E-Commerce Security Enhancement and Anomaly Intrusion Detection Using Machine Learning Techniques

A thesis submitted for the degree of
Doctor of Philosophy

Xuan Dau Hoang
B.IT. (Hons), M.Eng. (RMIT University)

School of Computer Science and Information Technology,
Science, Engineering, and Technology Portfolio,
RMIT University,
Melbourne, Victoria, Australia.

February 2006

Declaration

This thesis contains work that has not been submitted previously, in whole or in part, for any other academic award and is solely my original research, except where acknowledged. The work has been carried out since the beginning of my candidature on 22 January 2002.

Xuan Dau Hoang
School of Computer Science and Information Technology
Royal Melbourne Institute of Technology
February 22, 2006

Acknowledgments

I would like to thank my supervisors, Dr. Jiankun Hu and Dr. Peter Bertok for their constant assistance and support in the preparation of this thesis.

I also would like to thank all staff and my fellow research students in the Distributed Systems and Networking Discipline for their valuable comments on my work.

Many thanks to Rita Healy, Chin Scott, and Beti Dimitrievska, the administrative staff of the School of Computer Science and Information Technology, RMIT University.

I also would like to thank Dr. Maria Hurley, who is from Writing Clear Science, for her great assistance in proofreading of my thesis.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- Xuan Dau Hoang and Jiankun Hu. New Encryption model for secure E-commerce transactions using dsp–host, board and server communication. In *IEEE International Conference on Telecommunications*, Vol.1, pages 166–170, June 2002.
- Xuan Dau Hoang, Jiankun Hu and Peter Bertok. Intrusion Detection Based on Data Mining. In *International Conference on Enterprise Information Systems*, April 2003.
- Xuan Dau Hoang, Jiankun Hu and Peter Bertok. A multi-layer model for anomaly intrusion detection using program sequences of system calls. In *International Conference on Network - ICON2003*, Vol.2, pages 531–536, Sydney, Australia, September 2003.
- Xuan Dau Hoang and Jiankun Hu. An Efficient Hidden Markov Model Training Scheme for Anomaly Intrusion Detection of Server Applications Based on System Calls. In *International Conference on Network - ICON2004*, Vol.2, pages 470–474, Singapore, November 2004.

Note

Unless otherwise stated, all fractional results have been rounded to the displayed number of decimal figures.

Contents

List of Figures

List of Tables

Abstract

With the fast growth of the Internet and the World Wide Web, security has become a major concern of many organizations, enterprises and users. Criminal attacks and intrusions into computer and information systems are spreading quickly and they can come from anywhere on the globe. Intrusion prevention measures, such as user authentication, firewalls and cryptography have been used as the first line of defence to protect computer and information systems from intrusions. As intrusion prevention alone may not be sufficient in a highly dynamic environment, such as the Internet, intrusion detection has been used as the second line of defence against intrusions. However, existing cryptography-based intrusion prevention measures implemented in software, have problems with the protection of long-term private keys and the degradation of system performance. Moreover, the security of these software-based intrusion prevention measures depends on the security of the underlying operating system, and therefore they are vulnerable to threats caused by security flaws of the underlying operating system. On the other hand, existing anomaly intrusion detection approaches usually produce excessive false alarms. They also lack in efficiency due to high construction and maintenance costs. In our approach, we employ the "defence in depth" principle to develop a solution to solve these problems.

Our solution consists of two lines of defence: preventing intrusions at the first line and detecting intrusions at the second line if the prevention measures of the first line have been penetrated. At the first line of defence, our goal is to develop an encryption model that enhances communication and end-system security, and improves the performance of web-based E-commerce systems. We have developed a hardware-based RSA encryption model to address the above mentioned problems of existing software-based intrusion prevention measures. The proposed hardware-based encryption model is based on the integration of an existing web-based client/server model and embedded hardware-based RSA encryption modules. DSP embedded hardware is selected to develop the proposed encryption model because of its advanced security

features and high processing capability. The experimental results showed that the proposed DSP hardware-based RSA encryption model outperformed the software-based RSA implementation running on Pentium 4 machines that have almost double clock speed of the DSP's clock speed at large RSA encryption keys.

At the second line of defence, our goal is to develop an anomaly intrusion detection model that improves the detection accuracy, efficiency and adaptability of existing anomaly detection approaches. Existing anomaly detection systems are not effective as they usually produce excessive false alarms. In addition, several anomaly detection approaches suffer a serious efficiency problem due to high construction costs of the detection profiles. High construction costs will eventually reduce the applicability of these approaches in practice. Furthermore, existing anomaly detection systems lack in adaptability because no mechanisms are provided to update their detection profiles dynamically, in order to adapt to the changes of the behaviour of monitored objects. We have developed a model for program anomaly intrusion detection to address these problems.

The proposed detection model uses a hidden Markov model (HMM) to characterize normal program behaviour using system calls. In order to increase the detection rate and to reduce the false alarm rate, we propose two detection schemes: a two-layer detection scheme and a fuzzy-based detection scheme. The two-layer detection scheme aims at reducing false alarms by applying a double-layer test on each sequence of test traces of system calls. On the other hand, the fuzzy-based detection scheme focuses on further improving the detection rate, as well as reducing false alarms. It employs the fuzzy inference to combine multiple sequence information to correctly determine the sequence status. The experimental results showed that the proposed detection schemes reduced false alarms by approximately 48%, compared to the normal database scheme. In addition, our detection schemes generated strong anomaly signals for all tested traces, which in turn improve the detection rate.

We propose an HMM incremental training scheme with optimal initialization to address the efficiency problem by reducing the construction costs, in terms of model training time and storage demand. Unlike the HMM batch training scheme, which updates the HMM model using the complete training set, our HMM incremental training scheme incrementally updates the HMM model using one training subset at a time, until convergence. The experimental results showed that the proposed HMM incremental training scheme reduced training time four-fold,

compared to the HMM batch training, based on the well-known Baum-Welch algorithm. The proposed training scheme also reduced storage demand substantially, as the size of each training subset is significantly smaller than the size of the complete training set.

We also describe our complete model for program anomaly detection using system calls in chapter 8. The complete model consists of two development stages: training stage and testing stage. In the training stage, an HMM model and a normal database are constructed to represent normal program behaviour. In addition, fuzzy sets and rules are defined to represent the space and combined conditions of the sequence parameters. In the testing stage, the HMM model and the normal database, are used to generate the sequence parameters which are used as the input for the fuzzy inference engine to evaluate each sequence of system calls for anomalies and possible intrusions. The proposed detection model also provides a mechanism to update its detection profile (the HMM model and the normal database) using online training data. This makes the proposed detection model up-to-date, and therefore, maintains the detection accuracy.

Chapter 1

Introduction

With the fast growth of the Internet and the World Wide Web, computer and information systems have increasingly become the targets of criminal attacks and intrusions. Attacks spread very quickly and they can come from anywhere on the globe [114]. The reported number of computer and network attacks rises sharply every year [11, 35]. Therefore, finding the best possible ways to protect valuable information and computer systems against intrusions is crucial.

An attack, or intrusion, on a system is a security policy breach. Most attacks cause security policy breaches in very specific ways [5]. For example, certain attacks may enable an attacker to read specific files, but they do not allow the attacker to modify any system components. Another attack may cause a system service disruption to authorized users, but it does not allow the attacker to access any files. Athough computer and network attacks vary in types and capabilities, they usually cause breaches of four different security properties of the system [3, 5, 70]:

- *Confidentiality*: An attack causes a confidentiality breach if it allows unauthorized access to data.
- *Integrity*: An attack causes an integrity breach if it allows unauthorized modification to the system state or data.
- *Availability*: An attack causes an availability breach if it keeps authorized users from accessing a particular system resource when they need it.
- *Control*: An attack grants an attacker privilege to interfere with system operation in violation of the access control policy of the system. This can lead to a subsequent confidentiality,

integrity, or availability breach.

In regard to the violation to these four security properties, there are two common types of attacks: denial of service (DoS) attacks and system penetration attacks. DoS attacks may cause an availability breach because they attempt to slow or shut down the targeted systems or services. For example, "ping of death" attacks can crash certain Microsoft Windows systems [5]. In contrast, system penetration attacks may cause breaches to all four security properties. System penetration attacks usually involve unauthorized acquisition and/or modification of system privileges, resources, or data. For example, a "Remote Disk Read" attack allows an attacker on the network to read private data files on the target system without the owner's authorization [5].

There have been several approaches to protect computer and information systems from criminal attacks and intrusions. Among them, *defence in depth* is the best asset-protection strategy, in which several security mechanisms are combined to create a defence wall with multiple protection layers [3, 76]. This strategy is a result of the fact that all security mechanisms can be vulnerable to some form of compromise. Furthermore, it may require an enormous cost to substantially increase the strength or reliability of any given security mechanism [76]. In light of the *defence in depth* strategy, intrusion prevention has been used as the first line of defence, and intrusion detection has been used as the second line of defence, to protect computer and information systems from intrusions.

Intrusion prevention: first line of defence

Generally, intrusion prevention measures, which include a set of access control mechanisms, such as authentication, firewalls and cryptography, have been used to protect computer and information systems as the first line of defence [81, 70]. Each access control mechanism can be used to protect a resource in a very specific way. For example, authentication can be used to verify the identity of a user, a process or a computer, that attempts to gain access to a resource. On the other hand, a firewall can be used to protect a local system or a private network from network-based security threats by controlling the incoming and outgoing traffic. In our approach, we focus on improving the security and performance of cryptography-based intrusion prevention measures for E-commerce applications.

Intrusion prevention measures for E-commerce systems must ensure two important security

aspects: communication security and end-system security [32]. Communication security has to ensure the confidentiality, integrity, non-repudiation and authenticity of the information exchanged among communicating parties. End-system security requires strong security measures to protect sensitive information, such as long-term private keys.

Cryptography techniques have been long in use to address the communication security problems, such as confidentiality, integrity, non-repudiation of origin and authenticity. Cryptography plays a crucial role in software-based security packages and protocols, such as Secure Socket Layer (SSL) and Transport Layer Security (TLS). Researchers have pointed out that software-based security packages in general and SSL/TLS in particular, have problems with the protection of long-term private keys and the degradation of performance [32, 39, 61]. Furthermore, since the security of these software-based security packages depends on the security of the underlying operating system, they are susceptible to threats caused by security flaws of the underlying operating system. Therefore, improving the communication security, end-system security and performance of existing E-commerce systems is a major objective of this research.

Intrusion detection: second line of defence

Intrusion detection is usually needed as the second line of defence, when previous intrusion prevention measures of the first line of defence have been bypassed. The major benefits of an intrusion detection system (IDS) include [5, 81]:

- Real-time reporting of intrusions and misuses so that appropriate actions can be taken.
- Auditing of system configurations and vulnerabilities.
- Acting as quality control for security design and administration, especially of large and complex enterprises.
- Providing useful information about intrusions that take place, allowing enhanced diagnosis, recovery, and elimination of causative factors.

There are two broad categories of intrusion detection techniques: *misuse detection* and *anomaly detection* [26, 93]. In misuse detection, known attacks or system weaknesses are first encoded into patterns or signatures. Then, these signatures are used to verify the current activities, and any matched activity is considered an attack. In anomaly detection, a detection

profile is first constructed to characterize normal behaviour or historical activity patterns of an object. Then, the detection profile is used to verify the object's current activities, and any significant deviation from normal behaviour is considered a possible attack. While misuse detection can only effectively detect known attacks, anomaly detection has the potential to detect novel and unknown attacks. This is because no advance knowledge about specific attacks is required in anomaly detection [26, 93]. In our approach, we focus on anomaly detection, specifically in detecting anomalies in the execution of programs, as well as intrusions against programs.

Accuracy, efficiency and adaptability are the most important properties of any intrusion detection systems [81, 82]. Generally, existing anomaly detection systems usually produce excessive false alarms, which will virtually disrupt the normal operations of the monitored systems. In addition, several anomaly detection approaches, such as those presented in [102, 134] suffer a serious efficiency problem due to high training costs of the detection profiles. Furthermore, existing anomaly detection systems lack in adaptability, because no mechanisms are provided to update their detection profiles dynamically, in order to adapt to the changes of the behaviour of monitored objects [81, 82]. Therefore, improving the detection accuracy, efficiency and adaptability of existing program anomaly intrusion detection approaches is another major objective of this thesis research.

1.1 Thesis Aims

We propose a solution to protect computer and information systems against intrusions with two lines of defence: preventing intrusions at the first line, and detecting intrusions at the second line if the prevention measures of the first line have been bypassed.

Specifically, we focus on enhancing the following aspects of existing E-commerce systems at the first line of defence:

- Communication security
- End-system security
- Performance.

And we aim at improving the following characteristics of existing program anomaly intrusion

detection approaches at the second line of defence:

- Accuracy
- Efficiency
- Adatability.

1.2 Thesis Contributions

This thesis research provides contributions to three fields: E-commerce security, machine learning and anomaly intrusion detection.

1.2.1 Hardware-based RSA encryption model

This thesis develops a hardware-based RSA encryption model for the enhancement of security and performance for E-commerce applications.

This thesis presents a hardware-based RSA encryption model, aiming at enhancing security and performance for E-commerce applications with the following contributions:

1. Hardware-based RSA encryption modules are developed to enhance the security and performance for E-commerce applications. The security of the proposed encryption model is assured by the strength of RSA encryption at large keys and the security of a hardware-based cryptosystem. On the other hand, the model's performance is guaranteed by the hardware's high processing capability.

   Hardware-based cryptosystems are generally considered more secure than software-based counterparts since a hardware based cryptosystem runs on its own physical memory space, and therefore it can effectively prevent malicious access from external programs. This also means that the security of a hardware-based cryptosystem does not depend on the security of the underlying operating system. Furthermore, hardware can then be used to safely store sensitive information, such as long-term private keys.

   Experimental results showed that the proposed hardware-based RSA encryption model performs better than the software-based RSA implemetation, running on Intel Pentium 4 machines that have almost double clock speed of the DSP's clock speed at large RSA

encryption keys. The results confirmed that the proposed encryption model is more scalable than software based implementations, and it can be used to efficiently improve the security and performance of E-commerce applications.

2. Techniques are developed to integrate hardware-based RSA encryption modules into existing web-based client/server E-commerce systems. These techniques also include a method to improve the throughput of communication channels between hardware-based encryption modules and computer applications.

Some parts of our work in this area, including the above contributions has been published in [41].

1.2.2 HMM incremental training scheme for program anomaly detection

This thesis develops a new HMM incremental training scheme which aims at reducing HMM training time and storage demand. The proposed HMM incremental training scheme can be used to solve the efficiency problem for existing HMM-based program anomaly detection approaches.

Hidden Markov model (HMM) is a powerful data modelling tool in a wide range of applications [103, 104]. The application of Hidden Markov models to characterize the normal behaviour of monitored objects in anomaly detection has also reported high detection accuracy [42, 102, 134]. However, these HMM-based anomaly detection approaches suffered a serious efficiency problem because the construction of HMM models, using the well-known Baum-Welch algorithm, is very expensive in terms of training time and space. The time and space complexities of Baum-Welch algorithm are $O(N^2T)$, where N is the number of HMM states and T is the length of the observation sequence [77, 104]. The high training costs make HMM-based anomaly detection approaches less efficient for use in practice, especially when N and T are large. This thesis presents an efficient HMM incremental training scheme with the following contributions:

1. **An optimal initialization method for HMM training**: Random initialization is a common method used in HMM training, where random generated values are used as initial values for HMM parameters. The HMM training convergence, based on random initialization, is usually slow and this leads to longer training time. This thesis proposes an optimal initialization method to improve the HMM training convergence. Experimental results showed that HMM training based on the proposed initialization method reduced

training time up to 67.36%, compared to HMM training based on random initialization. The proposed initialization method consists of:

(a) An algorithm to compute the *prior probabilities* from training data.

(b) A method to initialize HMM parameters using a combination of the computed prior probabilities and random generated values for HMM training.

2. **An HMM incremental training scheme with optimal initialization**: We improve the HMM training scheme in [16] and make it incremental. Unlike the Baum-Welch algorithm which uses the complete set of training data to update the HMM model, the proposed HMM incremental training scheme only uses one subset of training data at a time to update the HMM model until convergence. This results in faster training convergence and lower storage demand. The optimal initialization method is incorporated into the proposed HMM incremental training scheme to further reduce training time. Experimental results showed that the proposed HMM incremental training scheme with optimal initialization, reduced training time about four times, compared to the Baum-Welch algorithm. The reduction of storage demand is K times, where K is the number of data subsets used. The proposed HMM incremental training scheme includes:

   (a) Weighted merging equations for HMM parameters $\lambda(A, B, \pi)$. These equations are used to merge two same-size HMM models based on their weights.

   (b) An HMM incremental training algorithm based on weighted merging, which updates the HMM model using one subset of training data at a time until convergence.

Some parts of our work in this area, including the above contributions has been published in [43].

1.2.3 HMM-based model for program anomaly detection

This thesis develops an HMM-based model for program anomaly detection using system calls, which aims at improving the detection accuracy, efficiency and adaptability of existing program anomaly detection approaches.

This thesis presents an HMM-based program anomaly detection model using system calls with the following contributions:

1. **A temporal-based anomaly signal evaluation scheme to improve the strength of anomaly signals**: In [25], the anomaly signal was measured by the percentage of the number of abnormal sequences detected in the whole test trace. However, since abnormal sequences usually distribute in small groups across the test trace, this anomaly signal measurement method does not accurately represent the distribution of abnormal sequences in the test trace. A temporal-based anomaly signal evaluation scheme is proposed to improve the strength of anomaly signals by grouping individual abnormal sequences into temporally local regions. The proposed anomaly signal evaluation scheme is used to evaluate the anomaly signal in the proposed HMM-based program anomaly detection model.

2. **A two-layer detection scheme to reduce false positive alarms**: The proposed two-layer detection scheme uses an HMM model and a normal database [25] to create a double-layer test in order to reduce false alarms. Sequences of system calls of test data are first evaluated by the normal database to find mismatched and rare sequences. Then only mismatched and rare sequences are evaluated by the HMM model to find abnormal sequences. Experimental results showed that the proposed two-layer detection scheme effectively reduced the false alarms by about 28%, compared to the normal database scheme [25].

3. **A fuzzy-based detection scheme to further reduce false positive alarms and to increase detection rate**: It is a fact that the normalcy and the abnormalcy are not truly crisp concepts, and therefore it is not always possible to correctly classify an object's behaviour as normal or abnormal using crisp conditions [18, 33, 34]. A fuzzy-based detection scheme is developed to improve this classification problem. In this detection scheme, first fuzzy sets and rules are used to represent the space and combined conditions of parameters of each test sequence of system calls. The sequence parameters are generated by an HMM model and a normal database. Then, the fuzzy reasoning is used to evaluate each test sequence by combining the sequence parameters based on the pre-defined fuzzy sets and rules. Experimental results showed that the fuzzy-based detection scheme reduced false alarms by about 28% and 48%, compared to the two-layer detection scheme and the normal database scheme [25], respectively. Furthermore, it also produced strong anomaly signals for all tested abnormal traces, which helped to improve detection accuracy. The

proposed fuzzy-based detection scheme includes:

(a) An empirical method to represent space and combined conditions of sequence parameters using fuzzy sets and rules.

(b) A fuzzy reasoning method to evaluate each sequence of system calls by combining sequence parameters from multiple sources.

4. The efficient HMM incremental training scheme is used to construct and update the HMM model in the proposed HMM-based anomaly detection model. This improves the proposed detection model's efficiency.

5. **A scheme for the online updating of the detection model**: An online update scheme is developed to update the HMM model and the normal database, which are the two main components of the proposed HMM-based anomaly detection model, using the online test data. First, the online test data is cleaned from anomalies to produce an online training set. Then, the online training set is used to incrementally update the HMM model and the normal database.

Some parts of our work in this area, including the above contributions has been published in [42, 43].

1.3 Thesis Structure

The rest of the thesis is organized as follows: Part I, which consists of Chapter 2, focuses on preventing intrusions for E-commerce systems using hardware-based encryption. Chapter 2 first investigates the problems of the existing software-based security solutions, and then presents the proposed hardware-based RSA encryption model for the enhancement of security and performance for E-commerce systems.

Part II, which includes six chapters, from Chapter 3 to Chapter 8, focuses on detecting intrusions on programs using anomaly detection. Chapter 3 discusses the advantages and disadvantages of the existing approaches for program anomaly detection, and then presents the aims and the outline of our anomaly intrusion detection approach. Chapter 4 gives a brief introduction to hidden Markov models, and their three basic problems and solutions, which will be used to develop the proposed HMM-based anomaly detection model. Chapter 5 describes our basic

model for program anomaly detection, in which HMM modelling is used to characterize normal program behaviour using system calls. It also presents our two-layer detection scheme, which is an extension to the basic model, aimed at reducing false alarms. Chapter 6 describes the proposed efficient HMM incremental training scheme for program anomaly detection. Chapter 7 presents our fuzzy-based detection scheme, aimed at further reducing false alarms and increasing the detection rate. Chapter 8 describes the proposed complete HMM-based program anomaly detection model using system calls.

Chapter 9 summarizes the thesis and outlines some possible improvements for future work, followed by the Appendices and Bibliography.

Part I

Intrusion Prevention: Hardware-based Encryption Model for E-commerce Applications

Chapter 2

E-Commerce Security Enhancement Based on DSP Hardware RSA Encryption

This chapter presents a hardware-based encryption model for the enhancement of E-commerce security. The proposed encryption model focuses on two main security issues: (i) information confidentiality and long-term private key protection, and (ii) performance improvement for E-commerce applications. The model's architecture is based on the existing web-based client/server model, and is enhanced with hardware-based encryption modules. A prototype E-commerce system, based on the proposed encryption model, has been built for the proof of concepts. Performance evaluation shows that the proposed encryption model is suitable for the business-to-customer E-commerce.

2.1 Introduction

As discussed in Chapter 1, the security of E-commerce applications depends on two aspects: communication security and end-system security [32]. Cryptography-based security packages and protocols, such as Secure Socket Layer (SSL) and Transport Layer Security (TLS) have been widely used to ensure communication security for Internet and E-commerce applications. These security protocols make use of TCP to provide a reliable end-to-end secure communication service. Figure 2.1 shows the relative location of SSL/TLS in the TCP/IP protocol stack. TLS

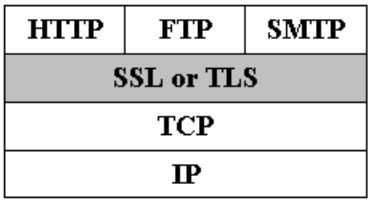

Figure 2.1: Relative location of SSL/TLS in the TCP/IP protocol stack

is approved by the Internet Engineering Task Force (IETF) as a standard protocol for secure communications. As TLS is very similar to SSL, thereafter we only discuss SSL. SSL's general features are as follows:

- SSL uses public key cryptography for communication parties to authenticate each other and to exchange shared secret keys.
- SSL uses secret key cryptography to encrypt and decrypt all information exchanged in communication sessions.
- SSL requires at least one communication party to hold a public key certificate.

SSL was developed by Netscape in 1994 and it quickly became an industrial standard for secure communications over the Internet. It has been implemented in most mordern Internet applications, such as web servers and web browsers. Although SSL is widely used in Internet and E-commerce applications, there are still some issues that need to be addressed:

1. *Protection of long-term private keys*: A communication party, that has a public key certificate, is also holding a private key corresponding to the public key stored in the certificate [32]. A common question is how this private key can be protected during its lifetime? Software, whether it is an operating system, a web browser, or an application package, that gets access to the private key, has to protect that key. In most cases, present technology will not be able to provide strong assurance for the protection of private keys [32].
2. *Degradation of performance*: Like almost all present security solutions, SSL provides the security at the expense of performance, because it uses system resources to establish secure communiation channels and to encrypt and decrypt all traffic between communication

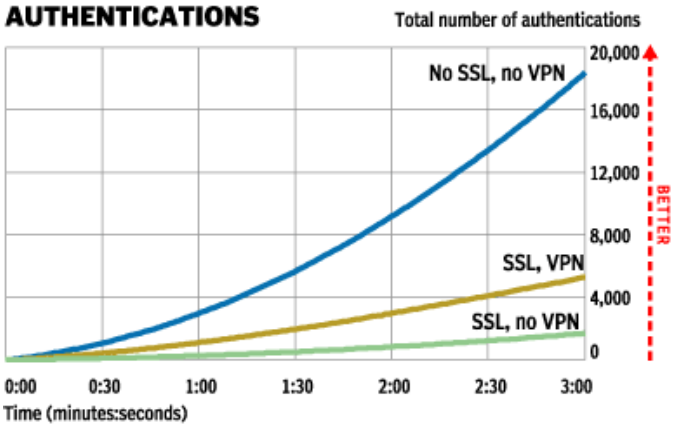

Figure 2.2: SSL Performance for HTTP 'authentication' requests [61]

parties. In order to measure the SSL's effect to the performance, Kaven [61] has conducted experiments on a client/server communication system using three options as follows:

(a) No SSL, no VPN[1]: the client sends requests to the server using a direct connection, without SSL and without VPN.

(b) SSL, VPN: the client sends requests to the server using a VPN connection over SSL.

(c) SSL, no VPN: the client sends requests to the server using a SSL connection.

The results of these experiments presented on Figure 2.2 and 2.3 [61] clearly show that SSL degrades the performance for both HTTP 'authentication' and 'get' requests significantly. Similarly, Secure HTTP (HTTP over SSL) is usually slower than normal HTTP. He's [39] experiments on the performance of Secure HTTP confirm that performance reduction of Secure HTTP compared to HTTP is 33% on average.

In general, SSL/TLS and software-based security packages, that are used to address the problems of E-commerce security, do not provide features that can address the security issues of E-commerce as a whole. In particular, SSL/TLS does not provide mechanisms to protect private keys during their lifetime. It is also noted that performance degradation is an issue of software-based security packages.

Hardware-based cryptography can overcome many security problems suffered by software-based security packages. Hardware-based cryptography has two main advantages over software-

[1]VPN stands for Virtual Private Network

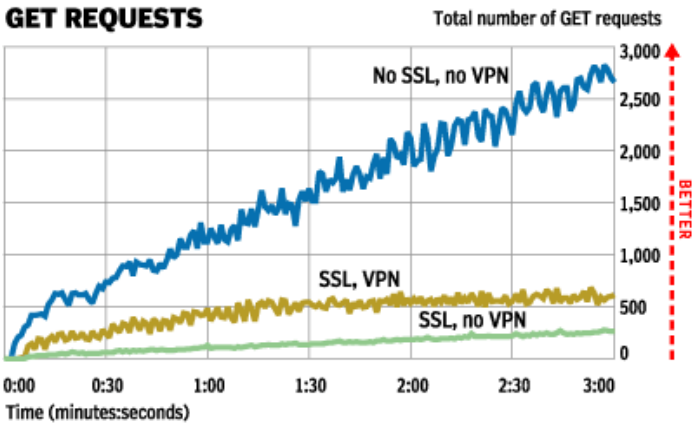

Figure 2.3: SSL Performance for HTTP 'get' requests [61]

based cryptography:

- *Security*: Hardware-based cryptosystems are considered more secure than software-based equivalents, because hardware cryptosystems run in their own physical memory space, which effectively prevents malicious memory access. Unlike software cryptosystems, hardware cryptosystems are not dependent on the security of the underlying operating system [6]. This means that they are not affected by operating system flaws that lead to security threats. Furthermore, sensitive information, such as long-term private key, which is burnt into a hardware chip, is effectively protected from illegal accesses and modifications [6]. Smart cards are the best example of using dedicated hardware to store important information [32].

- *Performance*: Generally, dedicated hardware provides high computing power, compared to that of general-purpose computers. Therefore, hardware can be used to speed up the encryption and decryption process. This also means that longer encryption keys can be used. And, this in turn improves security.

2.1.1 Aims

We propose a hardware-based encryption model for existing E-commerce systems, which focuses on three major aims:

1. To enhance the confidentiality of transaction data of E-commerce systems.

2. To improve the private key security of E-commerce end-systems.

3. To impvove the performance of E-commerce systems.

2.1.2 Outline of the Proposed Approach

In order to ensure the confidentiality of transaction data of an E-commerce system, we use the RSA public key encryption algorithm [108] as the core algorithm to encrypt all data at each end before they are transferred to the other end. The RSA encryption can provide a high level of security and its strength has been proven in practice [92, 113, 122]. As a public key algorithm, RSA has simpler key exchange mechanism, compared to that of secret key encryption [92, 113, 122]. RSA has been used in a wide-range of applications, to provide data confidentiality, non-repudiation of origin and digital signature [32, 92, 113, 122].

The private key security is guaranteed using an integrated hardware module as the key storage medium. Hardware has been proven to be the best medium to store important information, such as long-term private keys [6, 32]. Specifically, we use Digital Signal Processors (DSPs) to develop our hardware-based encryption modules. On one hand, DSPs provide more computing power than small microchip devices, such as smart cards. On the other hand, DSPs are more flexible than the dedicated VLSI (Very Large Scale Integrated) chips. In addition, DSP boards support several types of computer communication interfaces, such as PCI interface, COM ports and USB ports. This makes it more favourable for the development and deployment of DSP applications.

The use of hardware as the platform in the proposed encryption model also serves for the purpose of performance improvement. As discussed in Section 2.1, dedicated hardware can provide high computing power, compared to that of general-purpose processors. This is very important for performance improvement of RSA encryption implementations.

The proposed encryption model is based on the integration of the existing client/server model of E-commerce applications and hardware-based encryption modules. The hardware modules are integrated into both ends of an E-commerce system. They perform data encryption and decryption, as well as store the system's life-time private keys. In Sections 2.3 and 2.4, we discuss the architecture and implementation of the proposed encryption model in details.

2.2 Related Work in Hardware-based Cryptosystems

Hardware cryptography is an on-going research topic [67, 98]. Research and development on hardware-based RSA cryptosystems can be roughly classified into two categories:

1. *Reconfigurable Hardware Cryptosystems*:

   This solution is based on special hardware architectural design to build fast RSA cryptosystems. A new approach in this category is Field Programmable Gate Arrays (FPGAs). An FPGA is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks. In this way, one can create special purpose functional units that may be very efficient for a specific task. As FPGAs can be reconfigured dynamically, it is theoretically possible to optimize them for more complex special tasks at speeds that are higher than what can be achieved with general-purpose processors [96, 135].

2. *Embedded Hardware Cryptosystems*:

   Smart cards and Digital Signal Processors (DSPs) are leading hardware in this category. Generally, a smart card is a plastic card with a special integrated circuit (IC) chip imbedded in the surface of the card. There are two principal types of smart cards, namely microprocessor-based and memory-based. The IC chip in the card provides intelligence sufficient to protect its information from theft or damage. Smart cards usually have limited processing power and storage capacity [32, 45].

   A DSP is a specialized digital microprocessor used to efficiently and rapidly perform calculations on digital signals that were originally analog in form [135]. DSPs provide more computing power and storage capacity than smart cards. For example, a Texas Instruments's TMS320C6416 DSP board has a 32-bit processor running at 1000 MHz, and external RAM memory of 16 megabytes [127], as opposed to a typical smart card that has a 16-bit microprocessor and 8 kilobytes of RAM [45].

On the DSP platform, Hu et al [46] reported a DSP-based RSA cryptosystem. In this work a Texas Instruments' TMS320C6201 DSP PCI card is used as the hardware platform to implement the RSA encrytion algorithm. This DSP-based RSA cryptosystem was reportedly to outperform the RSA implementation in software environment using C++ language at PC level. The use of a DSP PCI card is an advantage in this system, because it can reduce overhead of communications between the host computer and the DSP. A common point between this work and our approach

is that DSP hardware of Texas Instruments' TMS320C6000 DSP family was used as the platform to support the RSA encryption. However, the reported work is a stand-alone DSP-based RSA cryptosystem while our hardware-based encryption modules are integrated into an E-commerce system in order to improve the system's security and performance.

In other DSP-based approach, Er et al [20] presented a design and implementation of an RSA cryptosystem using multiple DSP chips. The reported system makes use of parallel processing in order to speed up the RSA encrytion. The most interesting feature of this system design is that it allows for additional DSP chips to be inserted in allocated slots to improve performance. A master DSP chip is used to control and co-ordinate all the slave DSP chips, while the slave DSPs carry out the actual task processing. With 8 slave DSPs, the system's performance was found 70 times faster than the corresponding software implementation at the PC level. In this system, the DSP cryptosystem is controlled by a host PC application through UART serial channels.

On reconfigurable hardware platform, Buldas et al [10] presented a VLSI implementation of the RSA and IDEA encryption engine. The VLSI module can generate and hold RSA keys in its firmware. It uses the RSA encryption for secure key exchange, and the IDEA algorithm for block encrytion. The reported VLSI module only supports maximum RSA moduli of 768 bits which is too short for an RSA key for practical applications [108, 122].

In [96], an FPGA implementation of the RSA algorithm is reported. The FPGA structures are built using a modified Montgomery multiplier, where the multiplication and modular reduction operations are carried out in parallel rather than interleaved as in the traditional Montgomery multiplier. This helps the reported RSA structures outperform structures which are built based on the traditional Montgomery multiplier. The proposed implemenation reportedly had double throughput rate, compared to that of implementations, based on the traditional Montgomery multiplier. Another advantage of this implementation is it can reduce the size of the chip.

Kim and Lee [65] proposed an integrated crypto-processor which is a special-purpose microprocessor. This crypto-processor is optimized for cryptography algorithms. The proposed system supports secret key algorithms, such as AES, KASUMI, triple-DES, and public key algorithms, including RSA and ECC. The crypto-processor consists of a 32-bit RISC processor block and co-processor blocks. Each co-processor block is a dedicated hardware implementation of a cryptographic algorithm. The co-processor block allows fast execution of encryption, decryption

and key scheduling operations. The RISC processor block can be used to perform several cryptographic algorithms, such as hash, and other application programs, such as user authentication and IC card interface. The proposed crypto-processor can be used for various security applications, such as storage devices, embedded systems, network routers, security gateways using IPSec and SSL protocol. There are many other reports along this line, as in [64, 109, 118, 125, 126].

2.3 The Proposed DSP-Based RSA Encryption Model

We introduce a new DSP hardware-based encryption model using the RSA public key algorithm [108] for E-commerce systems, as shown in Figure 2.4. The model's architecture is based on the web-based client/server model. The server consists of a web server, a server DSP communication module and a DSP hardware RSA decryption engine. The client includes a web browser, a client DSP communication module and a DSP hardware RSA encryption engine. The server and the client communicate with each other using TCP/IP networks, such as the Internet. For simplicity, one way of exchanging information from the client to the server is presented. First, a plaintext message is encrypted by the client's hardware encryption engine using the server's public key to produce a ciphertext message. The ciphertext message is then sent to the server. Next, the ciphertext message is decrypted by the server's hardware decryption engine using the server's private key.

Regarding public key distribution from the server to the client, the most common method is that, the client gets the server's public key certificate that includes the server's identity and public key. Generally, a public key certificate is a certificate which uses a digital signature to bind a public key together with an identity that can be the information about an organization, or a person. In a typical public key infrastructure (PKI) scheme, public key certificates are issued by trusted parties called Certificate Authorities (CAs) [32, 135]. A CA certificate can be used to verify an individual's identity and that a public key belongs to an individual [135]. In our experiments, for simplicity we incorporate the server's public key into a web page and send it to the client.

A detailed description of components of the server and the client in the proposed encryption model is as follows:

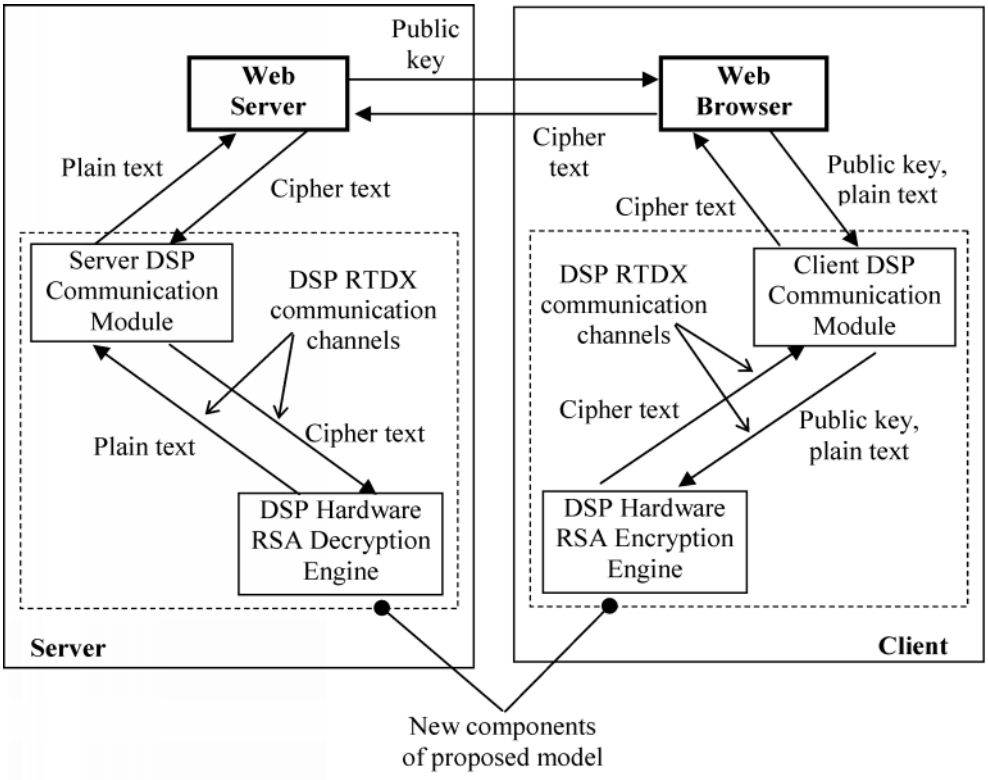

Figure 2.4: Architecture of proposed DSP hardware RSA encryption model. The parts wrapped by dashed lines are new components that are added to an existing client/server model.

1. **The Server**:

   (a) *Web server* is used to host a functional E-commerce website. It serves website's content to client and accepts information submitted from client.

   (b) *The server DSP communication module* runs on the server and works on the web server's requests. It is responsible for all communications between the web server and the DSP hardware RSA decryption engine.

   (c) *The DSP hardware RSA decryption engine* runs on DSP hardware to decrypt cipher messages received from the web server. This decryption engine stores the server's RSA long-term private key.

2. **The Client**:

   (a) *Web browser* is responsible for sending requests, managing downloads and displaying

web pages. It can also submit user data forms to the web server.

(b) *The client DSP communication module* runs embeded inside the client web browser as a plug-in. It is responsible for all communications between the client web browser and the DSP hardware RSA encryption engine.

(c) *The DSP hardware RSA encryption engine* runs on the DSP hardware to encrypt plaintext messages received from the client web browser. The encryption engine is also used to store the client's RSA private key for the decryption of messages from the server.

On the client side, the client DSP communication module is designed to be downloadable[1] and installable automatically from server to client. The procedure of sending information from the client to the server in the proposed encryption model has the following steps:

1. On the user's behalf, the client web browser sends a request to the server for a web page, such as an online credit card payment page, where the user's payment information needs to be encrypted before being sent to the server. The client DSP communication module is embeded into the requested web page. The client browser loads the web page, and installs and activates the client DSP communication module. The client DSP communication module is only downloaded and installed if it is not already present in the client system. The server's public key is also incorporated into the requested web page that is sent to the client web browser.

2. When the user enters data (plain text) into the data entry form in the web page and submits the information, the client DSP communication module captures the plain text. It then establishes an input communication channel to the DSP hardware RSA encryption engine, and then sends the plain text and public key to the encryption engine.

3. Upon receiving the plain text and public key, the DSP hardware RSA encryption engine encrypts the plain text to produce the cipher text. It then sends the cipher text back to the client DSP communication module through an output communication channel.

4. Next, the client DSP communication module forwards the cipher text to the browser, and then the browser submits the cipher text to the server.

[1]The security of downloadable ActiveX components is discussed in Section 2.4.2

5. When the server gets the cipher text, it passes the message to the server DSP communication module through its object interface. In turn, the server communication module opens an input communication channel to the DSP hardware RSA decryption engine, and then sends the cipher text to the decryption engine.

6. Upon receiving the cipher text, the DSP hardware RSA decryption engine decrypts the cipher text to get the plain text, using the server's stored private key. It then sends the plain text back to the server DSP communication module through an output communication channel.

7. In the end, the server DSP communication module forwards the plain text back to the server.

2.3.1 Communications between host computer and DSP

In the proposed encryption model, as shown in Figure 2.4, all communications between the client web browser and the DSP hardware RSA encryption engine, as well as communications between the web server and the DSP hardware decryption engine, are supported by the Real Time Data Exchange (RTDX) channels. RTDX comes as a software library of the DSP application development tools. Figure 2.5 shows the RTDX-based communication mechanism between the client browser and the DSP encryption engine. Figure 2.6 describes the RTDX-based communication mechanism between the web server and the DSP decryption engine.

RTDX is a new technology of Texas Instruments (TI), that enables two ways of communications between the DSP applications and computer applications. The RTDX's application programming interface includes the *RTDX built-in library* and the *RTDX exported library*. The RTDX built-in library is used in DSP applications to support RTDX communications on DSP target side while RTDX exported library is included in computer applications to support RTDX communications on computer side. The RTDX communication channels can support real-time and full duplex data exchanges between the host computer and the target DSP [62].

RTDX communication channels can be operated in two modes: non-continuous and continuous mode. In non-continuous mode, the transferred data is recorded into a log file that has been specified in RTDX configuration. As data is saved in a log file, non-continuous mode is suitable for off-line processing. In continuous mode, data is logged into a circular memory buffer in the

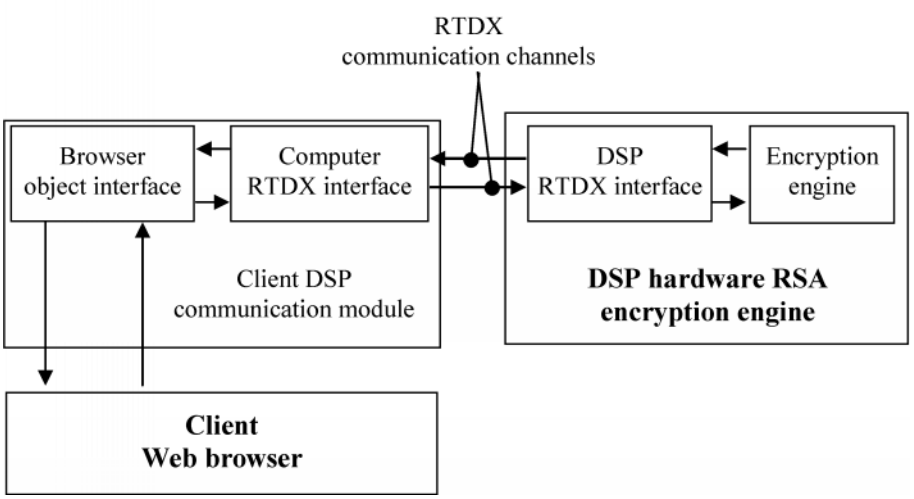

Figure 2.5: Communications between the client web browser and the DSP hardware RSA encryption engine using RTDX channels

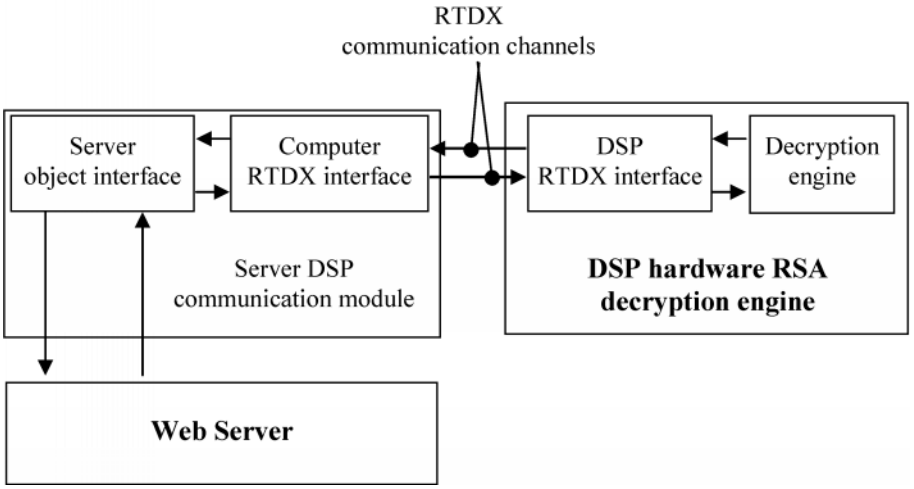

Figure 2.6: Communications between the web server and the DSP hardware RSA decryption engine using RTDX channels

RTDX Host Library, which helps continuously obtain and display live data from a target DSP application [62]. In our experiments, we use RTDX in continuous mode to get the real time data, as well as to avoid storing plain text data to computer's hard disk.

2.3.1.1 RTDX communications on the client side

As shown in Figure 2.5, the client web browser exchanges information with the DSP hardware RSA encryption engine, using RTDX channels supported by the client DSP communication module. The data flow from the client web browser to the DSP encryption engine is as follows:

- First, the browser sends user data that needs to be encrypted to the client DSP communication module through its object interface.
- Next, the browser object interface forwards the data to the computer RTDX interface of the client DSP communication module.
- Then, the computer RTDX interface opens an input RTDX communication channel to send the data to the DSP RTDX interface of the DSP encryption engine.
- Upon receiving the data, the DSP RTDX interface transfers it to the DSP encryption engine for encryption.

On the opposite direction, the data, which is an encrypted message, can be sent from the DSP encryption engine to the client web browser using the following procedure:

- First, the DSP encryption engine forwards the data to its DSP RTDX interface.
- Next, the DSP RTDX interface opens an output RTDX communication channel to send data to the computer RTDX interface of the client DSP communication module.
- Upon receiving the data, the computer RTDX interface transfers it to the browser object interface.
- In the end, the browser object interface forwards the data to the client web browser. The client web browser in turn submits the data to the server.

2.3.1.2 RTDX communications on the server side

As shown in Figure 2.6, the web server exchanges information with the DSP hardware RSA decryption engine, using RTDX channels supported by the server DSP communication module. The RTDX-based communication mechanism on the server side is very similar to that of the client side. The only difference is that the web server communicates with the server DSP

communication module using the *Server Object Interface*, instead of the Browser Object Interface of the client. In addition, the web server sends the encrypted message (submitted from the client web browser) to the DSP decryption engine and then gets back the decrypted message.

2.4 Implementation of the Proposed Encryption Model

In this section, we discuss the implementation details of crucial components of the proposed hardware-based encryption model, including the *DSP hardware RSA encryption engine*, the *client DSP communication module*, the *DSP hardware RSA decryption engine* and the *server DSP communication module*. We chose Microsoft Windows as our software platform and DSP hardware and development tools from Texas Instruments (TI) to develop these components. Specifically, Microsoft Visual C++ was used to develop the client DSP communication module and the server DSP communication module. The DSP hardware RSA encryption engine and the DSP hardware RSA decryption engine were built using the TI's TMS320C6416 DSP and Code Composer Studio. TI's Code Composer Studio is a C programming development environment for DSP applications.

2.4.1 DSP hardware RSA encryption engine

The DSP hardware RSA encryption engine consists of two parts: the *RSA encryption engine* and the *DSP RTDX interface*. The RSA encryption engine is the implementation of the RSA algorithm on DSP hardware, while the DSP RTDX interface is the communication interface between the RSA encryption engine and the client DSP communication module.

2.4.1.1 Implementation of RSA encryption engine

As discussed in Section 2.1, the RSA public key encryption algorithm [108] is selected as the core encryption algorithm for the proposed encryption model. The RSA algorithm was invented by R. Rivest, A. Shamir and L. Adleman in 1977. Table 2.1 summaries the RSA encryption scheme [122]. The RSA encryption protocol consists of two major processes: (1) - *Key Generation* to generate RSA key pair, which are public and private keys, and (2) - *Encryption* and *Decryption* of messages, using RSA key pair. Only the private key of the RSA key pair needs to be kept secret, while the public key can be made publicly available and transmitted through

Table 2.1: RSA public key encryption algorithm

Key Generation	
Select p, q	p and q both primes, $p \neq q$
Calculate $n = p * q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$
The modulus	n
Encryption component	e
Decryption component	d
Encryption	
Plaintext:	m ; $m < n$
Ciphertext:	$c = m^e \bmod n$
Decryption	
Ciphertext:	c
Plaintext:	$m = c^d \bmod n$

open communication environment, such as the Internet.

Important notes on the generation of RSA keys

In the RSA key generation process, the prime numbers p and q should be selected so that the factoring $n = p * q$ is computationally infeasible. The major restriction on p and q, in order to avoid the elliptic curve factoring, is that p and q should be roughly the same bit length and sufficiently large [92]. For example, if a 1024-bit modulus n is to be used, then each of p and q should be about 512 bits in length. Another restriction on the primes p and q is that the difference $p - q$ should not be too small. If $p - q$ is small, then $p \approx q$ and hence $p \approx \sqrt{n}$. Thus, n could be factored efficiently, simply by trial division by all odd integers close to $\sqrt{n}$. If p and q are chosen at random, then $p - q$ will be appropriately large with overwhelming probability [92]. In addition to the mentioned conditions, it is recommended that p and q be strong primes.

A prime p is said to be a strong prime if it satisfies the following three conditions [92]:

(i) $p-1$ has a large prime factor, denoted r;

(ii) $p+1$ has a large prime factor; and

(iii) $r-1$ has a large prime factor.

Use of small encryption component

Small encryption component e, such as $e = 3$, may be used to improve the efficiency of the RSA encryption [92]. A common encryption component e may be shared among a group of entities, and each entity in the group must have its own distinct modulus n. If an entity wants to send a message to other entities in the group, it encrypts the original message using common encryption component and a modulus from each of other entities to produce one ciphertext message for each entity. However, this scheme is vulnerable to eavesdropping attacks, in which an eavesdropper can recover the original message by analyzing corresponding ciphertext messages sent to other entities. Padding a random bit-string to the original message on each encryption can be used to prevent this type of attach [92]. In our proposed model, we chose encryption component size as same size of the modulus in order to prevent attacks that exploit the weaknesses of small encryption components.

Use of small decryption component

Similar to the use of small encryption component e, small decryption component d may be used to improve the efficiency of the RSA decryption [92]. However, if the decryption component d is small and $gcd(p-1, q-1)$ is small, d can be efficiently computed from the modulus and the encryption component (n, e). Hence, in order to avoid this type of attacks, we chose the decryption component size roughly the same size of the modulus n [92].

Size and format of encryption blocks

In the RSA algorithm, the equation $c = m^e \bmod n$ is used to encrypt message m using the public key (n, e). In order to make the RSA encryption work properly, the message m must be represented as an integer, and $integer\_value\_of(m)$ must be smaller than $integer\_value\_of(n)$ [92]. Therefore, for a large message M, it must first be divided and formatted into r encryption blocks $(m_1, m_2, ..., m_r)$ such that:

$$integer\_value\_of(m_i) < integer\_value\_of(n), \quad i = 1, r$$

The size of encryption blocks could be a fixed or changeable value, depending on the system's design. Theoretically, the encryption block size could be any non-zero value, provided that $integer\_value\_of(m_i)$ is smaller than $integer\_value\_of(n)$. However, the block size should not be too small because a block cipher with small block size may be vulnerable to the attacks, based on statistical analysis [92]. One such attack involves simple frequency analysis of cipher text blocks. Therefore, to get the maximum security, the block size should be as large as possible [92] and roughly the same size as the size of the modulus n.

The format of RSA encryption blocks has been formally defined in "PKCS #1: RSA Encryption, version 1.5 and 2.0" [55, 56]. An encryption block EB, that consists of a block type BT, a padding string PS and data D, is a formatted octet string as follows [55, 56]:

$$\text{EB} = 00 \;||\; \text{BT} \;||\; \text{PS} \;||\; 00 \;||\; \text{D}$$

where || is the concatenation sign. If the modulus n has k octets, each encryption block also has the same k octets. The leading 00 octet makes sure that the converted integer value of the encryption block is smaller than the value of the modulus n. The block type BT is one octet which is 00 or 01 for private key operation, and 02 for public key operation. The padding string PS consists of $k - 3 - ||\text{D}||$ octets, where $||\text{D}||$ is the length of the data D in number of octets. For public key operation, the padding string should be pseudorandomly generated and nonzero. The padding string should be at least eight octets long, which is a security constraint for public-key operations, that prevents an attacker from recoving data by trying all possible encryption blocks [55, 56]. Therefore, the minimum size of the encryption block EB is 12 octets to satisfy the defintion of the encryption block format (minimum 1 octet for data, and 11 octets for the leading octet, the BT, the PS and 1 octet between PS and the data).

For the purpose of performance testing, we chose to use a simple method to format a large message into encryption blocks in our implementation. Our encryption blocks consist of only the data. The size of encryption blocks in the number of octets is calculated as follows:

$$\text{size of encryption blocks (octets)} = k - 2$$

where k is the size of the modulus n in number of octets.

As we use the radix of 2^{16} (as explaining in next heading) to represent the big integers for RSA operations, the size of a big integer's digit is 16 bits or 2 octets. Therefore, the substraction of 2 octets (which is equivalent to one meaningful big integer digit) from the modulus size is to make sure that $integer\_value\_of(m_i)$ is always smaller than $integer\_value\_of(n)$.

Big integer representation for RSA implementation

Most modern 32-bit computing architectures, including the TI's TMS320C6416 DSP and personal computers, support standard types of integers, which are represented using fixed number of bits. The maximum size of integers supported by these computing systems is 32 bits. This means the maximum value representable in these system is $2^{32} - 1 = 4,294,967,295$. By the requirement of the RSA algorithm, RSA keys would be in the level of 10^{100}, or even larger, which cannot be held by computing systems in the standard types of integers. One solution is to use arrays of fixed integers to represent big integers, or multiple precision integers. This representation allows the accommodation of any integer values, and is therefore limited only by the size of the physical memory.

Takagi [125] reported a fast radix-4 modular multiplication hardware algorithm. Numbers are represented in a redundant representation, and additions and subtractions are performed without carry propagation. The algorithm was found efficient especially in applications where modular multiplications are carried out iteratively. However, this specialized hardware algorithm is not suitable for DSPs which have general-purpose processors.

In our work, we select 65536, or 2^{16} as the base for multiple precision integer representation. The major advantage of representing integer in base 65536 over decimal representation is that it requires much less memory. This is especially important in the embedded systems, such as DSPs that usually have limited memory space.

Another advantage of base 65536 representation over smaller bases, such as 256 or 2^8 is that a big integer on base 65536 has fewer number of digits than the same integer value on smaller bases. This means that arithmetic operations on base-65536 integers require less number of memory references, as well as less number of standard-integer-atomic operations. These in turn help to improve the computational performance. A larger base than 65536 would make the implementation of the arithmetic operations more complicated. This is because on most current computing systems, where the largest integer supported is 32-bit long, and the results of arithmetic operations on such integers would overflow the 32-bit registers.

Table 2.2: Definition of BigInt Data Struture

```
typedef struct {
    unsigned short *digits;   // pointer to array of digits
                              // the least significant digit at index 0
    unsigned int size;        // number of digits of the big integer
    short sign;               // sign of the big integer,
                              // sign = -1 for negative number, and 1 otherwise
} BigInt ;
```

In addition, on base 65536 (2^{16}), fast shift operations can be used to replace the high-cost multiplication and division operations. This in turn gives better computational performance.

We define a simple data structure *BigInt* to store big integers for the implementation of the RSA encryption algorithm, as shown in Table 2.2. Each digit in the BigInt type can store a 16-bit integer number, ranging from 0 to 65535. The maximum size of the big integer is $2^{32} = 4294967296$ digits. Hence, the biggest integer value can be represented is $65536^{4294967296}$. This great integer value is sufficient to represent very large RSA keys, such as 2048-bit key, which is about 616 base-10 digits.

BigInteger package implementation

BigInt package is the heart of the RSA algorithm implementation. The package consists of necessary arithmetic and logic functions on big integers to support RSA encryption and decryption. The implementation of arithmetic functions is based on efficient arithmetic algorithms, given in [92]. Auxiliary functions, such as input and output, are also implemented to convert text messages to BigInt and vice verse. Table B.1 (page 180) shows major functions of the BigInt package.

2.4.1.2 Implementation of the DSP RTDX interface

The RTDX built-in library is included into the implementation of the DSP RTDX interface, to support real-time communication between the RSA encryption engine and the client DSP communication module. The DSP RTDX interface provides functions to initialize the RTDX communication sessions, open communication channels, read and write data to channels and

close channels. In order to make the RTDX communication channels work properly, they need to be configured with correct mode and suitable buffer size. We selected the RTDX continuous mode to get the real time data transfer. Table B.2 (page 182) lists the core RTDX built-in functions that are used in the DSP RTDX interface.

2.4.2 Client DSP communication module

Since the client DSP communication module plays the role of a communication bridge, it must be able to communicate with both the client web browser and the DSP hardware RSA encryption engine. The module exchanges information with the DSP encryption engine using RTDX channels, and it communicates with the client web browser via the browser's object interface. The browser object interface is the Object Linking and Embedding (OLE) interface on Microsoft platform. The implementation of the communication module consists of two communication interfaces: the *Browser Object Interface* and the *Computer RTDX Interface* (as shown in Figure 2.5, page 26). The browser object interface's functions are callable from the browser, using browser's scripting languages, such as JavaScript. The computer RTDX interface is implemented, based on RTDX exported library that provides functions to establish communication channels to the DSP application. Table B.3 (page 183) lists the core RTDX exported functions that are used to build the client DSP communication module. Table B.4 (page 184) lists the major functions in the implementation of the client DSP communication module.

The client DSP communication moldule is designed and implemented as a web browser plug-in, using Microsoft's ActiveX technology. There is a security concern on the downloadable ActiveX control code [27]. However, a digital signature scheme called "Authenticode", which is developed by Microsoft, can be used to sign the code. This scheme allows certifying the authenticity and integrity of programs [27].

In order to use the client DSP communication module, it is embedded into web pages, using the HTML `<OBJECT>` tag. Table 2.3 shows a segment of HMTL code that is used to incorporate the client DSP communication module into a web page. The module was designed to be downloadable and installable automatically on the client machine if it is not already present in the client system. When the communication module is loaded into the client browser, its interface functions can be invoked using browser's scripting languages, such as JavaScript. Table 2.4 shows a segment of JavaScript code that is used to call an interface function of the

Table 2.3: Sample HTML code to embed the DSP communication module into a web page

```
<OBJECT NAME="dsp_comm" ID="dsp_comm" WIDTH=0 HEIGHT=0
 CLASSID="CLSID:08E5AB55-BD25-437F-A03F-7FB44B6FEFE4"
 CODEBASE="/dsp/ClientComm.ocx">
</OBJECT>
```

Table 2.4: Sample JavaScript code to invoke the DSP communication module from browser

```
// get the plain text
var plainText = new String(document.myForm.plainText.value);
if (plainText.length>0){
    // assign the board name and DSP CPU name
    var boardName = new String("C6416 Revision 1.1 DSK");
    var cpuName = new String("CPU_1");
    // get the RSA public key pair
    var rsa_keys = new String(document.myForm.eKey.value+"."+
        document.myForm.nKey.value);
    // invoke the DSP encryption engine to encrypt the message
    // through the DSP client communication module
    // and assign the result to the cipher text control
    // "dsp_comm" is the name of the embedded client DSP communication module
    document.myForm.cipherText.value =
       document.dsp_comm.DSPEncrypt(boardName, cpuName, rsa_keys, plainText);
}
```

communication module from the client web browser.

Improvement of RTDX communication performance

The RTDX library provides basic input and output functions which can be used to read data from and write data to RTDX communication channels. However, in order to maximize the performance of RTDX channels, the number of read and/or write operations to a channel

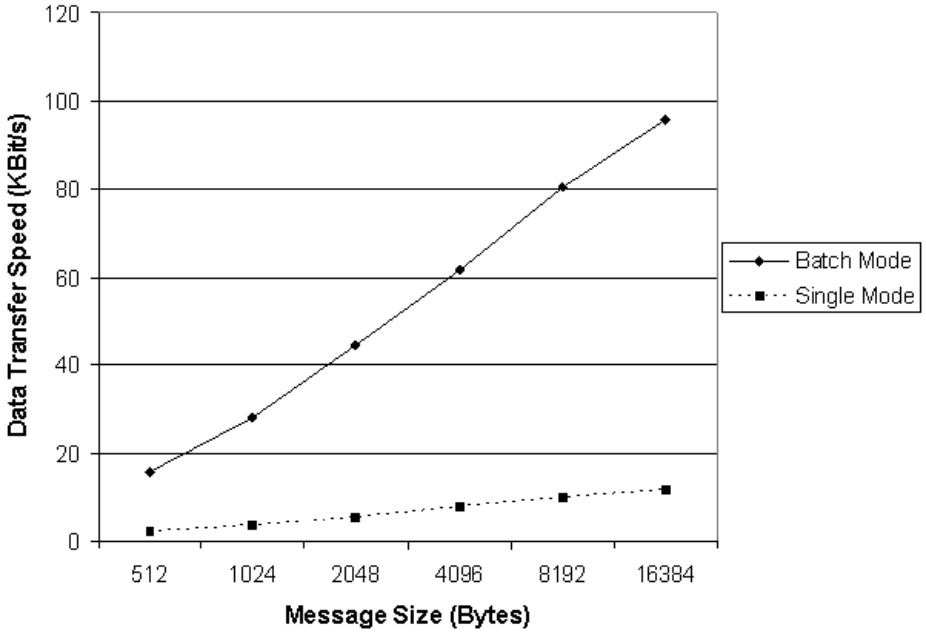

Figure 2.7: RTDX data transfer speed in single and batch modes

in a communication session should be kept to a minimum [62]. In a simple term, if the amount of data to transfer is 100 bytes, the data transfer is more efficient if the data is wrapped in a compound, and sent at one time than sending 100 times with one byte in each message. We encapsulated the data into a Microsoft Visual C++ `SAFEARRAY` of bytes, and sent them to the target DSP, using the `Write()` function (as shown in Table B.3 (page 183)). To get the data from the DSP target, we read the bulk of data into a `SAFEARRAY` of bytes, using `ReadSAI1()` function (as shown in Table B.3 (page 183)), and then extract the bulk data to appropriate data types. Figure 2.7 shows the RTDX data transfer speed in two cases: in the first case, we use single read and write (Single Mode), and the next case, we use the `SAFEARRAY` of bytes, as above mentioned (Batch Mode). The performance data shows that the data transfer speed of the latter is about 8 times faster than the former, on average. For the DSP target application, there is no need to use a compound data type, because the RTDX built-in functions, such as `RTDX_read()` and `RTDX_write()`, support both single and bulk data transfer (as given in Table B.2 (page 182)).

Table 2.5: Sample VBScript code to invoke the decryption function

```
' declare variables
Dim boardName, cpuName
Dim cipherText, plainText
Dim rsaObj
' get the cipher text from client
cipherText = Request.Form("cipherText")
' Specify the DSP board and CPU used
boardName = "C6416 Revision 1.1 DSK"
cpuName = "CPU_1"
' create an instance of server DSP communication object
Set rsaObj = Server.CreateObject("ServerComm.DSPComm")
' pass the cipher text to DSP decryption engine through the server
' DSP communication module for decryption and get back plaintext
plainText = rsaObj.DSPDecrypt(boardName, cpuName, cipherText)
' release the server object after use
Set rsaObj = nothing
```

2.4.3 DSP hardware RSA decryption engine

The implementation of the DSP hardware RSA decryption engine uses the same core arithmetic and logic functions on big integers, as those of the DSP hardware RSA encryption engine, as discussed in Section 2.4.1. The decryption engine has two components which are the DSP RTDX interface and the RSA decryption engine. The DSP RTDX interface is responsible for communications with the server DSP communication module, and RSA decryption engine is the implementation of the RSA decryption. In order to decrypt a cipher message, the decryption engine uses the server's private key which is also stored in the DSP.

2.4.4 Server DSP communication module

The server DSP communication module plays the same role as the client DSP communication module on the client side. Therefore, the core implementation of this module is very similar to the

implementation of client DSP communication module. It has two communication components, including the server object interface and the computer RTDX interface. The server object interface is used to communicate with the web server, and the computer RTDX interface is used to exchange information with the DSP decryption engine. The server DSP communication module is developed using Microsoft Visual C++, and linked in the form of a Dynamic Link Library (DLL). Its server object interface provides interface functions that are callable from server-side scripts, such as Microsoft VBScript and JScript. Table B.5 (page 184) lists the major functions in the implementation of the server DSP communication module. Table 2.5 shows a VBScript code segment that is used to invoke an interface function of server DSP communication module.

2.5 Experimental Design and Results

In the first part of this section, we describe the hardware and software, which are used in the development process and in the experiments of the proposed encryption model. Then, the design of experiments for the evaluation of the proposed model's security and performance is discussed. In the last part, we present the experimental results.

2.5.1 Hardware and software

We chose Microsoft Windows as the software platform, which includes the operating system and applications, running on server and client machines for our experiments. The DSP hardware and software development tools are from Texas Instruments Inc.

2.5.1.1 Hardware

- *The client machine* was a PC with an Intel Pentium 4 1.8 GHz CPU and 512MB RAM, running Microsoft Windows XP Professional.

- *DSP devices* were the Texas Instruments' TMS320C6416 1000 MHz DSP Starter Kits (DSK) [127]. The block diagram, board layout and major features of the TMS320C6416 DSK are given in Appendix A. TMS320C6416 is a fixed-point DSP chip, running at the default clock speed of 1000 MHz. The DSP can be configured to run at other clock speeds of 500, 600, 720, 850 and 1200 MHz. The DSP chip has 1024 KBytes fast internal

RAM memory for code and data. In addition, the DSP board has 16MB SDRAM main memory. The board also has a USB 1.0 port for communications with the host computer. The TMS320C6416 DSK is built on the TI's industry-leading line of low cost and high performance DSP development boards.

- *The server* used in experiments was a PC with an Intel Pentium 4 2.4 GHz CPU and 512MB RAM, running Microsoft Windows 2000 Server Family. A web server and a database server are installed on the server machine to host a functional E-commerce website. The website is used as the environment to test the proposed encryption model.

2.5.1.2 Software

- *Development Tools for TMS320C6416 DSP*: The Texas Instruments Code Composer Studio 3.0 and the attached RTDX library were used to develop the RSA encryption and decryption engines on DSP hardware.
- *Development Tools for client and server applications*: Microsoft Visual C++ 6.0 was used to build the server DSP communication module and the client DSP communication module.
- *Web browser*: The web browser used in our experiments was Microsoft Internet Explorer, because it supported the ActiveX/OLE technology that was used to develop the client DSP communication module.
- *Web and database servers*: The web server software used was Microsoft Internet Information Services 5.0. In order to test our encryption model, we built a simple but fully functional E-commerce website on the web server, with the Microsoft SQL Server 7.0 as database backend. The E-commerce website provides customers with online registration for various types of commercial events. The customers can register their details with the system, then select the interested events, and make the online payment. The design and implementation of the website and its database are not discussed in this thesis due to limited space.

2.5.2 Experimental design

In order to begin experiments on the proposed DSP hardware RSA encryption model, the first step is to generate RSA key pairs. We developed a simple module to generate RSA key pairs

for our experiments. The RSA keys, we choose to generate, are 128, 256, 512, 1024, 1536 and 2048 bits in length. For each RSA key pair, the public and the private components are roughly the same length, and their length is equal to the modulus's length.

In the next step, we used the generated RSA keys to conduct three types of experiments as follows:

1. Performance tests on the proposed encryption model: the aim of these experiments was to measure the model's performance, including DSP encryption/decryption speed, overhead of RTDX communications. We also measure the impact of RSA key length, DSP clock speed and message size, to the encryption speed. Since the RSA public (encryption) and private (decryption) keys had the same length, the complexities of the encryption and decryption were equal. Therefore, we only measure the DSP encryption performance to represent the model's performance. In order to get the performance information, first, we measure the amount of time taken for each task in the encryption process. We then compute the speed performance, base on the amount of time taken and the size of input message. The amounts of time taken are collected as follows:

   - Encryption time: The amount of time taken, from the time the web browser sends plain text message to the DSP encryption engine, to the time it gets back the corresponding cipher text message from the DSP.
   - Message transfer time: The amount of time taken for round trip of a message sent from the client browser to the DSP encryption engine, and then the DSP sends the message back to the client browser, without doing the encryption.

   In this performance testing, we also tested the proposed model with different DSP clock speeds, in order to measure the effect of the DSP clock speed to the model's performance. The TMS320C6416 DSP clock speeds, that are selected for experiment, are 500, 720, 1000 and 1200 MHz.

2. Testing the proposed encryption model with the online credit card payment: In this experiment, we measure the amount of time taken for the encryption of information of typical online credit card payments.

3. Performance testing of the RSA implementation in software: We measure the speed per-

formance of software-based RSA implementation, running on Pentium 4 PC machines with different clock speed CPUs, in order to compare with the performance of the DSP hardware-based RSA implementation.

The sizes of input messages used in experiments are 512, 1024, 2048, and 4096 bytes.

2.5.3 Experimental results

2.5.3.1 Model's performance

For each RSA key pair of 128, 256, 512, 1024, 1536 and 2048-bit, we conducted four tests, with different messages of 512, 1024, 2048 and 4096 bytes in size as mentioned in Section 2.5.2. The performance results are shown in Tables 2.6, 2.7, 2.8 and 2.9 for the TMS320C6416 DSP, running at 500, 720, 1000 and 1200 MHz, respectively. Figure 2.8 presents the relation between RSA key length and the encryption speed for a 2048-byte message, on the DSP running at 500, 720, 1000 and 1200 MHz. Figure 2.9 shows the relation between the 1000 MHz-DSP encryption speed and the message sizes, at encryption keys of 512, 1024 and 2048 bits. Figures 2.10 and 2.11 describe the effect of the DSP clock speed to the encryption speed for short encryption keys and long encryption keys, respectively. Figure 2.13 shows the RTDX data transfer speed between computer and the DSP, using USB port.

As shown in Figure 2.8, the encryption speed decreases dramatically, when the length of keys increases, for all DSP clock speeds. However, the decrement rates are vary for each DSP clock speed. Another important factor, that affects the encryption speed, is the DSP clock speed. It can be seen from Figures 2.10 and 2.11 that the encryption speed is proportional to the DSP clock speed. It is also noted on Figure 2.9 that the message size has small impact on the encryption speed. The encryption speed almost becomes stable when the message is large enough (2048 to 4096 bytes).

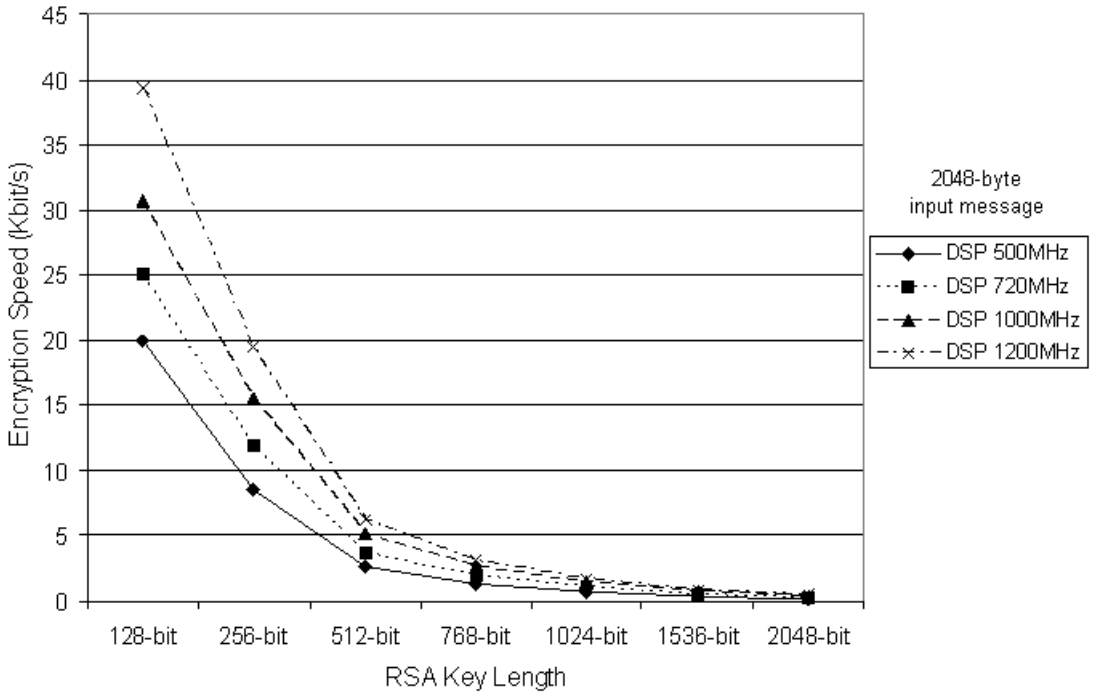

Figure 2.8: Dependence of the encryption speed on the RSA key length

Table 2.6: Encryption speed of the DSP running at 500 MHz

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	18.78316	8.44545	2.48505	1.20945	0.61789	0.31697	0.13476
1024	20.20637	8.39237	2.62102	1.31286	0.68694	0.31795	0.16172
2048	19.95210	8.58101	2.61442	1.31674	0.72779	0.34685	0.17969
4096	19.13739	8.62978	2.66670	1.31620	0.74973	0.34671	0.19010

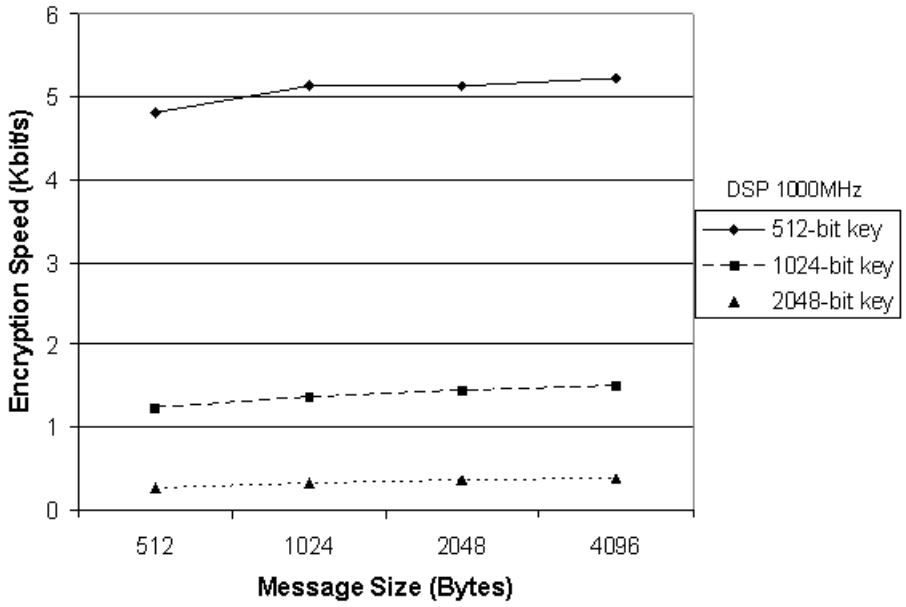

Figure 2.9: Dependence of the encryption speed on the size of input messages

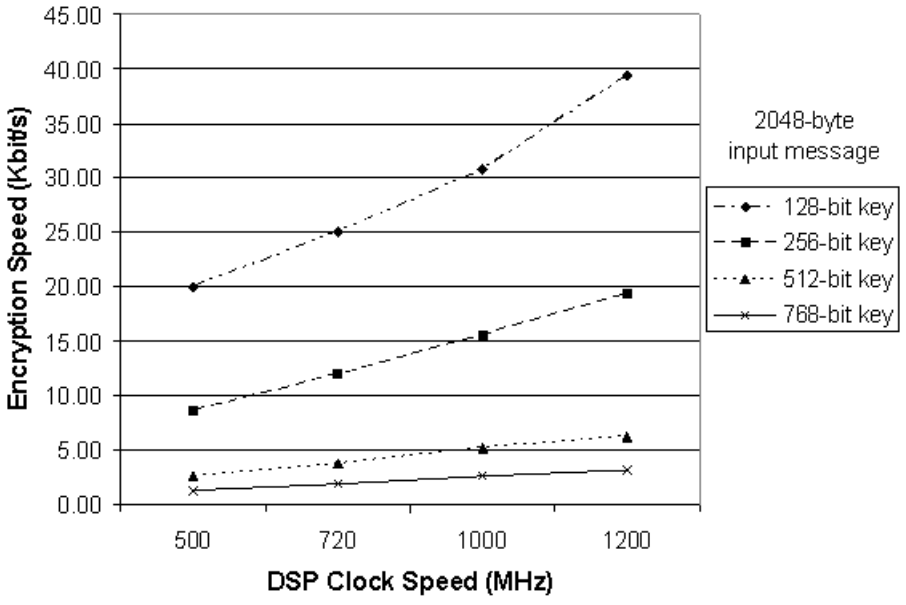

Figure 2.10: Dependence of the encryption speed on the DSP clock speed for short keys

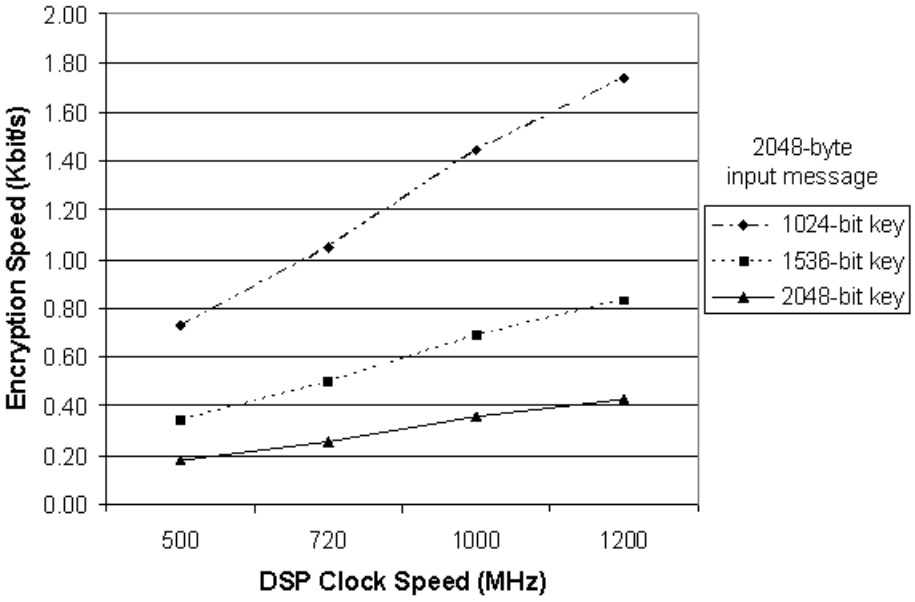

Figure 2.11: Dependence of the encryption speed on the DSP clock speed for long keys

Table 2.7: Encryption speed of the DSP running at 720 MHz

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	30.72000	12.59016	3.60563	1.74150	0.88992	0.45851	0.19409
1024	28.98113	12.59016	3.80198	1.89630	0.98462	0.45755	0.23305
2048	24.97561	11.95331	3.74634	1.88814	1.04561	0.49886	0.25856
4096	25.49378	11.88395	3.78092	1.88351	1.07620	0.49886	0.27361

Table 2.8: Encryption speed of the DSP running at 1000 MHz

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	34.90909	15.67347	4.80000	2.37771	1.22684	0.63002	0.26881
1024	30.72000	14.91262	5.12000	2.59459	1.35929	0.63288	0.32222
2048	30.72000	15.51515	5.12855	2.60560	1.44497	0.69065	0.35867
4096	33.57377	16.21108	5.21562	2.60670	1.49126	0.69166	0.37975

Table 2.9: Encryption speed of the DSP running at 1200 MHz

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	51.52203	20.77430	5.95493	2.92085	1.49144	0.76269	0.32229
1024	39.41620	19.95615	6.21941	3.13529	1.64114	0.76157	0.38750
2048	39.38462	19.44304	6.21862	3.11858	1.73651	0.83049	0.43013
4096	39.40040	19.38361	6.26300	3.12349	1.78762	0.83017	0.45589

Table 2.10: Increment rate of the encryption speed on the DSP clock speed. The encryption speed of 500 MHz DSP is used as the unit speed.

DSP Clock	**Increment rate of encryption speed (times)**						
Speed (MHz)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
500	1.00	1.00	1.00	1.00	1.00	1.00	1.00
720	1.25	1.39	1.43	1.43	1.44	1.44	1.44
1000	1.54	1.81	1.96	1.98	1.99	1.99	2.00
1200	1.97	2.27	2.38	2.37	2.39	2.39	2.39

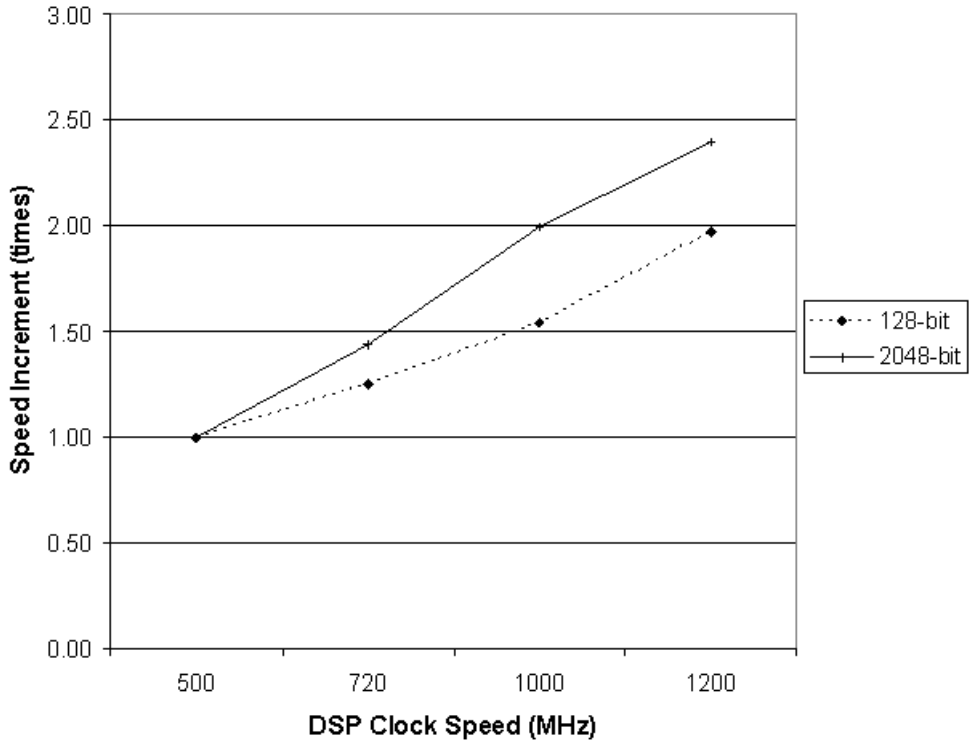

Figure 2.12: Increment rate of the encryption speed on the DSP clock speed for 128 and 2048-bit keys. The encryption speed of 500 MHz DSP is used as the unit speed.

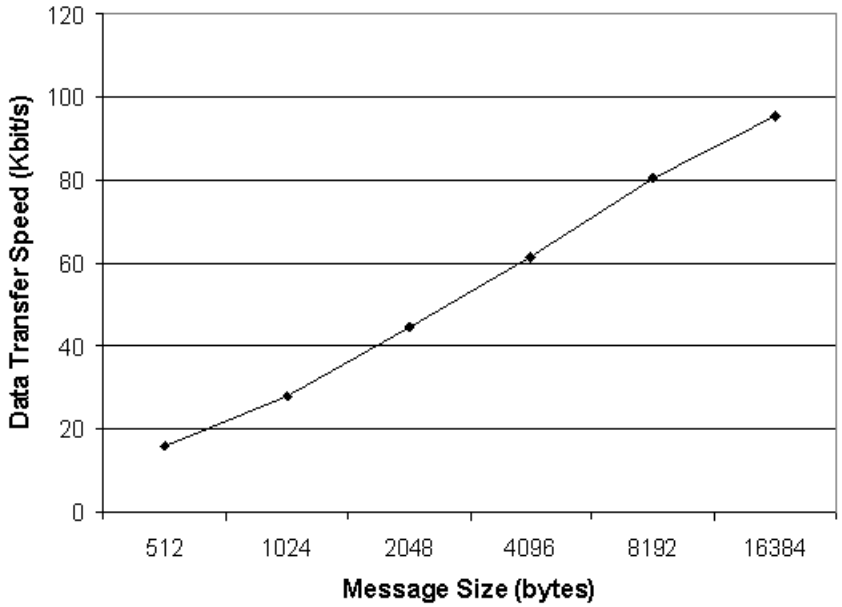

Figure 2.13: RTDX data transfer speed on message size

Table 2.11: Encryption time of a typical set of the payment information on DSP at 1000 MHz

	Encryption time on RSA key length (s)						
	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
Time (s)	0.01213	0.03009	0.10496	0.21398	0.41906	0.81947	1.93924

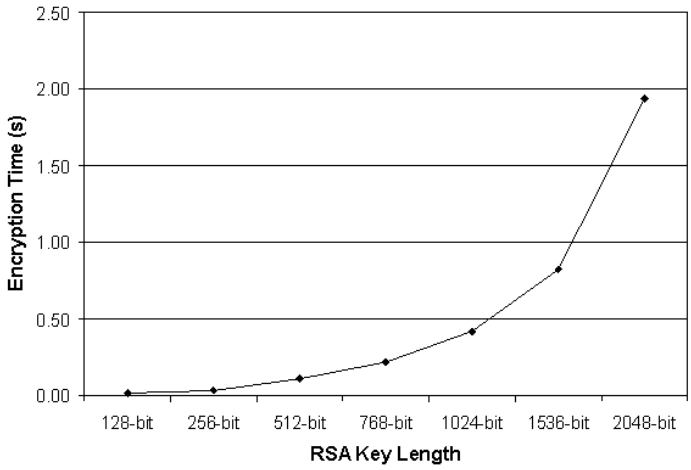

Figure 2.14: The relationship between the RSA key size and the encryption time of a typical set of the credit card payment information on DSP at 1000 MHz

2.5.3.2 Experiments with online credit card payment

We simulate the online credit card payments to measure the model's performance for payment transactions. In this experiment, the DSP hardware RSA encryption engine is used to encrypt the customers' card payment information (which consists of card number, expiry date, card security code and name on card, in total of 80 bytes maximum), using the server's public key. The encrypted payment message is then submitted to the server. Upon receiving the encrypted payment message, the server requests its DSP decyption engine to decrypt the encrypted payment message, using its own private key. The server then gets the actual payment information for verification. In the end, the server sends its response to client that the payment transaction is accepted, or rejected. Table 2.11 and Figure 2.14 show the amounts of time in seconds needed to encrypt a typical set of payment information on different key lengths. According to the results of this experiment, as shown in Figure 2.14, the encryption time of the payment information is less than 2 seconds at 2048-bit key, and less than 0.5 seconds at 1024-bit key.

Table 2.12: Encryption speed of the software implementation on the Intel Pentium 4 1.8 GHz machine

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	48.00000	17.06667	4.46512	2.08130	1.04348	0.52531	0.22152
1024	56.88889	17.65517	4.77019	2.26215	1.15402	0.52603	0.26551
2048	56.88889	17.96491	4.77019	2.27051	1.22439	0.57507	0.29490
4096	56.88889	18.12389	4.84161	2.26049	1.26108	0.57544	0.31200

Table 2.13: Encryption speed of the software implementation on the Intel Pentium 4 2.4 GHz machine

Message	**Speed on RSA key length (Kbit/s)**						
(Bytes)	128-bit	256-bit	512-bit	768-bit	1024-bit	1536-bit	2048-bit
512	69.81818	23.27273	6.00000	2.81319	1.42222	0.71243	0.29733
1024	76.80000	23.27273	6.40000	3.06587	1.56895	0.71144	0.35671
2048	78.76923	24.38095	6.37344	3.06893	1.66775	0.77792	0.39496
4096	78.76923	24.28458	6.50159	3.06893	1.71908	0.77812	0.41939

2.5.3.3 Experiments of RSA software implementation on different PC machines

In addition to the experiments on DSP hardware RSA encryption model, we have also conducted experiments to test the performance of software implementation of the RSA encryption on some PC computer systems, using the same inputs (message size and key length) as those used in the tests on the proposed encryption model. Performance data are presented in Table 2.12, and 2.13 for experiments on the computer system running the Intel Pentium 4 CPU at 1.8 GHz and 2.4 GHz, respectively. Figure 2.15 shows the relationship between the RSA key length and the encryption speed on Intel Pentium 4 1.8 GHz and 2.4 GHz machines.

It can be seen from Figure 2.15 that the size of encryption keys has similar impact on the ecnryption speed of P4-based machines as that of DSP. The encryption speed drops sharply when the key length increases, especially with keys from 128 to 512 bits. The performance

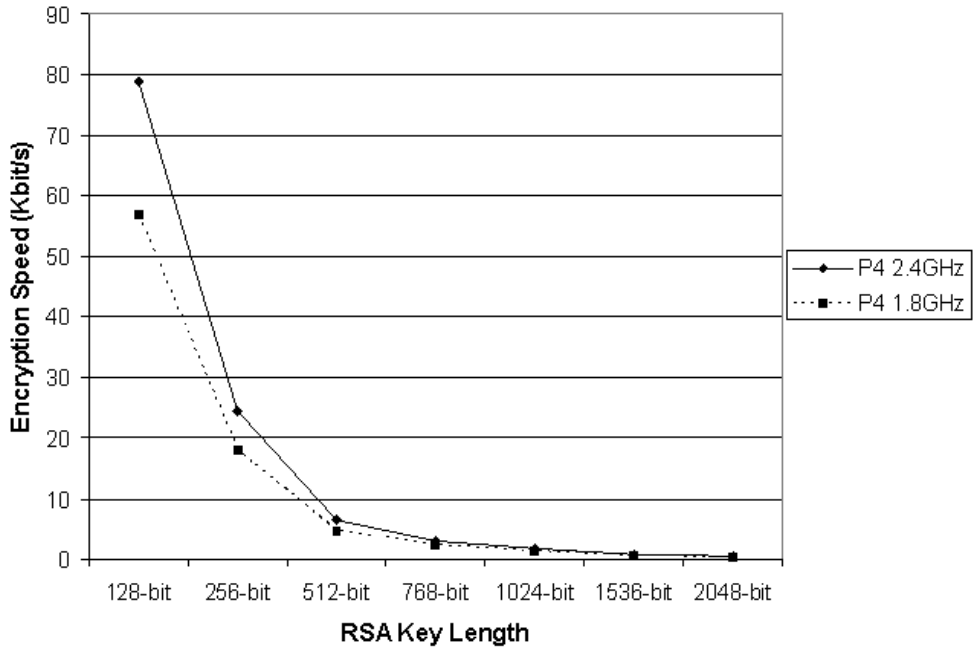

Figure 2.15: The relationship between the RSA key length and the encryption speed of the software implementation on the Intel P4 1.8 GHz and 2.4 GHz machines

comparison between 1200 MHz-DSP and P4-based machines, as shown in Figures 2.16 and 2.17 indicates that the DSP performs better than these Intel P4-based machines at large keys (from 512 to 2048 bits).

Figures 2.16 and 2.17 show the comparison of the encryption speed for the 2048-byte message on the DSP hardware at different clock speeds and the software implementation running on PC systems, using Pentium 4 1.8 GHz and 2.4 GHz CPUs.

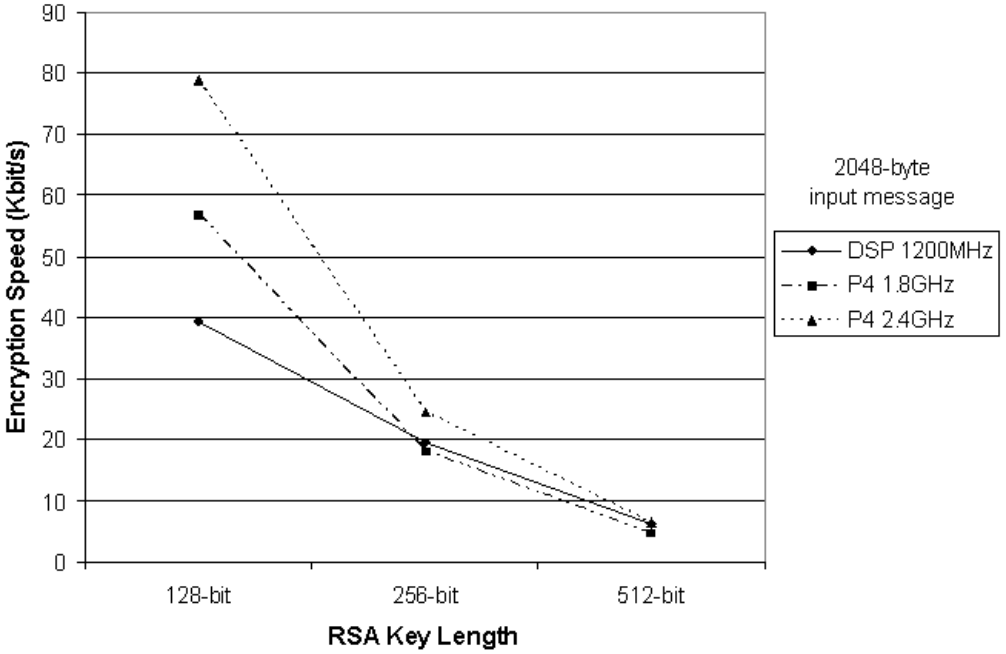

Figure 2.16: Encryption speed of DSP and PC machines for short encryption keys

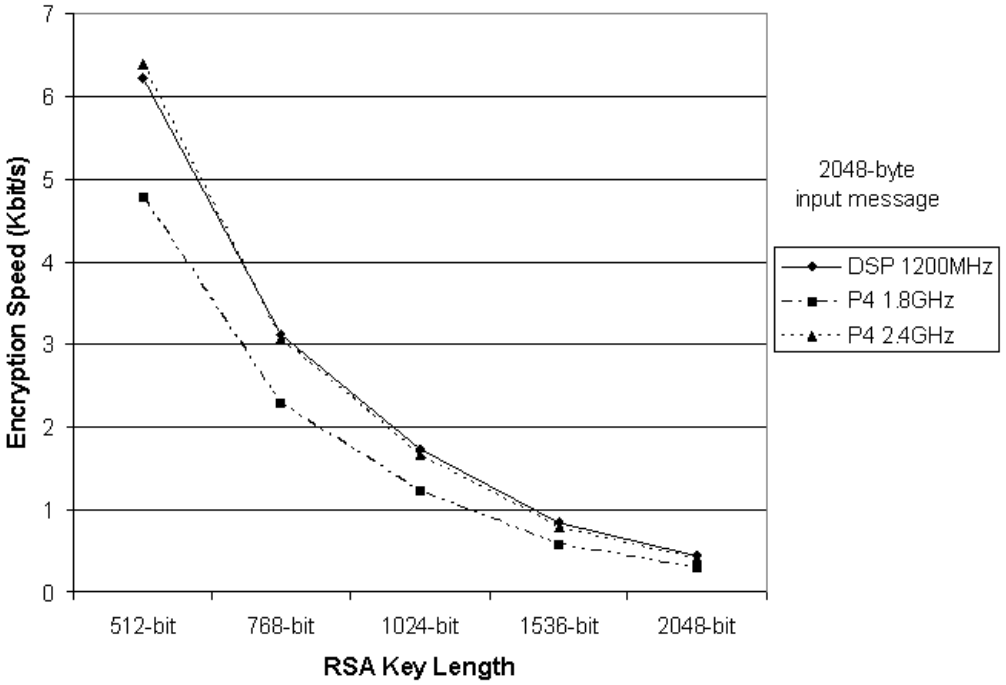

Figure 2.17: Encryption speed of DSP and PC machines for long encryption keys

Table 2.14: RSA Security's recommendation of RSA key sizes [108]

Protection Lifetime of Data	Present 2010	Present 2030	Present 2031 and Beyond
Minimum RSA key size	1024 bits	2048 bits	3072 bits

2.6 Discussion

2.6.1 Model's performace

In this section, we analyze the proposed model's performance results, compared to that of RSA software implementation on Pentium-based PC machines. In addition, main factors which affect the model's performance, such as length of encryption keys, DSP clock speed and the size of input messages, are also discussed.

2.6.1.1 Performance comparison with RSA software implementation on Pentium based PC machines

From the experimental results presented on Figure 2.16 and 2.17 (page 50), the 1200 MHz DSP performs better than Pentium 4 based PC machines on large keys, although it is slower than these PC machines on short keys. At short keys of 128 bits the DSP is slower than the Pentium 4 1.8 GHz machine, however, it performs slightly better than this PC machine at 256-bit keys and outperforms the P4 machine at larger keys. At 2048-bit keys the DSP is about 46% faster than the P4 1.8 GHz machine. Similarly, the DSP performs better than the P4 2.4GHz machine at encryption keys of 768 bits and larger, although it is slower than the PC machine at short keys of 128 bits, 256 bits and 512 bits.

The DSP's performance gain at large keys indicates that the proposed DSP based encryption model is more suitable for practical applications, as it is more scalable than the software implementation running on modern Pentium based PC machines. Practical RSA security requires large-size keys, because short keys of 128 bits and 256 bits can be factored quickly, and even 512-bit keys are not secure at present [108]. It is recommended by RSA Security Inc. [108], as shown in Table 2.14, that a reasonable RSA key size should be at least 1024 bits for the present time, and 2048 bits for the next 20 to 25 years [108, 122].

2.6.1.2 Dependence of speed performance on the length of encryption keys

It is clearly shown on Figure 2.8 (page 42) that the length of encryption keys has a big impact on the proposed model's speed performance. For all supported DSP clock speeds, the encryption speed drops dramatically when the length of encryption keys increases from 128 bits to 2048 bits. When the length of keys is doubled from 128 bits to 256 bits, the encryption speed drops by half, and encryption speed reduces to one fourth when the key size increases from 1024 bits to 2048 bits. This shows how computationally intensive RSA encryption is, especially for large keys [92]. Therefore, an crucial question is how to determine the size of RSA encryption keys that can help to balance the level of security and the performance. Although the model's speed is relatively slow with large keys, it is fast enough for small and medium size messages. The time taken to encrypt a message of a typical online credit card payment information is about 0.105 second and 0.419 second at 512-bit and 1024-bit keys respectively, as can be seen in Table 2.11 (page 47).

2.6.1.3 Dependence of speed performance on DSP clock speed

When the DSP clock speed increases, the encryption speed increases proportionally, depending on the length of encyption keys used (refer to results presented in Figures 2.10 (page 43), 2.11 (page 44), and 2.12 (page 46)). When the DSP clock speed increases from 500 MHz to 1000 MHz, the encryption speed increases 1.54 times and 2.0 times respectively, at encryption keys of 128 bits and 2048 bits respectively (as shown in Table 2.10 (page 45) and Figure 2.12 (page 46)). This means that the encryption speed gains more at longer encryption keys, with the same increase in DSP clock speed. Therefore, in order to get further gains on performance, especially for the RSA operation on the server side, a choice of a high-speed multi-processor DSP is the ideal solution.

2.6.1.4 Dependence of speed performance on the size of input messages

It can be seen from Figure 2.9 (page 43) that the size of messages has different impacts on the speed performance of the proposed encryption model, depending on the length of encryption keys used. On the tested message sizes, the larger a message is, the higher the encryption speed is. The DSP-1000 MHz encryption speed increases 8.65%, when the message size raises from 512 bytes to 4096 bytes at the encryption key of 512 bits, while speed gain is 41.27% at 2048-bit

encryption key, for the same increase in message size. The reason for the speed performance gain for large messages is that for any message size, the initialization and cleaning work of the encryption process gets the same overheads. However, when the message is large enough, the speed performance will be stable. This is because the overhead of the initialization and cleaning job accounts for only a small portion of the total time for the encryption process.

2.6.1.5 Communication performance

As shown in Figure 2.13 (page 46), the communication speed is proportional to the message size due to the use of data transfer in batch mode. However, the RTDX communcation, based on USB 1.0 is still not fast enough to support the transfer of large amounts of data. A possible solution to improve the communication throughput, is to use faster communication interfaces such as USB 2.0 or PCI interface. A faster DSP-computer communication interface will improve the communcation performace as well as the overall performance of the proposed encryption model.

2.6.2 Security

The security of the proposed encryption model is ensured by the strength of the RSA encryption algorithm and the hardware cryptography features. RSA security is based on the difficulty of factoring big prime integers. Generally, the larger the key, the more secure the system [92, 108, 122]. Table 2.15 [108, 122] shows the estimated cost for factoring RSA keys from 428 bits to 1024 bits, in which the amount of time required to break the key is expressed in the total number of arithmetic operations. A 1024-bit key will be 52 million times harder to break in time, and about 7200 times harder to break in terms of space, compared to the 465-bit key. We have tested the proposed encryption model with RSA encryption keys, ranging from 128 bits to 2048 bits, in which large-size keys (from 1024 bits to 2048 bits) are considered good keys (for use in practice) [108, 122].

As discussed in Section 2.4.1, we used a simple method to format input message into encryption blocks for performance tests. In order to meet security requirements, standard format of encryption blocks [55, 56] must be complied. In addition, the padding string of each encryption block should be generated independently, especially if the same data is entered into more than one encryption block [55, 56].

Table 2.15: Estimated cost for factoring RSA keys [108]

Keysize (bit)	Total Time	Factor Base	Sieve Memory	Matrix Memory
428	$5.5 * 10^{17}$	600 Kbytes	24 Mbytes	128 Mbytes
465	$2.5 * 10^{18}$	1.2 Mbytes	64 Mbytes	825 Mbytes
512	$1.7 * 10^{19}$	3 Mbytes	128 Mbytes	2 Gbytes
768	$1.1 * 10^{23}$	240 Mbytes	10 Gbytes	160 Gbytes
1024	$1.3 * 10^{26}$	7.5 Gbytes	256 Gbytes	10 Tbytes

2.6.2.1 Security of hardware cryptography

Hardware cryptography has built-in features to address the security issues of communication and end system security, as discussed in Section 2.1. The biggest advantage of hardware-based cryptographic implementations, over their software counterparts, is that a hardware cryptosystem runs in its own physical memory space, therefore, it effectively prevents illegal memory access and modification by external programs. Running on a separate physical environment also means hardware cryptosystems are protected from underlying operating systems' security flaws which usually lead to security risks to software implementations. Moreover, hardware can be used to safely store long-term private keys [6, 32]. This is one of the most important reasons that hardware was selected as the platform to implement the proposed encryption model. In our DSP hardware based RSA encryption model, the RSA private key is kept in DSP hardware, and is thus not directly accessible, even if the host is compromised.

From an architectural aspect, a *secure communication tunnel* is created between the client and the server by the integrated hardware based encryption modules. All encryption and decryption tasks are securely carried out inside these hardware modules. This is very important to the proposed model's security, because this mechanism can effectively prevent access violation by malicious propgrams to the model's code and protected data.

2.7 Summary

In this chapter we presented a new hardware-based encryption model for the enhancement of security for E-commerce applications. The proposed model is based on the integration of

an existing E-commerce model and embedded hardware based RSA encryption modules. The security of the proposed model is guaranteed by the strength of RSA encryption at large keys and the features of hardware-based cryptosystems, such as difficult-to-break-in features and protected-from-operating-system flaws.

A successful implementation of DSP hardware-based RSA encryption and decryption modules using Texas Instruments' TMS320C6416 DSK boards was presented to test the concepts. We also provided the details on how to communicate efficiently between the host computer applications (the web browser and the web server) and the DSP hardware based encryption modules. A fully functional E-commerce system, including the web and database applications, was built to test the proposed model's performance and security. From our experimental results, it is clear that, at large RSA encryption keys, the proposed DSP based RSA encryption system outperforms a 1.8GHz Intel Pentium 4 PC, and it has comparable performance to a 2.4GHz Intel Pentium 4 PC that has almost double clock speed of the DSP. The results confirm that our model is more scalable than software-based implementations and it can be used to efficiently improve the security and performance of E-commerce applications.

Part II

Intrusion Detection: Hidden Markov Model Based Program Anomaly Detection

Chapter 3

Program Based Anomaly Intrusion Detection

3.1 Introduction

Intrusion detection is the process of monitoring events that occur in a computer system or network, and analyzing these events for evidence of intrusions [5, 70]. An intrusion detection system (IDS) usually consists of three functional components: information sources, analysis and response. Information sources can be drawn from different levels of the target system. Networks, hosts, and applications are the most common sources in intrusion detection. The analysis component is responsible for modelling and analyzing the collected data events, to decide whether those events indicate that intrusions are occurring, or have already taken place. The response is a set of actions taken when an intrusion is detected. The actions can be passive measures, such as reporting intrusion alerts to administrators, or active measures, such as blocking intrusions.

3.1.1 Overview of intrusion detection

Based on the information sources, there are two main types of intrusion detection systems: network-based IDSs and host-based IDSs. Network-based IDSs detect attacks by capturing and analyzing network packets. A network-based IDS can be used to protect multiple hosts that are connected to a network segment, or a switch, which is monitored by the IDS. However, network-based IDSs may have difficulty processing all packets in a large, or busy network,

and therefore they may fail to recognize an attack launched during periods of high traffic. By contrast, host-based IDSs detect attacks by analyzing information collected within an individual computer system. This allows host-based IDSs to provide a comprehensive analysis of the system activities, to determine exactly which processes and users are involved in a particular attack on the system [5].

Based on analysis methods, intrusion detection techniques can be classified into two categories : *misuse detection* and *anomaly detection* [26, 93]. Misuse detection searches for signatures of known attacks and any matched activity is considered an attack. Misuse detection can detect known attacks effectively, but it is incapable of detecting novel, or unknown attacks. Anomaly detection attempts to characterize the *normal behaviour* of an object and any significant deviation from that behaviour is considered a possible attack. Anomaly detection has the potential to detect unknown attacks, because no advance knowledge about specific attacks is required [26, 93]. The work discussed in Part II deals with anomaly detection, specifically in detecting anomalies in the execution of programs, as well as intrusions against programs.

3.1.1.1 Misuse intrusion detection

Generally, misuse intrusion detection, also known as signature-based intrusion detection, first defines signatures of known intrusions, then monitors current activities for such signatures, and reports matched cases. An intrusion signature specifies a pattern of scenarios or events that lead to an attack or misuse. Intrusion signatures are not only useful to detect intrusions but also to discover intrusion attempts. A partial satisfaction of an intrusion signature may indicate an intrusion attempt [70].

There are several misuse intrusion detection approaches which use different representations of intrusion signatures, as well as apply different matching algorithms to search for intrusion patterns. For example, NIDES [87], a real-time intrusion-detetion expert system consists of a rule-based analysis component for misuse detection. The rule-based component uses an expert system to detect attempts which exploit security vulnerabilities of the monitored systems. The rule-based component encode known system vulnerabilities, attack scenarios and suspicious behaviour into rules. For example, one such rule is defined as “more than three consecutive unseccessful login attempts for the same useid within five minutes is a penetration attempt”. A monitored activity that satisfies any of these pre-defined rules is flagged.

The key advantage of misuse intrusion detection approaches is that they are very effective at detecting known attacks whose signatures have been encoded into rules, or patterns without generating an overwhelming number of false alarms. However, misuse intrusion detection approaches are not effective at detecting unknown attacks, because there are no matched signatures for such attacks. In addition, since attacks change their patterns quickly and new attack techniques are invented rapidly, the attack signatures of misuse detection systems must be updated frequently across many platforms in order to be able to detect newly created attacks. This maintenance work is very labour-intensive task [81].

3.1.1.2 Anomaly intrusion detection

Anomaly intrusion detection involves in first (i) building a profile of normal behaviour of a monitored object, and then (ii) monitoring the object's actual behaviour to detect significant deviations from normal behaviour recorded in the profile. The monitored objects can be users, programs, network traffic or other resources in a system. Generally, a object's profile is constructed from historical data (also known as training data) collected over a period of normal operation of the object.

The main assumption of anomaly intrusion detection is that intrusive activity is highly correlated to anomalous activity [93]. In other word, intrusive activity can be considered as a subset of anomalous activity. In practice, an intrusive activity can be implemented as a set of individual activities, and none of these individual activities is independently anomalous [70]. Since the set of anomalous activities is not generally the same as the set of intrusive activities, the evaluation result for an activity, by an anomaly intrusion detection system, can be one of the following four possibilities:

1. *Intrusive* and *anomalous*: An activity is intrusive and it is correctly reported as intrusive because it is also anomalous. This is *true positive* case because the intrusion detection system correctly reports the activity as intrusive.

2. *Not intrusive* and *not anomalous*: An activity is not intrusive and it is not reported as intrusive because it is not anomalous. This is *true negative* case because the intrusion detection system does not report the activity as intrusive.

3. *Not intrusive* but *anomalous*: An activity is not intrusive but it is reported as intrusive

because it is anomalous. This is *false positive* case because the intrusion detection system falsely reports the activity as intrusive.

4. *Intrusive* but *not anomalous*: An activity is intrusive but it is not reported as intrusive because it is not anomalous. This is *false negative*, or *missed* case because the intrusion detection system fails to detect the intrusive activity.

Obviously, both false negatives and false positives are not desirable. One possible way to reduce false negatives is to lower thresholds that define the abnormalcy. However, low thresholds may result in large volume of false positives, which virtually disrupt the normal operation of the monitored system and reduce the effectiveness of the anomaly intrusion detection system.

There are several anomaly intrusion detection approaches which use different information sources as well as apply different modelling techniques to characterize normal behaviour of monitored objects. Users, programs and network traffic are the most common monitored objects in anomaly detection. Data modelling techniques, such as statistics, data mining and machine learning, have been used in many anomaly intrusion detection systems [70, 82, 93].

The main advantage of anomaly intrusion detection approaches is that they have the potential to detect novel and unknown attacks without advance knowledge about these attacks [26, 93]. This is because anomaly detection approaches focus on detecting abnormal, or unusual behaviour based on the normal behaviour defined in profiles. However, anomaly detection approaches usually produce high volume of false alarms, compared to misuse detection counterparts, due to the unpredictable behaviour of users and networks. In addition, anomaly detection approaches often require extensive system resources to contruct profiles of normal behaviour from historical data.

3.1.2 Detecting intrusions by monitoring program behaviour

Monitoring program behaviour to detect intrusions has a major advantage over either monitoring user behaviour and network traffic, because of its stability [25, 26, 66, 93]. Unlike the unpredictable nature of human user behaviour or the unstable pattern of network activity, the behaviour of a program is relatively stable during its execution because it is generally designed to perform a limited set of specific tasks [26]. More specifically, the monitored programs in anomaly intrusion detection are commonly *system* or *privileged programs*. These programs are

usually running in the background to provide services to users or other programs. The behaviour of these programs should not be changed without our awareness [93]. Therefore, if intrusions can be detected as deviations from normal program behaviour, false alarms caused by changes in the historical patterns of user behaviour can be eliminated [93].

The remaining problem is how normal program behaviour should be characterized. Intuitively, a program's normal behaviour can be extracted from the semantic of its source code. However, this approach is generally unsolvable because the source code of programs are not always available, especially for commercial software. In addition, the program behaviour extraction from its source code can be very complicated due to the growing comlexity of modern computer software. A more practical approach is to consider a program as a *black box* and monitor its activity and interaction with the underlying operating system [25, 26]. The program's observed information is then used to characterize its normal behaviour. There exists a mechanism, known as *kernel auditing*, and supported by many operating systems, which allows auditing systems to record programs' behaviour in terms of their interaction with the operating system [93]. For example, many Unix systems support the *Basic Security Modules* (BSM) which provide detailed audit trails for some running processes.

Forrest et al [25] proposed the use of *short sequences of system calls*, made by an Unix privileged program to the operating system kernel during its normal execution, to characterize its normal behaviour. Their approach is based on two assumptions: (i) temporal-ordering short sequences of system calls of an Unix privileged process are consistent during its normal operation, and (ii) abnormal sequences of system calls are produced by anomalies or intrusive activities. This program-based anomaly detection method is relatively simple. Firstly, *normal traces* of system calls produced by a program, during its normal execution, were collected. Then, a *normal database*, characterizing normal program behaviour, was constructed from normal traces. The normal database was built by sliding a window of k system calls across normal traces, one system call at a time, and recording system calls within the sliding window as one short sequence at each move. During this process, only unique short sequences were added to the normal database. Then, this normal database was used to classify short sequences of system calls of test traces to find anomalies. If a short sequence was not found in the normal database, it was considered as abnormal. The anomaly signal was measured by counting the total number of abnormal short sequences in test traces. Experimental results showed that short sequences of system calls

were consistent for some Unix privileged programs and the proposed method was able to detect several instrusions on Unix *sendmail* and *lpr* programs.

Forrest et al [26], in an extension to their previous work [25], conducted further experiments on a wide range of Unix privileged programs in synthetic[1] and real[2] working environments. Instead of simply counting the number of abnormal sequences, the *Hamming distance* method was used in the latter to compute the differences between a mismatched short sequence and short sequences in the normal database. The minimum value of the computed differences was used as the anomaly signal. Their experimental results also confirmed that short sequences of system calls are a good discriminator between the normal and abnormal behaviour of programs. Their proposed detection method [25, 26] is efficient because it requires only one pass through the training data (normal traces) to build the normal database. However, due to the growing complexity of modern software applications, the behaviour of a program can be very complicated, which makes it difficult to construct a complete normal database that covers all execution scenarios of the program. The incompleteness of the normal database is likely to lead to a high level of false alarms. Therefore, it is desirable to find more expressive modelling methods to characterirze normal program behaviour [93].

Several other modelling techniques, such as data mining [78, 79], finite state automata (FSA) [68, 93, 117] and hidden Markov models [42, 43, 102, 134] have been used to construct models of normal program behaviour using system calls. In an effort to find the best modelling method for normal program behaviour using system calls, Warrender et al [134] have investigated various modelling techniques through extensive experiments. They used the normal database method [25, 26], frequency based method, data mining, and the hidden Markov model to construct detection models from the same normal traces of system calls. Their experimental results have shown that the hidden Markov model method can generate the most accurate results on average, although the training cost of the hidden Markov model method is high. Other program anomaly detection approaches based on hidden Markov models also reported promising results [42, 43, 102].

[1] A type of artificial or simulated environment created to test programs. Detailed description of synthetic environments is given in [26].

[2] Also called live environment [26].

3.1.3 Aims

The hidden Markov model (HMM) is a very powerful data modelling technique [103, 104], and it has been widely used to construct signal models in speech recognition [16, 28, 29, 30, 72, 104], and in protein sequence analysis in biology [47, 58, 59, 69, 115]. In anomaly detection, HMM modelling has been used to increase the detection rate, as well as to reduce the false alarm rate [42, 43, 102, 134]. However, the major drawback of these HMM-based anomaly detection approaches is that the HMM training using the well-known Baum-Welch algorithm [104] demands excessive system resources, which makes these HMM-based detection approaches less efficient for practical use.

We propose an HMM-based program anomaly intrusion detection method using system calls, which focuses on three major aims:

1. To improve detection accuracy, by reducing false alarms and by increasing the detection rate. Generally speaking, existing anomaly intrusion detection approaches usually produce too many false alarms, which will virtually disrupt the normal operations of the monitored systems. Therefore, reducing the false alarm rate as well as increasing the detection rate are crucial for any anomaly detection approaches [81, 82].

2. To improve efficiency in terms of training costs and testing costs. As mentioned, HMM-based anomaly detection approaches consume excessive system resources due to the high HMM training costs, which also affects the normal operations of the monitored systems. Therefore, improving efficiency is critical to make HMM-based anomaly detection approaches practical.

3. To improve adaptability of existing anomaly detection approaches. The proposed detection method should be adaptable to the changes of normal program behaviour.

3.1.4 Outline of the proposed approach

In order to improve the detection accuracy, we employ the hidden Markov model (HMM) as the main data modelling technique to construct a model of normal program behaviour using system calls because HMM is a powerful modelling technique in a wide range of applications [103, 104]. HMM modelling in anomaly intrusion detection has also brought promising results, but is still in an early stage [42, 43, 102, 134]. Futhermore, in order to increase the detection rate and

to reduce the false positive rate, we propose two detection schemes: (i) a two-layer detection scheme and (ii) a fuzzy-based detection scheme. Both detection schemes use a normal database [25, 26] and an HMM model to evaluate test traces of system calls for possible intrusions. The two-layer detection scheme aims at reducing false positive alarms by applying two sequential tests on each short sequence. On the other hand, the fuzzy-based detection scheme focuses on improving the detection rate, as well as reducing false positive alarms, using fuzzy inference to combine multiple sequence parameters generated by the normal database and the HMM model.

Efficiency improvement is also critical to HMM-based anomaly detection approaches. A major factor that affects the overall efficiency of an intrusion detection system is the efficiency of the construction of the detection model (also known as detection profile, rule base) [25, 26, 134]. Therefore, the efficiency improvement of the HMM construction or HMM training is crucial to any HMM-based anomaly detection approachs. Traditional HMM batch training using the Baum-Welch algorithm [104] was found to be expensive in terms of time and storage requirements [104, 134]. We introduce an HMM incremental tranining scheme with optimal initialization for program anomaly detection, which aims at reducing training time and storage demands.

While accuracy and efficiency are crucial requirements of an intrusion detection system, its adaptability is also important in a networked computing environment [82]. This means that an intrusion detection system should support the dynamic update of its detection model in order to adapt to the changes of the monitored object's behaviour. Our HMM-based detection method supports the online update of its detection model, which is based on the model's incremental training scheme using cleaned online training data. This makes the proposed detection model up-to-date, and therefore, maintains the detection accuracy.

In next section, we review some monitoring techniques in anomaly detection, including monitoring user behaviour, monitoring network activity and monitoring program behaviour. We then focus our discussion on program anomaly detection approaches using system calls.

3.2 Related Work in Anomaly Detection

Anomaly intrusion detection is based on the assumption that "intrusions are highly correlated to abnormal behaviour exhibited by either a user, or the system" [93]. Based on this assumption, anomaly detection techniques usually focus on detecting deviations of an object's current behaviour from its past behaviour. If a deviation is significant it is considered a possible intrusion

[26, 93].

Initially, one must decide which object to monitor to detect intrusions. The monitored object can be a single object, or a set of objects, in computer and network systems. Once the monitored object is selected, an anomaly detection system can be developed in two stages:

1. The construction of the detection model (also known as detection profile, rule base). This is the process of building a model of the object's normal behaviour. This task can also be done in two steps as follows:

   (a) Collect training data: collect data for the construction of the detection model. It is usually assumed that the training data is clean and free from anomalies and intrusions.

   (b) Construct the detection model: the detection model is built using suitable data modelling techniques. Data modelling techniques such as data mining, machine learning and statistics have been commonly used to build detection models [70, 82, 93].

2. Testing for intrusions: monitoring the object's current behaviour to find deviations from its normal behaviour, which is recorded in the detection model for possible intrusions.

In subsection 3.2.1, we discuss techniques used to detect intrusions by monitoring objects' activities. In subsection 3.2.2, we discuss the methods used for modelling normal program behaviour using system calls. In subsection 3.2.3, we review the methods used to improve detection accuracy, efficiency, adaptability and extensibility of current anomaly detection techniques. Subsection 3.2.4 analyses some open issues of current anomaly detection approaches.

3.2.1 Monitoring techniques in intrusion detection

In one of the earliest studies in anomaly detection, Anderson [1] suggested that intruders can be detected by monitoring the abnormal changes in historical patterns of use for individual users. In a broader context, this suggestion can be applied to detect intrusions by monitoring objects' activity such as user behaviour, network activity and program behaviour.

3.2.1.1 Monitoring user behaviour

In light of Anderson's suggestion [1], Denning [17] proposed a generic detection model that aims to detect a wide range of security violations including break-in attempts by outsiders and abuses

by insiders. Denning's generic detection model [17] is based on the hypothesis that "security violations can be detected by monitoring a system's audit records for abnormal patterns of system usage" [17]. The generic detection model [17] has six main components including the subjects, objects, audit records, profiles, anomaly records and activity rules, which can be considered as a rule-based pattern matching system. For a given subject with respect to a given object, an *activity profile* is created to represent the normal pattern of the subject's activity, which is used to determine whether the subject's current activity is normal or anomalous.

More recently, Lane et al [73] used machine learning to construct user profiles from sequences of user commands in order to detect user anomalies. Each record in a user profile is a *characteristic sequence* which is an ordered and fixed-length set of temporally adjacent actions. Lee et al [82], on the other hand, applied data mining techniques such as frequent episodes to learn the normal patterns of user activity from input sequences of shell commands of user sessions. In the learning process, a new pattern of user commands is compared with the historical patterns and a *similarity score* is given. Only patterns with a high similarity score are merged into historical patterns to create the user profile [82].

A major difficulty of user anomaly detection is that human user behaviour is generally unstable and unpredictable in nature [93]. This means that the current user activity does not always match the historical user activities. Therefore, user-based anomaly detection systems usually tend to generate a high level of false positive alarms. In addition, there are gradual changes in user behaviour over a long period of time, which makes it even more difficult to detect user anomalies.

3.2.1.2 Monitoring network activity

Monitoring network activity to detect intrusions has been the attention of many reseachers [9, 23, 82, 84, 97, 99, 116]. In [99], Paxson described a system, called Bro, which is a real-time and stand-alone system for detecting network intrusions by monitoring network traffic. Bro's security policies are encoded into event handler scripts written in Bro language which is a specialized language used to express a site's security policy. Network traffic streams are filtered into a series of high-level events by an *event engine*. These events are then processed by event handler scripts which are interpreted by a *policy script interpreter*. Bro reportedly achieves some advanced features including high-speed monitoring, real-time notification, separation between

mechanism and policy, and extensibility [99]. However, the creation of site security policy scripts using the Bro language still requires enormous amount of manual engineering work [82].

Aiming at the automatic creation of detection rules from training data, Lee et al [82], first used various data mining algorithms such as classifications, association rules and frequent episodes to compute frequent patterns from network connection data. Next, they extracted the predictive features from these frequent patterns, and then these predictive features were used to generate rules to detect known network intrusions [82]. The main advantage of this approach is that it can reduce the amount of manual engineering work to encode the signatures of known intrusions. However, as a misuse detection approach, it still needs at least one real trace of each intrusion in order to extract the intrusion pattern, which is not possible for unknown attacks.

Instead of using real traces of known intrusions for detection rule learning, Fan et al [23] proposed an algorithm called DBA2 (Distribution-based Artificial Anomaly) to generate *artificial anomalies* for known network intrusions. These artificial anomalies were used to form initial anomaly classes, which help a data mining based inductive learner to accurately discover the boundary between known classes (normal connections and known intrusions) and anomalies. It is reported that the proposed method performed well on unknown intrusions and comparably to signature-based approaches on known intrusions [23].

Focussing on detecting intrusions that exploit the weaknesses of implementations of network protocols, Lee and Heinbuch [84] proposed a network anomaly detection method which is based on neural networks. The proposed system's detection model is composed of a hierarchy of back propagation neural networks. Artificial network data that are generated based on network protocols' specifications, are used to train these neural networks. According to the simulation results, the proposed model was able to detect 100% of specific attacks in the experiment without any prior knowledge about these attacks [84].

3.2.1.3 Monitoring program behaviour

Recently, there have been significant research interest in anomaly detection by monitoring program behaviour [25, 26, 42, 43, 53, 54, 66, 68, 78, 93, 102, 112, 117]. Ko et al [66] proposed a method to detect a program's vulnerabilities by monitoring its execution. The normal program behaviour is defined using a program policy specification language, in which the allowed operations of the program are formally specified. It is reported that the program policies are

concise and clear, and the proposed approach is capable of detecting exploitations of known and unknown vulnerabilities of programs [66]. However, this is a type of security enforcement technique that requires special knowledge about the program's internal functionality to create an appropriate program security policy. In addition, this approach cannot detect intrusions that do not cause violations of the pre-defined program security policy. For example, Denial of Service (DoS) attacks usually flood server programs with huge number of requests, which may not cause any violations of the internal program security policy.

Other program-based anomaly detection approaches consider a program as a *black box* and focus on modelling normal program behaviour using the program's observed information such as *language library calls* [54], *temporal signatures* [53], and *system calls* [25, 26, 42, 43, 68, 78, 93, 102, 112, 117]. Jones et al [54] described a program-based anomaly detection method using language library calls used by a program during its execution. They applied the method of [25] to construct a *normal database* from normal traces of language library calls to represent normal program behaviour. Their experimental results using C language library calls have shown that the proposed method can correctly detect some intrusions such as buffer overflow, trojan programs and DoS attacks [54]. The language library call-based method reportedly gave stronger anomaly signals for a Trojan attack on a *mSQL* program, than the system call-based approach. However, there was not sufficient evidence provided in [54] to conclude that the language library call-based method performed better than the system call-based approach in different attacks on other programs. In general, low-level system calls represent the system resource usage of a program better than high-level language library calls [25, 26].

Jones and Li [53] used a *temporal signature* to characterize normal program behaviour. The temporal signature was created by measuring time duration between adjacent system calls of a program. This is an interesting approach because it has the potential to detect intrusions which do not create sufficient anomalies in the temporal ordering of system calls (which is one of the basic assumptions of the approaches in [25, 26]). The measurement of elapsed time of consecutive calls was conducted multiple times, for each unique sequence of system calls, in order to capture elapsed time patterns. After removing high variance data and noise, the temporal signature was found relatively stable for a number of applications. However, there are many factors such as CPU clock speed, CPU load, and the availability of system resources, that affect execution time of a system call or elapsed time between two consecutive system calls. Because of high variation

of these factors, it is difficult to remove their effect on the temporal signature. In addition, execution time of system calls is machine (CPU) dependent, which makes the deployment of the proposed method even more difficult.

3.2.2 Modelling normal program behaviour using system calls

Program-based anomaly detection using system calls has received a growing attention by many researchers since the successful initiative of Forrest et al [25]. Inspired by the properties of natural immune systems, Forrest et al [25] introduced the concept of *self* of a program to represent its normal behaviour. A program's *self* is defined as a *normal database*, which is a list of unique and fixed-length short sequences of system calls as discussed in Section 3.1. Since then, several other approaches have been proposed to represent normal program behaviour using system calls. Among them, approaches that are based on data mining [78, 134], finite state automata (FSA) [68, 93, 117] and hidden Markov models (HMM) [42, 43, 102, 134] have been the most successful.

3.2.2.1 Data mining-based approaches

Lee et al [78, 79] proposed a method to learn detection rules from program traces of system calls based on data mining. They used a data mining-based rule induction program called RIPPER [13] to learn detection rules from training data [78, 79]. Two separate sets of detection rules were learnt: (i) the first set was used to classify a test sequence as normal or abnormal, and (ii) the second set was used to predict a system call in a test sequence to find violations. The proposed method's rule base requires less storage than the normal database of [25, 26]. Another advantage over [25, 26] is that it requires less training data to build the detection model in order to achieve the same level of detection performance. However, the construction of the detection model is much more expensive than the construction of a normal database of [25, 26] in terms of training time and system resoure requirements. This is because there are additional overheads generated by the learning of initial abnormal classes using a normal database and by the rule induction using the RIPPER program.

3.2.2.2 Finite state automata-based approaches

Finite State Automata (FSA) is a natural way to represent normal program behaviour as it can capture program structures such as loops and branches [117]. In addition, the FSA states

can remember short-term and long-term correlations among system calls [117]. Kosoresow and Hofmeyr [68] manually built a Deterministic Finite Automaton (DFA) to characterize normal program behaviour using system calls. The DFA was reportedly able to detect some intrusions. However, the manual construction of finite state automata from system calls is a time-consuming task, especially for complicated programs and long running traces of system calls.

Sekar et al [117] proposed an automated method to learn FSA from program execution information, which effectively solves the DFA manual learning problem [68]. The FSA can learn the program structures, therefore it is able to capture both short-term and long-term relationship among system calls. The proposed method [117] makes use of operating system mechanisms to get program execution information which includes: system calls, values of program counter (the position from which each system call was made), and the stack information when each system call was made. Each state in the FSA corresponds to a distinct value of the program counter, and a transition is a system call. The FSA was reportedly light-weight and had fast training convergence. Moreover, the proposed method had low false positive rate, and was able to detect several attacks such as buffer overflow, malicious code changes, password guessing and DoS. However, learning a program's structures without its source code and special knowledge about the program's internal functionality, is an extremely difficult task. In addition, modern applications usually make heavy use of language library functions, and functions from dynamic linked libraries, which makes it even more difficult to learn program structures correctly.

Aiming at improving detection performance, Micheal and Ghosh [93] proposed two simple state-based algorithms for program-based anomaly detection using system calls. Unlike Sekar et al's [117] FSA that requires values of program counter, these finite state automata were automatically constructed only from normal traces of system calls [93]. In the first algorithm, a finite state automaton was constructed from training data of system calls to represent normal program behaviour. The FSA was able to detect the unseen short sequences in the training data. The second algorithm built a finite state machine, of which each state was associated with the probability distribution of the successor sequence of system calls. This algorithm was designed to detect *statistical deviations* of current program behaviour from its normal behaviour. The second algorithm had an advantage over the first algorithm as its state machine had the capability of modelling long-term dependencies in normal program behaviour. The performance of the proposed detection algorithms was reportedly almost comparable to the normal database

method [25, 26] while they required much less training data.

3.2.2.3 Hidden Markov model-based approaches

Modelling normal program behaviour, using hidden Markov models (HMM), has some common characteristics with modelling normal program behaviour, using finite state automata, because HMM is a special type of finite state machines. Although the hidden Markov model is a powerful data modelling technique [19, 103, 104], its applications in anomaly detection are still in an early stage. HMM modelling was used to construct a model of normal program behaviour from normal traces of system calls [134]. The HMM-based approach [134] can give the best detection accuracy on average, compared to other modelling methods including normal database method [25, 26], frequency-based method, and data mining [78].

In an attempt to find the original factor which determines the emission of program system calls, Qiao et al [102] first constructed an HMM from the training data, then used the HMM to find the sequence of hidden states for the observation sequence of system calls. Next, they built a *normal database* from these hidden states using the approach of [25, 26] to represent normal program behaviour. It is reported that the proposed method [102] had a smaller detection database, compared to that of [25, 26], and it required less training data.

HMM-based anomaly detection approaches [42, 102, 134] face a common problem of efficiency degradation. This is because the training of hidden Markov models from traces of system calls using the Baum-Welch algorithm [104] was found expensive in terms of time and storage [77, 104, 134]. Therefore, it is crucial to improve the efficiency of HMM training in order to make HMM-based anomaly detection approaches more practical. This is one of the main objectives of our research.

3.2.3 Improving intrusion detection performance: existing methods

An intrusion detection technique can be evaluated based on the following four characteristics [26, 81, 82]:

1. *Accuracy*: Accuracy can be evaluated as a combination of detection rate and false postive rate. A good intrusion detection technique should have a high detection rate as well as low a false positive rate.

2. *Efficiency*: In a narrow context, the efficiency of an intrusion detection technique can be measured by: (i) the cost of the detection model construction for a monitored object, and (ii) the cost of testing the object's current activity against its recorded normal activity. Each of these costs consists of several factors including time and system resources.

3. *Adaptability*: An intrusion detection method should be able to detect novel/unknown attacks. Furthermore, the intrusion detection method should support the dynamic update of its detection model in order to adapt to the changes of the object's behaviour.

4. *Extensibility*: This relates to what extent an intrusion detection system supports extensible features such as reuse, customization and upgrade.

Next, we discuss the methods that have been used to improve accuracy, efficiency, adaptability and extensibility in anomaly intrusion detection.

3.2.3.1 Accuracy

The common methods that have been used to improve anomaly detection accuracy can be broadly classified into four categories: (i) improvement of the detection model (also known as detection database, rule base), (ii) enhancing of the evaluation method of anomaly signal, (iii) enrichment of the training data quality, and (iv) combination of misuse and anomaly detection in an unified system.

Among these four methods, improvement of the detection model, or more specifically, finding more appropriate representations of normal behaviour of monitored objects in anomaly detection, has been the most effective way to increase detection accuracy. In [68, 93, 117], more expressive FSA models have been used to learn normal program behaviour using system calls as opposed to normal databases in [25, 26]. This is because FSA models can capture program structures, and short-term and long-term correlations among system calls. The proposed method in [117] reportedly had a much lower false positive rate than the normal database method in [25, 26]. Hidden Markov model was found to give the highest detection accuracy, on average, when compared to other modelling techniques, including normal database, frequency-based method, and data mining [134].

Enhancing the evaluation method of the anomaly signal is also an effective way to increase the detection rate and reduce false alarms. Forrest et al [25] in their initial work, simply counted

the number of mismatched sequences in the whole test trace and considered this number as the anomaly signal. However, this result depends on the length of the test trace. In addition, it is not a good indication of how much difference there is between a mismatched sequence and sequences stored in the normal database. A better solution on how to measure anomalies was proposed in [26], which computes the minimum value of *Hamming distances* between a mismatched sequence and all sequences in the normal database. Although the latter method gives better detection accuracy than the former, it is still not suitable for online detection because it does not take the temporal locality of test sequences into account. Methods which evaluate the anomaly signal based on temporal frames of test sequences, are more suitable for online detection [93, 117, 134].

Improving the quality of training data for detection model construction can be used to increase the detection accuracy. Fan et al [23] generated *artificial anomalies* for known network intrusions, which were labelled and injected into the training data to help the inductive learner correctly recognize the boundary between known classes and anomalies. In another approach to program anomaly detection, Sekar et al [117] used operating system mechanisms to gather information of program execution, which includes the program counter and stack information at the time each system call was made. While the program counter values were used to correctly learn program structures, the stack information helped to detect several types of buffer overflow attacks. Information, such as parameters of system calls, can also be used to enrich the system call training data for program anomaly detection.

Misuse detection and anomaly detection can be combined in one system that can take advantages of advanced features of both misuse and anomaly detection techniques in order to improve detection accuracy. Lunt [87] described a hybrid intrusion detection system, called NIDES, which has an anomaly detection module and a rule-based misuse detection module. The two detection modules work in parallel and independently to each other. Fan et al [23], on the other hand, integrated misuse detection and anomaly detection in one module with an unified decision making process. The integration of misuse detection and anomaly detection achieved a higher detection rate and a lower false positive rate than each separate detection technique [23].

3.2.3.2 Efficiency

Building concise and simple detection models is a common approach to reduce both the model construction cost and the cost of testing for intrusions. The construction of a *normal database*

is very efficient in terms of training time and system resources because it requires only a single pass through the training data [25, 26]. On the other hand, the construction of detection models needs multiple passes through the training data using other modelling methods such as finite state automata [68, 93], data mining [78, 79, 23], and hidden Markov models [102, 134]. However, since these latter methods provide better detection accuracy, the balance between efficiency and accuracy must be taken into account.

Reducing the required amount of training data for the construction of detection models is also an effective approach. Lee et al [78] reported two data mining based methods for the construction of the detection model, which require only 80% of the training data set used by the normal database method [25, 26] for the same level of detection performance. Finite state automata based approaches also reported a significant reduction in the required amount of training data to build detection models [93, 117].

3.2.3.3 Adaptability

Unlike misuse detection, anomaly detection is capable of detecting novel/unknown intrusions in its characteristics [26, 93]. However, due to the changes of the monitored object's behaviour over time, an anomaly intrusion detection system must support adaptive learning, in order to maintain its detection accuracy. The detection rule base is continuously updated using inductive learning from new training data, which keeps the detection model up-to-date [79]. This is one of the most effective approaches to make a detection model that can adapt to the changes of the object's behaviour [74, 81, 83].

3.2.3.4 Extensibility

Applying hierarchical architectures has been commonly used to support customization and extension [79, 82, 100, 142]. In these systems high-level detection modules combine evidences from multiple local detection modules, each of which is responsible for monitoring a set of specific objects. The decision making process can be either centralized or distributed, depending on the system design. However, distributed approaches achieve better extensibility [83, 142]. In another approach, Paxson's Bro [99] separates "mechanism from policy" to support customization and extension. Nevertheless, the extension of a site's security policy can only accommodate signatures of known intrusions.

3.2.4 Other issues of existing anomaly detection approaches

Apart from the main problems of existing anomaly detection approaches, as discussed in Section 3.1.2 and Section 3.1.3, such as excessive false alarms, lack of efficiency and lack of adaptability, there are still other unsolved issues that also need to be taken into consideration. They are as follows:

- There is not a good method for the selection of important parameters. In most anomaly detection approaches, many critical parameters have been chosen manually or empirically for each monitored object. For example, there is no formal method available to select crucial parameters, such as: the length of short sequences in the construction of a *normal database* [25, 26, 102, 134]; the number of states of hidden Markov models [102, 134]; and most importantly, the detection threshold, which is used to determine whether the object's current activity is normal or anomalous.

- Detection models are built from noisy data. The assumption of many anomaly detection approaches is that the input training data is clean and free from anomalies and intrusions. However, this assumption does not always hold up in practice. Eskin [21] first built a probability model from the original (noisy) training data, and then used the probability model to process each original data element to find anomalies. When an anomaly was found, it was removed from the original training data, and the probability model was re-calculated. This training method [21] reportedly gives good results provided that the amount of anomalies in the original training data is small, about 5%.

3.3 Summary

Unlike misuse detection, anomaly detection first constructs a model of normal behaviour of a monitored object, and then uses that model to evaluate the object's current behaviour to find possible intrusions. User behaviour, network activity and program behaviour are the most common objects to be monitored in computer and network systems in anomaly intrusion detection.

Introduced by Forrest *et al* [25], short sequences of system calls produced by a program during its execution have been extensively used in many program anomaly detection approaches. Forrest *et al*'s [25] normal database method, which uses short sequences of system calls to represent the

normal program behaviour, is simple and efficient to build. However, it produces a very low level of anomaly signals, which is likely to lead to a low detection rate and high false positive alarms.

Some program anomaly detection approaches, based on HMM modelling to characterize normal program behaviour, reported a higher overall detection accuracy than those based on other modelling methods such as normal database, data mining and frequency-based methods [42, 102, 134]. However, these HMM-based anomaly detection approaches suffer a serious efficiency problem because the HMM training costs are very high in terms of time and storage.

As discussed in Section 3.1.3, our major aims are to improve the detection accuracy, efficiency and adaptability. In chapters 5 and 7, we incorporate HMM modelling of normal program behaviour into a two-layer detection scheme and a fuzzy-based detection scheme to improve the detection accuracy. Chapter 6 discusses our solutions to the HMM training efficiency problem. In Chapter 8, we present the complete description of the proposed program anomaly detection model which also provides a mechanism to support adaptability.

Chapter 4

Hidden Markov Models

As indicated in Section 3.1.3, we have chosen the hidden Markov model (HMM) as the main modelling technique to characterize normal program behaviour. In this chapter, we give a brief introduction to hidden Markov models, and the HMM three basic problems and their solutions. These HMM problems and solutions will be extensively used in next chapters to construct and test HMM-based detection models. The information presented in this chapter is based on Dugad and Desai [19], Rabiner and Juang [103], and Rabiner [104].

4.1 Markov Models and Hidden Markov Models

Hidden Markov model (HMM) is an extension of the traditional Markov model [103, 104]. Unlike a Markov model's states that are observable, a hidden Markov model's states are not observable or hidden [104]. Only observations emitted from these hidden states can be observed [104].

4.1.1 Markov models

A traditional Markov process consists of a set of N distinct states, $\{S_1, S_2, ..., S_N\}$ and transitions among states. An example of a Markov process is shown in Figure 4.1, where the number of states $N = 3$. In this Markov process, states are fully connected, and transitions are allowed from one state, to any other states, and to itself. The transitions among states are determined by the transition probability matrix, $A = \{a_{ij}\},\ i, j = 1, .., N$. For a discrete, first-order and

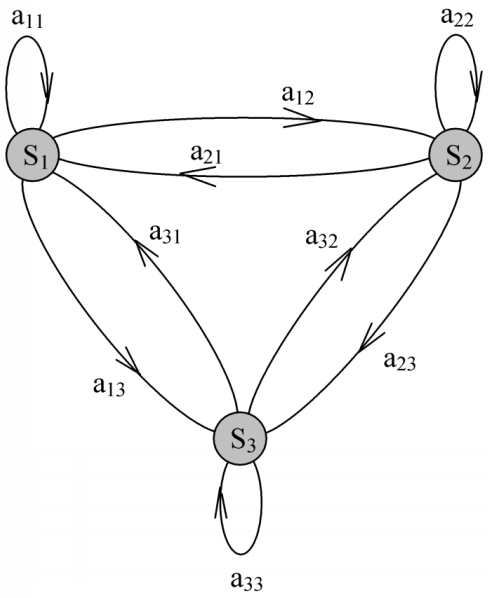

Figure 4.1: A Markov process with 3 states $\{S_1, S_2, S_3\}$; All states are fully connected.

stationary Markov process, the state transition probabilities $\{a_{ij}\}$ are defined as:

$$a_{ij} = P(q_t = S_j, q_{t-1} = S_i), \quad 1 \leq i, j \leq N$$

with the standard stochastic constraints:

$$a_{ij} \geq 0, \quad 1 \leq i, j \leq N$$

$$\sum_{j=1}^{N} a_{ij} = 1, \quad i = 1, N$$

In a first-order Markov process, the next state is dependent only on the current state, whereas in an n-order Markov process, the next state depends on previous n states. In a stationary Markov process, the state transition probability distribution is independent of time. The basic Markov process is called an observable Markov model because the output of the process is a set of states at each instant of time, and each state corresponds to an observable event [104]. The restriction of Markov models, that a state is correspondent to one observable event, limits their application to many real-world problems. Therefore, hidden Markov models have been introduced to overcome the Markov models' limitation. A hidden Markov model's states are not observable, and an observation is probabilistic function of a state.

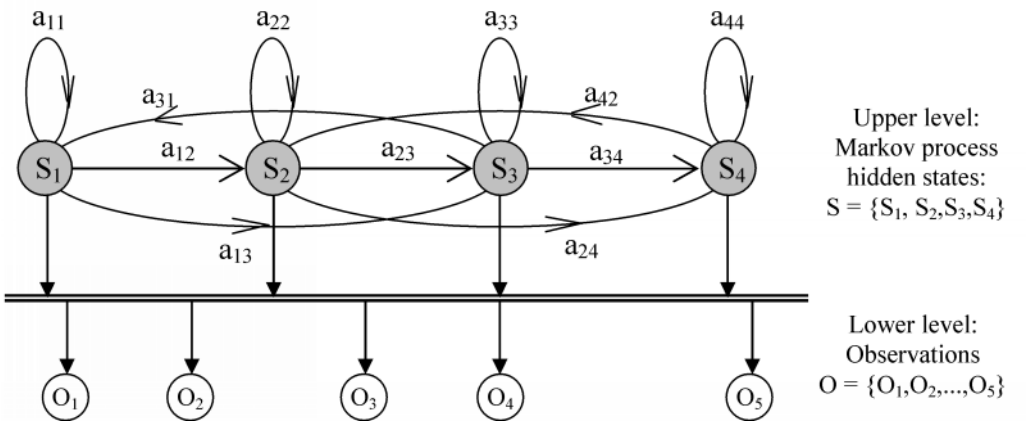

Figure 4.2: A 4-state HMM $\{S_1, S_2, S_3, S_4\}$ with selected state transitions, and emitted observations $\{O_1, O_2, O_3, O_4, O_5\}$

4.1.2 Hidden Markov models and their notations

Hidden Markov Model (HMM) is a doubly embedded stochastic process with two hierarchy levels. The upper level is a Markov process, in which the states are not observable. The lower level consists of observations, each of which is a probabilistic function of an upper-level Markov state. Different Markov states will have different observation functions. Figure 4.2 illustrates a hidden Markov model with 4 states: $\{S_1, S_2, S_3, S_4\}$, some selected transitions, and observations: $\{O_1, O_2, O_3, O_4, O_5\}$, which are emitted from states.

HMM is a powerful data modelling tool [103, 104], and it has been widely used in biology for protein sequence analysis [47, 58, 59, 69, 115], and in speech recognition to build signal models [16, 28, 29, 30, 72, 104]. The application of hidden Markov models in anomaly intrusion detection, to construct models of the normal behaviour of monitored objects, has also reported promising results [42, 43, 102, 134].

Hidden Markov models can be classified roughly into two types based on the observation density: discrete and continuous HMMs. For discrete HMMs, observations generated from distinct states are finite in number, while for continuous HMMs observations are infinite and continuous. As program behaviour can be described by finite states, and there is only a finite number of system calls, we only consider discrete HMMs hereafter.

In order to represent the cross relations among hidden states that emit system calls, we select ergodic hidden Makov models for our detection model, in which hidden states are fully connected

and transitions are allowed from any state to any other states [104]. We define notations for our hidden Markov models to be used throughout this chapter as follows:

N : Number of hidden states in the model.

M : Number of distinct observation symbols.

T : Length of observation sequence, that is the number of symbols observed.

S : Set of hidden states, $S = \{S_1, S_2, ..., S_N\}$.

V : Set of possible observation symbols, $V = \{v_1, v_2, ..., v_M\}$.

π : Initial state distribution, $\pi = \{\pi_i\}$, where $\pi_i = P(q_1 = S_i),\ 1 \leq i \leq N$, is the probability of being in state S_i at $t = 1$.

A : State transition probability matrix, $A = \{a_{ij}\}$, where $a_{ij} = P(q_{t+1} = S_j, q_t = S_i)$, $1 \leq i, j \leq N$, is the probability of being in state S_j at time $t + 1$, given that the model was in state S_i at time t.

B : Observation probability distribution, $B = \{b_j(k)\}$, where $b_j(k) = P(v_k \text{ at } t | q_t = S_j)$, $1 \leq j \leq N,\ 1 \leq k \leq M$, is the probability of observing symbol v_k at time t, given that the model is in state S_j.

Q : Sequence of hidden states, $Q = \{q_1, q_2, ..., q_t, ..., q_T\}$, where q_t is the model's state at time t.

O : Sequence of observations, $O = \{O_1, O_2, ..., O_t, ..., O_T\}$, where $O_t,\ 1 \leq t \leq T$, denotes observation symbol observed at time t.

λ : Entire HMM model, $\lambda = (A, B, \pi)$ is used as compact notation to denote an HMM.

$P(O|\lambda)$: The probability of the occurrence of observation sequence O, given the HMM λ.

$P(O, Q|\lambda)$: The joint probability of the occurrence of the observation sequence O for the state sequence Q, given the HMM λ.

HMM assumptions:

- The first-order Markov assumption: The next state is dependent only upon the current state.

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i) = P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, ...) \tag{4.1}$$

- The stationarity assumption: State transition probabilities are independent of the actual time at which a transition takes place.

$$P(q_{t1} = S_j | q_{t1-1} = S_i) = P(q_{t2} = S_j | q_{t2-1} = S_i) \tag{4.2}$$

Standard stochastic constraints to HMM parameters:

- Initial probability π:

$$\pi_i \geq 0,\ 1 \leq i \leq N \tag{4.3}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{4.4}$$

- Transition probability matrix A:

$$a_{ij} \geq 0,\ 1 \leq i, j \leq N \tag{4.5}$$

$$\sum_{j=1}^{N} a_{ij} = 1,\ i = 1, N \tag{4.6}$$

- Observation probability distribution B:

$$b_j(k) \geq 0,\ 1 \leq j \leq N,\ 1 \leq k \leq M \tag{4.7}$$

$$\sum_{k=1}^{M} b_j(k) = 1,\ j = 1, N \tag{4.8}$$

Basic conditional probability rules:

- Conditional probability:

$$P(A|B) = \frac{P(AB)}{P(B)} \tag{4.9}$$

- Joint probability:

$$P(A, B) = P(A|B)P(B) \tag{4.10}$$

- The Bayes' rule:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \quad , P(B) > 0 \tag{4.11}$$

- The Markov chain rule:

$$\begin{aligned} P(q_1, q_2, ..., q_T) &= P(q_T|q_1, q_2, ..., q_{T-1})P(q_1, q_2, ..., q_{T-1}) \quad (4.12) \\ &= P(q_T|, q_{T-1})P(q_1, q_2, ..., q_{T-1}) \\ &= P(q_T|q_{T-1})P(q_{T-1}|q_{T-2})P(q_1, q_2, ..., q_{T-2}) \\ &= P(q_T|q_{T-1})P(q_{T-1}|q_{T-2})...P(q_2|q_1)P(q_1) \quad (4.13) \end{aligned}$$

4.2 The Three HMM Problems and Their Solutions

There are three basic problems for most HMM applications:

Problem 1 : Given the HMM $\lambda = (A, B, \pi)$, how to compute $P(O|\lambda)$, the probability of the occurrence of the observation sequence $O = \{O_1, O_2, ..., O_T\}$.

Problem 2 : Given observation sequence $O = \{O_1, O_2, ..., O_T\}$ and the HMM $\lambda = (A, B, \pi)$, how to choose $Q = \{q_1, q_2, ..., q_T\}$, a corresponding state sequence so that the joint probability $P(O, Q|\lambda)$ is maximized.

Problem 3 : Given the observation sequence O, how to adjust the HMM model parameters $\lambda = (A, B, \pi)$ so that $P(O|\lambda)$ is maximized.

Problem 1 can be viewed as the evaluation problem in which the question is, given a model and an observation sequence, how do we efficiently compute the probability that the observation sequence was produced by the model. Problem 2 is a decoding problem in which we try to find the optimal sequence of hidden states given a model and an observation sequence. Problem 3 is a learning problem where we attempt to find a model that best suits the input sequence of observations. In the next section, we briefly discuss the solutions to Problems 1 and 3 as they are used in our HMM-based intrusion detection model. Problem 2 is not discussed here because it is not relevant to our HMM application.

4.2.1 Solution to Problem 1: forward and backward procedures

HMM Problem 1 is to efficiently compute $P(O|\lambda)$, the probability that the sequence of observations O was generated by the model λ. $P(O|\lambda)$ may be computed directly as follows:

- Let $Q = \{q_1, q_2, ..., q_T\}$ be a state sequence, where q_1 is the initial state.
- The joint probability of the observation sequence O for the state sequence Q is

$$P(O, Q|\lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) \tag{4.14}$$

$$= b_{q_1}(O_1)b_{q_2}(O_2)...b_{q_T}(O_T) \tag{4.15}$$

  with assumption that observations are independent.

- The probability of a state sequence Q is

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} ... a_{q_{T-1} q_T} \tag{4.16}$$

- Then we have

$$P(O|\lambda) = \sum_{Q} P(O, Q|\lambda)P(Q|\lambda) \tag{4.17}$$

$$= \sum_{Q} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) a_{q_2 q_3} ... a_{q_{T-1} q_T} b_{q_T}(O_T) \tag{4.18}$$

The direct computation of $P(O|\lambda)$, using the equation (4.18), involves in the order of $O(2N^T T)$ calculations that is computationally infeasible, even for small values of N and T [104]. For example, for $N = 5$ (states), and $T = 100$ (observations), the number of compuations is $2 * 5^{100} * 100 \approx 10^{72}$ [104]. Therefore, a more efficient procedure to the Problem 1 is required; such a procedure exists and is called forward-backward procedure. More precisely, we need only the forward part of the forward-backward procedure to solve the Problem 1. However, we also discuss the backward part here because it will be used to solve Problem 3.

Forward procedure to compute $P(O|\lambda)$ [19, 104]:

- Define forward variable $\alpha_t(i)$ as:

$$\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i|\lambda) \tag{4.19}$$

- $\alpha_t(i)$ is the probability of observing the partial sequence $\{O_1, O_2, ..., O_t\}$ and the state i at time t, given model λ. $\alpha_t(i)$ can be computed inductively as follows:

  1. Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{4.20}$$

  2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \tag{4.21}$$
$$1 \leq t \leq T-1, \quad 1 \leq j \leq N$$

  3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.22}$$

Backward procedure to compute $P(O|\lambda)$ [19, 104]:

- Define backward variable $\beta_t(i)$ as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T | q_t = S_i, \lambda) \tag{4.23}$$

- $\beta_t(i)$ is the probability of observing the partial sequence $\{O_{t+1}, O_{t+2}, ..., O_T\}$ given the state i at time t and the model λ. $\beta_t(i)$ can be computed inductively as follows:

  1. Initialization:

$$\beta_T(i) = 1 \tag{4.24}$$

  2. Induction:

$$\beta_t(j) = \sum_{i=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \tag{4.25}$$
$$t = T-1, T-2, ..., 1, \quad 1 \leq j \leq N$$

  3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1) \beta_1 \tag{4.26}$$

The complexity of the forward and backward procedures is the order of $O(N^2T)$ [104]. These computation methods of $P(O|\lambda)$ are much more efficient than the direct calculation in equation (4.18). For example, when $N = 5$, and $T = 100$, we need about 3000 computations for the forward method, compared to 10^{72} computations, which is required by the direct method. The proof of the forward and backward procedures are not discussed in this thesis due to space limitations. This information can be found in [19, 103, 104].

4.2.2 Solution to Problem 2: Viterbi algorithm

As stated at the beginning of Section 4.2, Problem 2 is not relevant to our work, and is not discussed. Additional materials are available [19, 103, 104].

4.2.3 Solution to Problem 3: Baum-Welch algorithm

The Baum-Welch algorithm is an Expectation-Maximization (EM) iterative procedure, which is used to estimate HMM parameters $\lambda = (A, B, \pi)$ to maximize the probability of the observation sequence O. The following is a brief description of the Baum-Welch re-estimation procedure of HMM parameters.

Preliminaries:

- Define $\xi_t(i, j)$ be the probability of being in state i at time t and in state j at time $t + 1$, given observation sequence $O = \{O_1, O_2, ..., O_T\}$ and $\lambda = (A, B, \pi)$:

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{4.27}$$

$$= \frac{P(q_t = S_i, q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} \tag{4.28}$$

- Considering the numerator of equation (4.28), we have:

$$P(q_t = S_i, q_{t+1} = S_j, O|\lambda) = P(q_t = S_i, O_1, O_2, ..., O_t, O_{t+1}, ..., O_T, q_{t+1} = S_j|\lambda) \tag{4.29}$$

$$= P(q_t = S_i, O_1, O_2, ..., O_t|\lambda)P(O_{t+1}, ..., O_T, q_{t+1} = S_j|\lambda)$$

$$= \alpha_t(i)P(O_{t+1}, ..., O_T, q_{t+1} = S_j|\lambda) \tag{4.30}$$

- Analyzing the second factor of equation (4.30), we get:

$$P(O_{t+1},...,O_T,q_{t+1}=S_j|\lambda) = P(q_{t+1}=S_j,O_{t+1}|\lambda)P(O_{t+1},...,O_T|q_{t+1}=S_j,\lambda) \quad (4.31)$$

$$= P(q_{t+1}=S_j,O_{t+1}|\lambda)\beta_{t+1}(j)$$

$$= a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \quad (4.32)$$

- Combining (4.28), (4.30), and (4.32), we get:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \quad (4.33)$$

$$= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \quad (4.34)$$

- Define $\gamma_t(i)$ as the probability of being in state i at time t, given observation sequence $O=\{O_1,O_2,...,O_T\}$ and $\lambda=(A,B,\pi)$:

$$\gamma_t(i) = P(q_t=S_i|O,\lambda) \quad (4.35)$$

$$= \frac{P(q_t=S_i,O|\lambda)}{P(O|\lambda)}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

$$= \sum_{j=1}^{N}\xi_t(i,j) \quad (4.36)$$

- By definition of $\xi_t(i,j)$ and $\gamma_t(i)$, we have:

$\sum_{t=1}^{T-1}\gamma_t(i)$ = Expected number of transitions from state i

$\sum_{t=1}^{T-1}\xi_t(i,j)$ = Expected number of transitions from state i to state j

Update rules of model parameters:

$$\bar{\pi}_i = \text{expected number of times in state } i \text{ at time } t=1$$

$$\bar{\pi}_i = \gamma_1(i) \quad (4.37)$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \quad (4.38)$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observed symbol } v_k}{\text{expected number of times in state } j}$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ O_t=k}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.39}$$

Baum-Welch algorithm training procedure:

1. Let initial model be λ_0.
2. Estimate new model λ based on λ_0 and observation sequence O using equations (4.37, 4.38, 4.39).
3. Compute $P(O|\lambda_0)$ and $P(O|\lambda)$; If $P(O|\lambda) - P(O|\lambda_0) < \Delta$ go to step 5.
4. Else set $\lambda_0 \leftarrow \lambda$, and go to step 2.
5. Stop.

where Δ is the convergence threshold that the HMM training stops when $P(O|\lambda)$ is stable.

Complexity of Baum-Welch algorithm:

For an HMM with N hidden states, M distinct observation symbols, and an observation sequence O of length T, the time and space complexities of Baum-Welch algorithm are $O(N(1 + T(M + N)))$ and $O(N(N + M + TN))$ respectively [77]. Since the length T of the observation sequence is usually much larger than the number of states N, and the number of distinct observation symbols M ($T >> N$, $T >> M$), we can consider the time and space complexities of Baum-Welch algorithm are $O(N^2T)$ [77, 104].

4.2.4 Scaling in probability computation and re-estimation of HMM parameters

Scaling is a technique which is used in the implementation of the Baum-Welch re-estimation of HMM parameters to prevent the value underflow problem. This is where the computation values head to zero exponentially and are out of the precision range of numeric representation of computers. For example, considering the definition of $\alpha_t(i)$ of the equation (4.19) in the forward

procedure to compute $P(O|\lambda)$. It can be seen that $\alpha_t(i)$ consists of the sum of a large number of terms, each of which has the form

$$\left(\prod_{s=1}^{t-1} a_{q_s q_{s+1}} \prod_{s=1}^{t} b_{q_s}(O_s)\right) \tag{4.40}$$

with $q_t = S_i$. Since each term a and b is generally significant less than 1, it can be seen that each term of $\alpha_t(i)$ starts to head exponentially to zero as t starts to get big [104]. For our HMM-based intrusion detection, t (the number of data items in the training set) is usually very large. Therefore, incorporating a scaling procedure is necessary to compute $\alpha_t(i)$ and $P(O|\lambda)$.

Instead of computing $P(O|\lambda)$, we compute $log(P(O|\lambda))$ in our implementation using forward procedure with scaling. In this procedure, a scaling coefficient c_t is used and it is computed as:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{4.41}$$

The forward procedure with scaling is describered as follows [104]:

1. Initialization:

$$\hat{\alpha}_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N \tag{4.42}$$

$$c_1 = \frac{1}{\sum_{i=1}^{N} \hat{\alpha}_1(i)} \tag{4.43}$$

$$\hat{\alpha}_1(i) \leftarrow c_1 \hat{\alpha}_1(i) \tag{4.44}$$

2. Induction:

$$\hat{\alpha}_{t+1}(j) = \left[\sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij}\right] b_j(O_{t+1}), \tag{4.45}$$

$$1 \le t \le T-1, \quad 1 \le j \le N$$

$$c_{t+1} = \frac{1}{\sum_{i=1}^{N} \hat{\alpha}_{t+1}(i)} \tag{4.46}$$

$$\hat{\alpha}_{t+1}(i) \leftarrow c_{t+1} \hat{\alpha}_{t+1}(i) \tag{4.47}$$

3. Termination:

$$log(P(O|\lambda)) = -\sum_{t=1}^{T} log(c_t) \tag{4.48}$$

4.3 Summary

Hidden Markov model is a doubly embedded stochastic process with two hierarchy levels including the upper level of states and the lower level of observations. Unlike the observable states of the traditional Markov model, the states of the upper level of the hidden Markov model are not observable. Each observation of the lower level is a probabilistic function of an upper-level Markov state. Due to the dual-level structure, hidden Markov models overcome the restriction of Markov models, where a state is correspondent to one observable event.

There exist three basic problems for most HMM applications. However, we focused on the solutions to HMM Problem 1, the evaluation problem and Problem 3, the training problem, because they will be used extensively in our work. Although the convergence of the Baum-Welch algorithm, the solution to HMM Problem 3, has been proven mathematically, the algorithm has been found very expensive in terms of time and space. This makes the HMM-based anomaly detection approaches less efficient for practical use. Our solution to the efficiency problem of HMM training is presented in chapter 6.

Chapter 5

A Basic Detection Model Based on HMM Modelling

In this chapter, a basic model, based on HMM modelling for program anomaly detection is presented, which will be used as a starting point for our approach. First, the two-stage development process of the basic model is given in Section 5.1. Then, a two-layer detection scheme is proposed to further improve the detection performance of the basic model in Section 5.2.1.

5.1 The Proposed Basic Model for Program Anomaly Detection Based on HMM Modelling

HMM modelling of normal program behaviour is the process of building a HMM model from training data. The training data is the traces of system calls, which are collected in normal runs of the program. In this section, we propose a basic model, in which HMM modelling is used to characterize normal program behaviour, using traces of system calls. The collection and pre-processing procedures of system call traces are also given. In addition, we present some preliminary results of using HMM models to generate anomaly signals for abnormal traces generated by some known intrusions.

5.1.1 Collecting system call data

Generally, application programs use system calls (also called API calls in some systems), or software interrupts as the mechanism, to request services and resources from the underlying

operating system. There are hundreds of system calls supported in present day operating systems. However, the number of distinct system calls used by a specific program is usually a small portion of the supported system calls. In Unix and Linux systems, popular system calls are *close, execve, fork, kill, open, read*, and *write* [135].

There have been several methods that can be used to collect traces of system calls produced by a program during its execution [26, 93]. One can use Basic Security Modules (BSM) supported in many Unix systems to get audit information of running programs. Another popular auditing and debugging tool, *strace*, is freely available in almost all Linux systems, and it can also be used to capture the program's traces of system calls and related information. On the other hand, Sekar et al integrated the operating system's kernel autditing mechanism into their monitoring engine, to capture system calls, as well as other related information, such as the program counter and stack information when each system call was made [117]. In our work, we use *strace* to collect system call data produced by programs running on RedHat Linux systems. In addition, we use data traces of system calls of some common Unix programs from the "Computer Immune Systems Project" at the University of New Mexico (UNM), USA [132] in our experiments.

5.1.1.1 Collecting system call data using *strace*

Strace is an open-source debugging tool which can trace system calls made by another process, or program [4]. When attached to a running process, *strace* can capture traces of system calls made by the process to the operating system kernel. In addition to the name and parameters of each system call, other system call information, such as timestamp, return value, and elapsed time are also available. Table 5.1 shows a sample of a raw trace of system calls of *sendmail* process, which was captured by *strace* on a RedHat Linux 9.0 system. In addition, *strace* can also follow and trace system calls of child processes, created by *fork* system calls, which are issued by the parent process. This is important because server processes, such as *sendmail* usually create child processes to handle new connections.

5.1.1.2 System call data from the UNM project

The system call data sets from the "Computer Immune Systems Project" at the University of New Mexico consist of several traces of system calls of some common Unix server applications, such as *sendmail, FTP, named, inetd* and *lpr* programs [132]. The procedures of generating

Table 5.1: A sample of a raw trace of system calls produced by *sendmail* process, captured by *strace*. Each line shows one system call, the first part is the timestamp, followed by the system call name, parameters and return value, and the elapsed time.

```
11:57:38 rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0 <0.000040>
11:57:38 time([1128563858])             = 1128563858 <0.000014>
11:57:38 open("/proc/loadavg", O_RDONLY) = 6 <0.000065>
11:57:38 fstat64(6, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000015>
11:57:38 read(6, "0.00 0.00 0.00 2/41 10368\n", 1024) = 26 <0.000024>
11:57:38 close(6)                       = 0 <0.000029>
11:57:38 time([1128563858])             = 1128563858 <0.000014>
11:57:38 rt_sigprocmask(SIG_UNBLOCK, [ALRM], [ALRM], 8) = 0 <0.000015>
11:57:38 select(4, [3], NULL, NULL, {5, 0}) = 0 (Timeout) <4.998346>
11:57:43 rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0 <0.000040>
11:57:43 time([1128563863])             = 1128563863 <0.000013>
11:57:43 open("/proc/loadavg", O_RDONLY) = 6 <0.000063>
11:57:43 fstat64(6, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000015>
11:57:43 read(6, "0.00 0.00 0.00 2/41 10368\n", 1024) = 26 <0.000024>
11:57:43 close(6)                       = 0 <0.000028>
11:57:43 time([1128563863])             = 1128563863 <0.000014>
11:57:43 rt_sigprocmask(SIG_UNBLOCK, [ALRM], [ALRM], 8) = 0 <0.000014>
11:57:43 select(4, [3], NULL, NULL, {5, 0}) = 0 (Timeout) <4.998376>
11:57:48 rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0 <0.000039>
11:57:48 time([1128563868])             = 1128563868 <0.000014>
11:57:48 open("/proc/loadavg", O_RDONLY) = 6 <0.000063>
11:57:48 fstat64(6, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0 <0.000015>
11:57:48 read(6, "0.00 0.00 0.00 2/41 10368\n", 1024) = 26 <0.000025>
11:57:48 close(6)                       = 0 <0.000029>
11:57:48 time([1128563868])             = 1128563868 <0.000014>
11:57:48 rt_sigprocmask(SIG_UNBLOCK, [ALRM], [ALRM], 8) = 0 <0.000015>
```

Table 5.2: A sample of a *sendmail* trace of system calls from the "Computer Immune Systems Project" of University of New Mexico [132]. Each pair of values in first two lines of this table forms a record in system call trace files.

Process identification (PID)	8840	8840	8840	...	8844	8844
System call number	4	2	66	...	19	105
Corresponding system call	write	fork	sstk	...	lseek	sigvec

and collecting these traces are described in [25, 26]. Each trace is a list of system calls produced by a program's running processes (including parent and child processes) during its execution. The first two lines of Table 5.2 shows the sample of a trace file, in which each line is a pair of the process identification (PID) and the system call number. The mapping between system call numbers and actual system call names is provided in a separate file. For example, in the system call mapping file "calls.txt.sun" [132], number 4 is corresponding to the system call "write", number 2 represents system call "fork", and number 19 is "lseek".

For each program, the collected system call data usually consist of two types of data traces: (i) *normal traces* - traces collected during the program's normal working condition, which are used to construct the detection model in the training stage, and (ii) *abnormal traces* - traces come from the program's abnormal runs that were produced by real intrusions, or generated by intrusion tools for some known intrusions, which are used in the model's testing stage.

The normal traces of system calls in UMN project were collected in two environments: (i) *synthetic* environment and (ii) *live* environment. In synthetic environment, a program's normal traces of system calls were collected from its execution in simulated conditions which were specifically designed to capture the program's behaviour in a number of scenarios. In live environment, on the other hand, normal traces of system calls were collected directly from the program's runs in real and normal working conditions (without anomalies and intrusions).

5.1.2 The proposed basic detection model using HMM modelling

Figure 5.1 shows the two stages of the proposed basic detection model using HMM modelling: (i) building a HMM model from the traces of system calls and (ii) testing input traces of system calls for possible intrusions, using the built HMM model. In HMM building stage, the raw traces

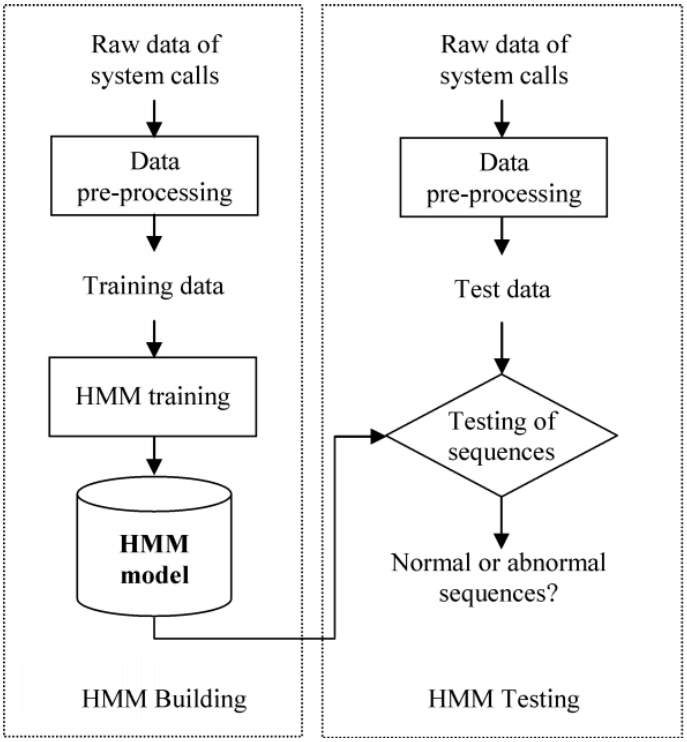

Figure 5.1: The proposed basic detection model using HMM modelling: HMM building and HMM testing

of system calls are processed and transformed to HMM observation sequences which are then used to train the HMM model. In HMM testing stage, a similar pre-processing procedure is applied to the raw traces of system calls in order to get the HMM test data, and then the HMM model is used to evaluate each short sequence of system calls for possible intrusions.

5.1.2.1 Pre-processing raw traces of system calls

Raw traces of system calls collected by process auditing tools, such as *strace*, must be processed and transformed into HMM observation sequences, before they can be used in the HMM model construction stage and the testing stage. First, we extract only plain system calls (system calls without parameters and other related information) from raw traces, and then traces of plain system calls are transformed into HMM observation sequences, using HMM notations defined in Section 4.1.2. For data sets from [132], the transformation of traces of system calls to HMM observation sequences can be applied directly, because the pre-processing of raw system calls has already been done.

In the transformation from system calls to HMM symbols, the first step is to determine the number of distinct HMM symbols M and the set of possible HMM observation symbols, $V = \{v_1, v_2, ..., v_M\}$. Since we consider each system call in raw traces correspoding to one symbol in HMM observation sequences, the number of distinct HMM symbols M can be chosen as the number of system calls supported by the operating system and the HMM symbol set can be the full set of system calls supported by the operating system. However, as the actual number of distinct system calls used by a program, is usually a portion of the full set of supported system calls, it is more practical to select M as the actual number of distinct system calls used, and V as the set of actual system calls used. This selection leads to a smaller number of distinct HMM symbols, which helps reduce the complexity of HMM training as well as testing for intrusions.

5.1.2.2 HMM training using traces of system calls

The purpose of HMM training is to estimate HMM parameters $\lambda(A, B, \pi)$ from training data which are traces of system calls in this case. After the number of distinct HMM symbols M is chosen, the next step in HMM training is to select the number of HMM hidden states N. It is suggested that number of hidden states N can be chosen based on the number of unique system calls used by the program [134]. In our HMM training, we choose the number of hidden states N also equal to the number of unique system calls used by the program. For example, we select $N = M = 35$ for an HMM model for *inetd* program, using data sets collected in live environment[1] [132], because *inetd* only actually used 35 unique system calls in its normal traces.

Generally, an HMM model can be trained from the training data of observation sequences, using the traditional Baum-Welch algorithm [19, 104], as discussed in Section 4.2. The Baum-Welch algorithm requires several passes through the training data to infer the destination model. As a special case of *Expectation - Maximization* (EM) algorithms, the Baum-Welch HMM training scheme consists of two steps at each pass through the training data: (i) *E-step* - values of the probability auxiliary function over the whole set of training data are computed, and (ii) *M-step* - update of HMM parameters $\lambda(A, B, \pi)$, using the values computed in E-step. HMM training using Baum-Welch algorithm can be seen as batch training, because the full set of training data is required in E-step.

However, HMM training using Baum-Welch algorithm was found very expensive in terms

[1] Also called real working environment.

of time and storage, especially for large-size HMM models and long observation sequences [77, 104, 134], although its convergence has been proven. The high cost of HMM training is likely to limit the HMM application in anomaly intrusion detection. In order to reduce the requirement of system resources and to speed up HMM training, we introduce an efficient HMM incremental training scheme with optimal initialization, which is discussed in details in Section 6.1.

5.1.2.3 Testing traces of system calls and measuring anomaly signals

In testing stage, the constructed HMM model is used to evaluate test traces of system calls to find possible intrusions. This task can be done using a number of ways. A standard way is to compute the probability that the HMM model generates a test trace. If the HMM model is a good model of normal program behaviour, normal traces should require only likely probabilities, while intrusive traces should require unlikely probabilities [134]. However, this method is sensitive to the length of the test trace. In another approach, Warrender et al considered the HMM model as a non-deterministic finite automaton, and tracked the state transition and symbol probabilities required to produce each system call in the test trace [134]. In our approach, we use the HMM model to classify fixed-length short sequences of system calls, which are formed from the test trace. In addition, we measure anomaly signals, based on temporally local regions of these short sequences. To start with, first, we empirically choose parameters for the testing stage:

- *Short sequence* is a sequence of consecutive system calls in the test trace. Short sequences have a pre-selected length of k system calls.
- *Region* is a series of non-overlapped consecutive short sequences in the test trace. Regions have a pre-selected length of r short sequences.
- *Short sequence probability threshold* $\hat{P}$ is the minimum probability that the HMM model produces a normal short sequence. The threshold $\hat{P}$ is used to determine if a short sequence is *normal* or *abnormal*. If a short sequence probability, $P \geq \hat{P}$, it is considered normal, otherwise it is abnormal.
- *Region score threshold* $\hat{A}$

  A region's anomaly score A is calculated as:

$$A = \frac{\text{number of abnormal short sequences in the region } c}{\text{length of region } r} \tag{5.1}$$

Region score threshold $\hat{A}$ is the minimum value of the anomaly score A, that indicates *alarm* status.

Next, we form test short sequences by sliding a window of k system calls, through the test trace, one system call at a time, and recording system calls within the sliding window as one short sequence at each move. During this process, we form one test region from every r consecutive short sequences. Non-overlapped regions are selected because a single abnormal short sequence may generate anomalies in many overlapping regions. The procedure for classifying each short sequence of system calls and measuring anomaly signal for a test region is as follows:

Input : HMM model $\lambda(A, B, \pi)$, and a test region of r consecutive short sequences of system calls, $O_{(R)} = \{O_{(1)}, O_{(2)}, ..., O_{(l)}, ..., O_{(r)}\}$. Each short sequence has k system calls.

Ouput : Status of a test region (normal or alarm). A region with alarm status can be used to trigger an intrusion alarm.

1. Reset the abnormal region counter, $c \leftarrow 0$.
2. For each consecutive short sequence $O_{(l)}$ in the test region, $1 \leq l \leq r$:
   (a) Compute the observation probability $P = P(O_{(l)}|\lambda)$ that the HMM model λ generates the short sequence $O_{(l)}$.
   (b) Compare probability P against the probability threshold $\hat{P}$;
   If $P < \hat{P}$, increase the abnormal region counter, $c \leftarrow c + 1$.
3. Compute the anomaly score for the region, $A = \frac{c}{r}$.
4. Compare the region anomaly score A against the region score threshold $\hat{A}$;
   If $A \geq \hat{A}$, set the region status to *alarm*; otherwise set the region status to *normal*.

Because the anomaly score is computed based on fixed-length temporal regions, this testing method is independent of the length of the test trace, and therefore it is suitable for online detection.

5.1.3 Experimental results

In order to evaluate the capability of HMM modelling of normal program behaviour in the basic detection model, we construct an HMM model for *sendmail* program and another HMM model for *inetd* program from their normal traces of system calls, and then use these HMM models to generate anomaly signals for their test traces.

5.1.3.1 Data sets

We use the system call traces of *sendmail* and *inetd* programs, which were collected in a live environment (also called real working environment), as given in [132] for this experiment. The data sets include:

- *Normal traces* are traces collected during the program's normal activities. These traces are used to construct HMM models.
  - The *sendmail* data include system call traces of *sendmail* daemon process and its child processes. There are 13,726 data traces with the total of 15,631,952 system calls.
  - The *inetd* data include a single trace with 541 system calls.
- *Abnormal traces* are traces that come from a program's abnormal runs produced by known intrusions. Abnormal traces include:
  - 105 traces generated by a denial-of-service attack (DoS) on *sendmail* program.
  - 1 trace generated by a DoS attack on *inetd* program.

5.1.3.2 Results

In the HMM building stage, the first task is to select the HMM size. As discussed in Section 5.1.2, the number of hidden states N and the number of distinct symbols M are chosen as the number of distinct system calls used in the program's normal traces. For this *sendmail* data set, we use the first 1,000,000 system calls of its normal traces to construct an HMM model with $N = M = 23$. For the *inetd* data set, we use the only available normal trace to construct an HMM model with $N = M = 35$.

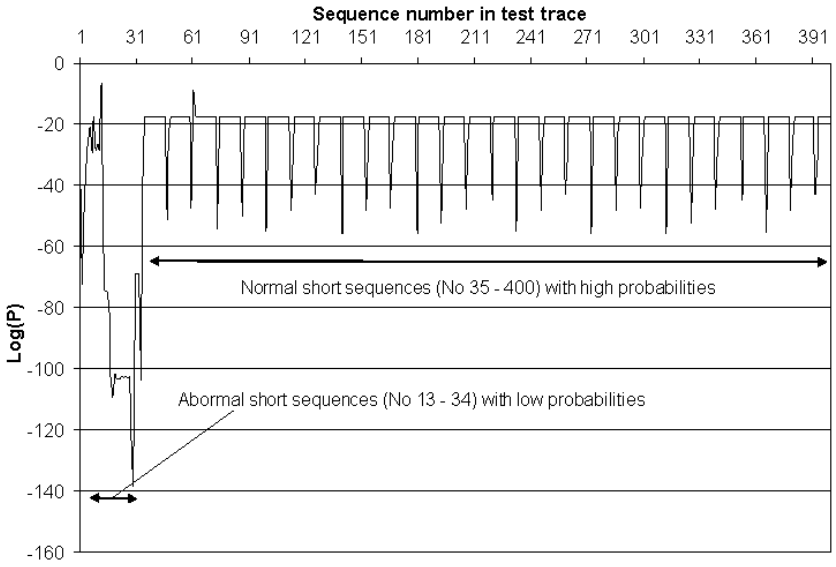

Figure 5.2: Anomaly signal (Log(P)) generated by *inetd* HMM model for an *inetd* DoS attack trace. Length of short sequences $k = 20$ system calls.

In the HMM testing stage, we compute log(P) for each fixed-length short sequence of system calls in test traces, where P is the probabilty that the HMM model produces a short sequence. log(P) is considered as the anomaly signal, that the model generates for the test traces. Normal short sequences should be generated by the HMM model with likely probabilites, while abnormal short sequences should be generated with unlikely probabilites.

Figure 5.2 shows the anomaly signal in the form of log(P), generated by the *inetd* HMM model for the test trace of system calls of a DoS attack on *inetd*. The test trace consists of 400 short sequences which are created using the sliding window method with length of $k = 20$ system calls. It can be seen that, the *inetd* model generates high probabilities ($log(P) \geq -40.0$) for short sequences from number 35 to the end of the trace, although there are some periodic irregularities. These short sequences are considered normal. On the other hand, short sequences from number 13 to 34 associated with unusual low probabilities ($log(P) < -60.0$) are considered abnormal. The discovery of these consecutive abnormal short sequences clearly indicates that the occurrence of DoS intrusion on *inetd* is detected. After the DoS attack period, the periodic pattern of $log(P)$ of short sequences from number 35 to the end of the trace represents the normal or recovered pattern of the system calls generated by repeated normal activities of the *inetd* program.

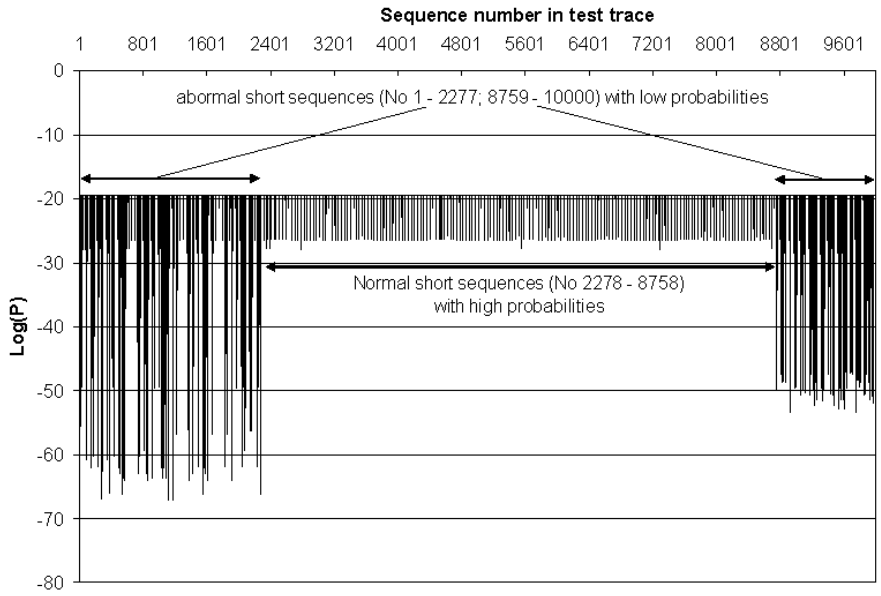

Figure 5.3: Anomaly signal (Log(P)) generated by *sendmail* HMM model for *sendmail* DoS attack traces. Length of short sequences $k = 20$ system calls.

Similarly, Figure 5.3 shows the log(P) anomaly signal generated by the *sendmail* HMM model for traces of system calls of a DoS attack on sendmail. 10,000 short sequences are formed from the *sendmail* test traces using the same method with the same sequence length, as those used in *inetd* test. It is noted that the *sendmail* model produces high probabilities ($log(P) \geq -30.0$) for short sequences from number 2278 to 8758. These sequences are considered normal. Mixed levels of high and low probabilities can be seen in two large groups of short sequences, from number 1 to 2277 and from 8759 to the end of test traces. However, the high density of short sequences that are associated with unlikely probabilities ($log(P) < -45.0$) indicates that the occurrence of DoS intrusion on sendmail is detected.

5.2 An Enhancement of the Basic Model: A Two-layer Detection Scheme

In Section 5.1.3, we conducted a simple experiment on the basic detection model, in which HMM models were used to generate anomaly signals for two abnormal traces of *sendmail* and *inetd* programs. Due to the limitation of the incomplete training data, the basic detection model

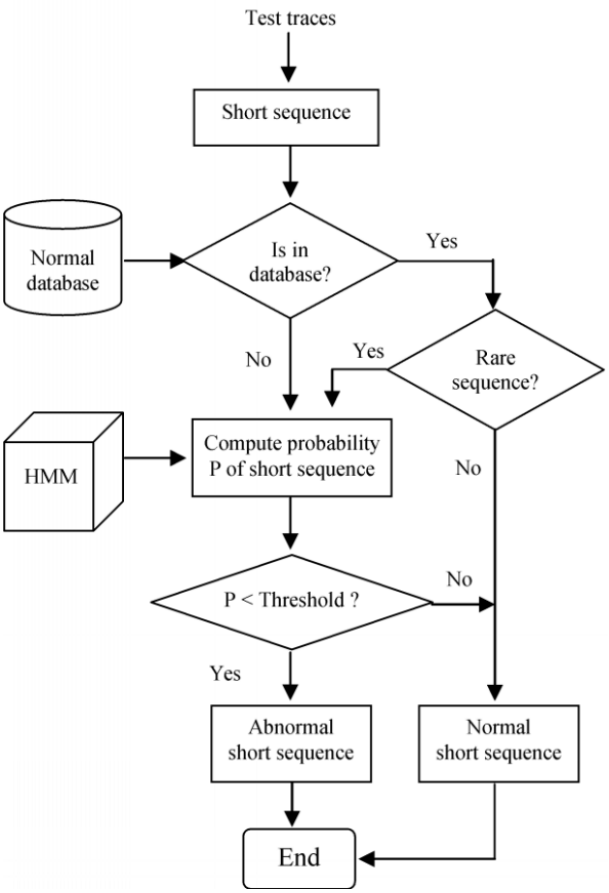

Figure 5.4: Two-layer detection scheme: evaluation of short sequences for anomalies and possible intrusions.

with a single HMM-based detection engine may still produce considerable false alarms. In this section, we propose a two-layer detection scheme which is an extension to the basic detection model. The two-layer detection scheme aims at reducing false positive alarms. In this detection scheme, a normal database [25] and an HMM model, are used to form a double-layer test to evaluate short sequences of system calls, to find anomalies and possible intrusions.

5.2.1 The proposed two-layer detection scheme

Figure 5.4 shows our two-layer detection scheme for the evaluation of short sequences of system calls of test traces for possible intrusions. The two-layer detection scheme is based on the basic detection model, discussed in Section 5.1 and the normal database model [25, 26]. The detection scheme is built as follows:

1. **Building a normal database and an HMM model from training data**:

   (a) Normal database: The normal database is an ordered list of all unique short sequences of system calls found in training data. The database is created from the training data of normal traces of system calls using the method given in [25, 26]. All short sequences in the normal database have the same number of system calls k, and k is used as the short sequence length. In addition, the frequency of each short sequence, that is the number of occurrences of that sequence in training data, is also recorded in the normal database.

   (b) HMM model: The HMM model is trained from normal traces of system calls, using HMM batch training, based on the Baum-Welch algorithm, as discussed in Section 4.2.3.

2. **Detecting anomalies**: First, test short sequences are formed from test traces of system calls, using the sliding window method. The length of each short sequence is k system calls, and is equal to the length of short sequences in the normal database. Then each short sequence is evaluated using the following procedure:

   (a) Test against the normal database: This test is applied for all short sequences of system calls in the test traces and is used to find mismatched and rare short sequences.

   (b) Test against the HMM model: This test is only used to verify short sequences which were classified as mismatched and rare by the normal database. For each mismatched, or rare short sequence, we compute the probability P of the HMM model producing the short sequence. If $P < \hat{P}$, where $\hat{P}$ is the sequence probability threshold, the short sequence is considered abnormal. Otherwise, it is normal, even though it is not found in the normal database.

5.2.2 Experimental results

In order to evaluate the performance of our proposed two-layer detection scheme, we conducted experiments to measure the false positive rate and the detection rate of the proposed two-layer detection scheme, and compared the results to those of the normal database scheme, as given in [25].

5.2.2.1 Data sets

We use *sendmail* traces of system calls collected in a synthetic environment[1], as given in [132]. The format of system call traces and the data collection procedures were discussed in Section 5.1.1. The data sets consist of the following:

1. *Normal traces* that are collected during the program's normal activity. Normal traces of *sendmail* program include 2 traces with the total of 1,595,612 system calls.

2. *Abnormal traces* that include known intrusions. The *sendmail* abnormal traces consist of 1 trace of *sm5x*, 1 trace of *sm565a*, 2 traces of *syslog-local*, and 2 traces of *syslog-remote* intrusions.

5.2.2.2 Experimental design

In order to measure the detection rate and the false alarm rate of our two-layer scheme, as opposed to the normal database scheme [25], our experiments were designed as follows:

1. **Measurement of the false positive rate**: In this test, we use the proposed detection scheme to evaluate normal traces of system calls which are not used in the construction of the normal database and the HMM model. Since the normal traces do not contain any intrusions, any reported alarms are considered false positive alarms. This experiment was set up as follows:

   (a) Select first 1,000,000 system calls of *sendmail* normal traces as the full training set.

   (b) Form 4 training sets which account for 30%, 50%, 80% and 100% of the size of the full training set.

   (c) Construct normal databases and HMM models from these training sets. The chosen values for the length of short sequences are $k = 5, 11$, and 15 system calls.

   (d) Select three test traces, each trace of 50,000 system calls from *sendmail* normal traces, which are not used in the training process to test for false positive alarms of our scheme and the normal database scheme [25]. Reported abnormal short sequences are counted for each test trace.

[1] A type of artificial or simulated environment created to test programs. Detailed description of synthetic environments is given in [26].

Table 5.3: Testing parameters: length of short sequences and probability threshold. The test set consists of 3 traces, each of which has 50,000 system calls.

Length of short sequence k	**Number of short sequences in each test trace (*)**	**Total number of short sequences in test set**	**Log probability threshold of short sequence $log(\hat{P})$**
5	49,996	149,988	-30.0
11	49,990	149,970	-40.0
15	49,986	149,958	-55.0
(*) Short sequences are formed by sliding a window of k system calls, one system call at a time, through the test trace of 50,000 system calls, and recording system calls within the sliding window as one short sequence at each move.			

2. **Measurement of anomaly signals and the detection rate**: In this test, we use the proposed detection scheme to evaluate abnormal traces to find possible intrusions. Since the abnormal traces have been collected from the program's abnormal runs generated by known intrusions, reported alarms in this case can be considered true alarms, or detected intrusions. This experiment was implemented as follows:

   (a) Construct a normal database and a HMM model from *sendmail* normal traces of system calls. We choose length of short sequences $k = 11$ to build the normal database from normal traces, and to form test short sequences from abnormal traces.

   (b) Use the proposed two-layer detection scheme to evaluate abnormal traces to find abnormal sequences.

   (c) Use temporally local regions to group individual abnormal sequences to measure the anomaly signals, as discussed in Section 5.1.2. The selected region length is $r = 20$.

   (d) Short sequence probability threshold $\hat{P}$ (or $log(\hat{P})$) and the region score threshold $\hat{A}$ are pre-selected.

Table 5.4: Overall false positive alarms produced by the normal database scheme [25] and by the proposed two-layer detection scheme, with the short sequence length, $k = 5, 11$ and 15.

Training data sets (% of full data set)	**Normal database scheme [25]**		**Two-layer scheme**	
	Reported number of abnormal short sequences	**False positive rate (%)**	**Reported number of abnormal short sequences**	**False positive rate (%)**
Sequence length, $k = 5$; 3 test traces with the total of 149,988 sequences				
30%	196	0.131	168	0.112
50%	149	0.099	118	0.079
80%	141	0.094	104	0.069
100%	141	0.094	104	0.069
Sequence length, $k = 11$; 3 test traces with the total of 149,970 sequences				
30%	291	0.194	255	0.170
50%	232	0.155	173	0.115
80%	225	0.150	160	0.107
100%	221	0.147	160	0.107
Sequence length, $k = 15$; 3 test traces with the total of 149,958 sequences				
30%	338	0.225	246	0.164
50%	264	0.176	182	0.121
80%	261	0.174	174	0.116
100%	257	0.171	174	0.116

5.2.2.3 Results

False positive alarms

Table 5.4 shows the total number of abnormal short sequences in three test traces with total of 150,000 system calls (each trace consists of 50,000 system calls), reported by the normal database scheme [25] and by the proposed two-layer scheme on different training sets, with short sequence length $k = 5$, $k = 11$ and $k = 15$. The overall false positive rate is calculated as the ratio of the total number of reported abnormal short sequences to the total number of short sequences in the three test traces. The total number of short sequences in each test trace is dependent on

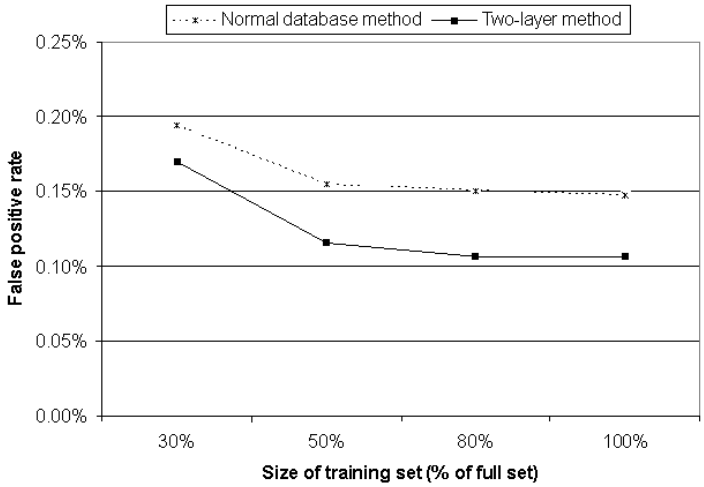

Figure 5.5: The relationship between the size of training sets and the false positive rate with $k = 11$

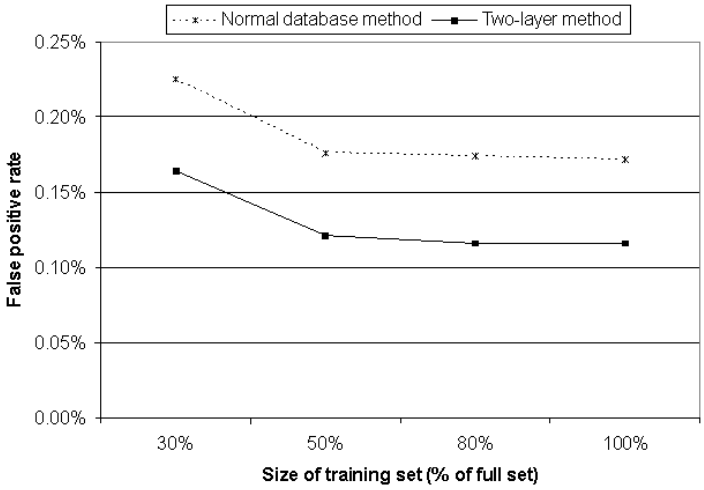

Figure 5.6: The relationship between the size of training sets and the false positive rate with $k = 15$

the short sequence length and is given in Table 5.3.

It is seen from Table 5.4 that the proposed two-layer scheme has lower false positive rate than the normal database scheme on all tested training sets and values of short sequence length. For example, on the training set of 80% of the full set, with $k = 15$, the false positive rate of the normal database scheme is 0.174%, while the false positive rate of our scheme is 0.116%. This is about 33.33% lower in the false positive rates.

Figures 5.5 and 5.6 describe the dependence of the false positive rate on the size of the training sets, with the short sequence length $k = 11$ and $k = 15$ respectively. When the size of the training set increases the false positive rate of both methods decreases significantly, especially from the training set of 30% of the full set, to the training set of 50% of the full set. For example, with $k = 11$, the false positive rates of our two-layer scheme are 0.170% and 0.115% on the training sets of 30% and 50% of the full set, respectively. This is a reduction of about 32.35%.

Anomaly signals and the detection rate

Table 5.5 presents a summary of the detection results, produced by the normal database scheme [25] and by the proposed two-layer scheme for some abnormal traces embedded with known intrusions. The detection performance results of the normal database scheme are extracted from Table 3, given in [25]. In [25], the anomaly signal is represented by the number of abnormal (mismatched) short sequences detected in the test trace, or by the equivalent percentage which is calculated as the ratio of the number of abnormal short sequences to the total number of short sequences of the test trace.

On the other hand, we measure the strength of anomaly signals, based on temporally local regions. The anomaly score A of a region is computed as the ratio of the number of abnormal short sequences detected in the region to the region length r, as given in equation (5.1). The average of anomaly scores is computed over abnormal regions which have the anomaly score A, with $A \geq \hat{A}$, where $\hat{A}$ is the region score threshold, and $\hat{A} = 40.0\%$.

It can be seen from Table 5.5 that the proposed two-layer scheme produced significantly better detection results than the normal database scheme [25] for all tested abnormal traces. For *sm565a* intrusion trace, the normal database scheme detected only 0.6% abnormal sequences, while the proposed scheme detected 38.46% abnormal regions. Similarly, the normal database scheme almost missed the "syslog-local" intrusion in the *syslog-local No2* trace, with only 1.7% abnormal sequences detected, while our scheme clearly detected the intrusion with 16.67% abnormal regions detected.

Furthermore, our two-layer detection scheme also generated strong anomaly signals on average. This indicates that the generated anomaly score of each abnormal region is high. The average of anomaly scores generated by the proposed detection scheme is ranging from 60.42% for *sm5x* trace to 74.74% for *syslog-remote-2* trace. Figures 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12

Table 5.5: Detection results produced by the normal database scheme and by the two-layer scheme for some abnormal traces with the short sequence length $k = 11$, region length $r = 20$ and the score threshold of abnormal regions, $\hat{A} = 40.0\%$.

Name of test abnormal traces	**Normal database scheme: % detected abnormal sequences [25] (#)**	**Two-layer scheme**	
		% detected abnormal regions (*)	**Average of scores of abnormal regions (%) (!)**
sm565a	0.60	38.46	68.00
sm5x	2.70	31.58	60.42
syslog-local No1	5.10	12.00	73.33
syslog-local No2	1.70	16.67	71.54
syslog-remote No1	4.00	28.26	72.31
syslog-remote No2	5.30	24.68	74.74

(#) Results produced by the normal database model are extracted from Table 3 of [25].
(*) The percentage of the number of detected abnormal regions out of the total number of regions of the test trace.
(!) The average of anomaly scores is computed over abnormal regions, those with $A \geq \hat{A}$.

show the anomaly signals produced by our proposed two-layer scheme for abnormal traces of *s5mx, sm565a, syslog-local No1, syslog-local No2, syslog-remote No1* and *syslog-remote No2*, respectively, at the short sequence length $k = 11$. It is noted that we compute the anomaly score based on temporally local regions, as discussed in Section 5.1.2.3. It can be seen from these figures that all embedded intrusions can be detected correctly at a region score threshold $\hat{A} = 0.40$ for all tested traces.

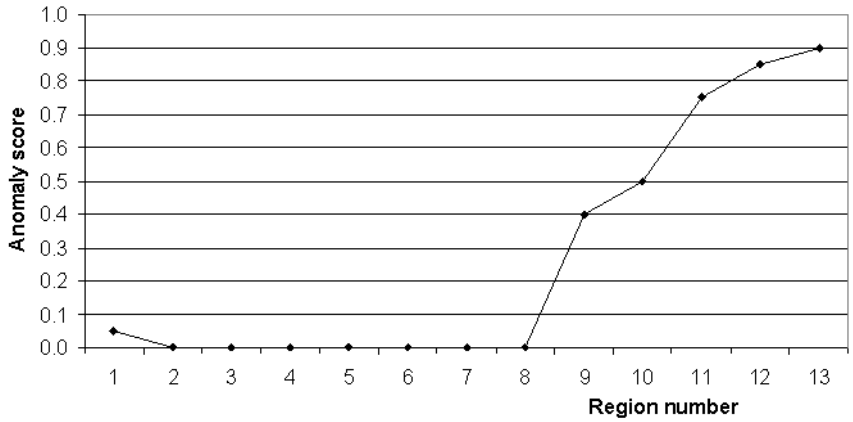

Figure 5.7: Anomaly signal generated for *sm565a* abnormal trace by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

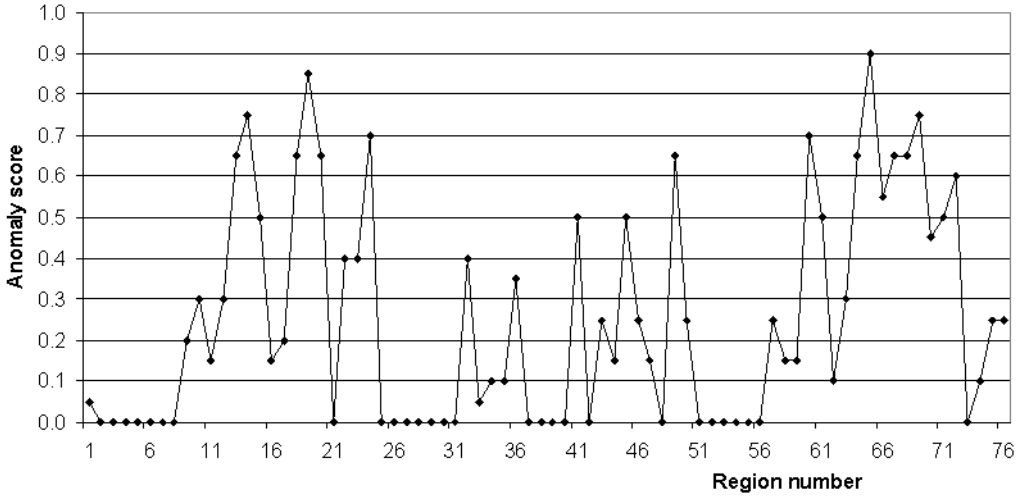

Figure 5.8: Anomaly signal generated for *s5mx* abnormal trace by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

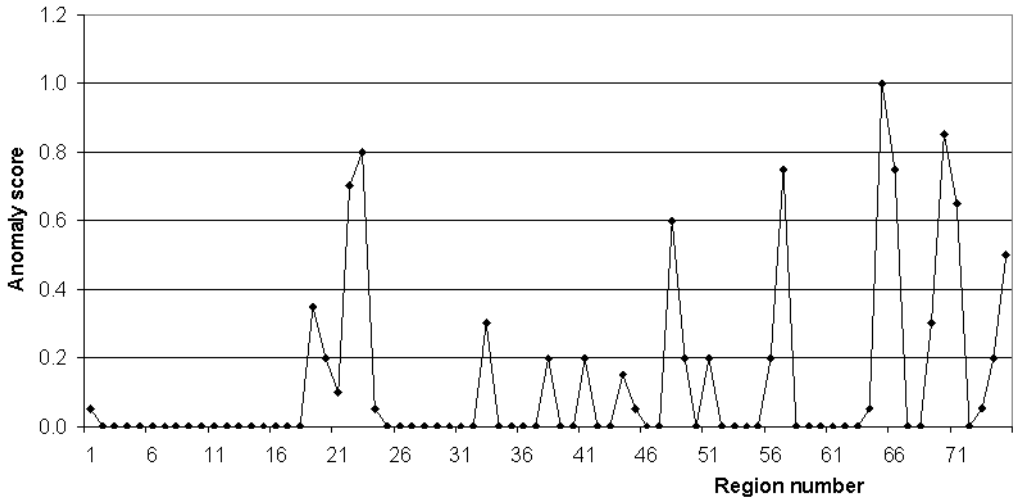

Figure 5.9: Anomaly signal generated for *syslog-local* abnormal trace No. 1 by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

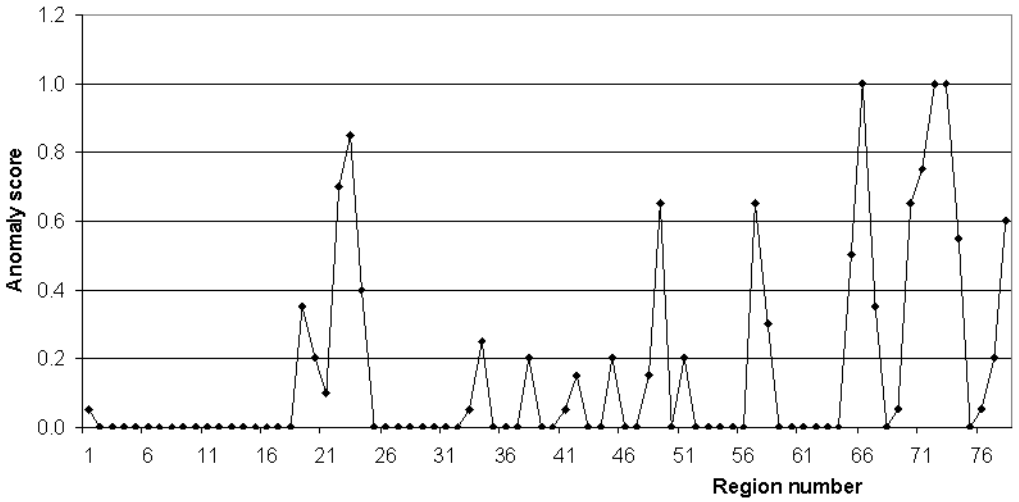

Figure 5.10: Anomaly signal generated for *syslog-local* abnormal trace No. 2 by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

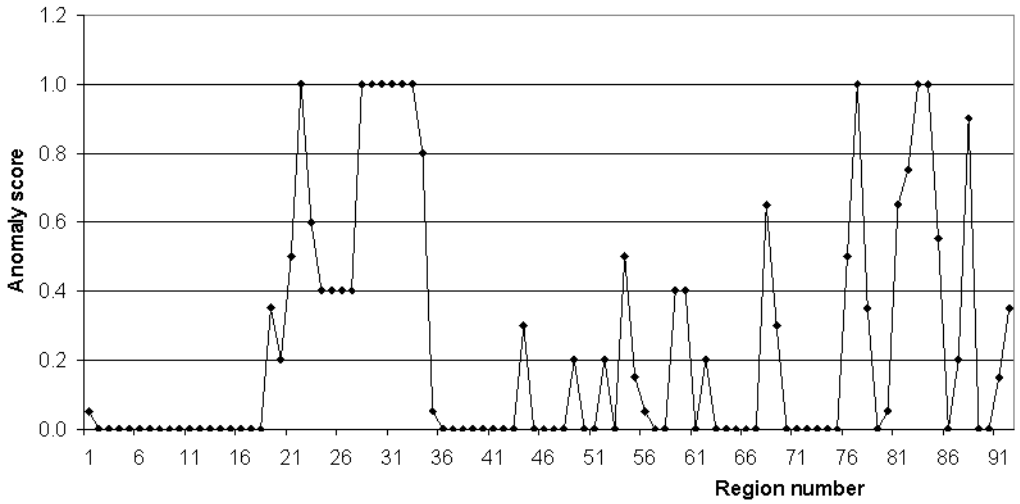

Figure 5.11: Anomaly signal generated for *syslog-remote* abnormal trace No. 1 by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

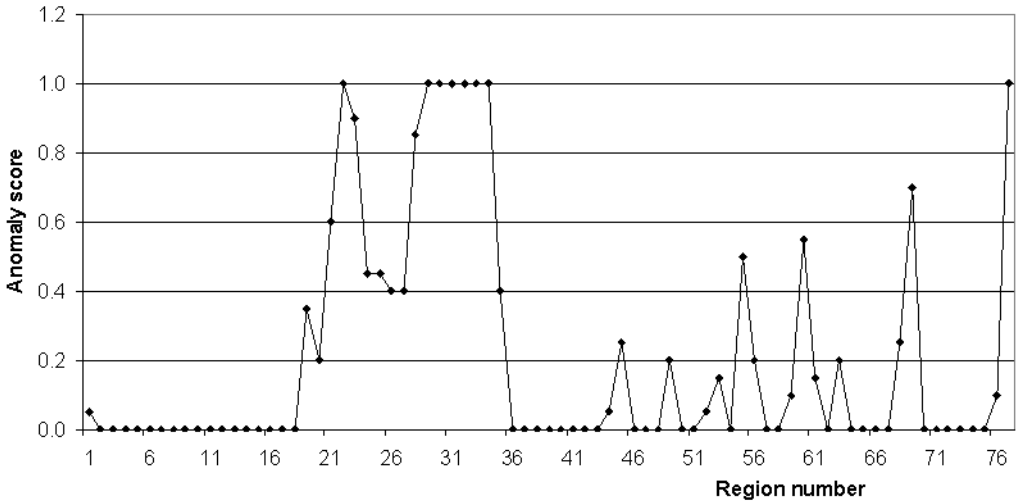

Figure 5.12: Anomaly signal generated for *syslog-remote* abnormal trace No. 2 by the proposed two-layer detection scheme with short sequence length k = 11, region length $r = 20$.

5.3 Summary

In this chapter, a basic anomaly detection model which is based on HMM modelling, using program system calls was presented. The model is developed in two stages: the building stage and the testing stage. In the building stage, an HMM model is constructed from training data to characterize normal program behaviour. In testing stage, the HMM model is used to evaluate short sequences of system calls to find possible intrusions. Also in testing stage, a scheme was given to measure the anomaly signal, which is based on the temporally local regions to group individual abnormal sequences. The collection and pre-processing procedures of traces of system calls were also given. The preliminary results showed that HMM-based basic detection model was able to generate strong and clear anomaly signals for two tested abnormal traces of system calls of *sendmail* and *inetd* programs. A strong anomaly signal helps the HMM-based detection correctly recognize the program's abnormal behaviour.

The proposed two-layer detection scheme which is an extension of the basic detection model produced substantially better overall detection results than the normal database scheme [25]. Our experimental results showed that the two-layer detection scheme correctly detected all intrusions embedded in the test traces for *sendmail* program. It also produced about 28% fewer false positive alarms than the normal database scheme (refers to Table 5.4 (page 105), for the case of $k = 11$ on the data sets of 50%, 80% and 100% of full data set). Futhermore, the proposed two-layer detection scheme generated stronger anomaly signals than the normal database scheme for several intrusion traces, as presented in Table 5.5 (page 108).

Since the basic detection model uses the traditional HMM batch training to train its HMM model, the model training cost is high, as discussed in Section 5.1.2.2. In Chapter 6, an efficient HMM training scheme is proposed to solve the HMM batch training efficiency problem.

Chapter 6

HMM Training for Program Anomaly Detection

As discussed in Section 5.1, existing HMM-based anomaly detection approaches suffer a serious efficiency problem, because the HMM batch training using the Baum-Welch algorithm is very expensive. In this chapter, an HMM incremental training scheme with optimal initialization to address the efficiency problem of HMM training is presented. The proposed HMM incremental training scheme aims at reducing training time, as well as storage demands.

6.1 HMM Training for Anomaly Detection

We propose an efficient HMM incremental training scheme as the solution to the efficiency problem of the traditional HMM batch training. The proposed HMM incremental training scheme consists two components: (i) an optimal initialization method for HMM parameters, and (ii) an HMM incremental training scheme. The optimal initialization method attempts to find optimal initial values for HMM parameters, which help to speed up the HMM training convergence. On the other hand, the HMM incremental training scheme focuses on reducing training time and storage demands by incrementally updating the HMM model, using one subset of the complete training data set at a time.

6.1.1 Optimal initialization of HMM parameters for HMM training

In HMM training process, the HMM parameters $\lambda(A, B, \pi)$ are adjusted so as to maximize the probability of the observation sequence O, $P(O|\lambda)$. Initialization is the first step in HMM training process, in which HMM parameters $\lambda(A, B, \pi)$ are assigned to pre-selected initial values. In theory, the values of HMM parameters in HMM training process should converge to a local maximum of likelihood function [104]. Choosing the initial values for HMM parameters, so that the local maximum is the global maximum of the likelihood function, is a crucial question. Unfortunately, there is no simple or straightforward answer for this question [104].

6.1.1.1 Random initialization

In random initialization, randomly generated values are used as the initial values for HMM parameters in HMM training. The random initialization can be described in the following simple algorithm:

1. Assign randomly generated values to HMM parameters, A, B, and π.

$$\pi_i \leftarrow rand(), \quad 1 \leq i \leq N \tag{6.1}$$

$$a_{ij} \leftarrow rand(), \quad 1 \leq i, j \leq N \tag{6.2}$$

$$b_j(k) \leftarrow rand(), \quad 1 \leq j \leq N,\ 1 \leq k \leq M \tag{6.3}$$

   where N is the number of HMM states, M is the number of HMM distinct observation symbols, and rand() is a function that returns a random value between 0 and 1.

2. Normalize A, B, and π, based on the stochastic constraints to HMM parameters, as given in Equations (4.4), (4.6) and (4.8).

Generally, random initialization is simple and it can be used to obtain useful re-estimates of HMM parameters in some cases [104]. However, since random generated values are used as initial values for HMM parameters, the HMM training convergence is usually slow. This leads to longer training time. Therefore, there is a need to find more suitable initialization methods which can improve HMM training convergence.

6.1.1.2 The proposed optimal initialization method

In order to find optimal initial values for HMM parameters, we propose to compute *prior probabilities* $C = \{c_{ij}\}, i = 1, N; j = 1, N$, where N is the number of hidden states, and use these values to initialize HMM training parameters. The prior probabilities are defined as the occurrence frequencies of two consecutive observation symbols in the input observation sequence. To compute prior probabilities, we count the number of occurrences of each pair of consecutive symbols $(O_t, O_{t+1}), t = 1...T-1$ in input sequence O. After normalization, prior probabilities C are used to initialize the HMM parameters in the training process. We use C as the initial values for HMM parameters A and B since C is relatively close to these parameters in probability terms, referring to the definition of A and B in section 4.1.2.

The prior probabilities are computed using the following simple algorithm:

Input : Observation sequence $O = \{O_t\}, t = 1, T$. Also noted that, we selected the number of hidden states equal to the number of distinct observation symbols, or $N = M$ as discussed in Section 5.1.2.2, page 95.

Output : The matrix of prior probabilities $C = \{c_{ij}\}, i = 1, N; j = 1, N$.

1. Set $c_{ij} = 0; i = 1, N, j = 1, N$
2. For each pair of consecutive observations O_t and O_{t+1} in sequence O, $t = 1, T-1$:
   (a) Find the corresponding element of C, $c_{ij} : i \leftarrow O_t, j \leftarrow O_{t+1}$.
   (b) Increase counter c_{ij} by 1: $c_{ij} \leftarrow c_{ij} + 1$.
3. Normalize the C matrix based on the transition probability distribution constraints defined in equation (4.6).

HMM training can then be performed with two types of initializations:

1. *Random initialization*: assign initial values to all HMM parameters $\lambda(A, B, \pi)$ randomly. Random initialization is a common initialization method in HMM training using the Baum-Welch algorithm. The random initialization is implemented based on equations (6.1), (6.2) and (6.3).

2. *Mixed initialization*: assign the combination of prior probabilities and random values to HMM parameters $\lambda(A, B, \pi)$ as intitial values. The prior probabilities are computed from the input observation sequence using the algorithm discribed above. We form three initialization options using prior probabilities:

   (a) *Init A*: assign prior probabilities to A as initial values, and random values to B and π as initial values. Specifically, A, B and π are initilized as:

$$\pi_i \leftarrow rand(), \quad 1 \leq i \leq N \tag{6.4}$$

$$a_{ij} \leftarrow c_{ij}, \quad 1 \leq i, j \leq N \tag{6.5}$$

$$b_j(k) \leftarrow rand(), \quad 1 \leq j \leq N,\ 1 \leq k \leq M, \quad (M = N) \tag{6.6}$$

   (b) *Init B*: assign prior probabilities to B as initial values, and random values to A and π as initial values. Specifically, A, B and π are initilized as:

$$\pi_i \leftarrow rand(), \quad 1 \leq i \leq N \tag{6.7}$$

$$a_{ij} \leftarrow rand(), \quad 1 \leq i, j \leq N \tag{6.8}$$

$$b_j(k) \leftarrow c_{ij}, \quad 1 \leq j \leq N,\ 1 \leq k \leq M, \quad (M = N) \tag{6.9}$$

   (c) *Init A&B*: assign prior probabilities to matrix A and the observation probability matrix B as initial values, and random values to π as initial values. Specifically, A, B and π are initilized as:

$$\pi_i \leftarrow rand(), \quad 1 \leq i \leq N \tag{6.10}$$

$$a_{ij} \leftarrow c_{ij}, \quad 1 \leq i, j \leq N \tag{6.11}$$

$$b_j(k) \leftarrow c_{ij}, \quad 1 \leq j \leq N,\ 1 \leq k \leq M, \quad (M = N) \tag{6.12}$$

6.1.2 HMM incremental training

Traditional HMM batch training, based on Expectation-Maximization (EM) algorithms, such as the Baum-Welch algorithm, updates HMM parameters using the full training data set at each iteration. Although the convergence of EM algorithms have been proven mathemetically, a large number of iterations is required to learn an useful model [36, 37]. This likely leads to long training time, especially for large-size models and long observation sequences. On the other

hand, incremental HMM training algorithms first split the full training data set into smaller subsets, and then incrementally update model parameters using one subset at a time, until convergence. It is suggested that incremental training algorithms converge faster than their EM batch counterparts [37, 95].

6.1.2.1 Existing HMM incremental training schemes

Gotoh et al [36, 37] proposed two HMM training schemes using incremental Maximum Likelihood (ML) and Maximum a Posterior (MAP) estimation algorithms for speech recognition. Their HMM incremental training algorithms reportedly converge significantly faster than the traditional HMM batch training. However, their algorithms only hold when the subsets of training data are independent. In our training data set, system calls are in interactive relations to form program functionality, and therefore the assumption of independent data subsets does not hold.

On the other hand, Davis et al [16] proposed a simple method to learn HMM models from a training set of multiple observation sequences. In this approach, first each observation sequence of the training set is used to learn one corresponding HMM model which we call an HMM sub-model. The learning of HMM sub-models are independent from each other and can be done in parallel. Then, when the learning of all HMM sub-models is complete, all HMM sub-models are merged together based on their weights to produce the final HMM model. The proposed method reportedly gives better recognition performance than the HMM batch training from multiple observation sequences, as described in [104].

6.1.2.2 The proposed HMM incremental training scheme

In our approach, we improve Davis et al's [16] training method to make it incremental. Our proposed HMM training scheme first divides a long training data set into a number of data subsets. Next, each data subset is used to train one HMM sub-model, and then the sub-model is incrementally merged into the final HMM model. Figure 6.1 describes our training scheme, in which λ_k is the HMM sub-model trained from the subset $O_{(k)}$, and $\lambda^{(k)}$ is the final HMM model for k data subsets. To make the merging process simple, we choose an uniform size (number of hidden states N and number of distinct observation symbols M) for the final HMM model and all HMM sub-models. The proposed training scheme can be described in the following steps:

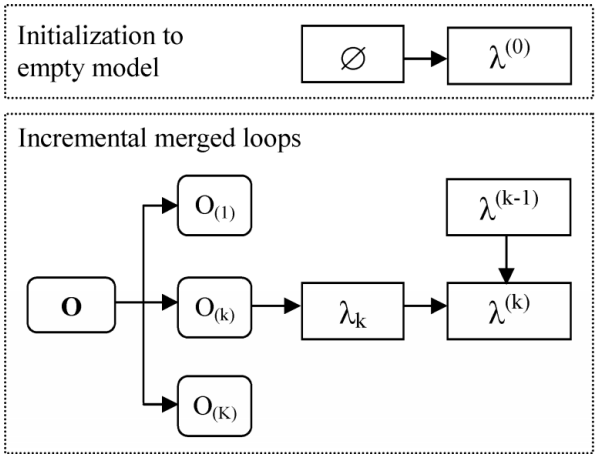

Figure 6.1: The proposed HMM incremental training scheme based on weighted merging

1. Initialize the final HMM model to empty, $\lambda^{(0)} \leftarrow \oslash$.
2. Divide the training data set O into K subsets $\{O_{(1)}, ..., O_{(k)}, ..., O_{(K)}\}$.
3. For each subset $O_{(k)},\ k = 1, K$:
   (a) Use subset $O_{(k)}$ to train HMM sub-model λ_k using the Baum-Welch training algorithm as described in Section 4.2.
   (b) Incrementally merge $\lambda_{(k)}$ into final HMM model, $\lambda^{(k)} \leftarrow \lambda_k + \lambda^{(k-1)}$.
4. End: $\lambda \leftarrow \lambda^{(K)}$.

The parameters of the final HMM model (A, B, π) at the subset k^{th} are computed using incremental weighted merging as follows:

$$\pi_i^{(k)} = w_k * \pi_{(k)i} + w^{(k-1)} * \pi_i^{(k-1)}, \quad 1 \le i \le N \tag{6.13}$$

$$a_{ij}^{(k)} = w_k * a_{(k)ij} + w^{(k-1)} * a_{ij}^{(k-1)}, \quad 1 \le i, j \le N \tag{6.14}$$

$$b_{ij}^{(k)} = w_k * b_{(k)ij} + w^{(k-1)} * b_{ij}^{(k-1)}, \quad 1 \le i \le N,\ 1 \le j \le M \tag{6.15}$$

where $(\{a_{(k)ij}\}, \{b_{(k)ij}\}, \{\pi_{(k)i}\})$ and $(\{a_{ij}^{(k-1)}\}, \{b_{ij}^{(k-1)}\}, \{\pi_i^{(k-1)}\})$ are HMM parameters of sub-model λ_k and model $\lambda^{(k-1)}$ respectively, and the weights w_k and $w^{(k-1)}$, which represent the contribution of new data and historical data respectively to the HMM final model, are

computed as:

$$w_k = \frac{1}{P(O_{(k)}|\lambda_k)},\ P(O_{(k)}|\lambda_k) \text{ is the probability that} \tag{6.16}$$
$$\text{the sub-model } \lambda_k \text{ generates subset } O_{(k)}.$$

$$w^{(k-1)} = \frac{1}{P(O_{(1)},...,O_{(k-1)}|\lambda^{(k-1)})},\ P(O_{(1)},...,O_{(k-1)}|\lambda^{(k-1)}) \text{ is the probability that} \tag{6.17}$$
$$\text{the model } \lambda^{(k-1)} \text{ generates subsets } \{O_{(1)},...,O_{(k-1)}\}.$$

In HMM online training, since the historical subsets $\{O_{(1)},...,O_{(k-1)}\}$ are not available, $w^{(k-1)}$ is computed approximately as:

$$w^{(k-1)} = \frac{1}{P(O_{(k-1)}|\lambda^{(k-1)})},\ P(O_{(k-1)}|\lambda^{(k-1)}) \text{ is the probability that} \tag{6.18}$$
$$\text{the model } \lambda^{(k-1)} \text{ generates subset } O_{(k-1)}.$$

6.2 Experimental Results

We measure the performance of the proposed HMM training scheme using two sets of experiments. In the first experiment set, we measure the training time of HMM batch training using random initialization and the proposed mixed initialization method. The best performed option of the mixed initialization method is named as the "optimal initialization method". Optimal initialization helps to improve the convergence rate of HMM training, and therefore to reduce the overall training time. In the second experiment set, we measure the training time of HMM batch training and the proposed HMM incremental training scheme. The optimal initialization method is used in both HMM batch training and the proposed HMM incremental training scheme in the second experiment set.

6.2.1 Optimal initialization results

In order to examine the effect of different initialization methods to HMM training, we conducted HMM training experiments with different data sets, using random initialization and mixed initialization with the three options mentioned above. Our experiments were set up as follows:

1. HMM training data: First, a *sendmail* data set of 1,000,000 system calls, collected in

Table 6.1: Length of the training subsets in number of system calls

Subset No	**1**	**2**	**3**	**4**	**5**
Length	100,000	300,000	500,000	800,000	1,000,000
% of full set	10%	30%	50%	80%	100%

synthetic environment[1] [132], is selected as the *full data set*. Then, from the full data set, we form 5 training subsets of 10%, 30%, 50%, 80%, and 100% of the size of the full data set, as shown in Table 6.1.

2. Size of HMM models: As discussed in Section 5.1.2, we select $N = M = 50$, which is the number of unique system calls used by *sendmail* in this *sendmail* data set.

3. Performance test of HMM training using different initialization methods: The Baum-Welch algorithm is used to train HMM models from the mentioned data subsets. For each data subset, we train HMM models using different initialization methods, as discussed in Section 6.1.1. The training time for each HMM model is recorded with common convergence criteria. An AMD Athlon XP-2200+ based PC with 1GB RAM memory is used in this performance test.

Table 6.2 shows the training time of HMM models trained from 5 data subsets, using random initialization and mixed initialization with the three options, as discussed in Section 6.1.1.2. The difference between the training time of HMM training using random initialization and option "Init A" of mixed initialization for each data subset are computed, normalized and given in the last column of the table in the percentage form for comparison.

It is seen from Table 6.2 that any option of mixed initialization gives better performance than random initialization. Among three options of mixed initialization, two options of "Init A" and "Init A & B" perform noticebly better than random initialization across the whole data set. For example, the training time of HMM training, based on "Init A" initialization option is reduced by 67.36%, compared to that of HMM training, based on random initialization, for the data subset of 100,000 system calls. The reductions of HMM training time for data subsets of

[1] A type of artificial or simulated environment created to test programs. Detailed description of synthetic environments is given in [26].

Table 6.2: HMM training time based on random initialization and mixed initializations (Init A, Init B, and Init A & B) on 5 training sets.

Length of training subset	**HMM training time (minutes)**				**Normalized time difference between Init A and Random**
	Random initialization	**Mixed initialization**			
		Init A	**Init B**	**Init A&B**	
A	B	C	D	E	F=(B-C)/B
100,000	24.77	8.08	9.10	13.08	67.36%
300,000	70.80	30.37	40.57	25.72	57.11%
500,000	100.30	50.50	56.90	53.75	49.65%
800,000	177.27	77.65	128.85	87.80	56.20%
1,000,000	201.37	123.92	180.88	111.07	38.46%

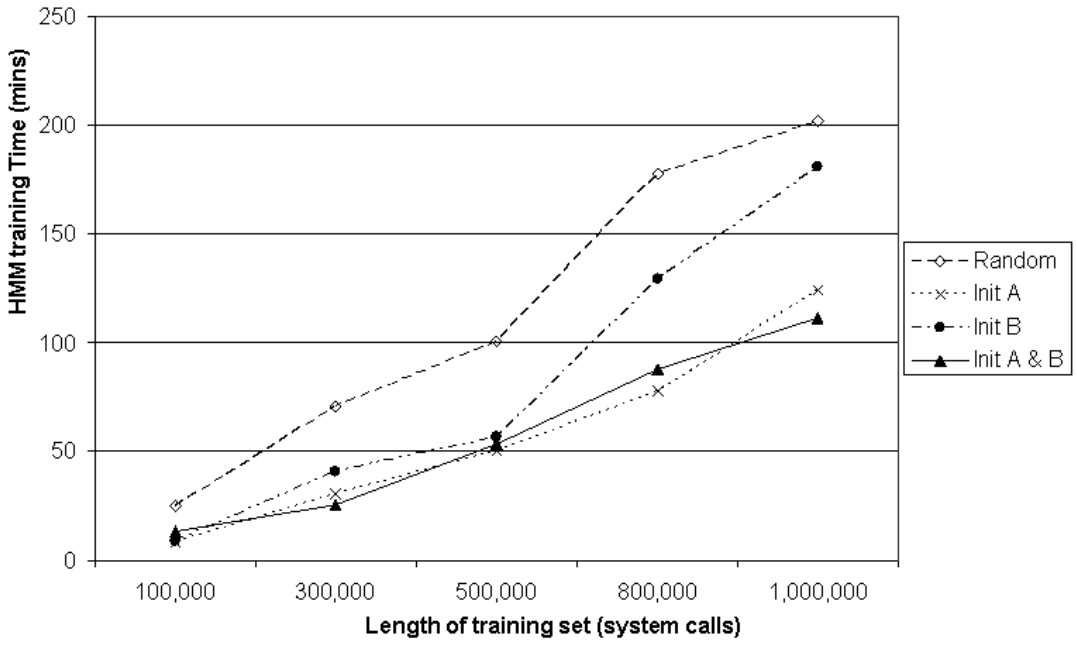

Figure 6.2: Performance comparison between random initialization and mixed initialization (Init A, Init B, and Init A & B) on 5 training sets.

300,000, 500,000, 800,000 and 1,000,000 system calls are 57.11%, 49.65%, 56.20% and 38.46%, respectively. "Init B" also performs well for small data sets of 300,000 and 500,000 system calls, but it produces only a small reduction of training time for large data sets of 800,000 and

1,000,000 system calls.

Figure 6.2 presents the performance comparison between random initialization and mixed initialization with three options, including "Init A", "Init B" and "Init A & B". "Init A" and "Init A & B" initialization options have about the same performance level. For 5 data subsets, "Init A" performs better on 3 subsets of 100,000, 500,000 and 800,000 system calls, while "Init A & B" does better on 2 subsets of 300,000 and 1,000,000 system calls. Although "Init B" option performs almost comparably to "Init A" and "Init A & B" on small data sets of 100,000 and 300,000 system calls, it does not perform well on larger data sets. The reason of performance gains of options "Init A" and "Init A & B" is that the prior probabilities C (the occurrence frequencies of two consecutive observation symbols), that are used as initial values of the transition matrix A in "Init A" and "Init A & B" options, are the most suitable to the transition probability distribution.

6.2.2 HMM incremental training results

In order to measure the performance of our incremental HMM training scheme, we conducted HMM training experiments using the Baum-Welch batch training and using the proposed HMM incremental training scheme on the different training sets. Our experiments were set up as follows:

1. Selected data set: we select a *sendmail* data set of 1,000,000 system calls collected in synthetic environment[1], as given in [132] for our experiments.

2. Size of HMM models: As discussed in Section 5.1.2, we select $N = M = 50$, which is the number of unique system calls used by *sendmail* in this data set.

3. Performance test of the proposed HMM incremental training scheme: in this performance test, we measure the training time of our HMM incremental training scheme and the traditional HMM batch training as follows:

   (a) Measure the training time of the Baum-Welch HMM batch training using the original training set of 1,000,000 system calls.

[1] A type of artificial or simulated environment created to test programs. Detailed description of synthetic environments is given in [26].

(b) Measure the training time of the proposed HMM incremental training. We form 7 incremental training options by dividing the original training set of 1,000,000 system calls into 2, 5, 10, 20, 30, 40 and 50 subsets. For each incremental training option, each subset is used to update the HMM model incrementally, until convergence.

(c) The mixed intitialization option "Init A", discussed in Section 6.1.1, is used in both HMM batch training and HMM incremental training.

(d) All HMM models have the same training convergence criteria.

An AMD Athlon XP-2200+ based PC with 1GB RAM memory is used in this HMM training performance test.

Table 6.3 shows the training time of the Baum-Welch HMM batch training and the proposed HMM incremental training scheme. The performance results of HMM batch training, based on the Baum-Welch algorithm, is given in the first row of the table. Other rows show performance results of the proposed HMM incremental training scheme. There are 7 incremental training options with the number of training subsets, ranging from 2 to 50. The difference between the training time of HMM batch training and each incremental training option was computed, normalized and given in the last column of the table in the percentage form for comparison.

As can be seen from Table 6.3 that the proposed incremental training scheme can reduce training time substantially on all incremental training options except the option with 2 subsets. While the training time of the HMM batch training on a single set is 123.92 minutes, the training time of HMM incremental training on 40 subsets is 48.47 minutes. This is 60.89% reduction. The corresponding reductions of the training time of other HMM incremental training options on 5, 10, 20, 30, and 50 subsets are 20.89%, 26.77%, 46.79%, 51.41%, 55.40%, respectively.

Figure 6.3 shows the dependence of the training time of HMM training on the number of subsets. When the number of subsets increases from 1 (batch mode) to 2, the training time increases slightly, from 123.92 minutes to 127.72 minutes. Then, the training time decreases significantly, from 127.72 minutes to 48.47 minutes. This is a reduction of about 2.6 times, when the number of subsets increases from 2 to 40. When the number of subsets increases to 50, the training time increases slightly again to 55.27 minutes. This means that the proposed HMM incremental training gives best performance on 40 subsets, or 25,000 system calls per subset.

Table 6.3: Training time of HMM batch training and incremental training on the number of subsets. The total length of each training set is 1,000,000 system calls.

Index i = 0,...,7	**Number of subsets**	**Length of each subset**	**Total training time (mins)**	**Average training time per subset (mins)**	**Normalized time difference bewteen incremental and batch training**
	A	B	C	$D_i = C_i/A_i$	$E_i = (C_0 - C_i)/C_0$
0	**1***	**1,000,000**	**123.92**	**123.92**	**0**
1	2	500,000	127.72	63.86	-3.07%
2	5	200,000	98.03	19.75	20.89%
3	10	100,000	90.75	9.08	26.77%
4	20	50,000	65.93	4.80	46.79%
5	30**	33,333	60.22	2.01	51.41%
6	40	25,000	48.47	1.21	60.89%
7	50	20,000	55.27	1.11	55.40%
* Batch training: number of subset is 1 ** There is 1 subset of 33343 system calls in this incremental option					

When the number of training subsets is too small, the proposed HMM incremental training scheme performs poorer than the HMM batch training scheme. For example, the training time of HMM batch training is 123.92 minutes, while the training time of HMM incremental training on 2 subsets is 127.72 minutes. This is because the time saving produced by incremental training is relatively small, when the number of training subsets is small. This time saving is not sufficient to compensate the extra training time caused by the additional overhead of the HMM incremental weighted merging. However, when the number of training subsets is increasing, the time saving is increasing, and this additional overhead is becoming insignificant.

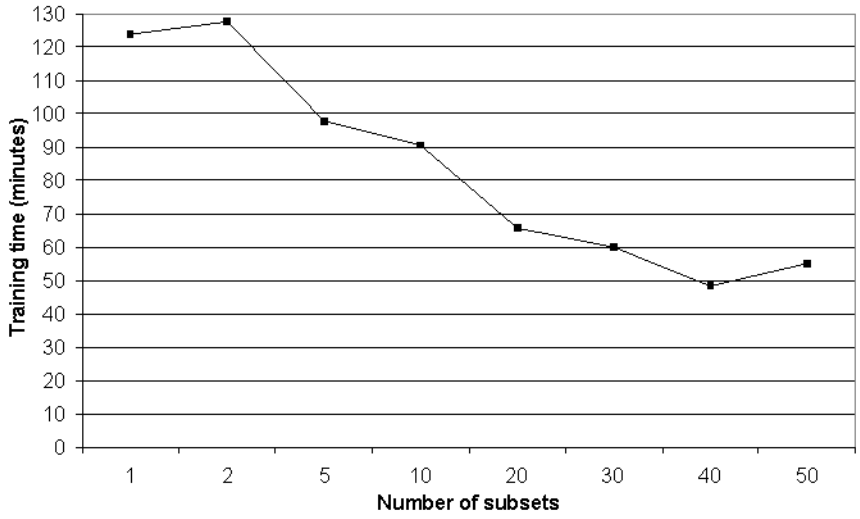

Figure 6.3: The dependence of training time on the number of subsets in HMM incremental training. 1 subset indicates HMM batch training. The total length of each training set is 1,000,000 system calls.

6.3 Summary

In this chapter, an HMM incremental training scheme with optimal initialization, which aimed at addressing the efficiency problem of the HMM Baum-Welch batch training, was presented. As compared to the HMM batch training, our HMM incremental training scheme reduced the training time by over 4 times, and reduced the required storage, by K times, where K is the number of training subsets used. This efficiency improvement is significant, and it makes our HMM-based detection model more efficient for practical use.

Another advantage of our incremental training scheme is that the HMM model is updated more frequently during the training process, because the model update is based on each subset which is significantly smaller than the full training set. For example, the average training time for each subset of 25,000 system calls is only 1.21 minutes, compared to the total training time of 123.92 minutes for HMM batch training on the full training set. The frequent update by the incremental training is suitable for online update of HMM-based detection models.

In Chapter 8, the proposed incremetal HMM training scheme will be used to construct the HMM model in the training stage of the proposed complete detection model, as well as to support online update of the HMM model, using online training data.

Chapter 7

Detecting Program Anomalies Based on Fuzzy Inference

In this chapter, a fuzzy-based extension of the two-layer detection scheme, which aims at further improving detection accuracy is presented. We are motivated by the fact that the normalcy or abnormalcy in anomaly detection are not truly crisp concepts [18, 33, 34]. Therefore, it is difficult to correctly classify an object's behaviour as normal or abnormal, using crisp conditions. Fuzzy logic and its fuzzy sets and rules are naturally suitable to be used to represent the space of these imprecise concepts. Furthermore, fuzzy inference's capability of combining information from multiple sources helps anomaly detection accurately evaluate the object's behaviour.

7.1 Introduction

One of the most difficult tasks in anomaly intrusion detection is to determine the boundaries between the normal and abnormal behaviours of a monitored object. A well-defined boundary helps an anomaly detection system correctly label the current behaviour as normal or abnormal. Unfortunately, the border between the normal and abnormal behaviours may not always be precisely defined, because the normal and abnormal behaviours can be overlapped or very closed to each other [18, 33, 34]. This likely leads to an increase of the false positive rate and a reduction of the detection rate. The work discussed in this chapter focuses on a fuzzy-based solution to this problem, specifically in the classification of system call sequences, using fuzzy inference for program anomaly detection.

In anomaly detection, the boundaries between the normal and abnormal behaviours can be divided into two types: (i) hard boundaries and (ii) soft boundaries. A hard boundary is usually used in the form of crisp conditions or fixed thresholds. For example, in [25], a test sequence of system calls is labelled as normal if it is seen in the training set. Otherwise, it is classified as abnormal. In the HMM-based anomaly detection model, discussed in Section 5.1, a probability threshold $\hat{P}$ is used to determine the status of test sequences. If a test sequence's probability P generated by the HMM model is equal or greater than the probability threshold ($P \geq \hat{P}$), it is considered normal. Otherwise, it is considered abnormal ($P < \hat{P}$).

On the other hand, a soft boundary is usually represented by fuzzy sets and rules, instead of crisp conditions or fixed thresholds. In [18], five fuzzy sets, namely LOW, MED-LOW, MEDIUM, MED-HIGH and HIGH are used to represent the space of each input network data source. In addition, a set of fuzzy rules is defined to combine a set of inputs in order to produce an output which is the status of current network activity. In another approach reported in [24], a set of fuzzy association rules is generated to represent the normal behaviour of network traffic.

Anomaly detection approaches, based on soft boundaries in general, or fuzzy sets and rules in particular, can produce better detection performance than those based on hard boundaries because of the following two reasons:

- Since the normalcy and abnormalcy are not truly crisp concepts, it is difficult to define a hard boundary that can create a sharp distinction between the normal and abnormal. Therefore, it is natural to use fuzzy sets to define a "soft" border between them [18, 34]. In fuzzy logic terms, the normal is represented by the degree of normalcy. Similarly, the abnormal is represented by the degree of abnormalcy.

- Anomaly detection systems, based on fuzzy inference, can readily combine inputs from multiple sources, which likely leads to better detection performance [18].

Although the application of fuzzy inference in intrusion detection is still in an early stage, promising results have been reported by a number of fuzzy-based anomaly detection approaches. Cho [12] reported a high detection rate and a significant reduction in the false positive rate, using fuzzy inference to combine inputs from three separate HMM models in a user anomaly detection system. In another approach, Luo et al [88] presented a real-time anomaly intrusion detection system, in which a set of fuzzy frequent episode rules was mined from the training data to

represent the abnormality. The proposed approach reportedly had lower false positive rates than those, based on non-fuzzy frequent episode rules. There have also been good detection results reported in [18, 24, 34].

In our approach, we propose a detection scheme which is based on fuzzy inference for program anomaly intrusion detection using system calls. We employ the fuzzy inference to combine multiple inputs generated by a normal database and an HMM model to evaluate each test sequence of system calls. Each test sequence of test traces is represented by three parameters, including the sequence probability generated by the HMM model, the sequence distance and the sequence frequency produced by the normal database. Instead of using crisp conditions or fixed thresholds, a group of fuzzy sets is defined to represent each parameter's space. A set of fuzzy rules is created to combine these input sequence parameters, in order to produce an output of the sequence status.

The proposed fuzzy-based detection scheme is an extension of the *two-layer detection scheme*, discussed in Section 5.2. Like the two-layer scheme, the fuzzy-based scheme also includes a normal database [25] and an HMM model. Therefore, the two methods share the same process of database building and HMM training. However, while the normal database and the HMM model are used directly in the classification of short sequences in the two-layer scheme, they are only used to generate inputs for the fuzzy inference engine. The fuzzy inference engine is responsible for the classification of short sequences in the fuzzy-based scheme.

The remaining of this chapter is organized as follows: in Section 7.2, we give a brief introduction to fuzzy logic which is the heart of fuzzy inference. Section 7.3 describes the fuzzy reasoning process for sequence classification and Section 7.4 describes our fuzzy-based detection scheme to find anomalies and possible intrusions. In Section 7.5, we present the experimental results. Section 7.6 is the summary of this chapter.

7.2 Introduction to Fuzzy Logic

Fuzzy logic is an extension of boolean logic, which specifically deals with the concept of partial truth [135]. The mathematical principles of fuzzy sets and fuzzy logic were first presented in 1965 by professor L.A. Zadeh at the University of California at Berkley, USA [137, 138]. Since then, fuzzy logic has rapidly become one of the most successful technologies in the development of control systems. The application of fuzzy logic is ranging from simple, small and embedded

micro-controllers to large data acquisition and control systems [14].

While a truth value in classical logic can always be expressed in binary terms (0 or 1, True or False, Yes or No), a truth value in fuzzy logic is represented by the degree of truth. The degree of truth can be any value in the range [0.0, 1.0], with 0.0 representing absolute Falseness and 1.0 representing absolute Truth.

In this section, we give a brief introduction to two important concepts of fuzzy logic, fuzzy sets and fuzzy rules. Fuzzy sets and fuzzy rules will be extensively used by our fuzzy-based detection scheme.

7.2.1 Fuzzy sets

Fuzzy sets are an extension of the mathematical concept of classical sets, which are used in fuzzy logic. In classical set theory, the membership of elements in relation to a set is assessed in binary terms according to a crisp condition, which means an element either belongs or does not belong to the set. Therefore, classical sets are also known as crisp sets. In fuzzy set theory, on the other hand, an element can belong to one or more fuzzy sets to some degree. The membership of elements in relation to a fuzzy set can be assessed gradually using a membership function [135]. Figure 7.1 shows an example of a fuzzy set and a crisp (classical) set.

Mathematically, a fuzzy set A is defined as follows:

$$A = \{(x, \mu_A(x)) \mid x \in U\} \tag{7.1}$$

where $\mu_A(x)$ is the membership function of the fuzzy set A, and U is the *Universe of Discourse*. A *Universe of Discourse*, or *Universe* in short is the range of all possible values for an input to a fuzzy system.

7.2.1.1 Membership functions

The membership function $\mu(x)$ of a fuzzy set is the function that maps each element x of the universe U to a value of the degree of membership. The degree of membership can be zero or any value in the range of [0, 1]. Generally, the membership function can be represented mathematically as:

$$\mu(x) : U \rightarrow [0, 1] \tag{7.2}$$

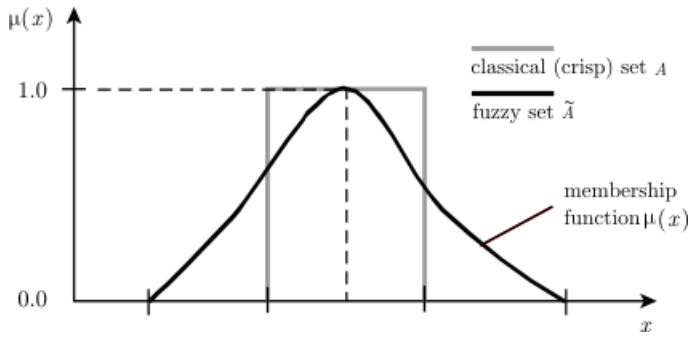

Figure 7.1: Fuzzy set and crisp (classical) set [135]

A membership function of a fuzzy set can either be continuous or discrete. In continuous form, the membership function is a mathematical function as presented in expression (7.2). For example, common membership functions are bell-shaped, s-shaped, triangular and trapezoidal. In discrete form, on the other hand, the membership function is a vector of discrete values as described in expression (7.3).

$$\mu(x) : \{x_1, x_2, ..., x_n\} \rightarrow [0, 1] \tag{7.3}$$

Figure 7.2 illustrates (a) a fuzzy set with a continuous trapezoidal membership function, and (b) a fuzzy set with a discrete membership function. The continuous trapezoidal membership function in this figure is defined as:

$$\mu(x) = \begin{cases} 1 & \text{if } x_0 \leq x < x_1 \\ ax + b & \text{if } x_1 \leq x < x_2;\ a = -\dfrac{1}{x_2 - x_1},\ b = \dfrac{x_2}{x_2 - x_1} \\ 0 & \text{if } x \geq x_2 \end{cases} \tag{7.4}$$

and the discrete membership function is expressed as:

$$\mu(x_1, x_2, x_3, x_4, x_5) = \{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5\} \tag{7.5}$$

A fuzzy set is continuous if its universe and membership functions are continuous. On the other hand, a fuzzy set is discrete if its universe and membership functions are discrete.

7.2.1.2 Fuzzy set operations

An operation on fuzzy sets creates a new fuzzy set from the original fuzzy sets. Since a fuzzy set is defined by its membership function, it is natural to define operations on fuzzy sets by means

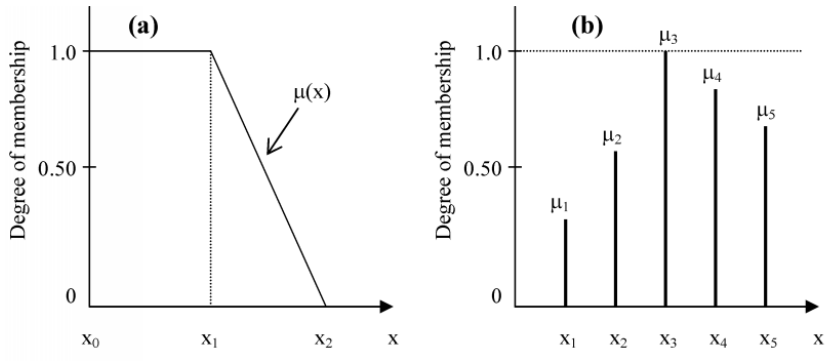

Figure 7.2: Fuzzy sets with (a) continuous and (b) discrete membership functions

of their membership functions. Due to the scope of our work, we only discuss some primitive operations on fuzzy sets.

Let A and B be fuzzy sets on a mutual universe U, and μ_A and μ_B be the membership functions of A and B respectively. Three primitive operations on fuzzy sets, namely Intersection, Union, and Complement are defined as follows:

Intersection: $A \cap B \equiv \min(\mu_A, \mu_B)$

The Intersection of two fuzzy sets A and B is a new fuzzy set that has the membership function as the minimum of the two indivudual membership functions.

Union: $A \cup B \equiv \max(\mu_A, \mu_B)$

The Union of two fuzzy sets A and B is a new fuzzy set that has the membership function as the maximum of the two indivudual membership functions.

Complement: $\bar{A} \equiv 1 - \mu_A$

The Complement of fuzzy set A is a new fuzzy set that has the membership function as the inverse of the membership function of A.

7.2.2 Fuzzy rules

Rules in fuzzy logic are used to combine and interpret inputs in order to produce an output. Fuzzy rules are usually expressed in the IF/THEN form as:

```
IF <variable> IS <fuzzy set> THEN <output>
```

For example, the following simple set of rules can be used to control a fan [135]:

```
IF temperature IS VeryCold THEN stop fan
IF temperature IS Cold THEN turn down fan
IF temperature IS Normal THEN maintain level
IF temperature IS Hot THEN speed up fan
```

where *temperature* is the input variable, {*VeryCold, Cold, Normal, Hot*} are input fuzzy sets, and {*stop, turn down, maintain level, speed up*} are output values. In this case, the output value is an action taken when the corresponding rule fires.

A rule is said to fire if its truth value is greater than 0. It is also noted that there is no "ELSE" clause in a fuzzy rule. All available rules in a fuzzy control system are evaluated because an input can belong to more than one fuzzy set. In the above fan control example, the *temperature* might belong to Cold set and Normal set at the same time, however, to some different degrees.

Like classical logic, fuzzy logic also supports AND, OR and NOT operators, which can be used to create more complex fuzzy rules. Let x and y be two fuzzy variables, and μ(x) and μ(y) be the degrees of membership of x and y, respectively, AND, OR and NOT operators are defined as:

x AND y = min(μ(x), μ(y))

x OR y = max(μ(x), μ(y))

NOT x = (1 - μ(x))

7.3 Fuzzy Inference for Sequence Classification

As our discussion in Section 3.2, the problem of program anomaly intrusion detection using system calls is reduced to the problem of classification of short sequences of system calls produced by the program during its execution. The two-layer detection scheme, presented in Section 5.2, classifies short sequences of system calls using a double-layer test supported by a normal database [25] and an HMM model. In this section, we present an enhanced sequence classification scheme which is based on fuzzy inference, aimed at further accuracy improvement.

Figure 7.3 presents the fuzzy inference engine for the classification of sequences of system calls. The engine accepts the sequence's parameters as the input and then applies the fuzzy sets and rules to produce the sequence's status as the output. The sequence parameters include the sequence probability P generated by the HMM model, and the sequence distance D and frequency F produced by the normal database.

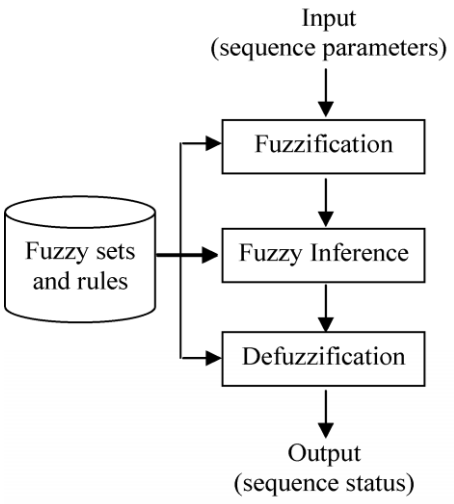

Figure 7.3: Fuzzy inference engine for the classification of sequences of system calls

7.3.1 Creation of fuzzy sets and rules

As shown in Figure 7.3, fuzzy sets and rules are used by the fuzzy inference engine to interpret the input in order to generate the output. While fuzzy sets are used to represent the sequence input and output parameters, the fuzzy rules are used to combine inputs in fuzzy reasoning process to produce the output.

7.3.1.1 Creation of fuzzy sets

We have created fuzzy sets for the three input parameters, including the sequence probability P, the sequence distance D and the sequence frequency F, and for the output parameter which is the anomaly score. Two common types of fuzzy sets, which are trapezium sets and triangle sets, are selected, based on analyzing the characteristics of each input and output sequence parameters.

Fuzzy sets for sequence probabilities

The probability of a test sequence is the probability that the HMM model produces the test sequence. Given the HMM model $\lambda(A, B, \pi)$ and the test sequence $O_t = \{O_{t1}, O_{t2}, ..., O_{tk}\}$, the sequence probability P is expressed as $P(O_t|\lambda)$. $P(O_t|\lambda)$ is computed using forward or backward procedure, as discussed in Section 4. In practice, $log(P)$ is usually computed, instead of P because of the number underflow problem [104].

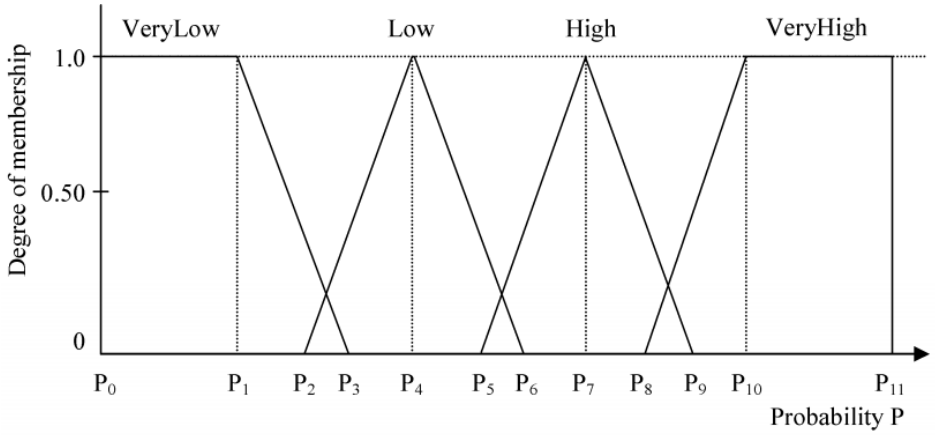

Figure 7.4: Fuzzy sets of sequence probability P

Four fuzzy sets are created for the sequence probabilities, which are named VeryLow, Low, High and VeryHigh, as shown in Figure 7.4. VeryLow and VeryHigh are trapezium sets while Low and High are triangle sets. The universe of the sequence probability fuzzy sets is $[0.0, 1.0]$, with 0.0 representing impossible sequence events and 1.0 representing certain sequence events. The membership functions of these fuzzy sets are determined empirically by computing the probability distribution of short sequences for some normal and abnormal traces of system calls. For example, (P_{10}) - the minimum probability that the HMM model produces a normal sequence is obtained from the sequence probability distribution in normal traces. And, (P_1) - the maximum probability that the HMM model produces an abnormal sequence is obtained from the sequence probability distribution in abnormal traces.

The sequence probability fuzzy sets are defined as follows:

- VeryLow: VeryLow set represents unlikely probabilities that the HMM model produces normal sequences. Its membership function is defined as:

$$\mu_{VeryLow}(P) = \begin{cases} 1 & \text{if } P_0 \leq P < P_1 \\ a * P + b & \text{if } P_1 \leq P < P_3;\ a = -\dfrac{1}{P_3 - P_1},\ b = \dfrac{P_3}{P_3 - P_1} \\ 0 & \text{if } P \geq P_3 \end{cases} \tag{7.6}$$

- Low: Low set represents relatively low probabilities that the HMM model produces normal

sequences. Its membership function is defined as:

$$\mu_{Low}(P) = \begin{cases} 0 & \text{if } P < P_2 \\ a_1 * P + b_1 & \text{if } P_2 \leq P < P_4;\ a_1 = \dfrac{1}{P_4 - P_2},\ b_1 = \dfrac{P_4}{P_4 - P_2} \\ a_2 * P + b_2 & \text{if } P_4 \leq P < P_6;\ a_2 = -\dfrac{1}{P_6 - P_4},\ b_2 = \dfrac{P_6}{P_6 - P_4} \\ 0 & \text{if } P \geq P_6 \end{cases} \tag{7.7}$$

- High: High set represents relatively high probabilities that the HMM model produces normal sequences. Its membership function is defined as:

$$\mu_{High}(P) = \begin{cases} 0 & \text{if } P < P_5 \\ a_1 * P + b_1 & \text{if } P_5 \leq P < P_7;\ a_1 = \dfrac{1}{P_7 - P_5},\ b_1 = \dfrac{P_7}{P_7 - P_5} \\ a_2 * P + b_2 & \text{if } P_7 \leq P < P_9;\ a_2 = -\dfrac{1}{P_9 - P_7},\ b_2 = \dfrac{P_9}{P_9 - P_7} \\ 0 & \text{if } P \geq P_9 \end{cases} \tag{7.8}$$

- VeryHigh: VeryHigh set represents likely probabilities that the HMM model produces normal sequences. Its membership function is defined as:

$$\mu_{VeryHigh}(P) = \begin{cases} 0 & \text{if } P < P_8 \\ a * P + b & \text{if } P_8 \leq P < P_{10};\ a = \dfrac{1}{P_{10} - P_8},\ b = \dfrac{P_{10}}{P_{10} - P_8} \\ 1 & \text{if } P_{10} \leq P \leq P_{11} \end{cases} \tag{7.9}$$

Fuzzy sets for sequence distance

We use the *Hamming distance* to measure the distance between two same-length sequences of system calls. The distance between a test sequence and normal sequences of the normal database is defined as:

$$\mathrm{D} = \min\{\mathrm{D(j)}\},\ \mathrm{j} = 1, \mathrm{N}$$

where N is the number of normal sequences in the normal database, and $D(j)$ is the Hamming distance between the test sequence and the normal sequence j. The distance D is then normalized to the length k of the sequence as: $\hat{D} = D/k$. Hereafter, we use the normalized distance as the sequence distance.

Based on the characteristics of the sequence distance, we create one crisp set and three fuzzy sets for this input parameter, as shown in Figure 7.5. The crisp set, namely Zero is used to

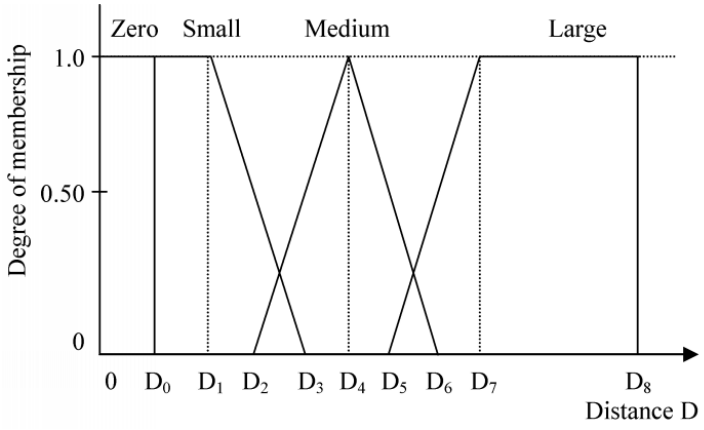

Figure 7.5: Fuzzy sets of sequence distance D

represent the "matched" cases when test sequences are found in normal database. Therefore, the sequence distance values of Zero set are always 0. Three fuzzy sets, namely Small, Medium and Large are used to represent the sequence distance when test sequences are "mismatched" or not found in normal database. Among three distance fuzzy sets, Small and Large are trapezium sets while Medium is a triangle set. The universe of the sequence distance fuzzy sets is $[0.0, 1.0]$, with 0.0 representing "matched" cases, and 1.0 representing "mismatched" cases in which a test sequence is entirely different from normal sequences in normal database. The membership functions of these fuzzy sets are also determined empirically by analyzing the distance pattern of sequences. For example, for the sequence length $k = 11$, we have $D_0 = 0.0$, $D_1 = 0.2$, $D_2 = 0.3$, $D_3 = 0.4$, $D_4 = 0.5$, $D_5 = 0.6$, $D_6 = 0.7$, $D_7 = 0.8$, and $D_8 = 1.0$.

The sequence distance fuzzy sets are defined as:

- Zero: Zero set has only a single distance $D = 0$. Its membership function is defined as:

$$\mu_{Zero}(D) = \begin{cases} 1 & \text{if } D = 0 \\ 0 & \text{if } D > 0 \end{cases} \tag{7.10}$$

- Small: Small set represents small differences between a test sequence and normal sequences. Its membership function is defined as:

$$\mu_{Small}(D) = \begin{cases} 1 & \text{if } D_0 \leq D < D_1 \\ a * D + b & \text{if } D_1 \leq D < D_3;\ a = -\dfrac{1}{D_3 - D_1},\ b = \dfrac{D_3}{D_3 - D_1} \\ 0 & \text{if } D \geq D_3 \end{cases} \tag{7.11}$$

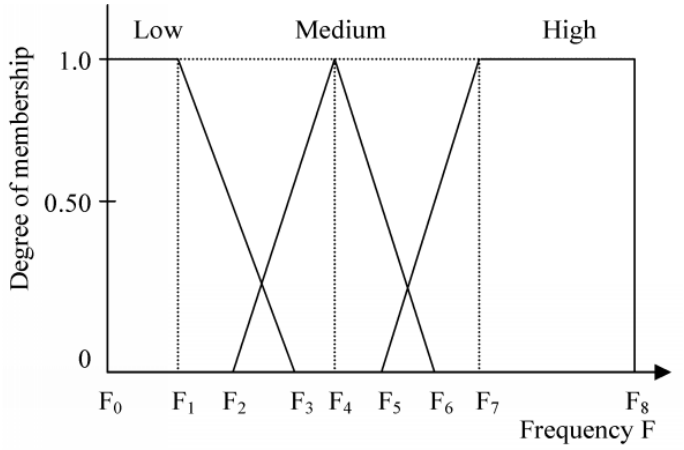

Figure 7.6: Fuzzy sets of sequence occurrence frequency F

- Medium: Medium set represents medium differences between a test sequence and normal sequences. Its membership function is defined as:

$$\mu_{Medium}(D) = \begin{cases} 0 & \text{if } D < D_2 \\ a_1 * D + b_1 & \text{if } D_2 \leq D < D_4;\ a_1 = \dfrac{1}{D_4 - D_2},\ b_1 = \dfrac{D_4}{D_4 - D_2} \\ a_2 * D + b_2 & \text{if } D_4 \leq D < D_6;\ a_2 = -\dfrac{1}{D_6 - D_4},\ b_2 = \dfrac{D_6}{D_6 - D_4} \\ 0 & \text{if } D \geq D_6 \end{cases} \tag{7.12}$$

- Large: Large set represents big differences between a test sequence and normal sequences. Its membership function is defined as:

$$\mu_{Large}(D) = \begin{cases} 0 & \text{if } D < D_5 \\ a * D + b & \text{if } D_5 \leq D < D_7;\ a = \dfrac{1}{D_7 - D_5},\ b = \dfrac{D_7}{D_7 - D_5} \\ 1 & \text{if } D_7 \leq D \leq D_8 \end{cases} \tag{7.13}$$

Fuzzy sets for sequence frequency

The occurrence frequency of a test sequence is the number of occurrences of the sequence in training data, which is recorded in normal database. Therefore, if a test sequence is not found in normal database (a mismatched sequence), the sequence frequency is 0.

We define three fuzzy sets for the sequence frequency, namely Low, Medium and High, as shown in Figure 7.6. Low and High are trapezium sets and Medium is a triangle set. The universe

of the sequence frequency fuzzy sets is $[0,\ T]$, where T is number of sequences formed from the training set. The membership functions of these fuzzy sets are also determined empirically by analyzing the distribution of sequence occurrences in the training data recorded in the normal database. The sequence frequency fuzzy sets are defined as:

- Low: Low set represents small number of occurrences of short sequences in training data. Its membership function is defined as:

$$\mu_{Low}(F) = \begin{cases} 1 & \text{if } F_0 \leq F < F_1 \\ a * F + b & \text{if } F_1 \leq F < F_3;\ a = -\dfrac{1}{F_3 - F_1},\ b = \dfrac{F_3}{F_3 - F_1} \\ 0 & \text{if } F \geq F_3 \end{cases} \tag{7.14}$$

- Medium: Medium set represents medium number of occurrences of short sequences in training data. Its membership function is defined as:

$$\mu_{Medium}(F) = \begin{cases} 0 & \text{if } F < F_2 \\ a_1 * F + b_1 & \text{if } F_2 \leq F < F_4;\ a_1 = \dfrac{1}{F_4 - F_2},\ b_1 = \dfrac{F_4}{F_4 - F_2} \\ a_2 * F + b_2 & \text{if } F_4 \leq F < F_6;\ a_2 = -\dfrac{1}{F_6 - F_4},\ b_2 = \dfrac{F_6}{F_6 - F_4} \\ 0 & \text{if } F \geq F_6 \end{cases} \tag{7.15}$$

- High: High set represents high number of occurrences of short sequences in training data. Its membership function is defined as:

$$\mu_{High}(F) = \begin{cases} 0 & \text{if } F < F_5 \\ a * F + b & \text{if } F_5 \leq F < F_7;\ a = \dfrac{1}{F_7 - F_5},\ b = \dfrac{F_7}{F_7 - F_5} \\ 1 & \text{if } F_7 \leq F \leq F_8 \end{cases} \tag{7.16}$$

Fuzzy sets for anomaly score

Anomaly score of a test sequence is the score assigned to the test sequence, which is used to determine the status of the test sequence. In our fuzzy-based model, we have created two fuzzy sets, namely Normal and Abnormal for the output sequence anomaly score parameter, as shown in Figure 7.7. Both Normal and Abnormal are trapezium sets. The universe of the sequence anomaly score fuzzy sets is $[0.0, 1.0]$, with 0.0 representing absolute normal and 1.0 representing absolute abnormal. The anomaly score fuzzy sets are used in the defuzzification

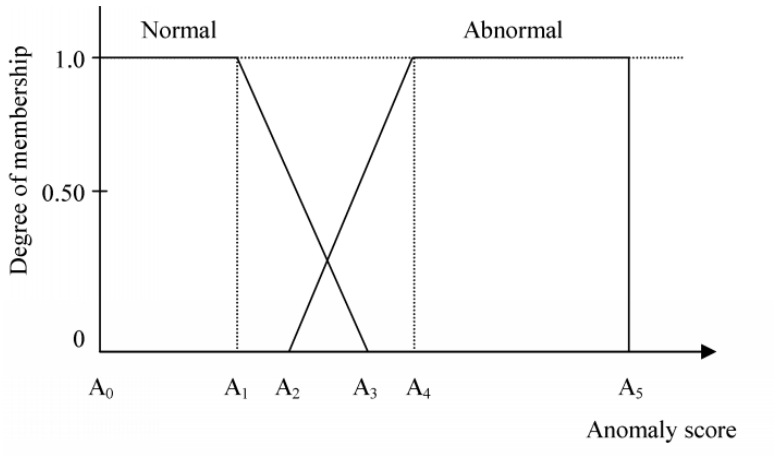

Figure 7.7: Fuzzy sets of sequence anomaly score A

process to convert the output fuzzy set to the actual anomaly score of the test sequence (refers to Figure 7.3). The Normal and Abnormal sets are defined as follows:

- Normal: Normal set represents the anomaly scores for normal sequences. Its membership function is defined as:

$$\mu_{Normal}(A) = \begin{cases} 1 & \text{if } A_0 \leq A < A_1 \\ a * A + b & \text{if } A_1 \leq A < A_3;\ a = -\dfrac{1}{A_3 - A_1},\ b = \dfrac{A_3}{A_3 - A_1} \\ 0 & \text{if } A \geq A_3 \end{cases} \tag{7.17}$$

- Abnormal: Abnormal set represents the anomaly scores for abnormal sequences. Its membership function is defined as:

$$\mu_{Abnormal}(A) = \begin{cases} 0 & \text{if } A < A_2 \\ a * A + b & \text{if } A_2 \leq A < A_4;\ a = \dfrac{1}{A_2 - A_4},\ b = \dfrac{A_4}{A_4 - A_2} \\ 1 & \text{if } A_4 \leq A \leq A_5 \end{cases} \tag{7.18}$$

7.3.1.2 Creation of fuzzy rules

Since the input sequence parameters of the fuzzy rules, which include probability P, distance D and frequency F, are generated by the HMM model and the normal database, the definition of fuzzy rules for our fuzzy-based detection scheme also follows the assumptions for the classification of test sequences by the normal database and the HMM model. Specifically, these assumptions are as follows:

1. A sequence is produced with a likely probability by the HMM model is considered to be normal.

2. A sequence is produced with an unlikely probability by the HMM model is considered to be abnormal.

3. A mismatched sequence (which is not found in the normal database) is more suspicious than a matched sequence (which is found in the normal database). The larger the distance between a test sequence and normal sequences is the more likely the test sequence is abnormal.

4. A matched sequence with low occurrence frequency is more suspicious than a sequence with high occurrence frequency.

In addition, it should be noted in the definition of fuzzy rules that there exists a close relationship between the distance and the frequency of a test sequence. If a test sequence is labelled as "mismatched" by the normal database, its distance is greater than 0 ($D > 0$, and $D \in$ {Small, Medium, Large}). In this case, the sequence frequency is 0 ($F = 0$). On the other hand, if a test sequence is labelled as "matched" by the normal database, its distance is 0 ($D = 0$, and $D \in$ {Zero}). In this case, the sequence frequency is greater than 0 ($F > 0$).

Based on the above mentioned assumptions, we devise a set of fuzzy rules as follows:

Rule 1 : IF probability IS *VeryLow* AND distance IS *Large* THEN the test sequence IS *abnormal*.

Rule 2 : IF probability IS *VeryLow* AND distance IS *Medium* THEN the test sequence IS *abnormal*.

Rule 3 : IF probability IS *VeryLow* AND distance IS *Small* THEN the test sequence IS *abnormal*.

Rule 4 : IF probability IS *VeryLow* AND distance IS *Zero* AND frequency IS *Low* THEN the test sequence IS *abnormal*.

Rule 5 : IF probability IS *VeryLow* AND distance IS *Zero* AND frequency IS *Medium* THEN the test sequence IS *abnormal*.

Rule 6 : IF probability IS *VeryLow* AND distance IS *Zero* AND frequency IS *High* THEN the test sequence IS *normal*.

Rule 7 : IF probability IS *Low* AND distance IS *Large* THEN the test sequence IS *abnormal*.

Rule 8 : IF probability IS *Low* AND distance IS *Medium* THEN the test sequence IS *abnormal*.

Rule 9 : IF probability IS *Low* AND distance IS *Small* THEN the test sequence IS *abnormal*.

Rule 10 : IF probability IS *Low* AND distance IS *Zero* AND frequency IS *Low* THEN the test sequence IS *abnormal*.

Rule 11 : IF probability IS *Low* AND distance IS *Zero* AND frequency IS *Medium* THEN the test sequence IS *normal*.

Rule 12 : IF probability IS *Low* AND distance IS *Zero* AND frequency IS *High* THEN the test sequence IS *normal*.

Rule 13 : IF probability IS *High* AND distance IS *Large* THEN the test sequence IS *abnormal*.

Rule 14 : IF probability IS *High* AND distance IS *Medium* THEN the test sequence IS *normal*.

Rule 15 : IF probability IS *High* AND distance IS *Small* THEN the test sequence IS *normal*.

Rule 16 : IF probability IS *High* AND distance IS *Zero* THEN the test sequence IS *normal*.

Rule 17 : If probability is *VeryHigh* THEN the test sequence is *normal*.

7.3.2 Sequence classification using fuzzy inference

As shown in Figure 7.3, the decision-making process of the fuzzy inference engine to classify each sequence of system calls begins when the engine accepts the sequence parameters as the input, and ends when the output is produced. This process is completed in three phases: (i) fuzzification, (ii) fuzzy inference, and (iii) defuzzification.

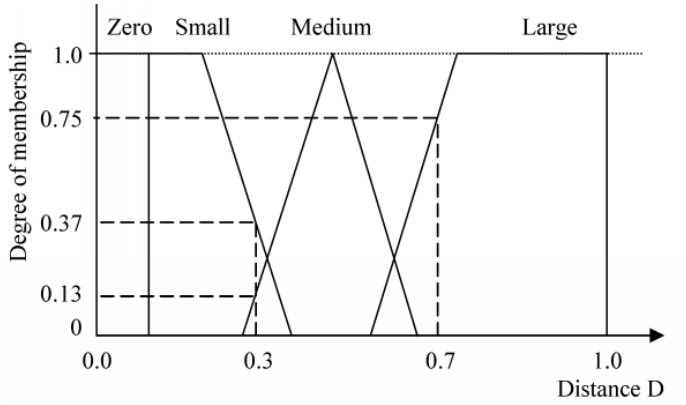

Figure 7.8: Fuzzification of sequence distance parameter

7.3.2.1 Fuzzification

Fuzzification is the process of transforming crisp input values into linguistic values which usually are fuzzy sets. There are two tasks involved in the fuzzification process: (a) input values are converted into linguistic values which are represented by fuzzy sets, and (b) membership functions are applied to compute the degree of truth for each matched fuzzy set.

Figure 7.8 illustrates two examples, in which input values of sequence distance are fuzzified. In the first example, for the input sequence distance $D = 0.7$, the matched fuzzy set for this input is Large. And Large's membership function is used to determine the degree of truth $\mu(0.7) = 0.75$. In the second example, two fuzzy sets of Small and Medium are found to be matched to the input sequence distance $D = 0.3$. Applying membership functions of Small and Medium to the distance input we get the degrees of truth of $\mu_{Small}(0.3) = 0.37$ and $\mu_{Medium}(0.3) = 0.13$.

7.3.2.2 Defuzzification

Defuzzification is the process of transforming the fuzzy value into a crisp value. In our fuzzy inference engine, the output anomaly score fuzzy set is defuzzified to produce the sequence's anomaly score. There are many defuzzification techniques available. Among them, some common defuzzification techniques can be listed as follows:

- Centriod (also known as center-of-area, center-of-gravity) method: This method deter-

mines the "center point" of the output fuzzy set as:

$$\bar{x} = \frac{\int_U \mu(x) * x dx}{\int_U \mu(x) dx} \tag{7.19}$$

where $\bar{x}$ is the output crisp value, U is the universe, $x \in U$, and $\mu(x)$ is the membership function of the output fuzzy set.

- Max-membership method: This method selects an element with maximum value of the degree of truth of the output fuzzy set.

$$\bar{x} : \mu(\bar{x}) = \max(\mu(x)), \quad x \in U \tag{7.20}$$

There exist cases that there is a set of points $\{\bar{x}_1, ..., \bar{x}_n\}$ which have a common maximum membership value. In these cases, $\bar{x}$ can be selected one of the following options:

$$\bar{x} = \min(\bar{x}_1, ..., \bar{x}_n) \tag{7.21}$$

$$\bar{x} = \max(\bar{x}_1, ..., \bar{x}_n) \tag{7.22}$$

$$\bar{x} = \frac{\min(\bar{x}_1, ..., \bar{x}_n) + \max(\bar{x}_1, ..., \bar{x}_n)}{2} \tag{7.23}$$

- Weighted average method: In this method, the output is calculated as the weighted average of the maximum membership values of fuzzy sets in the output region.

$$\bar{x} = \frac{\sum \mu(x_i) * x_i}{\sum \mu(x_i)} \tag{7.24}$$

where $\mu(x_i)$ is the maximum membership value of the fuzzy set i in the output region.

7.3.2.3 Fuzzy inference

In the fuzzy inference phase, all rules in fuzzy rule-base are applied to input parameters in order to produce an output. For each rule, first, each premise is evaluated, and then all premises connected by an AND are combined by taking the smallest value of their degree of membership as the combination value of rule's truth value. The final output fuzzy set of the fuzzy rule-base is the OR combination of results of all individual rules that fire. It is noted that the truth value of a rule that fires is non-zero. The output fuzzy set is defuzzified to produce a crisp output value.

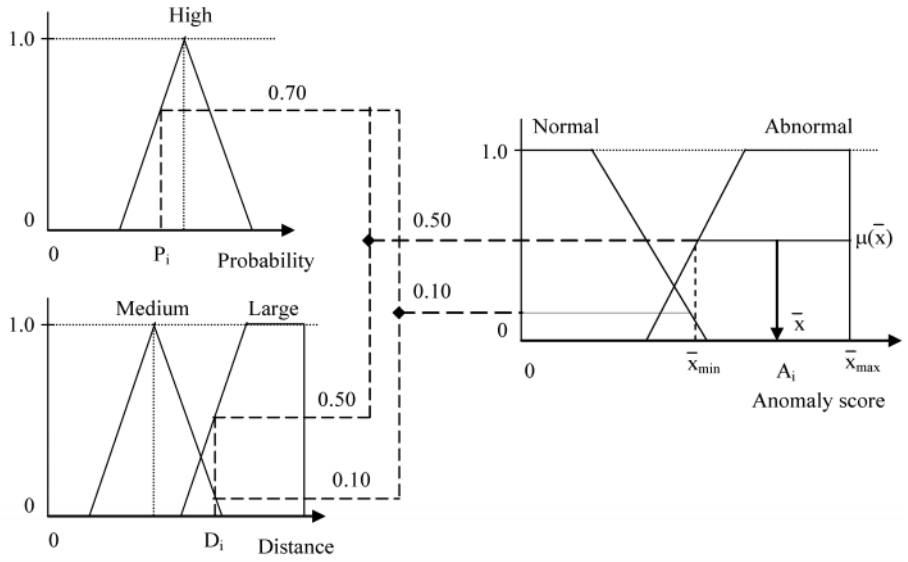

Figure 7.9: Calculation of sequence anomaly score using fuzzy inference

Figure 7.9 shows an example of sequence anomaly score calculation using fuzzy inference. The input sequence parameters are probability $P_i = e^{-35}$, distance $D_i = 0.60$ and frequence $F_i = 0$. The fuzzy inferrencing process can be described as follows:

1. Fuzzification: Represent input values (P_i, D_i, F_i) by corresponding fuzzy sets:

   - P_i belongs to the sequence probability High set.
   - D_i belongs to the sequence distance Medium and Large sets.
   - F_i belongs to the sequence frequency Low set.

2. Inference process: Apply fuzzy rules to inputs; only Rule 13 and Rule 14 fire (the fuzzy rule-base was presented in Section 7.3.1.2). It is noted that Rule 13 and Rule 14 support only two input parameters P_i and D_i. Therefore, F_i does not appear in Figure 7.9.

   - Rule 13 fires: the result of this rule is an abnormal fuzzy set with the combined truth value of 0.50.
   - Rule 14 fires: the result of this rule is an normal fuzzy set with the combined truth value of 0.10.

   The final output fuzzy set is the combination of two output sets produced by Rule 13 (part of Abnormal set) and Rule 14 (part of Normal set).

3. Defuzzification: Determine the output anomaly score based on the output fuzzy set from interence process. We use the Max-membership method to compute the output anomaly score as given in equations (7.20) and (7.23). Based on equation (7.20), we have $\mu(\bar{x}) = \max(0.50, 0.10) = 0.50$. Based on equation (7.23), we get $\bar{x} = (\bar{x}_{min} + \bar{x}_{max})/2$. The output anomaly score is $A_i = \bar{x} = (0.40 + 1.0)/2 = 0.70$.

7.4 Detecting Program Anomalies: A Fuzzy-based Detection Scheme

In this section, we incorporate the fuzzy inference engine for sequence classification, discussed in Section 7.3 to form a fuzzy-based detection scheme. As an extension of the two-layer scheme presented in Section 5.2, the fuzzy-based scheme utilizes fuzzy inference techniques to combine inputs from multiple sources, in order to further increase the detection accuracy.

Figure 7.10 shows the structure of the fuzzy-based detection scheme to evaluate short sequences of system calls of test traces for possible intrusions. The fuzzy-based detection scheme is built as follows:

1. **Building a normal database, an HMM model and fuzzy sets from training data**:

   (a) Normal database: The normal database is an ordered list of all unique short sequences of system calls found in training data. The database is created from the training data of normal traces of system calls using the method given in [25, 26]. All short sequences in the normal database have the same number of system calls k, and k is used as the short sequence length. In addition, the frequency of each short sequence, that is the number of occurrences of that sequence in training data, is also recorded in the normal database.

   (b) HMM model: The HMM model is trained from normal traces of system calls using HMM incremental training with optimal initialization, as discussed in Section 6.1.2.

   (c) Fuzzy sets: The fuzzy sets are created, as discussed in Section 7.3.1 of this chapter.

2. **Fuzzy-based testing**: First, test short sequences are formed from test traces of system calls using the sliding window method. The length of short sequences is k system calls

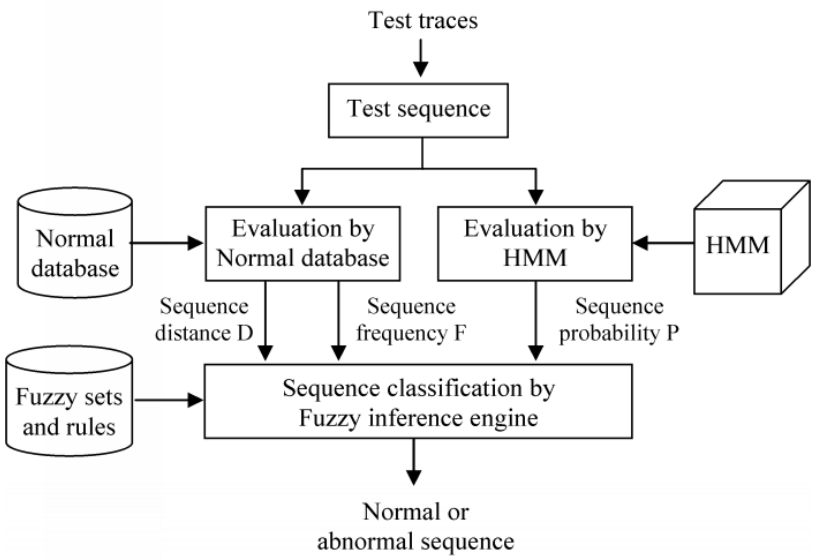

Figure 7.10: Fuzzy-based detection scheme: evaluation of short sequences for anomalies

and is equal to the length of short sequences in the normal database. Then, each short sequence is evaluated in two steps as follows:

(a) Evaluation of the short sequence by the normal database and by the HMM model: In this step, the normal database and the HMM model are used to compute the input parameters for fuzzy inference engine in next step. The input parameters include the sequence probability P generated by the HMM model, and the sequence distance D and frequency F produced by the normal database.

(b) Classification of the test sequence by the fuzzy inference engine: In this step, the fuzzy inference engine applies the fuzzy sets and rules to interpret the input parameters in order to produce the output which is the status of the short sequence: normal or abnormal.

7.5 Experimental Design and Results

In order to evaluate the performance of our proposed fuzzy-based detection scheme, we conducted experiments to measure the false positive rate and the detection rate of the fuzzy-based scheme. Then, the performance results are compared to those of the two-layer detection scheme presented

in Section 5.2.2 and the normal database scheme [25].

7.5.1 Data sets

We use *sendmail* traces of system calls collected in a synthetic environment[1], as given in [132]. The format of system call traces and the data collection procedures were discussed in Section 5.1.1. The data sets include:

1. *Normal traces* are traces collected during the program's normal activity. Normal traces of *sendmail* program include 2 traces with the total of 1,595,612 system calls.

2. *Abnormal traces* are traces that come from a program's abnormal runs generated by known intrusions. The *sendmail* abnormal traces consist of 1 trace of *sm5x*, 1 trace of *sm565a*, 2 traces of *syslog-local*, and 2 traces of *syslog-remote* intrusions.

7.5.2 Experimental design

In order to measure the detection rate and the false alarm rate of our fuzzy-based detection model, our experiments were designed as follows:

1. **Measurement of the false positive rate**: In this test, we use the proposed fuzzy-based detection scheme to classify normal traces of system calls which are not used in the construction of the normal database, the HMM model and fuzzy sets. Since the normal traces do not contain any intrusions, any reported alarms are considered false positive alarms. This experiment was set up as follows:

   (a) Select first 1,000,000 system calls of *sendmail* normal traces as the full training set.

   (b) Form 4 training sets which account for 30%, 50%, 80% and 100% of the size of the full training set.

   (c) Construct normal databases and HMM models from these training sets. The chosen values for the length of short sequences are $k = 5, 11$ and 15 system calls.

   (d) For each training set and on each selected sequence length, assign membership function values for fuzzy sets of the three sequence parameters as discussed in Section 7.3.1.

[1] A type of artificial or simulated environment created to test programs. Detailed description of synthetic environments is given in [26].

(e) Select three test traces, each trace of 50,000 system calls from *sendmail* normal traces, which are not used in the training process to test for false positive alarms of our scheme and the normal database scheme [25]. Reported abnormal short sequences are counted for each test trace.

2. **Measurement of anomaly signals and detection rate**: In this test, we use the proposed fuzzy-based scheme to classify abnormal traces to find possible intrusions. Since the abnormal traces have been collected from the program's abnormal runs generated by known intrusions, reported alarms in this case can be considered true alarms or detected intrusions. This experiment was implemented as follows:

   (a) Construct a normal database and an HMM model for *sendmail* program from normal traces of system calls. We choose length of short sequences $k = 11$ to construct the normal database from normal traces, and to form test short sequences from abnormal traces.

   (b) Assign membership function values for fuzzy sets of the three sequence parameters, as discussed in Section 7.3.1.

   (c) Use the proposed fuzzy-based detection scheme to evaluate abnormal traces to find abnormal sequences.

   (d) Use temporally local regions to group individual abnormal sequences to measure the anomaly signals, as discussed in Section 5.1.2. The selected region length is $r = 20$.

7.5.3 Experimental results

7.5.3.1 False positive rate

Table 7.1 shows the false positive rate for three test traces with total of 150,000 system calls (each trace consists of 50,000 system calls), reported by the normal database scheme, two-layer detection scheme and fuzzy-based detection scheme on different training sets, with short sequence length $k = 5, 11$ and 15. The overall false positive rate is calculated as the ratio of the total number of reported abnormal short sequences to the total number of short sequences in the three test traces. The total number of short sequences in the test traces is dependent on the short sequence length and is also given in Table 7.1.

Table 7.1: Overall false positive rate of the normal database scheme, the two-layer detection scheme and the fuzzy-based detection scheme with the short sequence length, $k = 5, 11$ and 15.

Training data sets (% of full data set)	**Normal database scheme (%)**	**Two-layer scheme (%)**	**Fuzzy-based scheme (%)**
Sequence length, $k = 5$; 3 test traces with the total of 149,988 sequences			
30%	0.131	0.112	0.067
50%	0.099	0.079	0.057
80%	0.094	0.069	0.049
100%	0.094	0.069	0.049
Sequence length, $k = 11$; 3 test traces with the total of 149,970 sequences			
30%	0.194	0.170	0.099
50%	0.155	0.115	0.081
80%	0.150	0.107	0.077
100%	0.147	0.107	0.077
Sequence length, $k = 15$; 3 test traces with the total of 149,958 sequences			
30%	0.225	0.164	0.107
50%	0.176	0.121	0.091
80%	0.174	0.116	0.085
100%	0.171	0.116	0.085

It can be seen from Table 7.1 that the false positive rate of the fuzzy-based detection scheme is much lower than that of the normal database scheme [25]. For example, the fuzzy-based detection scheme produced 48.23%, 48.89% and 50.96% fewer false positive alarms than the normal database scheme for the training set of 80% of full set, with the sequence length k = 5, k = 11 and k = 15, respectively.

It is also noted that there is a significant reduction in the false positive rate of the fuzzy-based detection scheme, compared to that of the two-layer detection scheme. For example, the fuzzy-based detection scheme produced 29.81%, 28.13% and 26.44% fewer false positive alarms than the two-layer detection scheme for the training set of 80% of full set, with sequence length

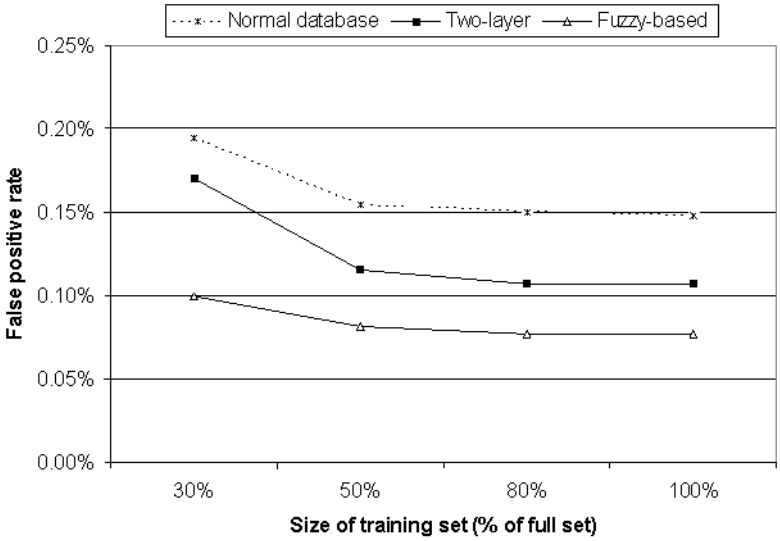

Figure 7.11: The relationship between the size of training sets and the false positive rate with $k = 11$

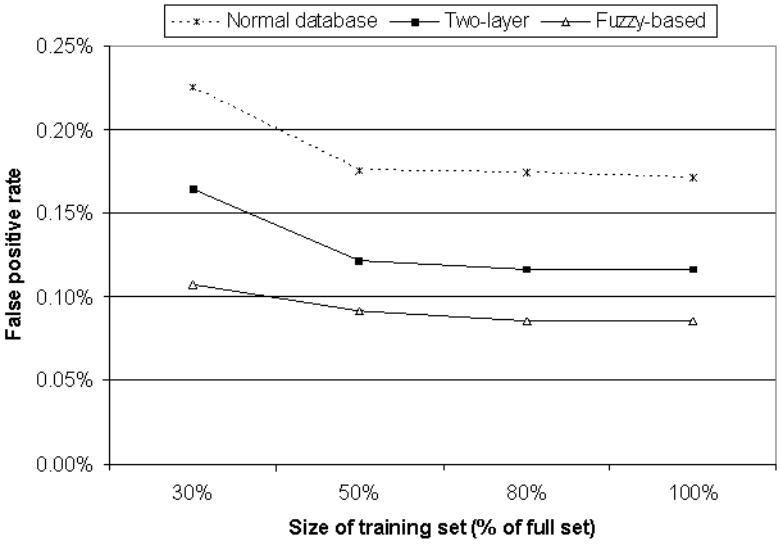

Figure 7.12: The relationship between the size of training sets and the false positive rate with $k = 15$

k = 5, k = 11 and k = 15, respectively (refers to Table 7.1).

Figures 7.11 and 7.12 describe the dependence of the false positive rate on the size of the training sets with the short sequence length $k = 11$ and $k = 15$ respectively. When the size of the training set increases, the false positive rate of the normal database scheme and the two-layer

scheme decreases considerably, especially from the training set of 30% of the full set to the set of 50% of the full set. Since the fuzzy-based scheme has already achieved a low false positive rate at the set of 30% of the full set, there is only a small reduction of the false positive rate when the size of the training set increases.

7.5.3.2 Anomaly signals and the detection rate

Table 7.2 shows a summary of the detection results of the two-layer scheme and the fuzzy-based scheme for some abnormal traces embedded with known intrusions. The detection performance results of the two-layer scheme are taken from Section 5.2. In both proposed methods, we measure anomaly signals based on temporally local regions. The anomaly score A of a region is computed as the ratio of the number of detected abnormal short sequences in the region to the length of the region r, as given in Equation (5.1). The average of anomaly scores is computed over abnormal regions which have the anomaly score A, with $A \geq \hat{A}$, where $\hat{A}$ is the region score threshold, $\hat{A} = 40.0\%$.

It can be seen from Table 7.2 that the fuzzy-based scheme produced significantly better detection results than the two-layer scheme in terms of the number of detected abnormal regions and the generated anomaly signal level. For "sm5x" trace, the rates of detected abnormal regions are 31.58% and 67.11% by the two-layer scheme and fuzzy-based scheme, respectively. Also for this test trace, the fuzzy-based scheme generated an average of anomaly scores of 72.55%, compared to the average of anomaly scores of 60.42% produced by the two-layer scheme.

Figures 7.14, 7.13, 7.15, 7.16, 7.17, and 7.18 show the anomaly signals produced by the two-layer scheme and the fuzzy-based scheme for some abnormal traces, namely *s5mx, sm565a, syslog-remote No1, syslog-remote No2, syslog-local No1* and *syslog-local No2*, respectively, at the short sequence length $k = 11$. It is noted that we measure anomaly signals based on temporally local regions for both methods, as discussed in Section 5.1.2.3. It can be seen from these figures that the fuzzy-based scheme generated much stronger and clearer anomaly signals than the two-layer scheme. For both schemes, all embedded intrusions are detected correctly at the region score threshold $\hat{A} = 40.0\%$ for all tested traces.

Table 7.2: Detection results produced by the normal database scheme [25], by the two-layer scheme and by the fuzzy-based scheme for some abnormal traces with the short sequence length $k = 11$, region length $r = 20$ and the score threshold of abnormal regions, $\hat{A} = 40.0\%$.

Name of test abnormal traces	**% detected abnormal sequences by [25] (#)**	**Percent of detected abnormal regions (*)**		**Average of scores of abnormal regions (!)**	
		Two-layer(%)	**Fuzzy-based(%)**	**Two-layer(%)**	**Fuzzy-based(%)**
sm565a	0.60	38.46	76.92	68.00	88.00
sm5x	2.70	31.58	67.11	60.42	72.55
syslog-local No1	5.10	12.00	60.00	73.33	84.67
syslog-local No2	1.70	16.67	60.26	71.54	86.49
syslog-remote No1	4.00	28.26	67.39	72.31	86.53
syslog-remote No2	5.30	24.68	61.04	74.74	83.40

(#) Results produced by the normal database scheme are extracted from Table 3 of [25].
(*) The percentage of the number of detected abnormal regions out of the total number of regions of the test trace.
(!) The average of anomaly signals is computed over abnormal regions, those with $A \geq \hat{A}$.

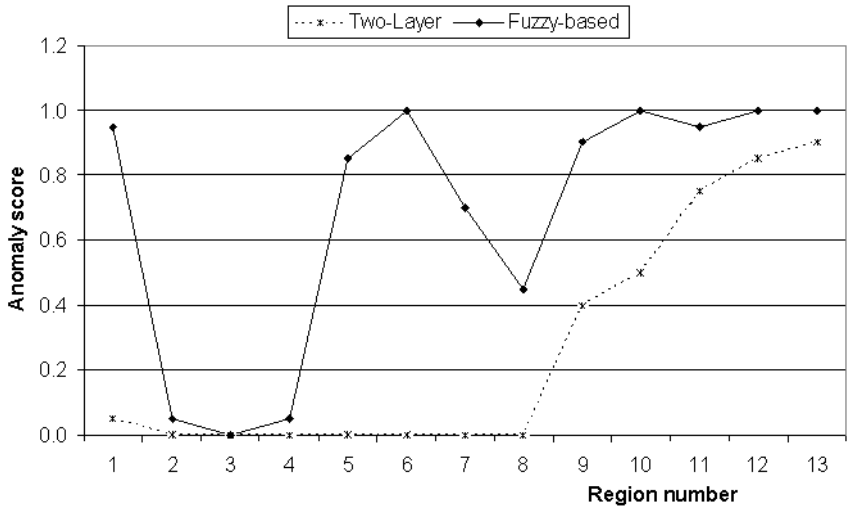

Figure 7.13: Anomaly signal generated for *sm565a* abnormal trace by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

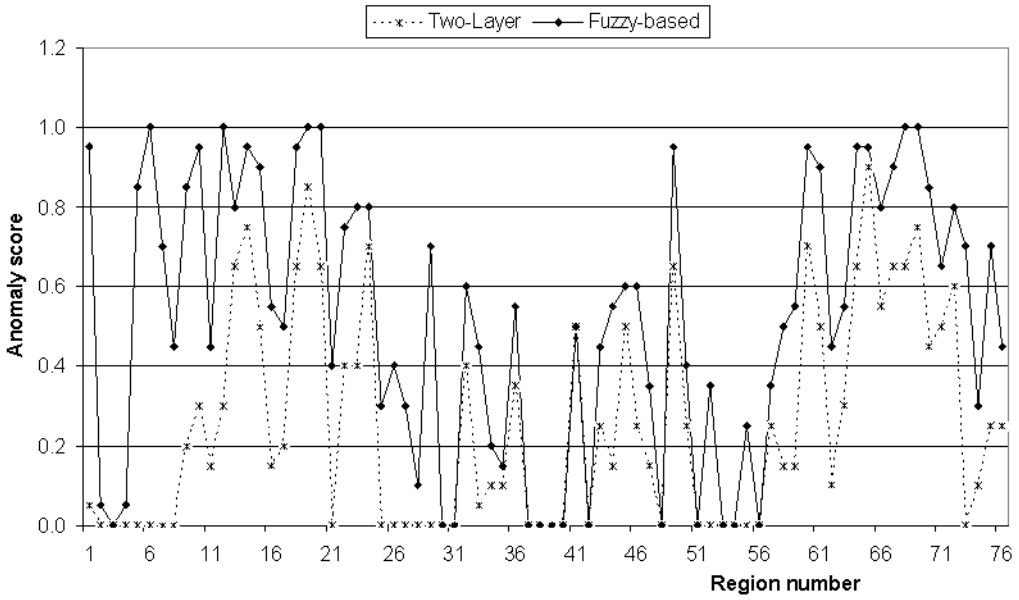

Figure 7.14: Anomaly signal generated for *s5mx* abnormal trace by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

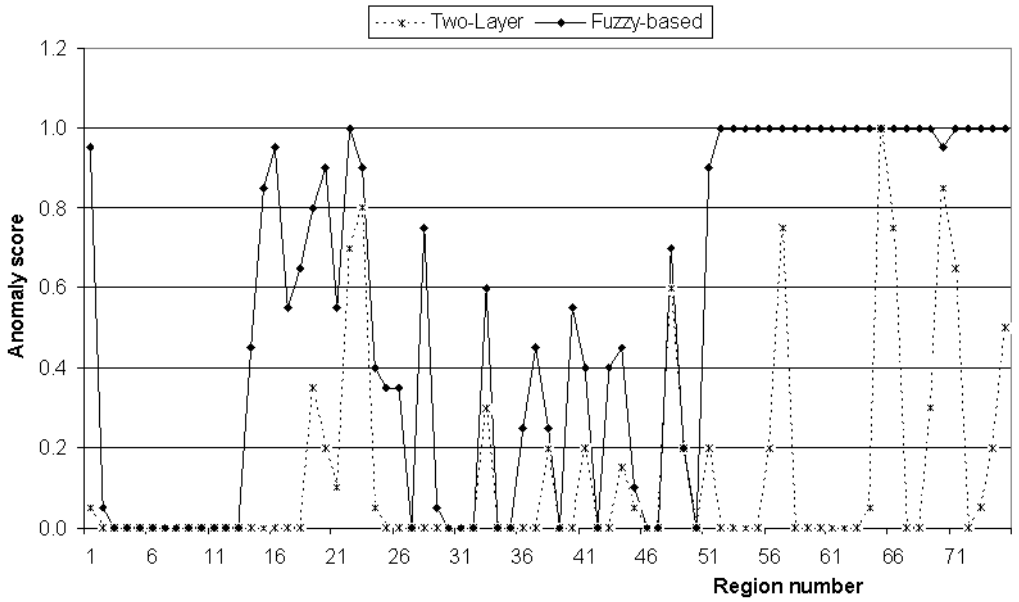

Figure 7.15: Anomaly signal generated for *syslog-local* abnormal trace No. 1 by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

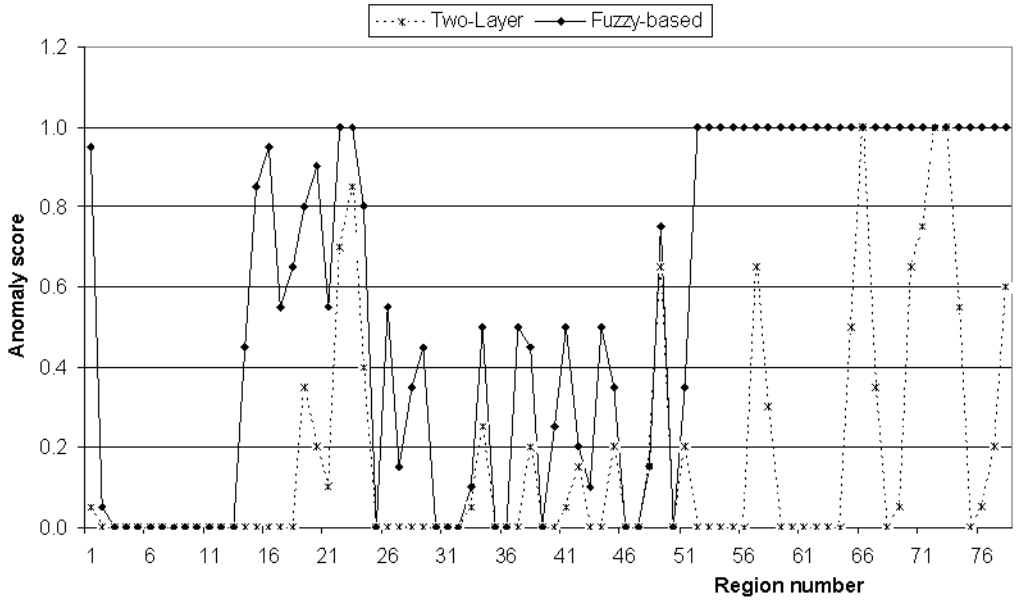

Figure 7.16: Anomaly signal generated for *syslog-local* abnormal trace No. 2 by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

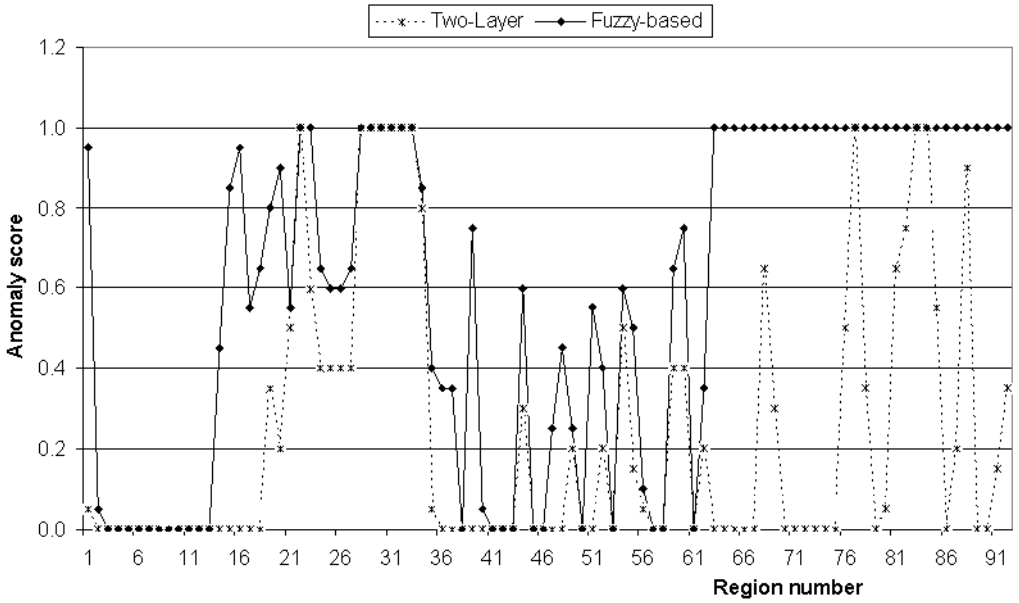

Figure 7.17: Anomaly signal generated for *syslog-remote* abnormal trace No. 1 by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

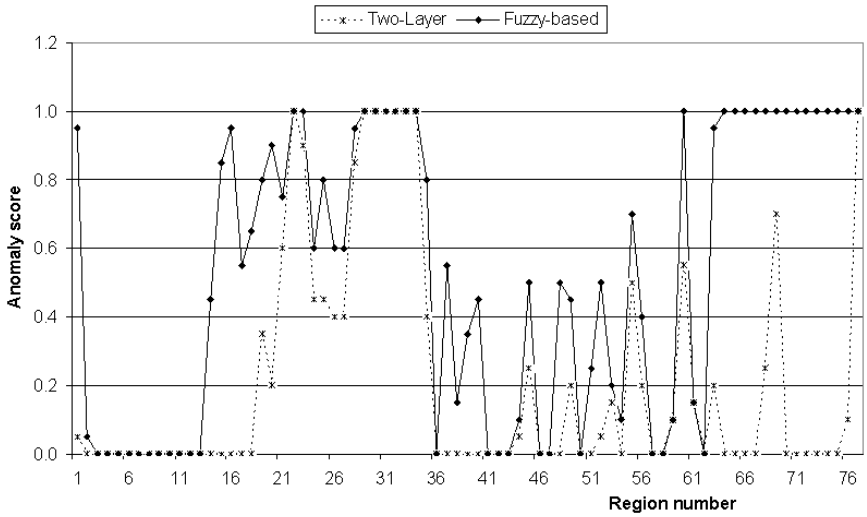

Figure 7.18: Anomaly signal generated for *syslog-remote* abnormal trace No. 2 by two-layer and fuzzy-based schemes with short sequence length k = 11, region length $r = 20$.

7.6 Summary

In this chapter, we presented a program anomaly detection scheme which utilizes the fuzzy inference techniques to increase the detection rate and to reduce the false positive rate. In the proposed fuzzy-based scheme, a normal database and an HMM model are used to produce inputs for the fuzzy inference engine. The fuzzy inference engine evaluates each sequence of system calls by combining the sequence's multiple inputs through a fuzzy reasoning process to correctly label the sequence. This in turn helps improve the detection accuracy.

Our experimental results showed that the proposed fuzzy-based scheme correctly detected "sm5x", "sm565a", "syslog-local" and "syslog-remote" intrusions in all tested abnormal traces. The fuzzy-based scheme produced about 48% fewer false positive alarms than the normal database model [25], and it produced about 28% fewer false positive alarms than the the two-layer scheme (refers to Table 7.1 (page 149), the case of $k = 11$ with data sets of 50%, 80% and 100% of full data set). Futhermore, the fuzzy-based scheme generated much stronger anomaly signals for all tested abnormal traces than the two-layer scheme and the normal database scheme, as results presented in Table 7.2 (page 152).

In Chapter 8, we present our complete model for program anomaly detection using system calls. The fuzzy-based detection scheme will be an important component of the complete model.

Chapter 8

Putting It All Together: The Proposed Program Anomaly Detection Model

In this chapter, a complete model for program anomaly detection using system calls is presented, which is a combination of all model components, discussed in previous chapters of this part. The HMM modelling is used to characterize normal program behaviour, and the HMM incremental training scheme is used to construct the HMM model from training data. Instead of using crisp conditions and thresholds to represent the normalcy and the abnormalcy of sequences of system calls, fuzzy sets and rules are used to represent the space and combined conditions of sequence parameters. To detect anomalies and possible intrusions, the fuzzy inference is used to combine the input sequence information from multiple sources to evaluate sequences of system calls.

8.1 The Proposed Program Anomaly Detection Model

Figure 8.1 shows the proposed detection model developed in two stages: (a) model training and (b) model testing. In the training stage, the detection model is constructed from the training data which is normal traces of system calls of the program. In the testing stage, the built detection model is used to evaluate test traces of system calls to find anomalies and possible intrusions.

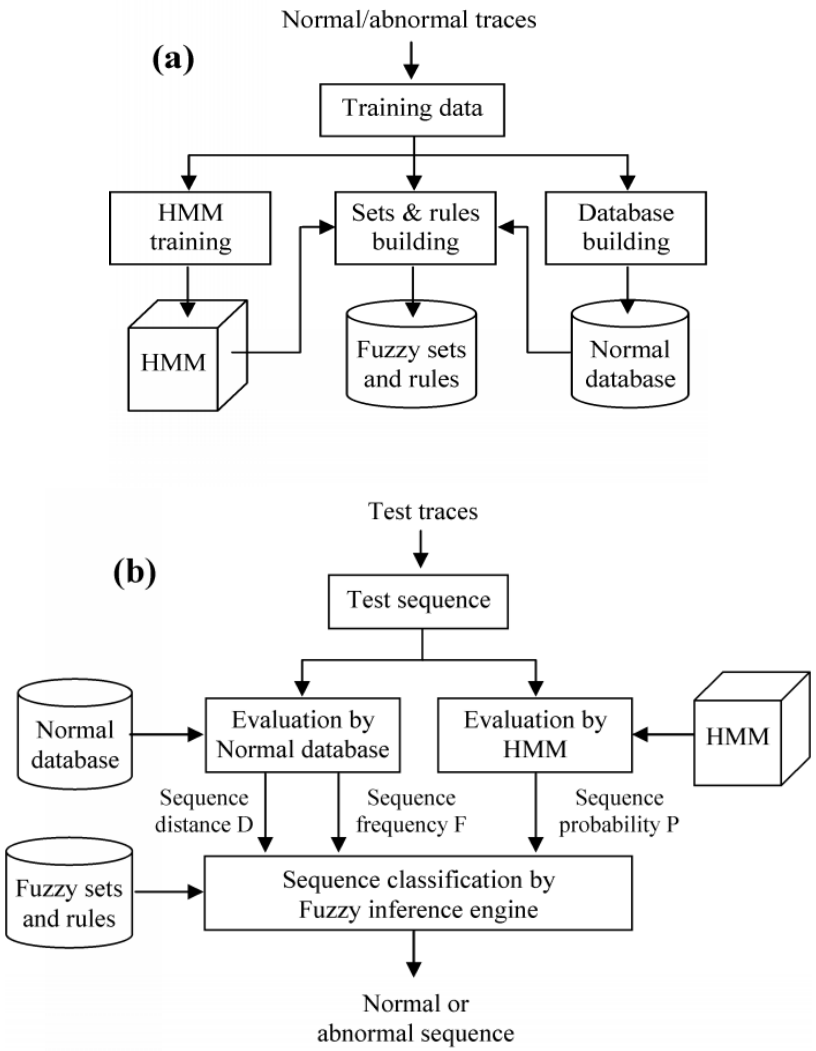

Figure 8.1: The proposed program anomaly detection model: (a) Training stage and (b) Testing stage

8.1.1 Model training stage

In the training stage, the proposed detection model's components which are (i) a normal database, (ii) an HMM model and (iii) fuzzy sets of sequence parameters are constructed from training data. The training data collection and the model construction can be described as follows:

1. Training data collection:

   (a) Data collection: Training data consist of normal and abnormal traces of system calls

collected from the normal and abnormal runs of a program. The normal traces are used to construct the normal database and the HMM model. The normal traces and certain abnormal traces are used to build fuzzy sets. The program's raw data can be collected in real working conditions or simulated conditions, as discussed in Section 5.1.1.

(b) Data pre-processing: Collected raw data must be processed to extract useful information from raw traces of system calls. The data is also transformed to observation sequences using HMM notations for the HMM training. The data pre-processing procedure was discussed in Section 5.1.2.1.

2. Model construction:

(a) Building of the normal database : The normal database is an ordered list of all unique short sequences of system calls found in training data. The database is created from the training data of normal traces of system calls using the method given in [25, 26]. All short sequences in the normal database have the same number of system calls k, and k is used as the short sequence length. In addition, the frequency of each short sequence, that is the number of occurrences of that sequence in training data, is also recorded in the normal database.

(b) HMM training: The HMM model is trained from normal traces of system calls using the HMM incremental training scheme with optimal initialization, as discussed in Section 6.1.2.

(c) Creation of fuzzy sets: The fuzzy sets are created from normal and abnormal traces of system calls, as discussed in Section 7.3.1.

8.1.2 Model testing stage

In the testing stage, the constructed detection model is used to evaluate test traces of program system calls to find anomalies and possible intrusions. There are three steps in the testing procedure: (i) formation of short sequences of system calls from test traces, (ii) evaluation of each short sequence, and (iii) calculation of the region anomaly score. The three steps of the testing stage are described as follows:

1. Formation of short sequences: Short sequences, each of which is used as the test unit in next step, are formed from test traces of system calls using the sliding window method, as discussed in Section 5.1.2.3. The short sequence length is k system calls, and is equal to the short sequence length of the normal database.

2. Evaluation of each short sequence by the detection engine:

   (a) First, the normal database and the HMM model are used to compute the short sequence's parameters which are used as the input for the fuzzy inference engine. The input parameters consist of the sequence probability P generated by the HMM model, and the sequence distance D and frequency F produced by the normal database.

   (b) Then, the fuzzy inference engine is used to evaluate the short sequence using sequence's parameters generated by the normal database and the HMM model. The fuzzy inference engine applies the fuzzy sets and rules to interpret the input sequence parameters in order to produce the output which is the status of the short sequence: normal or abnormal.

3. Calculation of the region anomaly score: As discussed in Section 5.1.2.3, we group individual abnormal short sequences into temporally local regions to measure the anomaly signal. A region's anomaly score A is calculated as the ratio between the number of detected abnormal sequences in the region to the total number of sequences of the region, as given in equation (5.1). A region with anomaly score A, is labelled as abnormal if $A \geq \hat{A}$, where $\hat{A}$ is region score threshold. A detected abnormal region can be used to raise an intrusion alarm.

8.1.3 Online update of the detection model

Although the behaviour of a program is relatively stable over time [25, 26, 66, 93], there are still good reasons for the program's anomaly detection model to be updated dynamically: (i) the detection model may not fully represent normal program behaviour due to the shortage of training data, (ii) updates to the program itself may change the program behaviour, and (iii) updates to the underlying operating system may also affect the program behaviour.

We propose an online update scheme for the proposed anomaly detection model, as shown in Figure 8.2. The proposed update scheme is capable of updating the detection model's two main

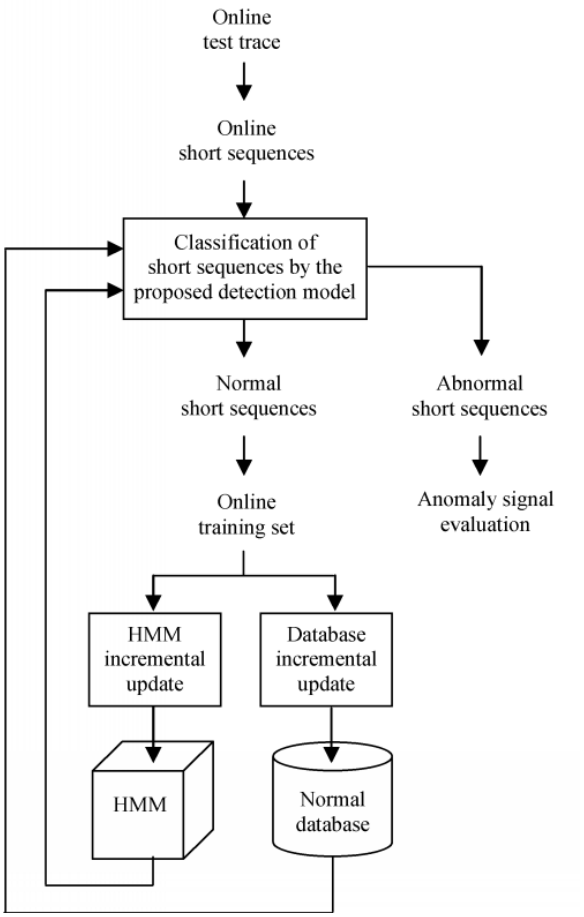

Figure 8.2: Online update scheme of the proposed anomaly detection model

components: the normal database and the HMM model. It is assumed that these components of the initial detection model have been built from the program's normal traces of system calls in the off-line training stage, as discussed in Section 8.1.1. The proposed online update scheme uses the detection model itself as a *filter* to remove *noisy data* (anomalies) from an online test trace before it is used for the online update. The proposed online update scheme can be described as follows:

1. First, *online short sequences* are formed from an *online test trace* using the sliding window method with length of k system calls.

2. Then, each of these short sequences is evaluated by the proposed detection model. This task is done in the model testing stage, as discussed in Section 8.1.2. Abnormal short

sequences detected in this process are used for the anomaly signal evaluation, while normal short sequences are added to the *online training set*.

3. When the online training set reaches a pre-defined size, it is used to update the normal database and the HMM model.

   - Update of the normal database: Short sequences that do not exist in the normal database are added, and the occurrence frequency of each short sequence in the normal database is also updated.
   - Update of the HMM model: The HMM model is incrementally updated using the HMM incremental training scheme, as discussed in Section 6.1.

4. The online training data set is reset after each successful update.

8.2 Discussion

In this section, we evaluate the proposed program anomaly detection model based on the three most important characteristics of an anomaly intrusion detection approach: accuracy, efficiency and adaptability.

8.2.1 Accuracy

The accuracy of an anomaly intrusion detection approach can be determined on its detection rate and false positive rate. An effective intrusion detection approach should have a high detection rate and a low false positive rate.

8.2.1.1 Detection rate

According to experimental results presented in Section 5.1.3.2, Section 5.2.2.3 and Section 7.5.3, our detection models correctly detected all intrusions embedded in all abnormal traces tested. The detected intrusions include a DoS intrusion on *sendmail* program, a DoS intrusion on *inetd* program, and *sm5x*, *sm565a*, *syslog-local* and *syslog-remote* intrusions on *sendmail* program. In contrast, the normal database scheme [25] missed the *sm565a* intrusion, with only 0.6% of abnormal sequences detected. This scheme [25] possibly also missed the *syslog-local* intrusion embedded in *syslog-local* trace No 2, with just 1.7% of abnormal sequences detected.

The better detection performance of the proposed detection model can be explained by (i) its application of the fuzzy inference and (ii) its ability to generate strong anomaly signals. On one hand, the model's fuzzy inference engine is capable of combining multiple sequence inputs using the fuzzy reasoning for the classification of sequences of test traces, which results in better detection rate. On the other hand, strong anomaly signals generated by the proposed model helps it to correctly recognize abnormal sequences in the test traces, which also results in a higher detection rate. More discussion on the gain of detection performance is given in following sections.

8.2.1.2 False positive rate

The proposed detection model generated much fewer false positive alarms than the normal database scheme [25], as shown in Table 7.1. For example, the false positive rate of the normal database scheme is 0.174%, as opposed to 0.085% of the proposed model, or a reduction of 50.96%, on the training set of 50% of the full set, with k = 15.

It is also noted that the proposed detection model achieved a much lower false positive rate on small-size training sets than the normal database scheme [25]. On the training set of 30% of the full set, the false positive rate of proposed detection model is lower than that of the normal database scheme on the training set of 100% of the full set. This means that the proposed detection model requires much less training data to achieve the same level of the false positive rate of the normal database scheme [25].

The fuzzy inference engine plays an important role in the reduction of false positive alarms. The engine's capability of incorporating multiple sequence information, generated by the normal database and the HMM model, through the fuzzy reasoning process, accurately classifies the sequence. This in turn reduces the false alarms.

8.2.1.3 Strength of anomaly signal

There are two types of anomaly signals generated by our detection models: (i) raw anomaly signals and (ii) region-based temporal anomaly signals. We evaluate the strength of each type of anomaly signals separately.

Raw anomaly signals

The raw anomaly signal is generated by our basic detection model for each test short sequence

of system calls, which is represented by log(P) - the logarithm of the probability that the HMM model produces a particular short sequence. Our basic detection model clearly generated two levels of raw anomaly signals for normal and abnormal short sequences in the test trace, according to the experimental results given in figures 5.2 and 5.3. Short sequences associated with likely probabilites are considered normal, and those associated with unlikely probabilites are considered abnormal. For both *sendmail* and *inetd* DoS intrusion traces, the difference between the probabilites of normal and abnormal sequences is substantial, which is greater than 50%.

Region-based temporal anomaly signals

The region-based temporal anomaly signal is created by grouping raw anomaly signals of individual short sequences into temporally local regions, as presented in Section 5.1.2.3. The proposed detection model produced much higher levels of region-based temporal anomaly signals, compared to the normal database model [25]. For example, for the *sm565a* trace, there are 68% of abnormal regions detected by the proposed model, as opposed to only 0.6% of abnormal sequences detected by the normal database model. Similar results can also be seen for other tested traces (refer to Table 7.2).

Our method of grouping individual abnormal short sequences into temporally local regions, generates stronger anomaly signals, because abnormal short sequences usually occur in local groups across the test traces. On the other hand, the anomaly signal evaluation method in [25] generates weak anomaly signals because it evaluates the anomaly signal by counting the total number of abnormal short sequences in the whole test traces. This anomaly signal evaluation method [25] does not properly represent the distribution of abnormal short sequences in the test traces.

8.2.2 Efficiency

As discussed in Section 3.2.3, we focused our discussion on the efficiency of our detection models in two aspects: (i) the cost of model construction, and (ii) the cost of testing for intrusions. Specifically, we consider these costs in terms of time (or computation) and storage (space) demands for model construction and model testing.

8.2.2.1 Model construction

HMM training accounts for a major portion of the time and storage demands in the construction of an HMM-based detection model. Therefore, we discuss the construction costs of our model, which is based on the HMM incremental training and the construction costs of other detection approaches, which are based on the HMM batch training.

Training time

Our detection model, based on the HMM incremental training, requires much less training time than other detection approaches [102, 134], which are based on the Baum-Welch batch training, using the same training data set. According to the experimental results shown in Table 6.2 and Table 6.3, the elapsed training time of HMM models, based on the Baum-Welch batch training, and our HMM incremental training are 201.37 and 48.47 minutes respectively. This is a time reduction of over 4 times, using the training set of 40 subsets.

It is also noted that the training time for each incremental update of the HMM model, using a data subset, is relatively small, which makes it possible for the online update of the detection model. The training time for each incremental update is 1.11 minutes on average, for the incremental training option of 50 subsets, as shown in Table 6.3.

Storage demand

As the training data set of T system calls has been broken up to K subsets of equal length T_k, we have $T_k = T/K$ system calls. And, since our HMM incremental training scheme only uses one subset at a time to update the HMM model, its storage demand is reduced by a factor of K. In other words, while the storage demand of the HMM Baum-Welch batch training scheme is $O(N^2T)$, the storage demand of our HMM incremental training scheme is only $O(N^2T_k)$, where N is the number of HMM states.

Since the training data set in anomaly detection using program system calls is extremely large, $T >> N$, the reduction of K times in storage demand is substantial. This in turn improves the efficiency and applicability of our HMM-based detection model in both online detection and offline analysis.

Amount of training data required

It can be seen in Table 7.1 that our proposed detection model requires less training data

to achieve the same levels of false positive rate as the normal database scheme [25]. For short sequence length $k = 11$, our model requires only 30% of the full training data set to achieve the false positive rate of 0.099%, which is lower than the rate of 0.147%, produced by the normal database scheme [25], on 100% training data set.

8.2.2.2 Model testing

In terms of computations for model testing, the proposed detection model is more expensive than the normal database model [25] because there is an additional HMM test on all short sequences to produce the sequence probability parameter for the fuzzy inference engine. In addition, there are some additional overheads for the fuzzy reasoning process to produce the output. However, the model testing costs are relatively small, compared to the model training costs. This is because the length of each short sequence is much smaller than the length of a training set.

8.2.3 Adaptability

An intrusion detection approach is said to be adaptable if it can (i) detect novel/unknown intrusions and (ii) support the dynamic update of its detection model. As anomaly detection approaches, HMM-based anomaly detection approaches are capable of detecting unknown intrusions. We use HMM modelling to characterize normal program behaviour, and attempt to detect deviations of the program's current behaviour from its normal behaviour. Therefore, any known and unknown intrusions, that cause significant changes to the program's current behaviour from its normal behaviour, can be detected by our HMM-based detection model.

In addition, the proposed detection model supports the online update of its detection model, which is based on the incremental update of the model's components, as discussed in Section 8.1.3. Since the online test data may include anomalies and intrusions, it needs to be cleaned by the detection model, before it can be used. And, only clean data (normal short sequences specifically) are added into the online training set for the periodic update. This online update keeps our detection model up-to-date, and therefore it is able to adapt to the changes of the program's behaviour. This in turn maintains the model's detection accuracy.

8.3 Conclusion

In Part II, we presented the development process of the proposed program anomaly intrusion detection model using system calls. The proposed anomaly detection model focuses on improving the detection accuracy, efficiency and adaptability of existing anomaly intrusion detection approaches.

8.3.1 Reducing false alarms and increasing the detection rate

The HMM modelling was used to characterize normal program behaviour using normal traces of system calls to improve the detection rate since HMM is a powerful modelling technique in many research areas [103, 104]. The strong anomaly signal generated by our HMM-based basic detection model for some abnormal traces embedded with DoS intrusions on programs indicated that the HMM model is a good model of normal program behaviour.

Reducing false alarms is an integral part of improving detection accuracy because a high level of false alarms will reduce the effectiveness of an intrusion detection system [81, 82]. We introduced two detection schemes in order to reduce false alarms: a two-layer detection scheme and a fuzzy-based detection scheme. The two-layer detection scheme is an extension of the basic detection model, which uses a normal database [25] and an HMM model to form a double-layer test on each sequence of system calls. Experimental results showed that the proposed two-layer scheme detection produced about 28% fewer false alarms than the normal database scheme [25].

On the other hand, the fuzzy-based detection scheme focuses on further improving the detection accuracy. It uses the fuzzy inference to combine the sequence information from multiple sources. Like the two-layer scheme, the fuzzy-based scheme also includes a normal database [25] and an HMM model. However, the normal database and the HMM model are only used to generate the inputs for the fuzzy inference engine. The fuzzy inference engine uses its fuzzy sets and rules to combine multiple sequence inputs to produce an output. Experimental results showed that the fuzzy-based detection scheme produced about 48% fewer false alarms than the normal database scheme [25]. Moreover, it generated a much stronger anomaly signal than the two-layer detection scheme for all tested abnormal traces.

Our anomaly signal measurement method, which groups individual abnormal sequences into temporally local regions, also helps produce a stronger anomaly signal since abnormal sequences usually occur in local groups across the test traces. A strong anomaly signal helps the detection

engine correctly classify test sequences, which in turn improves detection accuracy.

8.3.2 Improving the efficiency and adaptability

In our basic detection model, the HMM modelling of normal program behaviour generated strong abnormal signals, and therefore it produced a high detection rate. However, the basic detection model suffers a efficiency problem because the training of its HMM model, using the Baum-Welch algorithm, is very expensive in terms of training time and storage demand. Our HMM incremental training scheme with optimal initialization was designed to solve this efficiency problem. Experimental results showed that the proposed HMM incremental training scheme reduced training time and storage demand substantially. In addition, the proposed HMM incremental training scheme was used to support the online update of the proposed program anomaly detection model, which makes the detection model adaptable to the changes of the program's behaviour.

Chapter 9

Conclusions

9.1 Summary

This thesis developed a solution to protect computer and information systems with two lines of defence: (i) preventing intrusions at the first line and (ii) detecting intrusions at the second line if the prevention measures of the first line have been penetrated. We presented a security enhancement for E-commerce applications at the first line of defence, and an improvement of the detection accuracy, efficiency and adaptability of existing program anomaly intrusion detection approaches at the second line of defence.

9.1.1 Security enhancement for E-commerce applications

Part I of this thesis described a hardware-based encryption model for the enhancement of communication and end-system security, and performance for E-commerce applications. Although existing software based security solutions, such as SSL/TSL protocols have been widely used, they have problems with the protection of long-term private keys and the degradation of system performance. Moreover, the security of these software based security solutions depends on the security of the underlying operating system, and therefore they are vulnerable to threats caused by security flaws of the underlying operating system.

The proposed encryption model is based on the integration of an existing web-based client/server model and embedded hardware-based RSA encryption modules. The proposed encryption model consists of two main components: the client and the server. The client consists of a web browser, a hardware-based encryption engine and a client communication module. The server

includes a web server, a hardware-based decryption engine and a server communication module. Plaintext messages are encrypted by hardware encryption engines before they are transferred from one end to the other through an open environment, such as the Internet. The proposed model's security is assured by the strength of RSA encryption at large keys, and the security of a hardware based cryptosystem. On the other hand, the model's performance is guaranteed by the hardware's high processing capability.

Hardware was selected as the platform for the implementation of the proposed encryption model because of its advanced security features. Hardware based cryptosystems are generally considered more secure than software based counterparts, since a hardware based cryptosystem runs on its own physical memory space, and therefore it can effectively prevent malicious access from external programs. This also means that the security of a hardware based cryptosystem does not depend on the security of the underlying operating system. Furthermore, hardware can be used to safely store sensitive information, such as long-term private keys.

A successful implementation of DSP hardware based RSA encryption modules using Texas Instruments' TMS320C6416 DSK boards was presented. Each encryption module at each end of the proposed model supports both message encryption and decryption functions. We also provided the details on how to integrate these hardware modules into an existing E-commerce system, and how to communicate efficiently between the computer applications (the web browser and the web server) and the DSP hardware based encryption modules.

A fully functional E-commerce system, including the web and database applications was built to evaluate the performance and security of the proposed encryption model. According to our experimental results, the proposed hardware based RSA encryption model performs better than the software based RSA implemetation running on Intel Pentium 4 machines that have almost double clock speed of the DSP's at large RSA encryption keys. The results confirmed that our encryption model is more scalable than software based implementations, and it can be used to efficiently improve the security and performance of E-commerce applications.

9.1.2 Improvement of accuracy, efficiency and adaptability for program anomaly intrusion detection

Part II of this thesis described an HMM-based program anomaly intrusion detection model using system calls. Our goal is to improve the detection accuracy, efficiency and adaptability of

existing anomaly detection approaches. We started with an intensive investigation of existing anomaly intrusion detection approaches, and then focused on approaches that monitor program behaviour to detect intrusions. Generally, anomaly detection approaches are not effective as they usually generate a high level of false alarms. In addition, several anomaly detection approaches [102, 134] demand excessive system resources for training and testing.

We applied the HMM modelling to represent normal program behaviour using system calls in our basic anomaly detection model. The procedure for collecting and pre-processing raw traces of system calls from programs' runs was proposed. In the model training stage, an empirical method to select the number of HMM states and the number of distinct observation symbols for HMM training was discussed. In the model testing stage, we presented an anomaly signal measurement method which produces strong anomaly signals by grouping individual abnormal sequences into a temporally local regions. Our experimental results confirmed that the HMM-based basic detection model generated strong and clear anomaly signals for a DoS intrusion trace on the *sendmail* program and a DoS intrusion trace on the *inetd* program.

We then proposed an extension to the basic detection model, and presented a two-layer detection scheme to reduce false alarms, since an HMM-based single-layer detection engine of the basic detection model may generate relatively a high level of false alarms. The scheme's double-layer detection engine consists of a normal database [25] and an HMM model. In this scheme, each sequence of system calls goes through two sequential tests, first against the normal database and then against the HMM model. The HMM model test only applies to sequences which were classified as mismatched, or rare, by the normal database. Experimental results showed that the two-layer scheme effectively reduced false alarms by about 28%, compared to the normal database scheme.

We next examined the HMM training efficiency problem of the basic detection model. The basic detection model suffers an efficiency problem because the construction of its HMM model, based on the HMM Baum-Welch batch training, was found to be very expensive in terms of training time and storage demands, as discussed in Section 5.1.2.2. The basic detection model must be improved in terms of efficiency, before it can be used. The proposed HMM incremental training scheme with optimal initialization is our solution to the basic model's efficiency problem. Unlike the HMM batch training, our HMM incremental training scheme incrementally updates the HMM model using one training subset at a time until convergence. Our experimental results

showed that the proposed HMM incremental training scheme reduced training time four-fold, compared to the HMM batch training based on the Baum-Welch algorithm. Our training scheme also reduced the storage demand in proportion to the number of training subsets used, compared to the HMM batch training.

We next improved the two-layer detection scheme in order to further reduce false alarms and increase the detection rate. We were motivated by the fact that the normalcy and the abnormalcy are not truly crisp concepts. Therefore, it is not always possible to correctly classify an object's behaviour as normal or abnormal using crisp conditions [18, 33, 34]. We proposed a fuzzy-based detection scheme, in which fuzzy sets and rules were used to represent the space and combined conditions of parameters of each test sequence. A normal database and an HMM model were used to generate sequence parameters which are used as the input to the fuzzy inference engine. The fuzzy inference engine evaluates each sequence of system calls, by combining the sequence information using its fuzzy sets and rules to produce the sequence status. Our experimental results showed that the fuzzy-based detection scheme reduced false alarms by about 28% and 48%, compared to the two-layer detection scheme and the normal database scheme, respectively. In addition, the fuzzy-based scheme produced much stronger anomaly signals for all tested abnormal traces, as presented in Section 7.5.3.

In Chapter 8, we presented a complete model for program anomaly intrusion detection using system calls. The proposed detection model consists of two development stages: training stage and testing stage. In the training stage, an HMM model and a normal database were built from system call data to characterize normal program behaviour. The proposed HMM incremental training scheme was used to construct and update the HMM model. Fuzzy sets were empirically created from training data to represent the space of sequence parameters. In the testing stage, the fuzzy-based detection scheme was used to evaluate test traces of system calls to find anomalies and possible intrusions. In addition, the proposed detection model also supports an online update scheme which is used to update the HMM model and the normal database using online data.

9.2 Future Work

There are some possible improvements for the proposed hardware-based RSA encryption model and the proposed program anomaly detection model, which can be the topic of future research.

9.2.1 Possible improvements for the proposed hardware based RSA encryption model

- Further optimization of the RSA hardware implementation for higher performance: The current RSA hardware implementation using the C programming language performs better than the RSA software implementation running on Intel-based PCs at large RSA keys. The RSA hardware implementation can be further optimized at atomic integer operations, such as adders and multipliers, to fully exploit the processing capability of DSP hardware. In addition, the C implementation of critical modules of the *BigInt* package can be replaced with low-level languages, such as Assembler. This can reduce code size, as well as improve performance.

- Extend the proposed DSP-based RSA encryption model to support more platforms: Current implementation of the proposed encryption model only works on the Microsoft platform, specifically on Microsoft Internet Explorer web browser on the client side, and Microsoft web server on the server side. Making the proposed encryption model able to work on other web browsers, such as Firefox and Netscape on the client side, and other web servers, such as Apache on the server side, are other possible extensions.

9.2.2 Possible improvements for the proposed program anomaly detection model

- Automatic learning of fuzzy sets and rules from training data: In the current implementation of the proposed detection model, we manually created fuzzy sets and rules for the fuzzy inference engine to evaluate each sequence of test traces of system calls. The fuzzy rules were manually defined, based on the detection assumptions of the HMM model and the normal database. For example, the membership functions of fuzzy sets for parameters, such as sequence probability and sequence frequency, were empirically determined by analysing the parameter distributions in the training data. An automatic learning scheme, which can generates fuzzy sets and rules from training data, is a possible future improvement for the proposed detection model.

- Improve the weighted merging method of HMM models of the proposed HMM incremental training scheme to further reduce training time and storage demand: To make the weighted

merging method simple, we chose a uniform size (the number of HMM states N and the number of distinct observation symbols M) for the final HMM model and all HMM sub-models. However, the size of each HMM sub-model can be significantly smaller than the size of the final HMM model, since an HMM sub-model is trained from a training subset which is substantially smaller than the full training set. A possible solution is to select the size of each HMM sub-model based on the corresponding training subset. Small-size HMM sub-models help further reduce the overall training time and storage demands. The proposed HMM incremental scheme must be improved to support the HMM incremental merging of two HMM models with different sizes.

- Improve the region-based anomaly signal evaluation scheme to produce a stronger anomaly signal: We used the region-based scheme to group individual abnormal sequences into temporally local regions, in which a region's anomaly score is calculated as the ratio of the number of abnormal sequences detected in the region, to the total number of sequences in the region. This means that all abnormal sequences in a region are treated equally, although they have different anomaly scores. We believe that, an anomaly signal evaluation scheme, which takes the anomaly score of individual abnormal sequences into consideration, will generate a stronger anomaly signal. A stronger anomaly signal, in turn helps to improve detection accuracy.

9.3 Closing Remarks

This thesis developed an approach to secure computer and communication systems from intrusions with two lines of defence: preventing intrusions at the first line and detecting intrusions at the second line. For the first line of defence, we proposed a hardware based RSA encryption model. For the second line of defence, we proposed an HMM-based program anomaly intrusion detection model. The hardware based RSA encryption model and the program anomaly intrusion detection model have been designed, implemented and evaluated for the proof of concepts.

Appendix A

Main Features of TMS320C6416 DSP Starter Kit

The Texas Instruments' TMS320C6416 DSP Starter Kit (DSK) is an external DSP board with central processing unit C6416 running at the default clock of 1000MHz. In this appendix, we describe main technical specifications of the DSP board mentioned in Section 2.5.1. Table A.1 presents main features of the DSP board. Figures A.1 and A.2 show the board layout and the block diaram of the DSP board respectively.

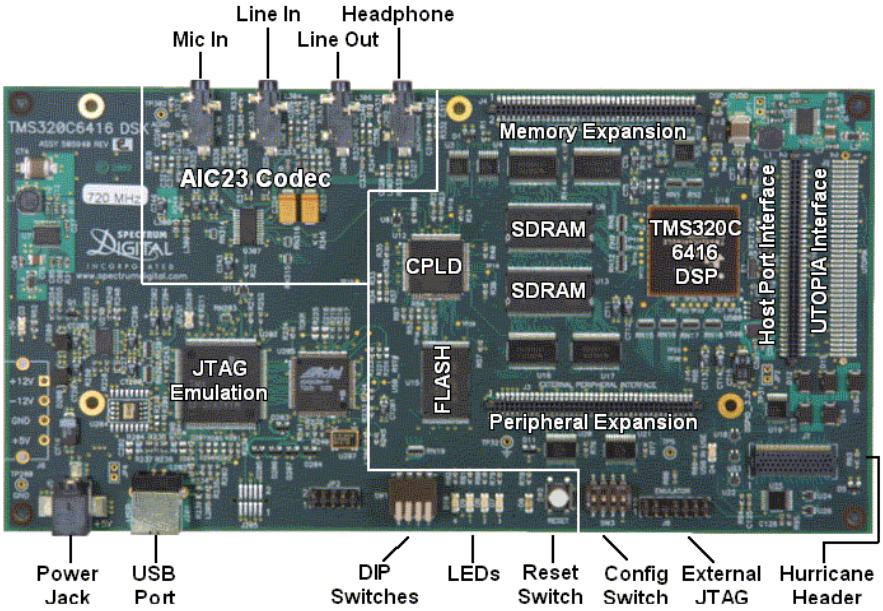

Figure A.1: Texas Instruments TMS320C6416 DSK board layout [121, 128]

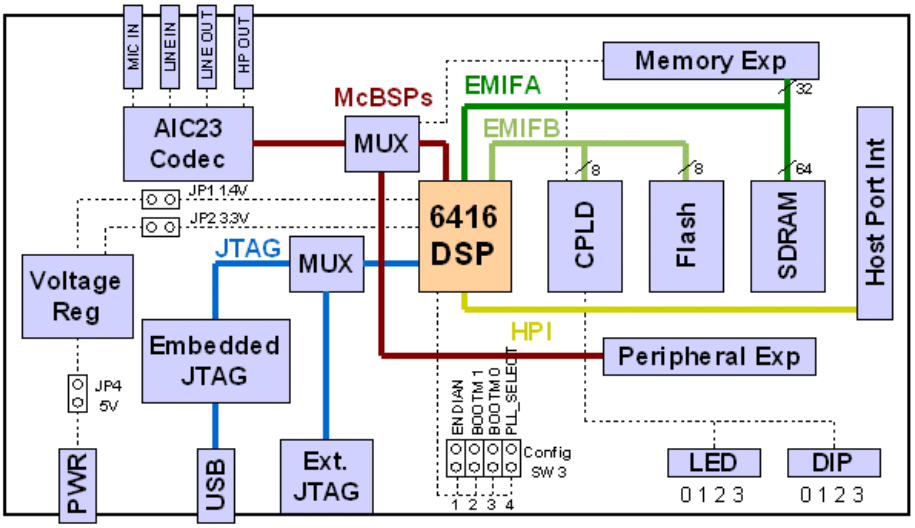

Figure A.2: Texas Instruments TMS320C6416 DSK block diagram [121, 128]

Table A.1: Main Features of TMS320C6416 DSK [127]

TMS320C6416 DSK Features
Highest-Performance Fixed-Point DSPs
2.00, 1.67, 1.39, 1.18, 1.00, 0.84 ns Instruction Cycle Time
500, 600, 720, 850, 1000, 1200MHz Clock Rate
Eight 32-Bit Instructions/Cycle
Twenty-Eight Operations/Cycle
4000, 4800, 5760, 6800, 8000, 9600 MIPS
Fully Software-Compatible With C62x
C6414/15/16 Devices Pin-Compatible
VelociTI.2 Extensions to VelociTI Advanced Very-Long-Instruction-Word (VLIW) TMS320C64x DSP Core
Eight Highly Independent Functional Units With VelociTI.2 Extensions: - Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per Clock Cycle - Two Multipliers Support Four 16 x 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 x 8-Bit Multiplies (16-Bit Results) per Clock Cycle
Non-Aligned Load-Store Architecture
64 32-Bit General-Purpose Registers
Instruction Packing Reduces Code Size
All Instructions Conditional
Instruction Set Features
Byte-Addressable (8-/16-/32-/64-Bit Data)
8-Bit Overflow Protection
Bit-Field Extract, Set, Clear
Normalization, Saturation, Bit-Counting
VelociTI.2 Increased Orthogonality
Viterbi Decoder Coprocessor (VCP)
Supports Over 600 7.95-Kbps AMR

TMS320C6416 DSK Features (continued)
Programmable Code Parameters
Turbo Decoder Coprocessor (TCP)
Supports up to 7 2-Mbps or 43 384-Kbps 3GPP (6 Iterations)
Programmable Turbo Code and Decoding Parameters
L1/L2 Memory Architecture
128K-Bit (16K-Byte) L1P Program Cache (Direct Mapped)
128K-Bit (16K-Byte) L1D Data Cache (2-Way Set-Associative)
8M-Bit (1024K-Byte) L2 Unified Mapped RAM/Cache (Flexible Allocation)
Two External Memory Interfaces (EMIFs)
One 64-Bit (EMIFA), One 16-Bit (EMIFB)
Glueless Interface to Asynchronous Memories (SRAM and EPROM) and Synchronous Memories (SDRAM, SBSRAM, ZBT SRAM, and FIFO)
1280M-Byte Total Addressable External Memory Space
Enhanced Direct-Memory-Access (EDMA) Controller (64 Independent Channels)
Host-Port Interface (HPI)
User-Configurable Bus Width (32-/16-Bit)
32-Bit/33-MHz, 3.3-V PCI Master/Slave Interface Conforms to PCI Specification 2.2
Three PCI Bus Address Registers: Prefetchable Memory, and Non-Prefetchable Memory I/O
Four-Wire Serial EEPROM Interface
PCI Interrupt Request Under DSP Program Control
DSP Interrupt Via PCI I/O Cycle
Three Multichannel Buffered Serial Ports
Direct Interface to T1/E1, MVIP, SCSA Framers
Up to 256 Channels Each
ST-Bus-Switching-, AC97-Compatible
Serial Peripheral Interface (SPI) Compatible (Motorola)
Three 32-Bit General-Purpose Timers

TMS320C6416 DSK Features (continued)
Universal Test and Operations PHY Interface for ATM (UTOPIA)
UTOPIA Level 2 Slave ATM Controller
8-Bit Transmit and Receive Operations up to 50 MHz per Direction
User-Defined Cell Format up to 64 Bytes
Sixteen General-Purpose I/O (GPIO) Pins
Flexible PLL Clock Generator
IEEE-1149.1 (JTAG) Boundary-Scan-Compatible

Appendix B

Implementation of the DSP Hardware RSA Encryption Model: Major Functions

This appendix provides more details about the implementation of the major components of the proposed DSP hardware RSA encryption model, presented in Section 2.4. Table B.1 shows the crucial functions of the BigInt package that is the core of the implementation of the RSA algorithm. Table B.2 lists the core RTDX built-in functions which are used to implement the DSP RTDX Interfaces of the DSP Hardware RSA Encryption Engine and the DSP Hardware RSA Decryption Engine. Table B.3 presents core RTDX exported functions that are used to develop the Client DSP Communication Module and the Server DSP Communication Module. Table B.4 shows main functions of the Client DSP Communication Module which are responsible the communications between the client web browser and the DSP Encryption Engine. Table B.5 shows main functions of the Server DSP Communication Module which are responsible the communications between the web server and the DSP Decryption Engine.

Table B.1: Major functions of BigInt package

=== Initialization, memory management and conversion functions ===
`int BigIntInit(BigInt *bn, unsigned int initLength)`: initialize a big integer of *initLength* digits, and assign all digits to 0.
`int BigIntInitStrHex(BigInt *bn, char *inputStr)`: initialize a big integer from a hexadecimal string.
`int BigIntInitStrBin(BigInt *bn, char *inputStr)`: initialize a big integer from an octet string.
`void BigIntToChar(BigInt *bn, char *outputStr)`: convert a big integer to an octet string.
`void BigIntToHex(BigInt *bn, char *outputStr)`: convert a big integer to a hexadecimal string.
`void BigIntRelease(BigInt *bn)`: deallocate memory occupied by a big integer.
=== Arithmetic functions ===
`int BigIntAdd(BigInt *bn1, BigInt *bn2)`: add two non-negative big integers; $bn1 \leftarrow bn1 + bn2$
`int BigIntAddSign(BigInt *bn1, BigInt *bn2)`: add two signed big integers; $bn1 \leftarrow bn1 + bn2$
`int BigIntSub(BigInt *bn1, BigInt *bn2)`: substract two non-negative big integers, $bn1 \geq bn2$; $bn1 \leftarrow bn1 - bn2$
`int BigIntSubSign(BigInt *bn1, BigInt *bn2)`: substract two signed big integers; $bn1 \leftarrow bn1 - bn2$
`int BigIntMulVal(BigInt *bn, unsigned short theValue, BigInt *result)`: multiply a non-negative big integer with a non-negative integer value; $result \leftarrow bn1 * theValue$
`int BigIntMulVal2(BigInt *bn, unsigned short theValue)`: multiply a non-negative big integer with a non-negative integer value; $bn1 \leftarrow bn1 * theValue$
`int BigIntShiftLeft(BigInt *bn, unsigned int pos)`: Shift all digits of a big integer *pos* position to the left.

Major functions of BigInt package (continued)
`int BigIntShiftRight(BigInt *bn, unsigned int pos)`: Shift all digits of a big integer *pos* position to the right.
`int BigIntDiv(BigInt *bn1, BigInt *bn2, BigInt *qt)`: big integer division of $bn1/bn2$; results: qt is quotient and bn1 is remainder.
=== Comparison functions ===
`unsigned short BigIntCmpZero(BigInt *bn)`: compare a big integer with 0.
`unsigned short BigIntCmp(BigInt *bn1, BigInt *bn2)`: compare two non-negative big integers.
`unsigned short BigIntCmpSign(BigInt *bn1, BigInt *bn2)`: compare two signed big integers.
=== Modular arithmetic functions ===
`int BigIntMontMul(BigInt *x, BigInt *y, BigInt *modulus, unsigned short modInverse, BigInt *a)`: Implementation of Montgometry multiplication that computes $a = x * y * R^{-1} \bmod modulus$, where $R = B^l$, B is the radix of the big integer, and l is the number of digits of $modulus$; and $modInverse = -modulus^{-1} \bmod B$
`int BigIntMontExp(BigInt *m, BigInt *exp, BigInt *modulus, unsigned short modInverse, BigInt *c)`: Implementation of Montgomery exponentiation that computes $c = m^{exp} \bmod modulus$

Table B.2: Core RTDX built-in functions [62, 128]

=== Target initialization for RTDX ===
`TARGET_INITIALIZE()`: initialize the RTDX communications on the DSP target.
=== Create RTDX channels ===
`RTDX_CreateInputChannel(ichan)`: create an input channel to get data from host computer with provided name 'ichan'.
`RTDX_CreateOutputChannel(ochan)`: create an output channel to send data to host computer with provided name 'ochan'.
=== Enable/diable RTDX channels ===
`void RTDX_enableOutput(RTDX_outputChannel *ochan)`: enable the output channel 'ochan'.
`void RTDX_disableOutput(RTDX_outputChannel *ochan)`: disable the output channel 'ochan'.
`void RTDX_enableInput(RTDX_outputChannel *ichan)`: enable the input channel 'ichan'.
`void RTDX_disableInput(RTDX_outputChannel *ichan)`: disable the input channel 'ichan'.
=== Read and write data to RTDX channels ===
`int RTDX_read(RTDX_inputChannel *ichan, void *buffer, int bsize)`: read the *bsize* bytes from the input channel and put it into the $buffer$; return the error code. The function causes a read request and blocks the current process until it gets the data.
`int RTDX_readNB(RTDX_inputChannel *ichan, void *buffer, int bsize)`: read the *bsize* bytes from the input channel and put it into the $buffer$; return the error code. The function causes a read request, but not blocks the current process.
`int RTDX_write(RTDX_outputChannel *ochan, void *buffer, int bsize)`: write *bsize* bytes stored in $buffer$ to the output channel. Return the error code.

Table B.3: Core RTDX Exported functions [62, 128]

=== RTDX COM initialization ===
`::CoInitialize(NULL)`: initialize COM.
`::CoUninitialize()`: uninitialize COM.
`rtdx.CreateInstance(L"RTDX")`: creates an RTDX instance.
`rtdx.Release()`: release RTDX instance.
`rtdx->SetProcessor(BSTR board, BSTR processor)`: set the board and processor used.
=== Open/close input/output RTDX channels ===
`rtdx->Open(char channel[], char mode[])`: open an RTDX channel for reading or writing data based on *mode*.
`rtdx->Close()`: close the RTDX channel.
=== Read/write data to the RTDX channels ===
`long ReadSAI1(VARIANT *pArr)`: read a $SAFEARRAY$ of 8 bit integer; $*pArr$ is a pointer to the array.
`long ReadI1(BYTE *pData)`: read 8 bit integer.
`long WriteI1(unsigned char data, long *numBytes)`: write 1 byte of *data* to the channel.
`long Write( VARIANT Arr, long *numBytes)`: Writes a message ($SAFEARRAY$ type) to a channel. The parameter Arr is a $VARIANT$ that contains a pointer to the $SAFEARRAY$. The $SAFEARRAY$ can contain: 8-bit integers, 16-bit integers, 32-bit integers, 32-bit floats, or 64-bit floats.
=== Query functions ===
`long GetMsgLength(long *pLength)`: returns the number of bytes in the current message. The value is returned in $*pLength$.

Table B.4: Major functions of the client DSP communication module

=== Interface - Public function ===
`BSTR DSPEncrypt(LPCTSTR boardName, LPCTSTR processorName, LPCTSTR rsaPublicKeys, LPCTSTR plainText)`: invoke the DSP encryption engine to encrypt the plaintext and return the ciphertext; the function is called from the client browser using JavaScript.
=== Private functions ===
`int RTDX_Initialize(BSTR boardName, BSTR processorName)`: initialize the RTDX communications with provided board name and process name.
`void RTDX_Remove()`: release RTDX communications.
`int RTDX_OpenChannel(char channel[],char mode[])`: open a communication channel. The mode can be "R" for reading or "W" for writing.
`int RTDX_CloseChannel()`: close a RTDX channel
`char *RTDX_ReadMsg()`: read a message from the RTDX channel.
`long RTDX_WriteMsg(char buffer[])`: write a message to the RTDX channel.

Table B.5: Major functions of the server DSP communication module

=== Interface - Public function ===
`BSTR DSPDecrypt(BSTR boardName, BSTR processorName, LPCTSTR cipherText)`: invoke the DSP decryption engine to decrypt the ciphertext and return the plaintext; the function is called from the web browser using VBScript/JScript.
=== Private functions ===
The server DSP communication module uses the same privates functions used by the client DSP communication module.

Bibliography

[1] J. Anderson. Computer Security Threat Monitoring and Surveillance. *Technical report, James P. Anderson Co., Fort Washington, PA, USA*, April 1980.

[2] D. Anderson, T. Frivold, A. Tamaru and A. Valdes. Next-generation Intrusion Detection Expert System (NIDES) A Summary. *Technical Report No.SRI–CSL–95–07, Computer Science Laboratory, SRI International*, May 1995.

[3] M. Andrews and J. A. Whittaker. Computer security. *IEEE Security & Privacy Magazine*, Vol.2, No.5, pages 68–71, September-October 2004.

[4] W. Akkerman. Strace Home page. *http://www.liacs.nl/~wichert/strace/*, 2005.

[5] R. Bace and P. Mell. Intrusion Detection Systems. *Special Publication 800-31, NIST, USA*, 2001.

[6] H. Bar-El. Security Implications of Hardware vs. Software Cryptographic Modules. *The Inforsec Writers Text Library*, January 2004.

[7] T. Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM*, Vol.43, No.4, pages 99–105, 2000.

[8] C. Borgelt. Finding Association Rules Hyperedges with the Apriori Algorithm. *http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html#assoc*, 2002.

[9] K. Boudaoud, H. Labiod, R. Boutaba and Z. Guessoum. Network security management with intelligent agents. *Network Operations and Management Symposium*, pages 579–592, 2000.

[10] A. Buldas and J. Pldre. A VLSI implementation of RSA and IDEA encryption engine. *The 15th NORCHIP Conference 1997*, pages 281–288, November 10-11, 1997.

[11] CERT Coordination Center. CERT/CC Statistics 1988-2005. *http://www.cert.org/stats/cert_stats.html*, 2005.

[12] S. Cho. Incorporating Soft Computing Techniques into a Probabilistic Intrusion Detection System. *IEEE transactions on systems, man, and cybernetics*, Vol.32, No.2, May 2002.

[13] W. W Cohen. Fast effective rule induction. *The Proceedings of 12th International Conference on Machine Learning*, pages 115–123, Lake Tahoe, California, USA, July 1995.

[14] E. Cox. Fuzzy fundamentals. *IEEE Spectrum*, Vol.29, No.10, pages 58–61, October 1992.

[15] A. Daly and W. Marnane. Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic. *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pages 40–49, Monterey, California, USA, February 24-26, 2002.

[16] R.I.A. Davis and B.C. Lovell. Improved Estimation of Hidden Markov Model Parameters from Multiple Observation Sequences. *International Conference on Pattern Recognition*, pages 168–171, Quebec City, Canada, August 2002.

[17] D.E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, Vol.SE-13, No.2, pages 222–232, 1987.

[18] J.E. Dickerson, J. Juslin, O. Koukousoula and J.A. Dickerson. Fuzzy Intrusion Detection. *Proceedings of North American Fuzzy Information Processing Society*, pages 1506–1510, Vancouver, Canada, July 25, 2001.

[19] R. Dugad and U. B. Desai. A tutorial on Hidden Markov Models. *Technical report No.SPANN-96.1*, May 1996.

[20] M.H. Er, D.J. Wong, A.L. Sethu and K.S. Ngeow. Design and implementation of RSA cryptosystem using multiple DSP chips. *IEEE International Symposium on Circuits and Systems*, Vol.1, pages 49–52, 1991.

[21] E. Eskin. Anomaly Detection over Noisy Data using Learned Probability Distributions. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 255–262, 2000.

[22] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo. A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data. *Applications of Data Mining in Computer Security*, Kluwer Academic Publishers, 2002.

[23] W. Fan, M. Miller and S.J. Stolfo. Using Artificial Anomalies to Detect Unknown and Known Network Intrusions. *Proceedings of the first IEEE International conference on Data Mining*, pages 123–130, 2001.

[24] G. Florez, S. M. Bridges and R.B. Vaughn. An Improved Algorithm for Fuzzy Data Mining for Intrusion Detection. *The 2002 Annual Meeting of the North American on Fuzzy Information Processing Society*, pages 457–462, June 27-29, 2002.

[25] S. Forrest, S.A. Hofmeyr, A. Somayaji and T.A. Longstaff. A sense of self for Unix processes. *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, pages 120–128, 1996.

[26] S. Forrest, S.A. Hofmeyr and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, Vol.6, pages 151–180, 1998.

[27] S. Garfinkel and G. Spafford. *Web Security & Commerce*. O' Reilley, 1997.

[28] J.L. Gauvain and C.H. Lee. Bayesian Learning of Gaussian Mixture Densities for Hidden Markov Models. *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 272–277, 1991.

[29] J.L. Gauvain and C.H. Lee. MAP Estimation of Continuous Density HMM: Theory and Applications. *Proceedings of the DARPA Speech and Natural Language Workshop*, 1992.

[30] J.L. Gauvain and C.H. Lee. Maximum A Posteriori Estimation for Multivariate Gaussian Mixture Observations of Markov Chains. *IEEE Transactions on Speech and Audio Processing*, Vol.1, No.2, pages 291–298, 1994.

[31] GHMM. The General Hidden Markov Model library (GHMM). *http://www.ghmm.org/*, 2004.

[32] D. Gollmann. E-Commerce Security. *Computing & Control Engineering Journal*, pages 115–118, June 2000.

[33] J. Gómez and D. Dasgupta. Evolving Fuzzy Classifiers for Intrusion Detection. *The Third Annual IEEE Workshop on Information Assurance*, New Orleans, Louisiana, USA, June 17-19, 2002.

[34] J. Gómez, F. González and D. Dasgupta. An Immuno-Fuzzy Approach to Anomaly Detection. *The IEEE International Conference on Fuzzy Systems*, Vol.2, pages 1219–1224, May 25-28, 2003.

[35] L.A. Gordon, M.P. Loeb, W. Lucyshyn and R. Richardson. 2005 CSI/FBI Computer Crime And Security Survey. *Security survey report, Computer Security Institute*, 2005.

[36] Y. Gotoh, M.M. Hochberg and H.F. Silverman. Using MAP estimated parameters to improve HMM speech recognition performance. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol.1, pages 229–232, Adelaide, Australia, Apr 1994.

[37] Y. Gotoh, M.M. Hochberg and H.F. Silverman. Efficient Training Algorithms for HMMs Using Incremental Estimation. *IEEE Transactions on Software Engineering*, Vol.6, No.6, pages 539–548, 1998.

[38] J. Grozschädl. The Chinese Remainder Theorem and Its Application in a High Speed RSA Crypto Chip. *The 16th Annual Conference on Computer Security Applications - ACSAC '00*, pages 384–393, December 11-15, 2000.

[39] X. He. A Performance Analysis of Secure HTTP Protocol. *STAR Lab Technical Report, Department of Electrical and Computer Engineering, Tennessee Tech University*, March 2003.

[40] R. Heady, G. Luger, A. Maccabe and M. Servilla. The architecture of a network level intrusion detection system. *Technical report, Computer Science Department, University of New Mexico*, August 1990.

[41] X.D. Hoang and J. Hu. New Encryption model for secure E-commerce transactions using dsp–host, board and server communication. *IEEE International Conference on Telecommunications*, Vol.1, pages 166–170, June 2002.

[42] X.D. Hoang, J. Hu and P. Bertok. A multi-layer model for anomaly intrusion detection using program sequences of system calls. *International Conference on Network - ICON2003*, Vol.2, pages 531–536, Sydney, Australia, September 2003.

[43] X.D. Hoang and J. Hu. An Efficient Hidden Markov Model Training Scheme for Anomaly Intrusion Detection of Server Applications Based on System Calls. *International Conference on Network - ICON2004*, Vol.2, pages 470–474, Singapore, November 2004.

[44] R. Housley. *Cryptographic Message Syntax*. RFC 2360, June 1999.

[45] Howstuffworks.com. Howstuffworks.com website *http://electronics.howstuffworks.com*, 2005.

[46] J. Hu, Z. Xi, A. Jennings, H.Y.J. Lee and D. Wahyud. DSP application in E-commerce security. *IEEE International Conference on Acoustics, Speech, and Signal Processing - ITT-P1*, Salt Lake City, Utah, USA, May 7-11, 2001.

[47] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS 12 - Computer Applications in the Biosciences*, pages 95–107, 1996.

[48] Q. Huo, C. Chan and C.H. Lee. Bayesian adaptive learning of the parameters of hidden Markov model for speech recognition. *IEEE Transactions on Speech and Audio Processing*, Vol.3, No.5, pages 334–345, September 1995.

[49] K. Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. *Master Thesis*, Computer Science Department, University of California, Santa Barbara, November 1992.

[50] K. Ilgun, R. Kemmerer and P. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, Vol.21, No.3, pages 181–199, March 1995.

[51] ITsecurity.com. The ITsecurity.com: Dictionary of Information Security. *http://www.itsecurity.com*, 2004.

[52] J. Jantzen. Tutorial On Fuzzy Logic. *Technical Report No.98-E 868, Technical University of Denmark*, August 1998.

[53] A. Jones and S. Li. Temporal signatures for intrusion detection. *Proceedings 17th Annual Conference on Computer Security Applications, 2001*, pages 252–261, December 10-14, 2001.

[54] A. Jones and Y. Lin. Application Intrusion Detection using Language Library Calls. *Proceedings 17th Annual Conference on Computer Security Applications, 2001*, pages 442–449, December 10-14, 2001.

[55] B. Kaliski. *PKCS #1: RSA Encryption, Version 1.5.* RFC 2313, March 1998,

[56] B. Kaliski. *PKCS #1: RSA Encryption, Version 2.0.* RFC 2347, October 1998.

[57] T. Kanungo. Hidden Markov Models. *http://www.cfar.umd.edu/~kanungo/software/software.html*, 1998.

[58] K. Karplus, C. Barrett and R. Hughey. Hidden Markov Models for Detecting Remote Protein Homologies. *Journal of Bioinformatics 14*, pages 846–856, 1998.

[59] R. Karchin, M. Cline, Y. Mandel-Gutfreund and K. Karplus. Hidden Markov models that use predicted local structure for fold recognition: alphabets of backbone geometry. *Journal of Proteins: Structure, Function, and Genetics*, pages 504–514, June 2003.

[60] C. Kaufman, R. Perlman and M. Speciner. *Network Security.* Prentice Hall PTR, 2002.

[61] O. Kaven. Performance Tests: SSL Testing. *http://www.pcmag.com/article2/0,1759,1202267,00.asp*, August 19, 2003.

[62] D. Keil. Real-Time Data eXchange. *Texas Instruments White Paper*, February 1998.

[63] S.M. Kerner. E-commerce Business Booming, But More Threats Looming. *Internet-news.com*, June 21, 2005.

[64] C.H. Kim, S. Oh and J. Lim. A New Hardware Architecture for Operations in GF (2^n). *IEEE Transactions on Computers*, Vol.51, No.1, pages 90–92, January 2002.

[65] H.W. Kim and S. Lee. Design and implementation of a private and public key crypto processor and its application to a security system. *IEEE Transactions on Consumer Electronics*, Vol.50, No.1, pages 214–224, February 2004.

[66] C. Ko, G. Fink and K. Levitt. Automated detection of vulnerabilities of privileged programs by execution monitoring. *Proceedings of the 10th Annual Conference on Computer Security Applications*, pages 134–144, December 1994.

[67] C.K. Koc and C. Paar. Guest Editors' introduction to the special section on cryptographic hardware and embedded systems. *IEEE Transactions on Computers*, Vol.52, No.4, pages 401–402, April 2003.

[68] A.P. Kosoresow and S.A. Hofmeyr. Intrusion Detection via System Call Traces. *IEEE Sofware Journal*, pages 35–42, September-October 1997.

[69] A. Krogh, M. Brown, I.S. Mian, K. Sjolander and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, February 1994.

[70] S. Kumar. Classification and Detection of Computer Intrusions. *Doctor of Philosophy Thesis*, Purdue University, USA, August 1995.

[71] S. Kumar and E.H. Spafford. A software architecture to support misuse intrusion detection. *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.

[72] S. Kwong, Q. He and Y.K. Chan. Using MAP estimated parameters to improve HMM speech recognition performance. *IEEE Region 10 International Conference on Electrical and Electronic Technology*, Vol.1, pages 350–354, Aug 2001.

[73] T. Lane and C.E. Brodley. An Application of Machine Learning to Anomaly Detection. *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 366–380, February 1997.

[74] T. Lane and C.E. Brodley. Approaches to Online Learning and Concept Drift for User Identification in Computer Security. *American Association for Artificial Intelligence press*, 1998.

[75] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, Vol.2, No.3, pages 295331, August 1999.

[76] T. Lane. Machine Learning Techniques for the Computer Security Domain of Anomaly Detection. *Doctor of Philosophy Thesis*, Purdue University, USA, 2000.

[77] J. Langford. Optimizing Hidden Markov Model Learning. *Technical Report, No.RI 02912, Toyota Technological Institute at Chicago, Providence*, 2003.

[78] W. Lee, S.J. Stolfo and P.K. Chan. Learning patterns from UNIX process execution traces for intrusion detection. *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50-56, July 1997.

[79] W. Lee and S.J. Stolfo. Data Mining Approaches for Intrusion Detection. *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, USA, 1998.

[80] W. Lee, S.J. Stolfo and K.W. Mok. Algorithms for Mining System Audit Data. *Data Retrieval and Data Mining*. Kluwer Academic Publishers, 1999.

[81] W. Lee. A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems. *Doctor of Philosophy Thesis*, Columbia University, USA, 1999.

[82] W. Lee and S.J. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, Vol.3, No.4, pages 227–261, 2000.

[83] W. Lee, S.J. Stolfo, P. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop and J. Zhang. Real Time Data Mining-based Intrusion Detection. *DARPA Information Survivability Conference and Exposition II*, pages 85–100, 2001.

[84] S.C. Lee and D.V. Heinbuch. Training a Neural-Network Based Intrusion Detector to Recognize Novel Attacks. *IEEE Transactions On Systems, Man, And Cybernetics*, Vol.31, No.4, pages 294–298, July 2001.

[85] X. Li, M. Parizeau and R. Plamondon. Training Hidden Markov Models with Multiple Observations - A Combinatorial Method. *IEEE Transactions on PAMI*, Vol.PAMI-22, No.4, pages 371–377, 2000.

[86] Z. Liu, S.M. Bridges and R.B. Baughn. Classification of anomalous traces of privileged and parallel programs by neural networks. *IEEE Conference on Fuzzy Systems*, pages 1225–1230, St. Louis Missouri USA, May 2003.

[87] T.F. Lunt. Detecting Intruders in Computer Systems. *Conference on Auditing and Computer Technology*, 1993.

[88] J. Luo, S.M. Bridge and R.B. Vaughn. Fuzzy Frequent Episodes for Real-time Intrusion Detection. *The IEEE International Conference on Fuzzy Systems*, Melbourne, Australia, December 2-5, 2001.

[89] H. Mannila, H. Toivonen and I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery 1.* Kluwer Academic Publishers, pages 259–289, 1997.

[90] R.C. Marchany and J.G. Tront. E-Commerce Security Issues. *IEEE Proceedings of the* 35^{th} *Annual Hawii International Conference on System Sciences*, Hawii, 2002.

[91] G.A. Marin. Network Security Basics. *IEEE Security & Privacy Magazine*, Vol.6, No.6, pages 68–72, November-December 2005.

[92] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Hanbook of Cryptography.* CRC Press, 1997.

[93] C.C. Micheal and A. Ghosh. Simple, state-based approaches to program-based anomaly detection. *ACM Transactions on Information and System Security*, Vol.5, No.3, pages 203–237, 2002.

[94] Microsoft Corporation. MSDN Library. *http://msdn.microsoft.com/library/default.asp*, 2002.

[95] R.M. Neal and G.E. Hinton. *A new view of the EM algorithm that justifies incremental and other variants.* Department of Computer Science, University of Toronto, 1993.

[96] O. Nibouche, M. Nibouche and A. Bouridane. High speed FPGA implementation of RSA encryption algorithm. *IEEE International Conference on Electronics, Circuits and Systems*, pages 204–207, Sharjah, United Arab Emirates, December 2003.

[97] Y. Okazaki, I. Sato and S. Goto. A New Intrusion Detection Method based on Process Profiling. *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT'02)*, pages 82–90, 2002.

[98] R. Oppliger. Shaping the Research Agenda for Security in E-Commerce. *IEEE 10th International Workshop on Database & Expert Systems Applications*, Florence, Italy, September 1999.

[99] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *The 7th Annual USENIX Security Symposium*, January 1998.

[100] J. Pikoulas, W.J. Buchanan, M. Mannion and K. Triantafyllopoulos. An agent-based Bayesian forecasting model for enhanced network security. *Proceedings of the Eighth Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems*, pages 247–254, Los Alamitos, California, USA, 2001.

[101] P.A. Porras and P.G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. *Proceedings of the 20th National Conference on National Information Systems Security*, pages 353–365, NIST/NCSC, Baltimore, Maryland, USA, October 1997.

[102] Y. Qiao, X.W. Xin, Y. Bin and S. Ge. Anomaly intrusion detection method based on HMM. *IEEE Electronic Letters Online No: 20020467*, June 20, 2002.

[103] L.R. Rabiner and B.H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, Vol.3, pages 4–15, 1986.

[104] L.R. Rabiner. A tutorial on hidden Markov model and selected applications in speech recognition. *Proceedings of the IEEE*, Vol.77, No.2, February 1989.

[105] R.L. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Vol.21, No.2, pages 120–126, 1978.

[106] E. Rescorla. *Preventing the Million Message Attack on Cryptographic Message Syntax.* RFC 3218, January 2002.

[107] RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard.* RSA Laboratories, 2002.

[108] RSA Security. RSA Security website. *http://www.rsasecurity.com*, 2004.

[109] A. Royo, J. Morn and J.C. Lpez. Design and Implementation of a Coprocessor for Cryptography Applications. *Proceedings of European Design and Test Conference, 1997. ED&TC 97*, pages 213–217, March 17-20, 1997.

[110] T.A. Runkler. Extended defuzzification methods and their properties. *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, Vol.1, pages 694–700, September 1996.

[111] D. Russell and G.T. Gangemi. *Computer Security Basics*. O' Reilley, July 1991.

[112] I. Sato, Y. Okazaki and S. Goto. An Improved Intrusion Detection Method based on Process Profiling. *IPSJ Journal*, Vol.42, No.11, pages 3316–3326, November 2002.

[113] B. Schneier. *Applied Cryptography, Protocols, Algorithms, and source code in C*. John Wiley & Sons, Inc., 1996.

[114] B. Schneier. The Speed of Security. *IEEE Security & Privacy*, Vol.1, No.4, July-August 2003.

[115] School of Medicine, Washington University. HMMER: profile HMMs for protein sequence analysis. *http://hmmer.wustl.edu/*, 2005.

[116] R. Sekar, Y. Guang, S. Verma and T. Shanbhag. A High-Performance Network Intrusion Detection System. *Proceedings of the 6th ACM conference on Computer and communications security*, pages 8-17, Singapore, November 1999.

[117] R. Sekar, M. Bendre, D. Dhurjati and P. Bullineni. A fast automaton-based method for detecting anomalous program behaviours. *IEEE Symposium on Security and Privacy*, pages 144–155, 2001.

[118] M. Shand, F. Bertint and J. Vuillemin. Hardware speedups in long integer multiplication. *ACM Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, pages 138–145, Island of Crete, Greece, 1990.

[119] M.H. Sherif. *Protocols for Secure Electronic Commerce*. CRC Press, 2000.

[120] D. Snyder. Online Intrusion Detection Using Sequences of System calls. *Master of Science Thesis*, Department of Computer Science, The Florida State University, USA, January 2001.

[121] Spectrum Digital, Inc. TMS320C6416T DSK Technical Reference. *Spectrum Digital, Inc, Document Number: 508035-0001 Rev.A*, November 2004.

[122] W. Stallings. *Cryptography and Network Security.* Pearson Education, Inc., 2003.

[123] J.C. Stiller and G. Radons. Online Estimation of Hidden Markov Models. *IEEE Signal Processing Letters*, Vol.6, No.8, pages 213–215, August 1999.

[124] D. Taipale. Implementing the Rivest, Shamir, Adleman Cryptographic Algorithm on the Motorola 56300 Family of Digital Signal Processors. *IEEE Proceedings of Southcon*, pages 10–17, Orlando, FL, USA, June 1996.

[125] N. Takagi. A Radix-4 Modular Multiplication Hardware Algorithm Efficient for Iterative Modular Multiplications. *IEEE Transactions on Computers, Special issue on computer arithmetic*, Vol.41, pages 949–956, August 1992.

[126] N. Takagi and S. Yajma. Modular Multiplication Hardware Algorithm with a Redundant Representation and Their Application to RSA Cryptosystem. *IEEE Transactions on Computers*, Vol.41, No.7, pages 887–891, July 1992.

[127] Texas Instruments. Online document - TMS320C6416 - fixed point DSP. *http://focus.ti.com/docs/prod/folders/print/tms320c6416.html*, 2005.

[128] Texas Instruments. TMS320C6000 Programmer's Guide. *Texas Instruments, Literature Number: SPRU198F*, February 2001.

[129] Texas Instruments. TMS320C6000 DSP/BIOS Application Programming Interface (API) Reference Guide. *Texas Instruments, Literature Number: SPRU403C*, April 2001.

[130] Texas Instruments. TMS320C6000 Optimizing C Compiler User's Guide. *Texas Instruments, Literature Number: SPRU187I*, April 2001.

[131] N. Tomabechi and T. Ito. Design of High Speed RSA Encryption Processor with Built-in Table for Residue Calculation of Redundant Binary Numbers. *IEEE International*

Symposium on Circuits and Systems - ISCAS 2000, Geneva, Switzerland, May 28-31, 2000.

[132] University of New Mexico, USA. Computer Immune Systems Project. *http://www.cs.unm.edu/~immsec/systemcalls.htm*, 1998.

[133] J. R. Vacca. *Public key infrastructure: building trusted applications and web services.* Auerbach Publications, 2004.

[134] C. Warrender, S. Forrest and B. Perlmutter. Detecting intrusions using system calls: Alternative data models. *Proceedings of the 1999 IEEE Computer Society Sympo-sium on Research in Security and Privacy*, pages 133–145, Los Alamitos, California, USA, 1999.

[135] Wikipedia encyclopedia. English Wikipedia encyclopedia. *http://en.wikipedia.org/*, 2005.

[136] D. Yuliang, M. Zhigang, Y. Yizheng and W. Tao. Implementation of RSA Cryptoprocessor Based on Montgomery Algorithm. *Proceedings of Solid-State and Integrated Circuit Technology*, pages 524–526, October 21-23, 1998.

[137] L.A. Zadeh. Fuzzy sets. *Information and Control Journal*, Vol.8, pages 338-353, 1965.

[138] L.A. Zadeh. Fuzzy algorithms. *Information and Control Journal*, Vol.12, pages 94–102, 1968.

[139] L.A. Zadeh. Fuzzy logic. *IEEE Computer Journal*, Vol.21, No.4, pages 83–88, April 1988.

[140] L.A. Zadeh. Soft computing and fuzzy logic. *IEEE Software Journal*, Vol.11, No.6, pages 48–56, November 1994.

[141] L.A. Zadeh. Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, Vol. 4, No.2, pages 103–111, May 1996.

[142] R. Zhang, D. Qian, C. Ba, W. Wu and X. Guo. Multi-agent based intrusion detection architecture. *The 2001 International Conference Proceedings of Computer Networks and Mobile Computing*, pages 494–501, 2001.