.

Building a system on the top of this protocol will transfer the traditional multi-level marketing model to the e-commerce world. Such a transfer would require a protocol that can act as the base for the exchange transaction. We believe that our protocol is an attempt to fulfill this role.

Taking the multi-level marketing model into an online model would require more than the core protocol. It would require a complete system that should include all aspects of payment, the hierarchical commissioning for distributors, and content distribution, etc.

### 2.3.1 TTP Involvement

The involvement of a trusted third party is important to have a low communication cost. A high communication cost occurs in graduate exchange protocols which do not involve a trusted third party in the transaction. It is also proven in Pagnia and Garnter [6] that exchange protocols over the Internet cannot achieve strong fairness without the involvement of a TTP [2]. Two-party protocols cannot achieve this property. Many Two party fair exchange protocols [7, 8, 9, and 10] that do not require the involvement of a TTP have been proposed but they fail to provide practical implementation of fairness. The help of a trusted third party in our protocol has helped in achieving the following security requirements:

### a. The Need for Manual Dispute Resolution

In order to provide true fairness, we use an online TTP which allows all parties to get their expected items and no harm is caused to any party under all circumstances even if one party misbehaves or permanently disconnects. In addition, phase three (recovery sub-protocol) acts as a dispute resolution protocol that provides this dispute resolution in a predefined automated way that does not require any manual involvement.

### b. Fair Exchange

Researchers have defined several levels for fair exchange including strong fairness, true fairness, and weak fairness [2]. But in general we describe an exchange to be fair if at the end of the exchange, either all parties get their expected items in the transaction or no party gets any piece of information about any item of other parties. The TTP helps achieve fair exchange between multiple parties and makes sure that all parties get their items at the end of the protocol even if one party or more behaves unfairly.

### c. Content Assurance

While fair exchange ensures that each party gets his item, content assurance ensures that each party gets the expected correct item it is waiting for. Phase one (the acquiring encrypted product sub-protocol) assures the client that the product that it will be purchasing matches a certified

description of the product, and prevents the merchant from providing a different product than the one the client requested. The involvement of the TTP allows ensuring this property.

We tried to reduce the overhead of the involvement of the TTP so that it does not become a performance or a security bottleneck. We tried to reduce the communication cost by reducing the number of messages exchanged with the TTP and reducing the processing cost by reducing the amount of processing of information and messages required by the TTP.

## 2.4 Multi-Party Fair Exchange Sub-Protocols

This protocol is composed of three phases or sub-protocols and each one of them has a smaller objective but if combined together they achieve the protocol overall objectives, these sub-protocols are:

1. Acquiring encrypted product sub-protocol: This sub-protocol delivers the Encrypted Product to the client, and provides content assurance.
2. Multi-party Exchange sub- protocol (Main Protocol): This sub-protocol allows the exchange Referral, payment decryption key, Receipt and Commission between the three parties, distributor, client, and merchant with the help of the trusted third party.
3. Recovery sub-protocol: This sub-protocol guarantees the achievement of the fairness property; this phase is considered a complementary of the Multi-party Exchange sub-protocol.

### 2.4.1 Phase 1: Acquiring Encrypted Product Sub-Protocol

The aim of this phase of the protocol is to deliver the Encrypted Product to the client, and to assure the client that the product that it received matches the specification described (the properties related to the content and its quality).

The client starts this phase by sending a good request *(GR)* to the merchant asking for a product using identification and specification information of the product included in the *(GR)*. The merchant replies with the Encrypted Good *(EG)* and the Encrypted Product Assurance *(EPA)* which provides content assurance to the client.

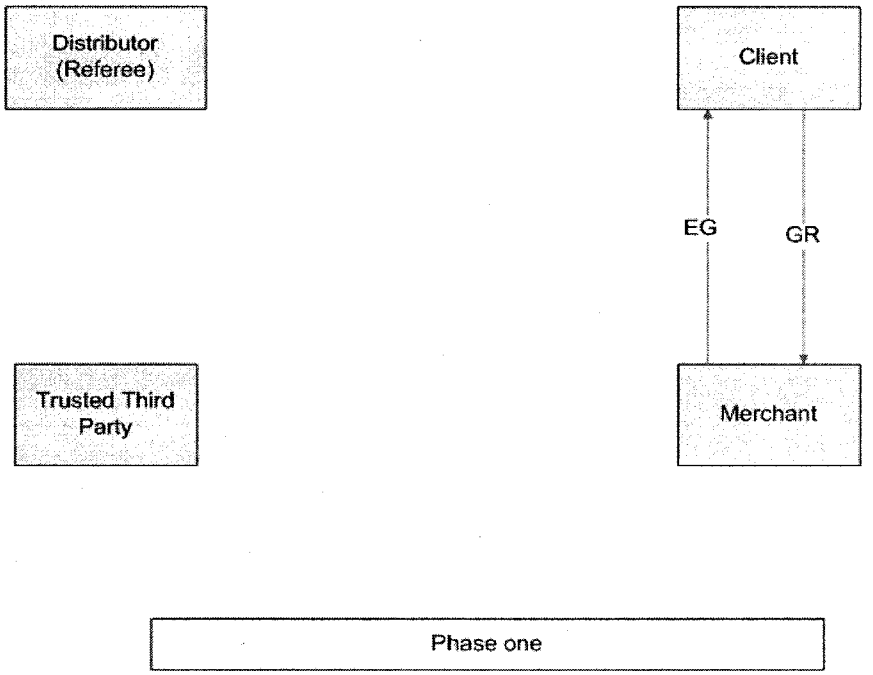


Figure 2.1: Acquiring encrypted goods sub-protocol description

| | | | |
|---|---|---|---|
| CL | → | MER | :GR |
| MER | → | CL | :EP, EPA |

**Message Description**

| | |
|---|---|
| **GR** | Good Request |
| **EG** | Encrypted Good (Product) |
| **EPA** | Encrypted Product Assurance |

**Table 2.1:** Acquiring encrypted goods sub-protocol description

This sub-protocol delivers an encrypted copy of the client requested product from the merchant to the client. The client has to make sure that the encrypted product matches the description provided by the merchant and signed by the trusted party by doing simple mathematical operations.

$$EPA = Sign_{TP}(Desc_P, H_P, H_{EP}, H_k)$$

Where:

$Desc_P$ is the description of product, its quality, and every other useful info that describes the product.

$H_P = H(P)$ is the hash value of the product *(P)*.

$H_{EP} = H(EG)$ is the hash value of the Encrypted product *(EG)*.

$H_k = H(K)$ is the hash value of the decryption Key K of the Encrypted Good *(EG)*.

$Sign_{TP}$ is the signature of the Trusted Party on the previous items.

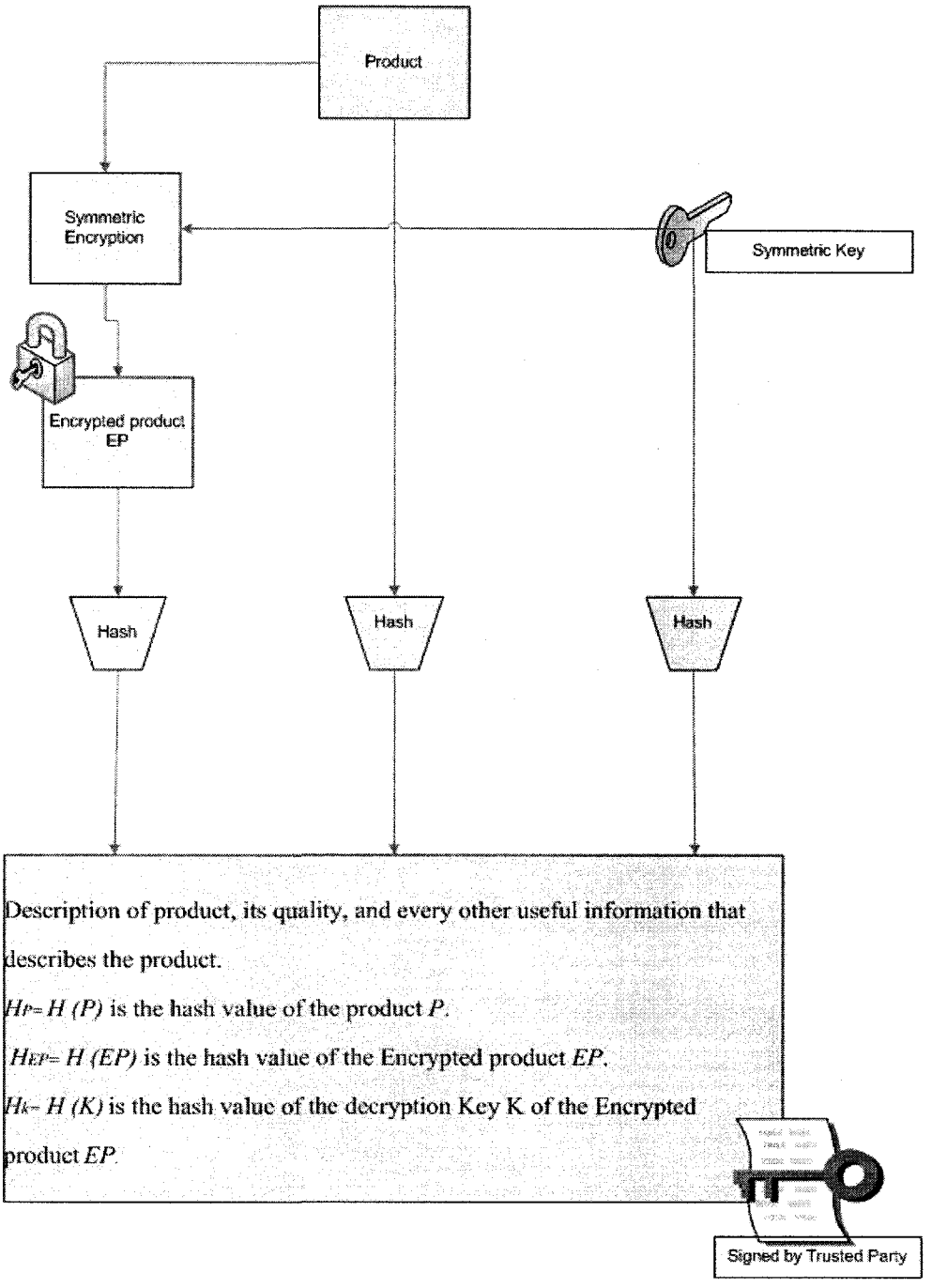


**Figure 2.2:** Encrypted Product Assurance

EPA links the encrypted good (EG), its description, its decryption key, and the product itself all together. The hash value of the product gives the client the ability to verify the product once she decrypts the EP while the hash value of the key allows the client to verify that she received the right decryption key from the merchant once she buy the product from that merchant. The digital signature provided by the trusted party ensures the correctness of this information, including the description of the product, hash value of it, the hash value of its encryption and the hash value of the decryption key.

### 2.4.2 Phase 2: Multi-Party Exchange Sub-Protocol (Main Protocol)

The purpose of the Multi-party Exchange sub-protocol and the Recovery sub-protocol is to perform a transaction between three entities: distributor, client, and merchant with the help of a Trusted Third Party (TTP). The transaction should provide fairness in a way that each party will obtain his/her expected items, so that the client gets the key for the Encrypted Product that matches the encrypted product description, the distributor gets his/her commission, and the merchant is credited for the price of the product.

The distributor starts this sub-protocol by sending a Referral (REF) to the client. The existence of a lookup service (LUS) can help the distributor find clients. Details of the client discovery are outside the scope of this thesis. This Referral should be signed by the distributor to prove the originality of the Referral. The client uses this Referral to order the referred product from the merchant by sending a Referred Order of Purchase (*ROP)* which should include the Referral, the Encrypted Good (EG), the product description, the product price and other items.

The payment is encrypted in a way that allows the trusted third party but not the merchant to access the payment information; the payment can be encrypted using the public key of the trusted third party. The Referred order of purchase *ROP* should be signed by the client to prove the originality of the order and to help in dispute resolution.

Once the merchant receives the Referred Order of Purchase she can proceed with the transaction by sending a Transaction Order (*TO*) to the trusted third party. This Transaction Order should be signed by the merchant to prove the originality of the order. The Transaction order should contain the Referral Order of Purchase with all its sub elements including the Referral, the encrypted payment, the product description, the product price, the product decryption key, and any other needed items.

When the trusted third party receives the Transaction Order, it does a series of verifications:

- It verifies the origin of the Referral, the Referred Order of Purchase, and the Transaction Order messages by checking the signature of the distributor, client, and merchant respectively.|
- It verifies the correctness of the encrypted payment information provided by the client. The trusted third party begins by decrypting the payment information using the *TTP* private key, and then it checks its validity (For example, it checks the validity of the credit card information).
- It verifies the signature of the Encrypted Product Assurances (EPA), and matches the information inside it including the requested product with the referred product by the distributor, and the requested product by the client. It also makes sure that the hash value of the key provided by the merchant matches the hash value of the key in the *EPA*.

If all the verification indicates a valid transaction, the trusted third party commits the transaction by doing the following:

- Charging the client for the purchase of the product.
- Crediting the merchant for the cost of the product.
- Crediting the distributor with a Commission for his Referral.
- Generating a Receipt (*RCT*) for the previous money transfers. The Receipt should also include the decryption key provided by the merchant, and a Commission that should be forwarded to the distributor. This Commission should be signed with the *TTP* private key.

If any of the verification indicates an invalid transaction for some reason, for example, the client payment information is not valid or the product decryption key does not match the one in the *TTP* signed Encrypted Product Assurance *EPA,* then the *TTP* will proceed by doing the following:

- The *TTP* will not charge the client, use the payment information, credit the merchant or the distributor, or does any money transfer.
- If the transaction was not completed because of the invalidity of the payment information, the *TTP* sends a No Money (*NM)* message signed by the *TTP* private key to the merchant indicating that no transaction took place because of the invalidity of the client payment information.

- If the transaction was not completed because of any reason other than the invalidity of the payment information, the *TTP* sends a No Transaction *(NT)* message signed with the *TTP* private key to the merchant indicating that no transaction took place specifying the reason for the incomplete transaction.

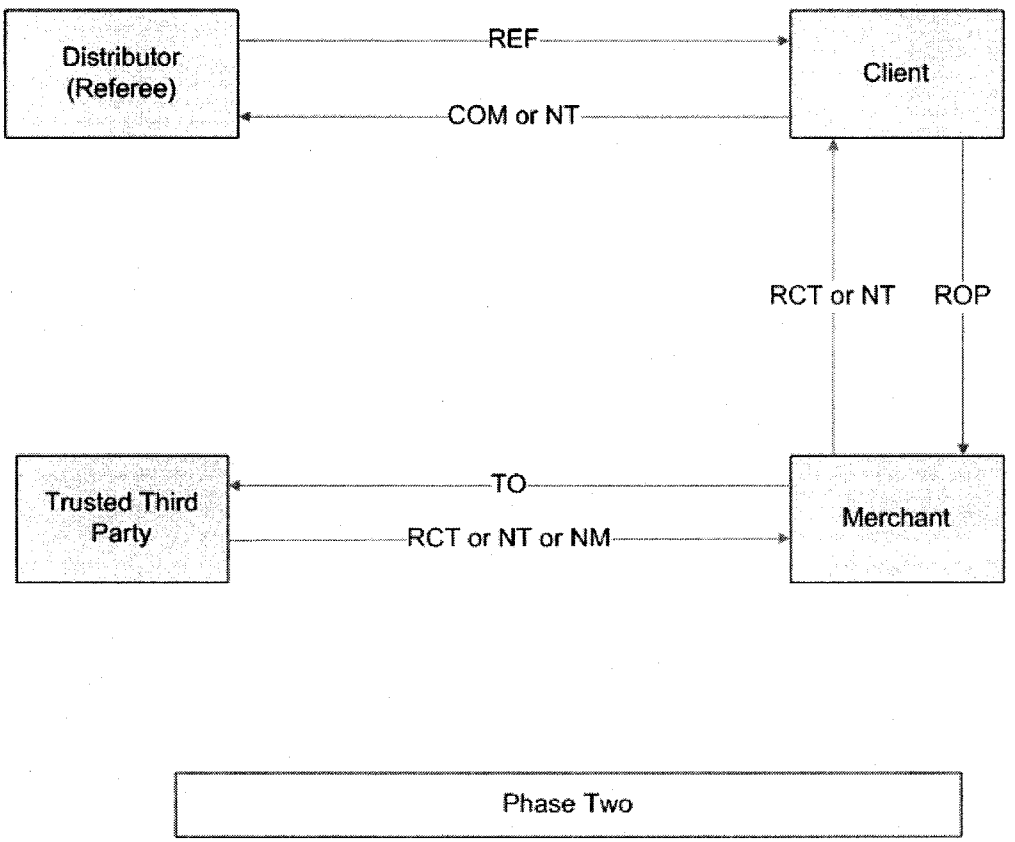


**Figure 2.3:** Multi-party exchange sub-protocol (main protocol)

| | | | |
|---|---|---|---|
| **DST** | → | **CL** | :REF |
| **CL** | → | **MER** | :ROP |
| **MER** | → | **TTP** | :TO |
| **TTP** | → | **MER** | :RCT |
| **MER** | → | **CL** | :[RCT, NT, NM] |
| **CL** | → | **DST** | :COM |

**Message Description**

| | |
|---|---|
| **REF** | Referral |
| **ROP** | Referred Order of Purchase. |
| **TO** | Transaction Order |
| **RCT** | Transaction Receipt |
| **COM** | Commission |
| **NT** | No Transaction |
| **NM** | No Money |

**Table 2.2:** Multi-party exchange sub-protocol (main protocol)

### 2.4.3 Phase 3: Recovery Sub-Protocol

The aim of the Recovery sub-protocol is to guarantee the achievement of the fairness property; this phase is considered complementary to the Multi-party Exchange sub-protocol. The execution of the Recovery sub-protocol is launched if an interruption or error in the execution of the main protocol took place.

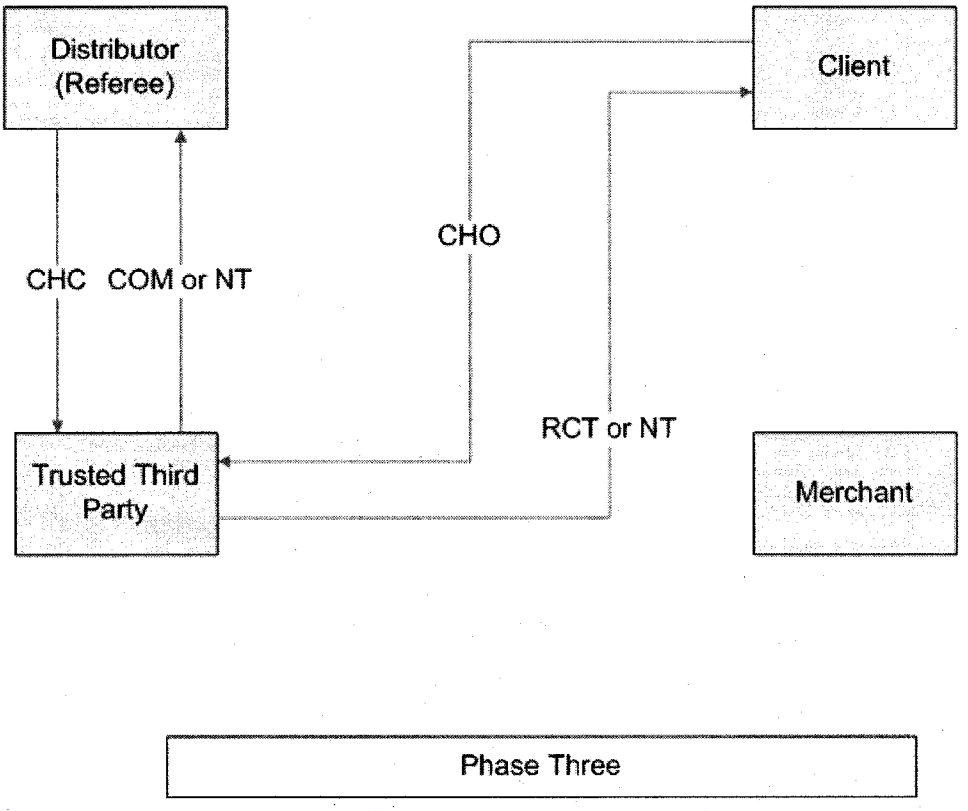


**Figure 2.4:** Recovery sub-protocol

| | | | |
|---|---|---|---|
| **CL** | → | **TTP** | :CHO |
| **TTP** | → | **MER** | :[RCT, NT] |
| **CL** | → | **DST** | :[COM, NT] |
| **DST** | → | **TTP** | :CHC |
| **TTP** | → | **DST** | :[COM, NT] |

**Message Description**

| | |
|---|---|
| **CHO** | Check Order |
| **CHC** | Check for Commission |
| **RCT** | Transaction Receipt |
| **COM** | Commission |
| **NT** | No Transaction |
| **NM** | No Money |

**Table 2.3:** Recovery sub-protocol message exchange

# Chapter 3

# Security analysis of the Multi-party Fair Exchange Protocol

In this section, we provide formal verification for the fairness property of our multi-party fair exchange protocol proposed in the previous chapter. We start by describing the behaviour of each party using state transition. Then we provide a brief survey of formal verification methods and tools that are used to verify protocol security properties and how we use these tools to present a formal verification of the fairness property of the multi-party fair exchange protocol.

## 3.1 Protocol parties' state transition

In what follows, we show how the four parties involved in the protocol (i.e., distributor, client, merchant, and trusted third party) move between different internal states by describing the behaviour of each party after sending messages, receiving messages, and processing information.

### 3.1.1 Distributor State Transition

Fig. 3.1 shows the state transition that the distributor (referee) goes through during the protocol execution and the inside details of the distributor's behaviour. The distributor moves from the initial state (*INIT_RE)* to the *REF_SENT_RE* state by sending a Referral to the client. The distributor stays in that state until one of the following events takes place: message received or a timeout occurs. We discuss both cases:

**a. Message Received**

If the message received is a Commission (*COM*), the distributor moves to the *SUCCESS_RE* state which is a final state indicating that the transaction completed was successful and the distributor gets his Commission for the Referral she made to the client, who used this Referral to make a purchase.

If the message received is a No Transaction (*NT*) message, then the distributor moves to the *NT_RE* state which is a final state indicating that the transaction was not completed successfully or did not take place at all. In this case, the distributor does not get a Commission for the Referral she made to the client because no transaction took place.

**b. Timeout**

Once a timeout occurs, the distributor sends a Check Commission (CHC) message to the Trusted Third Party and the state moves to *CHC_SENT_RE* state. The distributor waits for a reply from the TTP, if the reply message received is a *(COM),* the distributor moves to the *SUCCESS_RE* state. However, if it is a No Transaction message, the distributor moves to the *NT_RE* state. Both *SUCCESS_RE* and *NT_RE* are final states. Reaching the *SUCCESS_RE* state means that the distributor received the Commission and the transaction took place, while reaching the *NT_RE* state means that the distributor did not receive the Commission and no transaction took place.

If at any receiving state the distributor receives any other message than the expected one, it moves to *ERROR_RE* state which is also a final state.

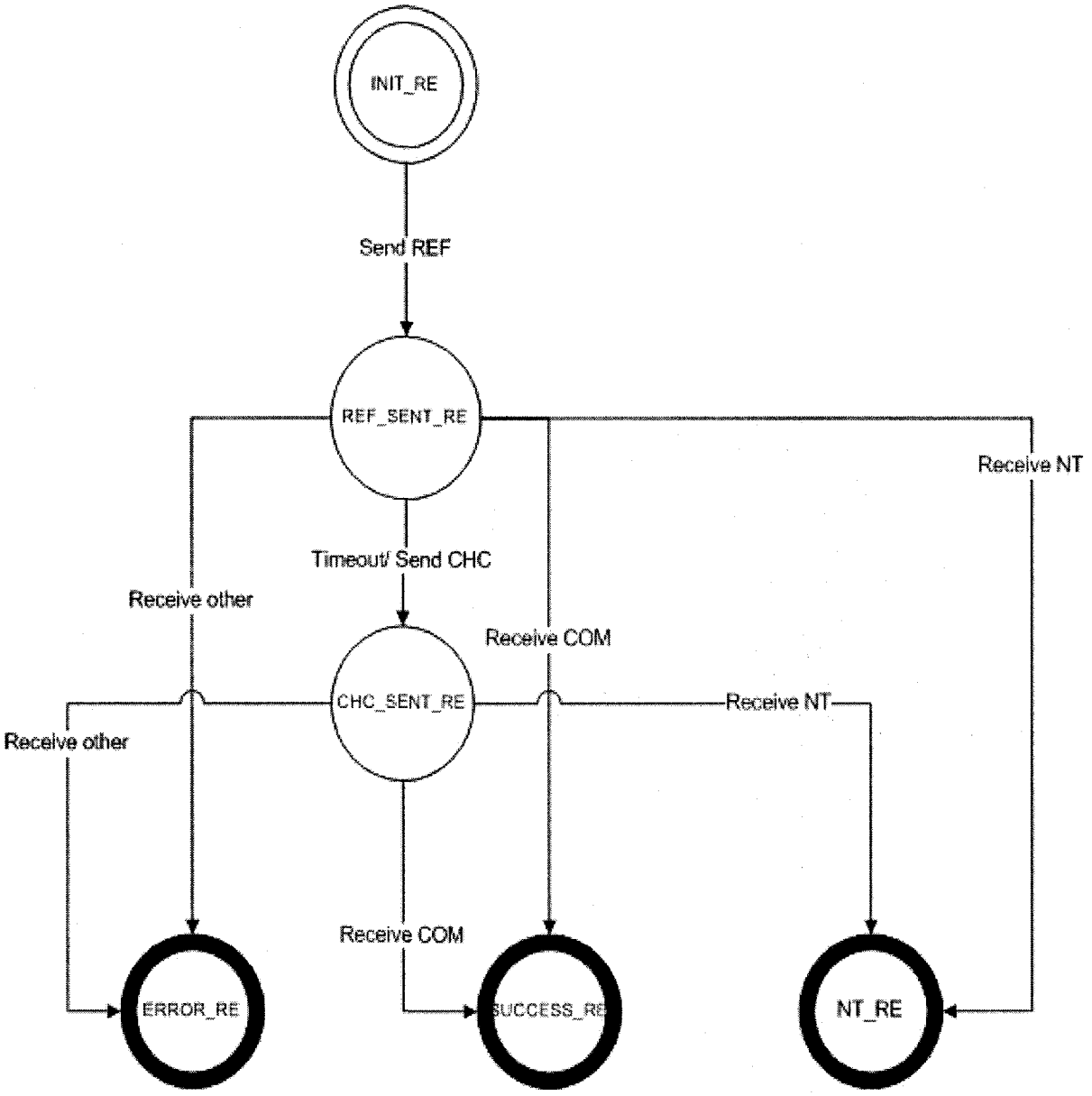


**Figure 3.1:** Distributor state transition

### 3.1.2 Client State Transition

Fig. 3.2 shows the state transition that the client goes through during the protocol execution and the inside details of the behaviour of the client. The client moves from the initial state *INIT_CL* to the *REF_REC_CL* once it receives a Referral (R*EF*) from the distributor. Then the client sends a Referred Order of Purchase (*ROP)* to the merchant and the state of the clients state moves to *ROP_SENT_CL*. The client stays in that state until it receives a message or a timeout passes.

**1. Message Received**

If the message received includes a Receipt (*RCT)*, the client moves to the *RCT_REC_CL,* this means that the client got her Receipt for the purchase she ordered from the merchant using the Referral she got from the distributor.

If the message received is a No Transaction occurred (*NT)* or No Money (NM) message, then the client moves to the *NT_CL* state.

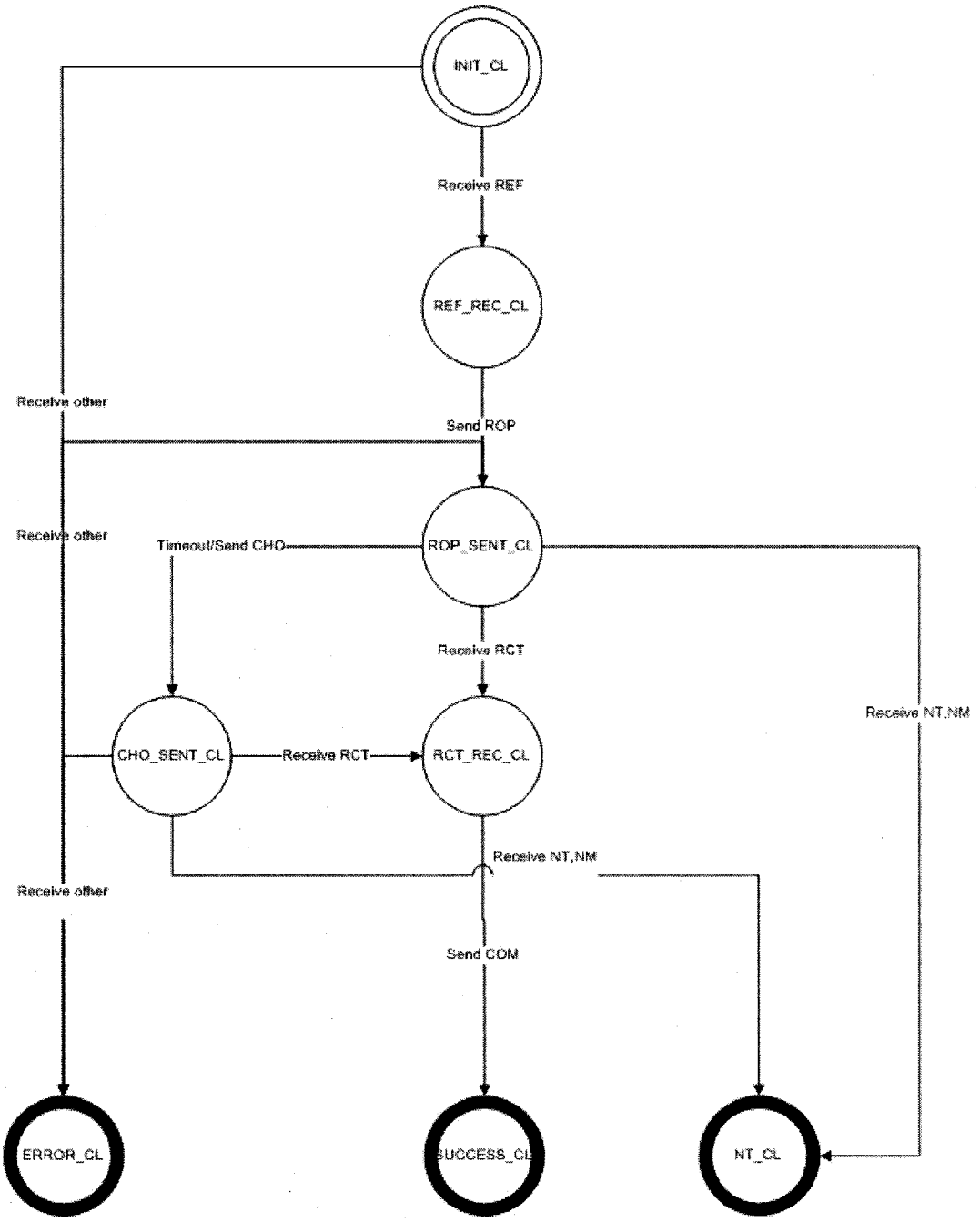


**Figure 3.2:** Client state transition

### 3.1.3 Merchant state Transition

Fig. 3.3 shows the state transition that the merchant goes through during the protocol execution and the inside details of the behaviour of the merchant. The merchant moves from the initial state *INIT_ME* to the *ROP_REC_ME* once it receives a referred Order of purchase *(ROP)* from the client. Then the merchant sends a Transaction Order *(TO)* to the TTP and the merchant moves to *TO_SENT_ME* state.

The merchant waits at the *TO_SENT_ME* state for a reply from the Trusted Third Party. If the TTP replies with a Receipt *RCT,* then the merchant forwards the *RCT* to the client, and this means that the transaction was completed successfully and the payment info provided by the client was correct. In this case, the merchant moves to *SUCCESS_ME* final state.

However if the TTP replies with a No Transaction occurred *(NT)* or No Money (NM) message, this means that transaction did not complete successfully. For example, if the TTP found that the payment info provided by the client is not correct, then the merchant moves to *NT_ME* which is a final state

If at *TO_SENT_ME* state the client receives any message other than the expected ones, it moves to *ERROR_CL* which is a final state.

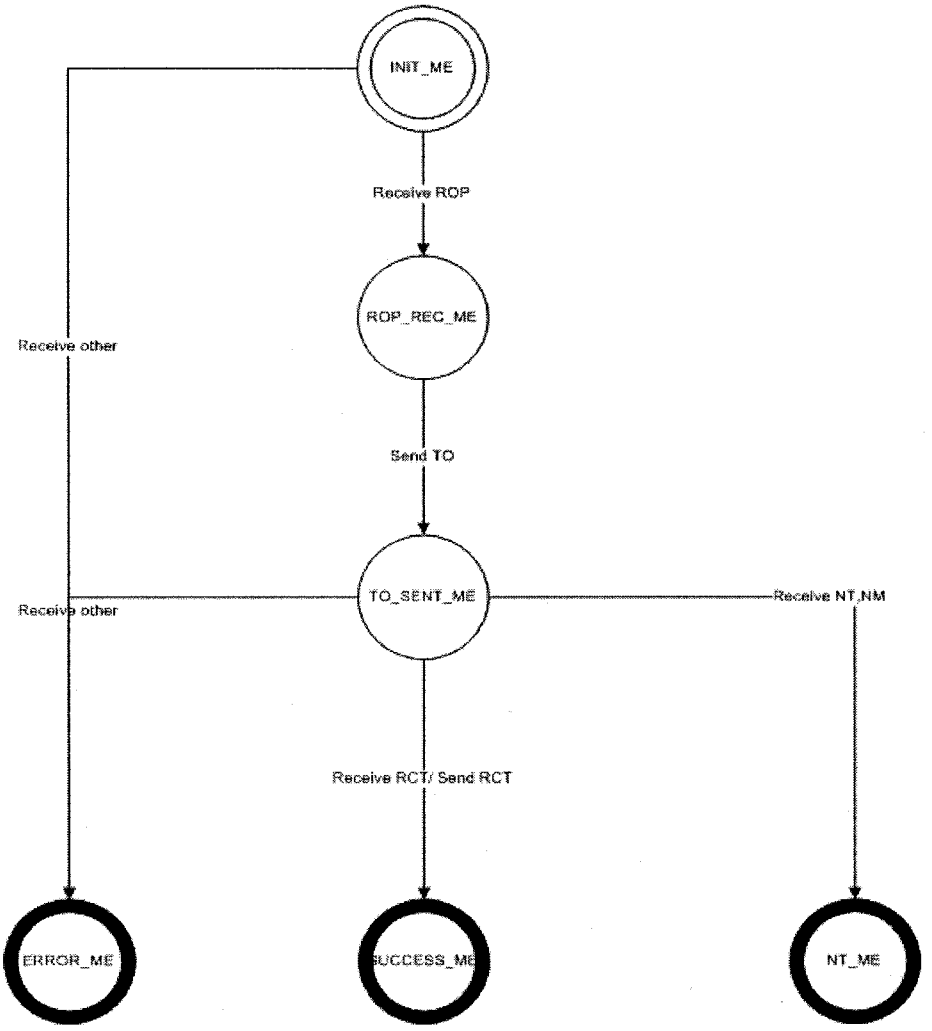


**Figure 3.3:** Merchant state transition

### 3.1.4 Trusted Third Party State Transition

Fig.3.4 shows the state transition that the Trusted Third Party goes through during the protocol execution and the inside details of the behaviour of the Trusted Third Party. The Trusted Third Party moves from the initial state *INIT_TP* to the *TO_REC_TP* state once it receives the Transaction Order (*TO*) from the merchant. Then the Trusted Third Party processes the Transaction Order. The processing of the Transaction Order can lead to one of two results:

#### a. Transaction OK

If the Trusted Third Party verifies all transaction parameters including payment and the matching good decryption key, and everything turns out to be correct according to the transaction process, then the TTP processes the transaction and sends back a Receipt (*RCT*) to the merchant. In this case, the TTP moves to *SUCCESS_TP* state.

#### b. Transaction NOK

If the Trusted Third Party fails to verify one or more of the transaction parameters including payment info and the decryption key, and cannot process a parameter according to transaction process (e.g., the payment information provided by the client was not valid), then, processing the transaction would lead the TTP into sending a No Transaction *NT* or No Money *NM* message indicating that the transaction was not processed successfully.

After the TTP reaches the *SUCCESS_TP* state, it can receive a request message related to phase three of the protocol. If it receives a Check Commission (*CHC*) message from the distributor, then the TTP replies with the Commission (*COM*) message. If the TTP receives a Check Order (*CHO)* message from the distributor, the TTP will reply with the Receipt *RCT* message.

After the TTP reaches the *NT_TP,* it can receive a request message related to phase three of the protocol. If it receives a Check Commission (*CHC)* message from the distributor, the TTP will reply with a No Transaction (*NT)* message. However, if it receives a Check Order (*CHO)* message from the distributor, the TTP will reply with a No Transaction (*NT)* message.

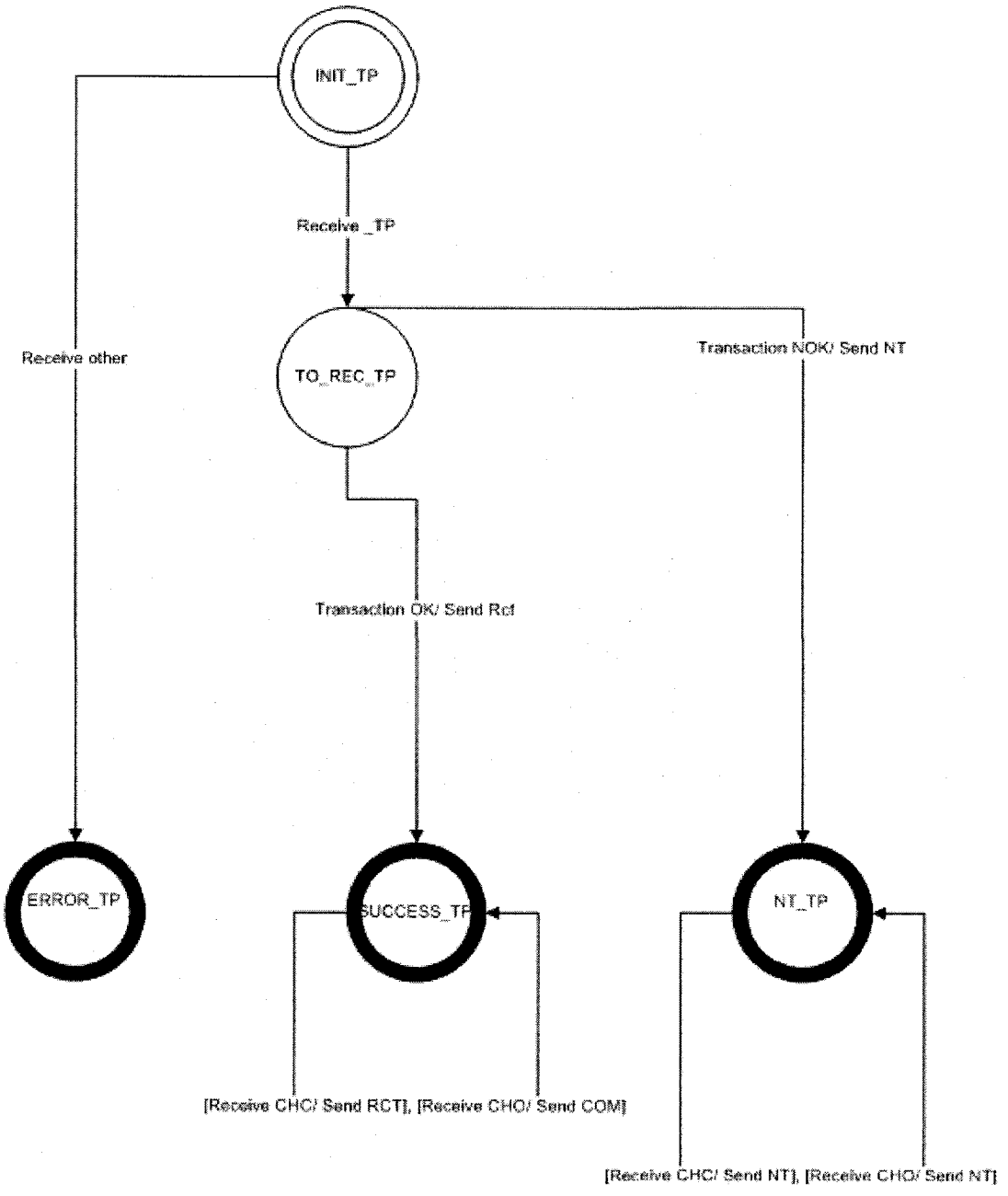


**Figure 3.4:** Trusted Third Party state transition

## 3.2 Formal Verification

The need for applying formal methods to check the security properties of security protocols has been widely recognized in the field of designing and verifying network protocols. Formal methods have been able to detect errors, to show weaknesses, and to provide attack exploitation scenarios in protocols that were assumed for a long time to provide certain security properties. One example is when Lowe [14, and16] showed a flaw in the design of the Needham- Schroeder public-key protocol [15]. These flaws have remained undetected for about 17 years till a formal method, Failures Divergences Refinement (FDR) checker for Communicating Sequential Processes (CSP) logic, was used. The flaw was detected in 1995 and the protocol was first proposed in 1978. The field of formal verification methods has since gained wide attention from researchers from all around the world.

### 3.2.1 Overview of Formal Verification Methods

Since the realization of the importance of formal methods in debugging and assuring security properties in network security protocols, many software tools have been introduced to assist in conducting the security analysis. These tools depend on different methodologies such as the finite state exploration, state enumeration, belief logics, model checking, and inductive theorem. Using these tools for this purpose has proven to be very successful in detecting bugs, and presenting attack scenarios on network and security protocols.

Although Formal methods have been used for the verification of cryptographic protocols including authentication protocols and key exchange protocols, only few used those methods to verify security properties of fair exchange protocols.

Zhou and Gollmann [17] applied SvO logic by formalizing the goals of non- repudiation services to study the validity of non-repudiation evidence. Protocols such as Kerberos, Secure Socket Layer 3.0 (SSL 3.0), and Needham-Schroeder protocols were analysed using the Finite-state exploration [19, 20, and 21]. While other protocols such as Secure Electronic Transaction (SET), Transport Layer Security (TLS) and Smart Cards protocols were verified using the inductive thermo proving method [25] in [22, 23, and 24]. Shmatikov and Mitchell analyzed a set of contract signing and fair exchange protocols [32, 33, and 34] using the finite-state analyser $Mur\Psi$. The automated model checker MOCHA was used by Kremer and Raskin [29, 30, and 31] to examine non-repudiation and fair exchange protocols.

A long list of software tools, based on different techniques, has been developed. The list includes but not limited to SPIN model checker [35], BLAST model checker [34], theorem prover Isabelle [36], model checkers Mocha [37], finite-state analysers $Mur\Psi$ [38], Markov Reward Model Checker (MRMC) [33], and Partial order reduction.

For the verification of our multi-party fair exchange protocol, we adopted model checking for formal verification of the fairness property of the protocols using the SPIN model checker. We used the specification language PROMELA for modelling the protocol and the Linear Temporal Logic (LTL) for encoding property formulas.

## 3.3 Model Checking

Model Checking is the process of verifying whether or not a finite state concurrent system satisfies a logical formula. This process is general enough to be applied to all kinds of logics, applied structures, and models [12].

Model checking has emerged as a successful technology for verifying design requirements. It has been used for verifying real industrial protocols such as the PCI local bus protocol, commercial products, and hardware designs such as sequential circuit designs, and real-time and safety critical systems [13].

A very common method of the model checking process is conducted by verifying that the structure and/or the design requirement of a hardware or software design, satisfies a property (specification). If the model checker finds that the structure or design accepts the property, the model checker will output a Yes result, otherwise it will output a No result. A No result is usually accompanied with a counter example that explains where the property is found not satisfied. The counter example can be very useful in correcting the structure by pointing at the error. Once the error is found, it should be corrected and another model checking cycle should be executed.

Temporal Logic formulas are often used to express the properties. While efficient algorithms are used to traverse the system finite space, model checkers test whether or not the temporal logic formulas hold.

## SPIN

SPIN is an open source automata-based model checking software tool. SPIN was developed at Bell Labs by Gerard J. Holzmann and others, and it is continuously evolving for more than 20 years. In 2001 SPIN was awarded the prestigious System Software award by the ACM. SPIN was also used to analyze the concurrent plan execution module in NASA's DEhEP SPACE 1 module, the Mars exploration Rovers [18].

SPIN depends on model checking based techniques for the formal verification, validation, and analysis of distributed software systems and data communication protocols. It does that by examining the logical consistency of concurrent systems. It can also be used as a simulator to explore all states in the execution paths through the system, and then it presents the resulting execution trace to the user. It can also present to the user the trace of possible execution paths that may lead to undesired states. However, simulation might face a problem of state space explosion due to the exponential growth of the generated finite state machine.

Traversing a large state-space model can be done in a relatively short period of time. However, state explosion problem is still common. Different techniques are used to avoid the state explosion problem including Counter Example Guided Abstraction Refinement (CEGAR) [39], partial reduction and symbolic algorithms.

SPIN is used to point out logical errors in concurrent systems including race conditions, deadlocks, unspecified reception, and process incompleteness. SPIN can also be used as a verification tool to prove the validity of a system against properties (specifications) or design requirements by scanning the whole state-space. Partial order reduction theory is used to

optimize the scan. In very large state-space systems, SPIN can be used as a proof approximation tool to validate properties and design requirements of a system's model.

## PROMELA

PROMELA (Process or Protocol Meta Language) is a process verification modelling language that supports the modeling of distributed systems as non-deterministic automata. It models the parallel systems in a way that can be verified using a model checker. A PROMELA model may consist of processes, variables and channels. While the process can specify the behaviour of the concurrent system, the global variables and channels specify the environment that processes communicate in.

A system that is modelled in PROMELA consists of three parts:

1. Dynamically created processes. Each process often represents different entities that are running in parallel.
2. Channels, which allow message exchange between the processes. The message exchange using the channels can be defined to be asynchronous (i.e., buffered) or synchronous (i.e., rendezvous).
3. Variables, which can be defined as global variables or local variables within the process.

Channels transfer messages between active processes. They can be defined locally with in a process or globally. We define a channel using the keyword *chan.* Default messages are stored in FIFO (First-in First-out) queue, so that the first message will be delivered first to the

receiving party and can also be defined to randomly choose a message from the queue. The following syntax:

$$Chan\ X, Y[n]$$

declares an uninitialized channel X and uninitialized array of $n$ elements. While the syntax:

$$Chan\ z = [m]\ of\ \{mtype\}$$

defines an array of *m* initialized messages. Each message is of type *mtype.*

## LTL (Linear Temporal Logic)

LTL is a modal temporal language with a model that allows describing time constrains. LTL allows writing properties and conditions in formulas. Those conditions are normally negated before being verified by the model checker. Evaluation of LTL formulas is performed over the execution paths of truth conditions. Describing time constrains allows expressing formulas that involves time as a factor.

## SPIN, PROMELA and LTL

Given a model system specified in PROMELA, SPIN can perform random or interactive simulations of the system's execution or it can generate a C program that performs a fast exhaustive verification of the system state space.

During simulations and verifications, SPIN ensures the absence of deadlocks, unspecified receptions, and un-executable code. The verifier can also be used to prove the correctness of system invariants and it can find non-progress execution cycles. Finally, it supports the verification of linear time temporal constraints; either with PROMELA never-claims [18] or by directly formulating the constraints in temporal logic.

PROMELA models can be analyzed with the any model checker, to verify that the modeled system produces the desired behaviour.

The verifier is designed to be fast while using a minimal amount of memory. The exhaustive verification performed by SPIN is conclusive. It provides confidence whether a system's behavior is error-free or not. Many verification runs, that cannot usually be performed with automated techniques, can be done in SPIN with a ``bit state space" technique [54]. With this method the state space is collapsed to a few bits per system state stored. Although this technique doesn't guarantee certainty, the coverage is better, and often much better, than that obtained with traditional random simulation

## XSPIN

XSPIN is a graphical interface of SPIN written in TCL/TK. XSPIN provides an easy way to use SPIN; it generates SPIN commands with the proper compile-time and run-time arguments and converts SPIN command output to a graphical representation one. The ease of use comes from the menu selection that provides different features such as syntax checking for PROMELA language code, and code compilation and execution.

## 3.4 Formal Verification of The Fairness Property of The proposed Multi-Party Fair Exchange Protocol

Each participant party in the protocol is defined as a process with the keyword *proctype*. A process describes the behaviour of a party including the communication description, such as sending and receiving messages, and processing data. For example,

$$Proctype\ partyX\ (\ chan\ c)$$

$$\{Set\ of\ statement\}$$

The most important step in model checking verification is to translate the protocol specification and message exchange into PROMELA language so that the model checker SPIN can read it. Fig. 3.5 shows a snapshot of the XSPIN graphical specification tool.

### 3.4.1 Model Implementation

In order to translate state transition shown in Figures [3.1, 3.2, 3.3 and 3.4] into a PROMELA model that can be verified by SPIN model checker, we build a process for each one of the four parties: client, distributor, merchant, and TTP. The statements inside each process implement the logic and behaviour of communication channels for each party.

During the simulation of the protocol, whenever a party faces a set of choices of behaviour, the party will randomly choose a move and execute it. For example, a timeout for receiving a message is implemented by randomly assigning a value to a Boolean timeout

variable. This way we simulate the two behaviours of a party in both cases receiving and not receiving the desired message in time. On the other hand, if the party has only one choice for a move, that move will be executed.

## Protocol Simulation

After modeling the multi-party exchange protocol in PROMELA language, we moved into performing simulations using the graphical user interface XSPIN. These random simulations provide assurance to an acceptable level that the protocol implementation is behaving in the intended way by performing a set of random protocol runs.

One simulation run showed the four parties exchanging messages and this simulation result in a success scenario where all parties get their expected items. The merchant gets her money included in the receipt and so does the client, while the Distributor gets her commission from the transaction. That means the TTP has successfully verified all the provided data in the received messages such as payment information.

Another simulation run showed us how the four parties exchange messages where the simulation run result in a situation where no party gets her expected item: The merchant did not get her money and neither did the client, while the Distributor got no commission because the transaction did not take place. This is means that the result of the TTP verification results in invalid provided information or non- matching information.

### 3.4.2 Protocol Verification

We chose to verify the fairness property in our protocol. A protocol is said to satisfy strong fairness if at the end of the protocol execution either all parties involved in the protocol get their expected items or no party gets any useful information.

Using LTL formulas, we try to express the fairness property, so that the model checker SPIN can verify the protocol against this property. The fairness property can be expressed by the following formula provided that the protocol will not reach a deadlock state:

$$[\ ](p \rightarrow (A\ ||B\ ))$$

It can also be expressed in the following formula:

$$<>\ (A\ ||B\ ),$$

where [] And <> are two LTL operators that stand for always and eventually respectively. So the formula says" either the Distributor, customer, and merchant get their expected items (property A) or none of them get her item (property B) at the end of the protocol execution (property P)".

Where:

Property P: defines when the end of the protocol execution is reached.

$$\#\ define\ p\ (TTPstop == true\ \&\&\ MEstop == true\ \&\&\ CLstop =$$

$$= true\ \&\&\ REstop == true\ )$$

Property A: When satisfied, all parties (the TTP, distributor, customer, and merchant) should have gotten their expected items and the TTP has successfully processed the transaction.

$$\#\,define\ a\ (TTPTranOK == true\ \&\&\ MEGotFound == true\ \&\&\ CLGetRct == true\ \&\&\ REGetCom == true\ )$$

Property B: When satisfied, all parties (the TTP, distributor, customer, and merchant) should have not gotten any useful information or item and the TTP has not processed the transaction successfully.

$$\#\,define\ b\ (TTPTranOK == false\ \&\&\ MEGotFound = = false\ \&\&\ CLGetRct == false\ \&\&\ REGetCom == false\ )$$

Finally, we verify the PROMELA model by loading the model into Spin. We load the fairness property, as an LTL formula, into the LTL Manager shown in Fig. 3.5. The result of the verification using SPIN has confirmed that the fairness property is really satisfied by the PROMELA model. By doing so, we have formally proved that the multi party fair exchange protocol is indeed a fair exchange protocol.

A significant point that worth mentioning about formal protocol verification is that although formal verification does a great help when verifying security properties of protocols, it is limited to the accuracy of the modeling of the protocol. A model of the protocol may be different from the protocol itself and the implementation of the protocol may also be different from both the model and the protocol. We understood this limitation in our security analysis and we provided both a formal and informal analysis to provide the confidence in the security properties of the protocol.

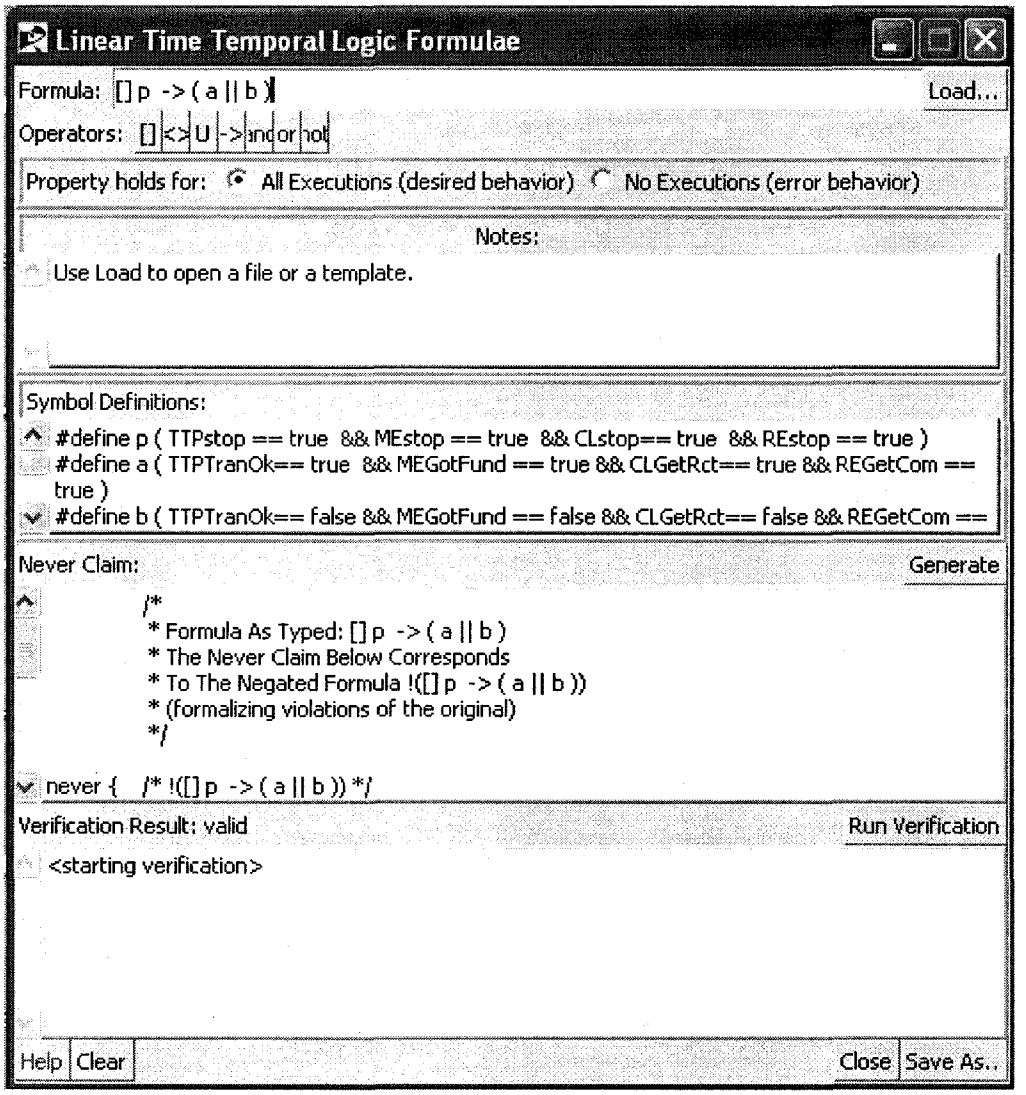


**Figure 3.5:** XPIN LTL manager

# Chapter 4

# Memory Disclosure Vulnerabilities

## 4.1 Background and Motivation

Many security solutions are proven to be secure only under a set of assumptions including the assumption of keeping cryptographic keys secret. Once the secret cryptographic key is revealed, the solution or the application's security is compromised. Typically, cryptographic libraries are used to provide the security services needed for securing many applications and systems. These libraries tend to keep the cryptographic keys in memory while performing different cryptographic operations which makes the underlying systems vulnerable if an attacker is able to locate the memory location where these keys are stored.

The problem of locating cryptographic keys in memory has first been recognized when an efficient method to locate cryptographic keys and encrypted data was described by Shamir and Someren [43]. One of the techniques that they described in their paper was entropy based; it uses the fact that the entropy of the cryptographic keys is much higher than the entropy of non-random data such as text files, images and executable codes. This technique does not calculate the exact entropy value; rather it examines the number of unique byte values in a sliding window of 64 bytes. Shamir and Someren [43] found that a 64 bytes window of non-random data

contains on average 30 unique byte values, while the same window of a typical cryptographic key may, on average, contain about 60 unique byte values. This method can be used to locate cryptographic keys in a large string of data such as hard disks, and memory dumps.

Contrary to common assumptions, the contents of a process that is running in the memory may contain critical information such as cryptographic keys. These cryptographic keys may become accessible using some tools and hence leave those keys vulnerable to other processes running on the system. Although good programming practices would not leave critical information in the memory vulnerable, it might become vulnerable if another user can get access to the memory of a running process using a malicious (or forensic) tool.

The remanence of cryptographic keys in memory has become a vulnerability that indirectly compromised the security of many organizations. The problem of employee's laptop losses and theft with company's sensitive data on it has reached epidemic levels. Many recent civil and military organizations have suffered from such accidents. For example, on January 2008, the British Royal navy lost a laptop with personal information of 600,000 new and potential recruits of Navy, Royal Marines, and Air Force on it. Another recent accident took place in West Penn Allegheny health system [53]. This accident put the personal information of 42,000 patients at risk when a laptop was stolen from a home care nurse. Although, it was previously believed that encryption can be used to protect the information held on mobile devices, some newly developed techniques managed to overcome the encryption of hard disks exploiting the remanence of cryptographic keys in memory. In particular, Halderman et al. [44], at the Center for Information Technology Policy at Princeton University, have proven that hard

disk encryption can be defeated using a "cold boot" attack. They found out that, when keeping memory at lower temperature levels using liquid nitrogen, only little RAM reading errors occur after 60 minutes. This observation can enable an attacker with physical access to copy the contents of the memory where she can find the encryption secret key. We recall their method: "To reconstruct an AES key, we treat the decayed key schedule as an error correcting code and find the most likely values for the original key. Applying this method to keys with 10 percent of bits decayed, we can reconstruct nearly any 128-bit AES key within a few seconds. We have devised reconstruction techniques for AES, DES, and RSA keys, and we expect similar approaches will be possible for other cryptosystems."[44].

In this chapter, we experimentally describe the extent and predictability of locating cryptographic keys and passwords in memory. Such predictability makes a lot of systems vulnerable to memory disclosure attacks.

Many applications and servers use the "SUNJCE" implementation of JCE (e.g. IBM® WebSphere® application Server). The "SUNJCE" implementation of JCE comes by default with most Java 2 SDK versions when a provider is accompanied. Many of these servers and applications that use directly or indirectly (e.g. via an application server such as Oracle Application Server 10g) this implementation of JCE are probably vulnerable to the described attacks. Those applications and servers are used in financial institutions, Health institutions, and governmental organizations. They utilize the JCE implementation to do different cryptography operations such as encrypting, decrypting and signing of messages.

We demonstrate that these attacks are applicable on both symmetric and public-key algorithms. We also believe that implementations of other algorithms are vulnerable to these simple memory search attacks as well. We experiment locating the cryptographic keys of both Advanced Encryption Standard (AES) and RSA in memory.

We also show the operating system's limitation in protecting passwords and cryptographic keys. While our main focus was on cryptographic keys and passwords, this approach can be extended to finding any sensitive data persisting in the memory. Hence, this sensitive data becomes vulnerable to the same attack. Then, we also suggest several techniques for partially mitigating this risk.

As a motivating example, consider a user using a computer in a library or a university laboratory connected to a secure password-protected website using a web browser. The user might be connected to his bank account, email, or any other secure website. In order to sign in his account, she will have to provide the username and password. Once the user finishes, she would normally signs out, close the web browser, and log off in best case scenarios. We demonstrate that if another user logged in with some privileges on the same computer and managed to scan the memory with some tool, then there is a big probability that he might find much critical information the previous user had entered, hanging in the memory.

Another motivating example might be a user logged into a computer in an Internet café. It is not hard to imagine different scenarios in which the user might be connected to an e-commerce website, or a financial institution (e.g. bank) to do an online transaction. Once the user is finished, another user who has captured the encrypted content sent over the network can locate

the cryptographic key using the technique and information that we provide in our experiment. The fact that this cryptographic key was used to encrypt that content allows the attacker, in this case, to have access to Credit Card Numbers, banking financial information and any other encrypted sensitive data sent over the network. These types of simple memory scan attacks may put secured websites working on Hypertext Transfer Protocol over Secure Socket Layer (HTTP over SSL), and the website's users' account information at risk.

## 4.3 Random Access Memory

The concept of memory is wide and includes all types of memory that can provide permanent or temporal storage of data and can be used by a digital computing system.

RAM stands for Random Access Memory; it is described as a volatile memory because in order to keep the data that it holds, it has to have power supply. The word "Random" comes from the fact that a series of data bytes can be retrieved from the RAM in a constant time regardless of the physical location of that series of data bytes. This is possible because the RAM is constructed as a two-dimensional grid of memory cells. In order to access any memory cell directly, we supply the column and row that intersects that memory cell.

In the history of digital computing systems, different types of memory have been developed and deployed. Evaluating these types takes into account different factors such as speed of operation, latency time, ease of construction and use, and cost.

The content of the RAM, contrary to what is commonly known, is not completely lost the moment the power is switched off. In fact, it starts fading away slowly after the power is turned off and this raises a security concern as traces of the cryptographic key may remain in the RAM.

The used storage mechanisms in RAM vary from magnetic discs, tapes, and optical discs. RAM that can be rewritten varies from Dynamic RAM, static RAM. Dynamic RAM is most commonly made of memory cells constructed of a paired capacitors and transistors. While static RAM is made of flip-flops that store bits of data.

Many computing systems have a memory hierarchy that consists of CPU registers, level one and level two cache, DRAM, and virtual memory which is stored on hard disks. This hierarchy serves different purposes while taking into consideration the compromise between performance and cost. The CPU registers provide the CPU with its requirement for a very fast access to data. Nevertheless, registers are relatively expensive so that small size is usually used. On the other hand, hard drives are not expensive so large size is used but the access time is relatively very high. When we go up in the memory hierarchy, the storage size and access time decrease while the cost per storage unit increases.

Swapping helps solving the problem of RAM shortage by using the hard drive as an additional memory, temporarily. This occurs usually when the computer system requires a lot of memory during intensive application cycles or when the computer system runs a lot of memory demanding applications. One disadvantage of Swapping is that it reduces the overall system performance because of its dependency on using the hard disk, which has high access time, as a computer memory.

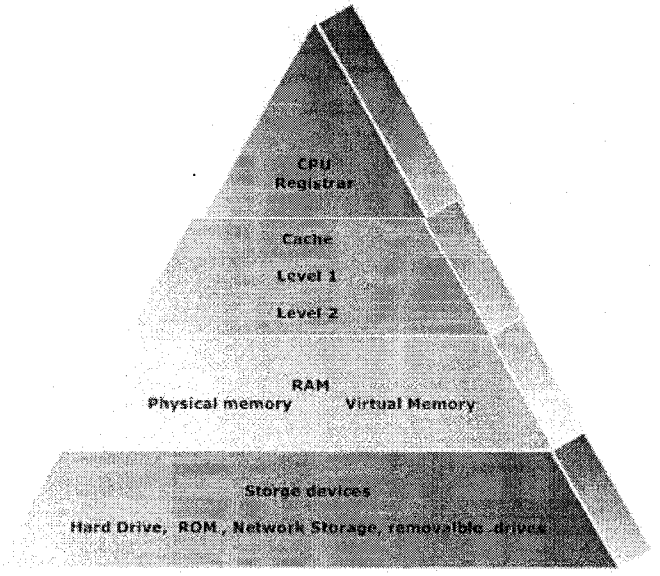


**Figure 4.1:** Memory hierarchy

### 4.3.1 Memory Management

Memory is an important resource of a computer system and needs to be managed efficiently. This has lead to the realization of memory management as a critical subsystem in modern operating systems.

One of the tasks of the memory management subsystem is to keep track of used memory parts and parts that are not in use, to allocate parts of memory to processes when they need it, and to free those parts once they are finished.

Another task of the memory management subsystem is to manage swapping. Swapping moves processes or parts of them back and forth between the main memory and the secondary storage. The swapping in and swapping out operations, which are relatively slow memory I/O, have to be managed efficiently by the memory management subsystem in order to optimize the performance.

The need for main memory is always growing and there has never been enough memory for the operating system and active processes. The operating system and the active processes use the memory to hold both data and the program code and hence allowing both them to run efficiently.

### 4.3.2 Memory Management Goals

The memory management subsystem provides many objectives that include the following:

1. Providing large memory address space using virtual memory to satisfy the need of active processes and the operating system for a large space that may be larger than the available physical memory.
2. Allowing virtual memory addressing: this separates processes' memory addresses from actual physical addresses. Virtual memory uses one of the following techniques or a combination of them. These techniques include:
    - Fixed Partitioning.
    - Dynamic partitioning.

- Simple paging.
- Simple segmentation
- Simple segmentation
- Virtual memory paging
- Virtual memory segmentation

3. Providing memory protection: Processes should be protected against accidental, intentional, and malicious interference from other active processes. Referencing memory addresses dedicated to a process by another process should not be allowed without permission.
4. Enforcing the memory protection requirement by the processor rather than the operating system allows efficient assessment of permissibility of referencing a memory address. While leaving this task for the operating system to do the assessment of memory referencing violations and to anticipate all memory referencing that a program will make, is not only time-consuming also not efficient as well.
5. Saving memory space by allowing multiple processes that are executing the same program's instructions to share the same copy of the program's instruction. Memory protection provides the flexibility of sharing memory between multiple processes executing some common libraries or any program's code. Dynamic libraries and command shells are examples of shared code between several processes.

6. Providing each process with its own virtual address space gives processes the impression of having access to a large share of physical memory in a transparent way. The memory management subsystem makes sure to divide the physical memory between processes in a fair way.

### 4.3.3 Process Isolation

A fundamental function of operating systems security is process isolation. Process isolation protects the system's integrity by isolating and preventing interference of one process with others processes, operating system code or data, or protected operating system resources. System resilience is another feature that the operating system can provide by the contribution of process isolation. System resilience provides failure boundaries that allow a process or part of the system to fail without affecting the rest of the system. Many mechanisms are used in operating systems for process isolation enforcement. Most used mechanisms rely on hardware protection such as page mapping, distinguishing user and kernel instructions, and memory segmentation.

The CPU memory management hardware enables process isolation by using two mechanisms:

a. Running process are allowed to access only certain pages of the memory
b. Implementing privilege levels in order to prevent un-trusted code from accessing certain pages of the memory.

Virtual memory and privilege levels increase the cost of inter-process communication and affect the operating system overall performance.

Windows NT keeps a separate page table directory for each process to prevent processes from manipulating other processes' address space. We recall how the windows NT apply process isolation "Depending on the process in execution, it switches to the corresponding page table directory. As the page table directories for different processes point to different page tables and these page tables point to different physical pages and only one directory is active at a time, no process can see any other process's memory. ... All the kernel pages are marked as supervisor pages; therefore, user-mode code cannot access them" [42].

Hardware protection and operating systems through memory management and other operating system security features try to protect processes running on the machine. However, attackers have been successful in writing malicious programs that are able to take advantage of the OS provided APIs to scan the memory looking for cryptographic keys.

## 4.4 Java Security

Java security includes a large set of specifications, APIs, tools, and implementations of commonly used security algorithms, mechanisms, and protocols. The Java security APIs cover a wide range of areas, including secure communication, cryptography, and public key infrastructure. Java security provides both a framework for writing secure applications, and a set of tools that are helpful in managing applications' security.

### 4.4.1 Java Cryptography Architecture

The Java cryptography Architecture (JCA) and the Java Cryptography Extension (JCE) are frameworks that are designed to provide cryptographic functionality for Java. The JCA is part of Java 2 Security API while JCE is an extension to the JCA. JCE provides encryption and decryption functionalities to the JCA. JCA and JCE were designed according to the following principles:

#### i. Implementation Independence

Implementation independence allows the ability of using cryptographic functions without worrying about the implementation. This feature is achieved using the provider-based architecture. This architecture allows Cryptographic Service Providers (CSP), or simply providers, to provide a package or more. Packages usually implement one or more cryptographic service in accordance with the defined specification.

#### ii. Implementation Interoperability

Implementation interoperability requires different implementations from different cryptographic service providers to offer services and accept services from each other. For example, for the same cryptographic algorithm and compatible keys, an encrypted message using a provider can

be decrypted using another provider, and a signature generated by one provider can be verified by another provider.

### iii. Algorithm Independence

Algorithm independence can be attained by categorizing cryptographic services, then identifying classes that provide the functionalities of these cryptographic services. These classes are called engine classes. Examples of these engine classes include the *Signature, MessageDigest, KeyFactory*, and *KeyPairGenerator* classes.

### iv. Algorithm Extensibility

Algorithm extensibility allows new algorithms that are categorized in one of the supported engine classes to be added easily.

Although JCE requirements provide the above principles, it does not focus on the implementation security nor does it give any guideline on how to write a secure implementation. It leaves the implementation to the provider to write a secure implantation of JCE.

### 4.4.2 Advanced Encryption Standard (AES) in JCE

In this section, we demonstrate how we can use Java to generate keys, and encrypt and decrypt messages using the Advanced Encryption Standard (AES). The provided code works with Java 2 Platform Standard Edition (J2SE)

The *KeyGenerator* class acts as a symmetric key generator. The *KeyGenerator* object can be re-used to generate symmetric keys. It is instantiated using one of the *getInstance* methods. The *KeyGenerator* object can be re-used for generating symmetric keys. The following code demonstrates how to instantiate a *KeyGenerator* object for a specified algorithm. In this example we generate it for the AES algorithm. The name of the requested key algorithm should be stated in compliance with the Java Cryptography Extension Reference Guide [41].

```
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(256);
```

The following few lines shows how we use the *Keygenerator* object to generate a secret symmetric key without utilizing a provider-based *SecretKeyFactory*. In order to get the key as an array of raw bytes, we call the *getEncoded* method. The *SecretKeySpec* object constructs a secret key from a given raw byte array. However, this constructor does not check any constraints on the secret key, is it of a given length, does it go along with constraints of a specific cryptographic algorithm ("AES" in our example), nor it does check if it is a weak or semi-weak key.

*SecretKey seckey = kgen.generateKey();*

*byte[] raw = seckey.getEncoded();*

*SecretKeySpec seckeySpec = new SecretKeySpec(raw, "AES");*

The Cipher object that we construct in the flowing code acts as the cryptographic service provider for operations such as encryption and decryption. The Cipher object can be instantiated using the Cipher's *getInstance* method which takes an argument the name of the transformation that need to be constructed. The transformation should always include the name of a cryptographic algorithm (e.g., AES, DES), and may be followed by a feedback mode (e.g., CBC, CFB, OFB) and padding scheme (e.g. PKCS5Padding). The transformation could be of the form "algorithm" or "algorithm/mode/padding".

*Cipher AES_cipher = Cipher.getInstance("AES");*

After constructing the Cipher Object, it has to be initialized to do one of the following four operations: encryption, decryption, key wrapping, or key unwrapping. Along with the operation mode (e.g. ENCRYPT_MODE, DECRYPT_MODE, WRAP_MODE or UNWRAP_MODE), we pass the secret key when calling the *init* method.

*AES_cipher.init(Cipher.ENCRYPT_MODE, seckeySpec);*

*AES_cipher.init(Cipher.DECRYPT_MODE, seckeySpec);*

The following few lines of code, shows how the Cipher object can be used to do both encryption and decryption operations. The First line shows how to encrypt a string message after converting it into an array of bytes. For the *Cipher.doFinal* method to do an encryption operation the Cipher object has to be initialized with an operation mode of encryption. Also, the encryption key has to be set (as we have shown in the previous lines of code). Encryption or decryption of data with a specific secret key depends on how the cipher object was initialized.

```
byte[] encryptedMsg=
    cipher.doFinal(("Hello: Please encrypt me! ").getBytes());
byte[] decryptedMsg =
    cipher.doFinal(encrypted);
```

### 4.4.3 RSA in JCE

We will demonstrate how we can use Java to generate keys, encrypt and decrypt messages using the RSA algorithm, this program works with Java 2 Platform Standard Edition (J2SE 5.0). RSA is a public-key cryptographic algorithm that is developed by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT in 1977. RSA is used for both encrypting and signing of messages, and it is considered a great advance in the public key cryptography field.

The RSA algorithm is widely used as an encryption and authentication algorithm. RSA is widely used in web browsers and many products including Intuit's Quicken Lotus Notes. RSA is

an Internet encryption and authentication system that uses an algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman.

RSA keys include a public key and a private key. The public key is used for encrypting messages and verifying digital signatures and is made publicly available. While private key is used for decrypting and signing messages, Messages encrypted using the public key can be decrypted using the private key only. Public keys are also used to verify the signature of messages singed using the corresponding private keys.

We will demonstrate how we can use the JCE API to generate public and private keys, and encrypt and decrypt messages using the public key and private key respectively.

The *KeyPairGenerator* class generates pairs of public and private keys. The *KeyPairGenerator* object can be re-used to generate pairs of keys. The *KeyPairGenerator* object is instantiated using one of the *getInstance* methods. The following code shows how to instantiate a *KeyPairGenerator* object for a specified algorithm. The *KeyPairGenerator* object has to be initialized with a source of randomness before the *KeyPair* is generated. In this example we generate it for the RSA algorithm. The name of the requested key algorithm should be stated in compliance with the Java Cryptography Extension Reference Guide [41]. Once the *keypair* object is constructed, a public key and a private key can be extracted from it.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
keyGen.initialize(1024, random);
```

*KeyPair key_pair = keyGen.generateKeyPair();*

*Private_Key priv_key = key_pair.getPrivate();*

*Public_Key pub_key = key_pair.getPublic();*

The following code illustrates how to construct the Cipher object. The Cipher object acts as the cryptographic service provider for operations such as encryption and decryption.

*Cipher RSA_Cipher = Cipher.getInstance("RSA");*

After constructing the Cipher Object, it has to be initialized to do one of the following four operations, encryption, decryption, key wrapping, or key unwrapping. Encryption and decryption of data with a specific secret key depend on how the cipher object is initialized. Along with the operation mode (e.g. *ENCRYPT_MODE, DECRYPT_MODE, WRAP_MODE or UNWRAP_MODE*), we pass the public key, when calling the *init* method, to initialize the cipher object to do the encryption operation using the provided public key.

*RSA_Cipher.init(Cipher.ENCRYPT_MODE, pub_key);*

*byte[] encrypted_Msg =*

*cipher.doFinal(("Hello: Please encrypt me! ").getBytes());*

For the *Cipher.doFinal* method to do a decryption operation, the Cipher object has to be initialized with an operation mode of decryption (operation mode of *DECRYPT_MODE*) and the key has to be set to the private key.

```
RSA_Cipher.init(Cipher.DECRYPT_MODE, priv_key );
byte[] decrypted_Msg = cipher.doFinal(encrypted);
```

## 4.5 Supporting Tools

With the help of a developed software tool and WinHex [50], we managed to find different key signatures that we will describe in section 4.6. An undocumented windows library "dumpmem.h" allows taking a snapshot of memory that can be searched to locate cryptographic keys and key signatures. While WinHex presents the memory pages of a process as a continuous block.

We ran our experiments on a machine running windows XP with Service Pack 2 installed and logged in an administrator privilege user account. We searched the memory looking for cryptographic keys and tried to determine the key signatures of different cryptographic algorithms on different experiment settings.

The tool allowed us to search the memory and locate the AES secret key. It also allowed us to perform a text search to find the text associated with the AES key signature.

The tool was useful while performing the RSA experiment, we used the binary search capability to find different cryptographic parameters of RSA such as the large prime number $p$, $q$, and the modulus n where:

$$n = p \times q$$

In the next section, we will be describing the key signatures for locating cryptographic keys. These signatures will allow locating keys of different cryptographic systems such as AES secret key and RSA private key.

## 4.6 Cryptographic Key Signature and Cryptographic Key Scanner

We define a cryptographic key signature as the pattern that allows finding a cryptographic key that is hidden in a large amount of data (e.g. gigabytes of data). The signature can be a characteristic byte-pattern; a fingerprint that can be used in locating the cryptographic key.

We introduce the idea of cryptographic key scanner, which is a program that uses the cryptographic key signature to find cryptographic keys in large amount of data.

### 4.6.1 AES Key Signature

In this section, we report the finding of AES secret keys in memory then we identify AES key signature; a pattern that allows finding AES keys in memory. We run experiments on many Java written programs using the "SUNJCE" provider as the implementation provider for JCE. All the

experimented programs use AES to do cryptographic operations. The key length of AES keys used was 128 bits. We ran the programs on a pc running Windows XP with service pack 2 and SUN JDK 1.6.0-b105 installed.

We report that whenever one of the described programs is doing an AES encryption or decryption operation on data, the AES secret key will be in the memory in address Y, where: Y= X + 0x48+ 0x 30.

| Address of | Shift address |
|---|---|
| String S1= "java.security.key " | X |
| String S2= "java/security/key" | X + 0x48 |
| AES secret key | Y= X + 0x48+ 0x30 |

**Table 4.1:** AES-128 key signature associated with encryption and decryption

Whenever we find the string *"java.security.key"* at address X, and String *"java/security/key"* at address X + 0x48, we were able to locate the AES secret key at address Y= X + 0x48+ ox30.

This means that we managed to identify an AES key signature for these described programs. An attacker running a cryptographic key scanner can use the above signature to locate AES secret keys in memory. What the cryptographic key scanner would do is that it will look for

the string S1 *("java.security.key")* in memory. It might look in the memory portion associated with a process or in the whole memory for that string. When S1 is located at address X, the scanner will check for string S2 *("java/security/key")* at address X + 0x48. If S2 is found, then the attacker will be able to locate an AES secret key at address Y= X + 0x48+ 0x 30.

| **Address of** | **Shift address** |
|---|---|
| String S3= "javax.crypto.spec.security.key " | X |
| String S4= "java/crypto/spec/securitykeyspec " | X + 0x68 |
| AES secret key | Y= X + 0x68+ 0x60 |

**Table 4.2:** AES-128 key signature associated with key generation

In addition, we report that whenever one of the described programs is doing an AES key generation operation, the generated AES secret key will be in the memory at address Y where:

$$Y = X + 0x68+ 0x\ 60.$$

Whenever we find the string "javax.crypto.spec.security.key" at address X and string "java/crypto/spec/securitykeyspec" at address X + 0x68, then there we were able to locate the AES secret key at address Y= X + 0x68+ ox60.

This means that we managed to identify an AES key signature for any program that generates AES keys. An attacker running a cryptographic key scanner can use the above signature to locate AES secret key. The cryptographic key scanner will look for the string S3 "javax.crypto.spec.security.key" in memory. It might look in the memory portion associated with a process or the whole memory for that string. When S3 is located at address X, the scanner will check for string S4 "java/crypto/spec/securitykeyspec" at address X + 0x68. If S4 is found, then the AES secret key will be found at address Y= X + 0x68+ 0x60.

The scanner can omit checking string S4 by directly checking the AES secret key at address Y once it finds S3 at address X. However, by doing this and not using the whole key signature, the accuracy of the scanner goes down and the ratio of false positives goes up.

This finding means that an attacker can run different types of attacks to extract AES secret keys which are still persisting in the memory. For example, a lunch time attack can locate an AES secret key for some critical information that the user has encrypted.

In some cases, the secret keys were not found exactly at address Y but rather a few bytes before or after Y. However, the keys were always very relatively close to where the key signature points.

When replacing Sun JDK with BEA JRockit 5.0 we found similar results. We have been able to locate AES Keys when doing encryption and decryption of messages and when generating keys. Tables [4.3, and 4.4] illustrate those results and define the AES key signatures when using BEA JRockit 5.0 in both cases.

| Address of | Shift address |
|---|---|
| String S1= "java.security.key" | X |
| S2=String "java.security.key" | X + 0x68 |
| AES secret key | Y= X + 0x48+ 0xc0 |

**Table 4.3:** AES -128 key signature associated with encryption and decryption when using BEA JRockit 5.0

| Address of | Shift address |
|---|---|
| String S3= "javax.crypto.spec.security.key" | X |
| String S4= javax.crypto.spec.secretkeyspec " | X + 0x80 |
| AES secret key | Y= X + 0x80 + 0x80 |

**Table 4.4:** AES-128 key signature associated with key generation when using BEA JRockit 5.0

### 4.6.2 RSA Key Signature

In this section we report the finding of RSA private keys in memory then we define an RSA Key signature; a pattern that allows finding RSA keys in memory. The host operating system that was used in these experiments was Windows XP with service pack 2 and SUN JDK 1.6.0-b105 installed. We run our experiments on many Java written programs using the "SUNJCE" provider as the implementation provider for JCE. All these programs use RSA to do cryptographic operations and they used RSA keys of 512 bits length.

We have not been able to associate the RSA key with fixed string as we did in AES. However, we managed to define a key signature using modulus n which is part of the public key. We report that whenever one of the described programs is doing an RS

A encryption or decryption operation on data, the RSA prime numbers $p$ and $q$ was in memory at addresses Y and Z respectively, where: Y= X + 0x48+ 0x30 and Z= X + 0x48+ [0x20:0x23].

| **Address of** | **Shift address** |
|---|---|
| Modulus n | X |
| $p$ | Y= X + 0xD0 |
| $q$ | Z= X + 0x48+ [0x20:0x23] |

**Table 4.5:** RSA-512 key signature associated with encryption and decryption

When we find the modulus n at address X, Modulus n is part of the public key and is not considered secret, we find the prime numbers $p$ and $q$ at addresses Y and Z defined in Table [4.5]. Finding these two prime numbers expose the private key.

This means that we managed to identify the RSA key signature for the described programs, and an attacker running a cryptographic key scanner can use the above signature to locate RSA private key. What the cryptographic key scanner would do is that it will look for the host modulus n in memory. It might look in the memory portion associated with a process or in

the whole memory for that string. When modulus n is located at address X, the scanner will check for $p$ at address Y= X + 0xD0 and $q$ at address Z= X + 0x48+ [0x20:0x23].

This finding means that an attacker can run different types of attacks to extract RSA private keys that are still persisting in the memory. For example, confidential information sent over the network can be compromised if an attacker could locate the RSA private key that was used to encrypt this information. An attacker can execute a lunch time attack to extract the RSA private key, getting access to a users private key allows the attacker to implement many attacks. The attacker can use this key to forge the host's digital signature or trick an authentication system into trusting him as the real user.

It should be noted that other sensitive financial information, such as credit card numbers, can be located using these memory scanning techniques. For example, it is known that credit card numbers satisfy simple check sum formula (mod 10). Although the main objective of this restriction is to facilitate the detection of errors when credit card numbers are used to carry any financial transactions, the same detections algorithm can be used to detect target credit card numbers in memory. Our experimental results show that using these well known check sums can allow the attacker to easily gather potential credit card numbers from the memory. However, because of the large number of false alarms, other signature information is needed to make such attacks more practical.

## 4.7 Countermeasures

In order to hide cryptographic keys so that it is not accessible to cryptographic key scanners, some techniques have been suggested. Some of these techniques make the task of the scanners only more difficult but still possible.

One of the best techniques is to keep the cryptographic keys inside a smart card. Not only the cryptographic key or part of it should not leave the smart card, but also encryption, decryption and signing of data should take place inside the smart card itself [55]. Other techniques include spreading the key over the program to lower the entropy of the key and hence avoid entropy attacks that exploits the randomness properties of keys. Some techniques require computation overhead such as constructing a set of values that when combined, they result into the key. Then, it suggests spreading these values over the program [56].

White-box implementation has been suggested as a solution to prevent secret key extraction on software running on malicious hosts. White-box implementation is widely used in Digital Rights Management (DRM) software where the attacker has full access to the implementation and execution of the software [45, 46, and 47]. Till now, white-box implementations were used to embed fixed keys inside the implementation. We believe that a dynamic-key white-box implementation, which is still a subject of ongoing research, can be a software solution for preventing key extraction out of programs.

Another suggested method suggests distributing the secret keys among several systems, a technique used to lower the chance of secret key exposure. A *Threshold cryptosystem* is one example. *Threshold cryptosystems* are mostly useful for organizations preferring a public key for

the entire organization rather than a public key for each individual. Desmedt et al. [49] proposed a method where for everyone in the organization to read a message, she must gather enough employees with the required number of shadows to decrypt the message. This suggested method alters the decryption process of RSA by requiring the employees to do partial calculations. Each shareholder will transmit the result of the partial calculation separately to the designated individual who will decrypt the message using those partial results.

Another approach to mitigate the risk of secret key exposure is called *Forward Security.* The basic concept of *Forward Security* is to reduce the usage time of secret keys to short time periods and make secrets from different time periods independent. Having short time periods mitigate the risk of exposure of one of secrets, and making the secret keys independent limits the risks to only messages encrypted during the short time period using the exposed key. However, changing keys too often might be a challenge to the system. For instance, in a public key infrastructure changing the public key frequently adds an inconvenience to many solutions. Dodis et al. [48] proposed a key-insulated-security system where even when an adversary learns a current signing key, she cannot generate signatures from future time periods. Two modules are used: a home base and a signer. The signer, which is usually mobile, possesses a secret signing key. The signer uses this secret signing key to sign messages. When the singing key expires at the end of every time period, the signer updates the secret signing key by contacting the home base then doing some local computations. The home base is usually stationary. The mobility of the singer makes it more vulnerable to key exposure. However, such a key exposure is less

helpful to an adversary because the key was used for only short period of time. This model provides the ability to limit the compromise of a key exposure to only one time period.

## 4.8 Summary and Conclusion

Techniques for efficiently locating secret keys in memory, as opposed to cryptanalysis computations of unknown secret keys, have started to receive much attention.

Revealing cryptographic keys used by enterprises has become a continuous threat. Such attacks are effective especially when users utilize a shared computer such as in libraries, computer labs, or Internet Café's.

We explained how modern operating system tries to protect its running processes through different methods such as process isolation. Then we gave an overview of Java cryptography Architecture (JCA) and Java Cryptography Encryption (JCE). Then, we showed how we used a set of tools to locate cryptographic keys. After that, we defined the terms key signatures and cryptographic key scanner.

We examined the key signature of programs using two different implementations of JCE to do cryptographic operations. We showed a detailed example of how to locate AES secret key and RSA private key in memory and how an attacker can use this vulnerability to lunch different types of attacks.

# Chapter 5

# Conclusion and Future works

In this chapter, we provide a summary of the main achievements of this thesis. We also point out some future research directions that can lead into enhancing the development of secure e-commerce protocols, and DRM systems.

## 5.1 Achievements

Throughout this work, we proposed a protocol that can be used as a building block for achieving a secure fair exchange of valuable items between multiple parties within the multi-level marketing business model and hence increase the users' confidence in e-commerce. Our protocol is characterised with the following features:

- Ability to provide exchange of items between a distributor, a client, and a merchant with the aid of a trusted third party.
- Fair exchange: That is, as the distributor provides a referral, she is assured to get her commission; the client is assured to get the product or service she pays for; and the merchant is also assured to get paid for the product or service she provides. In doing so, the trusted third party ensures either all three parties get their expected items or none of them does. In this sense, our protocol provides strong fairness.

- Content assurance: each one of the three parties should get her exact expected item. In a DRM system, a customer who has bought a song or a book with a certain quality in a certain format should get her purchased item in the exact right specification that she has ordered. And so does the merchant and distributor; they should get their exact amount of payment.

We provide some informal security analysis of the protocol. We provide formal verification of the fairness property by applying the following steps:

- Modelling the protocol using PROMELA modelling language.
- Defining the *strong fairness* property using the linear temporal logic.
- Running the SPIN model checker to verify the fairness property against the PROMELA model of the protocol. Verification provides the assurance of the correctness of the security requirements of the protocol.

The multi-party exchange protocol can be used as the core protocol that provides the exchange of merchandise between parties for an online implementation of multi-level marketing e-commerce business.

On the other hand, a secure protocol or application has to run in a secure environment. A system is assumed to be secure only under a set of assumptions including keeping the cryptographic keys secret. Once the cryptographic secret is vulnerable, the security of entire system becomes vulnerable as well.

In the second part of the thesis, we analyzed the predictability of locating cryptographic keys and passwords in memory by running multiple experiments on multiple cryptographic algorithms. Our main focus was on detecting key signature that can help locating cryptographic keys in the "SUNJCE" implementation of the Java Cryptography Extension. We presented the key signature of both a symmetric and a public-key algorithm namely, AES and RSA respectively as examples. We also discussed a set of countermeasures that makes exploiting such vulnerabilities more difficult.

## 5.2 Future Research Work

In this section, we provide some recommendations for future work that can further enhance our research. These recommendations include the following:

- Providing a formal analysis for the other security properties of the proposed multi-party fair exchange protocol.
- Designing and implementing different flavours of the multi-party fair exchange protocol with a Semi Trusted Third Party and offline Trusted Third Party.
- Designing a solution that implements a complete multi-level marketing business model and uses the multi-party fair exchange protocol as e-commerce protocol for payment and exchange of items. Such solution may help identifying additional functionalities that need to be added to the fair exchange protocol.
- Finding out the key signature for other symmetric and public key algorithms.

- Discovering whether other cryptographic libraries of other programming languages are vulnerable, and whether key signature can be determined for such libraries, for example, determining if the *System.Security.Cryptography* cryptography library of Microsoft .net languages is vulnerable to such attacks.
- Designing cryptographic libraries that are not suitable to such vulnerabilities. One cryptographic technique that can be useful for designing such a technique is Dynamic white box implementation.

# List of References


[1] W. Hanson, and K. Kalyanam, *Internet Marketing and e-Commerce.* South-Western College Pub, 2006.

[2] I. Ray and I. Ray, "Fair Exchange in E-commerce", in *ACM SIGEcomm Exchange,* September 2001.

[3] M. Geevarghese K., J. Manalel, and S. Zacharias, "Network Marketing: Exploitation of Relationships – Myth or Reality?," *International Marketing Conference on Marketing & Society*, 8 Oct. 2007.

[4] K. Harrison and S. Xu," Protecting Cryptographic Keys From Memory Disclosure Attacks," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07),* June, 2007.

[5] "Federal Trade Commission," last accessed: April 2008. [Online]. Available: www.ftc.gov

[6] H. Pagnia and F. Gartner. "On the Impossibility of Fair Exchange without a Trusted Third Party," TUD-BS-1999-02, University of Darmstadt, Germany, Technical Report, 1999.

[7] T. Tedrick, "How to exchange half a bit," in *Advances in Cryptology: Proceedings of Crypto'83*, pp. 147–151, New York, 1984. Plenum Press.

[8] D. Boneh and M. Naor, "Timed commitments," in *Advances in Cryptology: Proceedings of Crypto 2000*, vol. 1880 of Lecture Notes in Computer Science, pp. 236–254. Springer-Verlag, 2000.

[9] T. Tedrick, "Fair exchange of secrets," in *Advances in Cryptology: Proceedings of Crypto'84*, vol. 196, Lecture Notes in Computer Science, pp. 434–438. Springer-Verlag, 1985.

[10] O. Markowitch and S. Saeednia, "Optimistic fair-exchange with transparent signature recovery," in *5th International Conference, Financial Cryptography*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[11] S. Kosch, *Kosch's guide to network marketing in Canada*. Incor Publications, 1996.

[12] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual,* New York: Addison-Wesley Professional, 2003.

[13] A.Nenadić, "A security solution for fair exchange and non-repudiation in e-commerce," P.hd Thesis, University of Manchester, July 2005.

[14] G. Lowe, "An Attack on Needham-Schroeder Public-Key Authentication Protocol," *Information Processing Letters*, vol. 56, no. 03, pp. 131–133, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, 1995.

[15] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[16] G. Lowe, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR," *in Proceedings of the International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* - TACAS '96, vol. 1055, pp. 147–166. LNCS, Springer-Verlag, London, UK, 1996.

[17] J. Zhou and D. Gollmann, "Towards Verification of Non-Repudiation Protocols," *in Proceedings of the International Refinement Workshop and Formal Methods Pacific*, pp. 370–380. Springer, 1998.

[18] "LTL checking with SPIN," last accessed: April 2008. [Online]. Available: http://spinroot.com

[19] J. C. Mithcell, M. Mitchell, and U. Stern, "Automated Analysis of Cryptographic Protocols using *Mur*$\Psi$," *In Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 141–151. IEEE Computer Society Press, 1997.

[20] J. C. Mithcell, V. Shmatikov, and U. Stern, "Finite-State Analysis of SSL 3.0," *In Proceedings of the USENIX Security Symposium*, pp. 201–215, 1998.

[21] C. Meadows, "Analysing the Needham-Schroeder Public-Key Protocol: A Comparison of Two Approaches," *In Proceedings of the European Symposium on Research in Computer Security - ESORICS '98*, pp. 365–384. LNCS, Springer-Verlag, Berlin, Germany, 1996.

[22] G. Bella and L. C. Paulson, "Mechanising a Protocol for Smart Cards," *In Proceedings of the International Conference on Research in Smart Cards - eSmart '01*, vol. 2140, pp. 19–33. LNCS, Springer-Verlag, London, UK, 2001.

[23] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano, "Formal Verification of Cardholder Registration in SET," *In Proceedings of the European Symposium on Research in Computer Security - ESORICS '00*, vol. 1895, pp. 159–174. LNCS, Springer-Verlag, London, UK, 2000.

[24] L. C. Paulson, "Inductive Analysis of the Internet Protocol TLS," *ACM Transactions on Computer and System Security*, vol. 02, no. 03, pp. 332–351, 1999.

[25] L. C. Paulson, "Proving Properties of Security Protocols by Induction," *In Proceedings of the Computer Security Foundations Workshop*, pp. 70–83. IEEE Computer Society Press, 1997.

[26] V. Shmatikov and J. Mitchell, "Analysis of Abuse-Free Contract Signing," *In Proceedings of the International on Conference Financial Cryptography FC '00*, vol. 1962, pp. 174–191. LNCS, Springer-Verlag, London, UK, 2001.

[27] V. Shmatikov and J. Mitchell, "Analysis of a Fair Exchange Protocol," *In Proceedings of the Symposium on Network and Distributed Systems Security - NDSS '00*, pp. 119–128. Internet Society, 2000.

[28] V. Shmatikov and J. Mitchell, "Finite-State Analysis of Two Contract Signing Protocols," *Special issue of Theoretical Computer Science on Security*, 283(2):419–450. Elsevier Science Publishers Ltd., Essex, UK, 2002.

[29] S. Kremer and J-F. Raskin, "A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols," *Journal of Computer Security*, vol. 11, no. 03, pp. 399–429. IOS Press, Amsterdam, The Netherlands. 2003.

[30] S. Kremer and J-F. Raskin, "Game Analysis of Abuse-Free Contract Signing," *In Proceedings Computer Security Foundations Workshop - CSFW '02*, pp. 206–222. IEEE Computer Society Press, 2002.

[31] S. Kremer and J-F. Raski, "A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols, " *In Proceedings of International Conference on Concurrency Theory CONCUR '01*, vol. 2154, pp. 551–566. LNCS, Springer-Verlag, Berlin, Germany, 2001.

[32] S. Schneier, "Formal Analysis of a Non-Repudiation Protocol," In *Proceedings of the IEEE Computer Security Foundations Workshop*, pp. 54–65. IEEE Computer Society Press, 1998.

[33] "MRMC," last accessed: April 2008. [Online]. Available: www.cs.utwente.nl/~zapreevis/mrmc/

[34] "BLAST: Berkeley Lazy Abstraction Software Verification Tool," last accessed: April 2008. [Online]. Available: http://mtc.epfl.ch/software-tools/blast/

[35] J. Holzmann . *The SPIN Model Checker: Primer and Reference Manual*. New York: Addison-Wesley Professional, 2003.

[36] L. C. Paulson, "Isabelle: A Generic Theorem Prover," *LNCS 828*, Springer-Verlag, 1994.

[37] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tapcsiran , "MOCHA: Modularity in Model Checking," *In Proceedings of the International Conference on Computer-Aided Verification*, vol. 1427, pp. 521–525, LNCS, Springer-Verlag, 1998.

[38] D. L. Dill, A. J. Drexler, A. J. Hu, and C. Han Yang, "Protocol Verification as a Hardware Design Aid," *In Proceedings of the IEEE Conference on Computer Design: VLSI in Computers and Processors*, pp. 522-525. IEEE Computer Society, 1992.

[39] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. "Counterexample-guided abstraction refinement," *Proceedings of the 12th International*

*Conference on Computer Aided Verification (CAV '00)*, vol. 1855, Lecture Notes in Computer Science, pp. 154-169. Springer-Verlag, July 2000.

[40] W. Stallings, *Operating Systems: Internals and Design Principles (4th Edition)*. Alexandria, VA: Prentice Hall, 2000.

[41] "Java ™ Cryptography Extension (JCE), Reference Guide," last accessed: April 2008. [Online]. Available: http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html

[42] P. Dabak, M. Borate, S. Phadke , *Memory Management*, M&T Books, October 1999, Available: http://www.windowsitlibrary.com/Content/356/04/2.html

[43] A. Shamir, and N. van Someren, "Playing Hide and Seek with Stored Keys," *in Financial Cryptography*, pp. 118-124, 1999.

[44] J. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," [Online]. Available: http://citp.princeton.edu/pub/coldboot.pdf

[45] S. Chow, P. Eisen, H. Johnson, P. van Oorschot, "A white-box DES implementation for DRM applications," *In Proceedings of DRM 2002 - 2nd ACM Workshop on Digital Rights Management,* 2002.

[46] S. Chow, P. Eisen, H. Johnson, P. van Oorschot, "A white-box cryptography and an AES implementation," *In Proceedings of SAC 2002 - 9th Annual Workshop on Selected Areas in Cryptography*, Lecture Notes in Computer Science, pp. 250–270, Springer, 2002.

[47] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, "White-Box Cryptography and an AES Implementation," *Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002)*, Springer-Verlag lncs, August , 2002.

[48] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," *in Advances in Cryptology (Eurocrypt 2002)*, vol. 2332, pp. 65-82, 2002.

[49] Y. Desmedt, and Y. Frankel, "Threshold cryptosystems,". In Proceedings Crypto'89, pp. 307–315.

[50] "WinHex," last accessed: April 2008. [Online]. Available: http://www.x-ways.net/winhex/

[51] B. Bloch, "Multi-level marketing: What's the catch?" *Journal of Consumer Marketing*, 1996, pp. 18-26.

[52] "U.S. Securities and Exchange Commission," last accessed: April 2008. [Online]. Available: http://www.sec.gov/answers/ponzi.htm

[53] "BreachBlog," last accessed: April 2008. [Online]. Available: http://breachblog.com/

[54] "PROMELLA References," last accessed: April 2008. [Online]. Available: http://spinroot.com/spin/Man/Quick.html

[55] M. Blaze, J. Feigenbaum, and M. Naor, "A Formal Treatment of Remotely Keyed Encryption," in: *Eurocrypt '98*, Springer LNCS.

[56] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, Springer, 2004.

[57] "E-commerce," last accessed: April 2008. [Online]. Available: http://spinroot.com/spin/Man/Quick.html