UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

ANALYSING THE BEHAVIOUR OF A SMART CARD BASED MODEL FOR SECURE COMMUNICATION WITH REMOTE COMPUTERS OVER THE INTERNET

by

Deep Vardhan Bhatt

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

Department of Electrical, Electronic and Computer Engineering

UNIVERSITY OF PRETORIA

October 2010

SUMMARY

ANALYSING THE BEHAVIOUR OF A SMART CARD BASED MODEL FOR SECURE COMMUNICATION WITH REMOTE COMPUTERS OVER THE INTERNET

by

Deep Vardhan Bhatt

Promoters: Prof G. P. Hancke
Department: Electrical, Electronic and Computer Engineering
University: University of Pretoria
Degree: Master of Engineering (Computer Engineering)
Keywords: Smart card, cryptography, PKI, DES, security, ISO7816, SSL, SSH

This dissertation presents the findings of a generic model aimed at providing secure communication with remote computers via the Internet, based on smart cards. The results and findings are analysed and presented in great detail, in particular the behaviour and performance of smart cards when used to provide the cryptographic functionality. Two implemented models are presented. The first model uses SSL to secure the communication channel over the Internet while using smart cards for user authentication and storage of cryptographic keys. The second model presents the SSH for channel security and smart cards for user authentication, key storage and actual encryption and decryption of data.

The model presented is modular and generic by nature, meaning that it can easily be modified to accept the newer protocol by simply including the protocols in a library and with a minor or no modification to both server and client application software. For example, any new algorithm for encryption, key exchange, signature, or message digest, can be easily accommodated into the system, which proves that the model is generic and can easily be integrated into newer technologies.

Similarly, smart cards are used for cryptography. Two options are presented: first the smart cards only store the algorithm keys and user authentication, and secondly, smart cards are used for storing the algorithm keys, user authentication, and actual data encryption or decryption, as the requirement may dictate. This is very useful, for example, if data to be

transferred is limited to a few bytes, then actual data encryption and decryption is performed using smart cards. On the other hand, if a great deal of data is to be transferred, then only authentication and key storage are performed with smart cards. The model currently uses 3DES with smart card encryption and decryption, because this is faster and consumes fewer resources when compared to RSA. Once again, the model design is flexible to accommodate new algorithms such as AES or IDEA.

Important aspects of the dissertation are the study and analysis of the security attacks on smart card use. Several smart card attack scenarios are presented in CHAPTER 3, and their possible prevention is also discussed in detail.

OPSOMMING

DIE ONTLEDING VAN DIE GEDRAG VAN 'N SLIMKAARTGEBASEERDE MODEL VIR VEILIGE KOMMUNIKASIE MET AFGELEË REKENAARS OOR DIE INTERNET

deur

Deep Vardhan Bhatt

Promotors: Prof G. P. Hancke
Departement: Elektriese, Elektroniese en Rekenaaringenieurswese
Universiteit: Universiteit van Pretoria
Graad: Magister in Ingenieurswese (Rekenaaringenieurswese)
Sleutelwoorde: Sim kaart, kriptografie, PKI, DES, sekuriteit, ISO7816, SSL, SSH

Hierdie verhandeling bied die bevindinge van 'n generiese model wat daarop gemik is om veilige kommunikasie te voorsien met 'n afstandsrekenaar via die Internet en op slimkaarte gebaseer. Die resultate en bevindings word ontleed en breedvoerig aangebied, veral die gedrag en werkverrigting van slimkaarte wanneer hulle gebruik word om die kriptografiese funksionaliteit te voorsien. Daar word twee geïmplementeerde modelle aangebied. Die eerste model gebruik SSL om die kommunikasiekanaal oor die Internet te beveilig terwyl slimkaarte vir gebruikerbekragtiging en stoor van kriptografiese sleutels gebruik word. Die tweede model bied die SSH vir kanaalsekuriteit en slimkaarte vir gebruikergeldigheidvasstelling, sleutelstoor en werklike kodering en dekodering van data.

Die model wat aangebied word, is modulêr en generies van aard, wat beteken dat dit maklik gewysig kan word om die jongste protokolle te aanvaar deur bloot die protokolle by 'n programbiblioteek met geringe of geen wysiging van beide die bediener- en kliënttoepassingsagteware in te sluit. Byvoorbeeld, enige nuwe algoritme vir kodering, sleuteluitruiling, handtekening of boodskapbondeling kan maklik in die stelsel gehuisves word, wat bewys dat die model generies is en maklik in jonger tegnologieë geïntegreer kan word.

Slimkaarte word op soortgelyke wyse vir kriptografie gebruik. Daar word twee keuses aangebied: eerstens stoor die slimkaarte slegs die algoritmesleutels en gebruikergeldigheidvasstelling en tweedens word slimkaarte gebruik om die algoritmesleutels, gebruikergeldigheidvasstelling en werklike datakodering en –dekodering te stoor na gelang van wat vereis word. Dit is baie nuttig, byvoorbeeld, wanneer data wat oorgedra moet word, tot 'n paar grepe beperk is, word die eintlike datakodering en –dekodering uitgevoer deur slimkaarte te gebruik. Andersyds, indien 'n groot hoeveelheid data oorgedra moet word, word slegs geldigheidvasstelling en stoor met slimkaarte uitgevoer. Die model gebruik tans 3DES met slimkaartkodering en –dekodering omdat dit vinniger is en minder hulpbronne gebruik vergeleke met RSA. Die modelontwerp is weer eens buigsaam om nuwe algoritmes soos AES of IDEA te huisves.

Nog 'n belangrike aspek van die verhandeling is om die sekuriteitaanvalle op slimkaartgebruik te ondersoek en te ontleed. Verskeie slimkaartaanvalscenario's word in Hoofstuk 3 aangebied en die moontlike voorkoming daarvan word ook breedvoerig bespreek.

ACKNOWLEDGEMENTS

I would like to begin by thanking my supervisor, Professor G. P. Hancke, for his continuous encouragement and guidance, as well as financial support. A word of thanks goes to all the friends and colleagues who helped me with gathering and collecting most of the required documentation, knowledge, and software tools to facilitate the accomplishment of my tasks.

My special thanks go to Paul Greeff who gave me a great deal of his time and advice on several topics, especially Linux-based assistance.

Many thanks to Stephan Schultze and Francois Blignaut, who worked long hours, and spent many late nights implementing, integrating and testing the software.

Sincere thanks go to Mr. A. N. Pandey, Alcatel Cape Town, for allowing me to use their site facilities to test the system for its actual remote server connection.

Lastly, thanks to my wife who spared me many sleepless nights. Her unseen support was there with me throughout the project and it would not have been possible without it.

Not forgetting the ALLMIGHTY, no doubt nothing is possible without his mercy and blessings.

LIST OF ABBREVIATIONS

AC	Access Control
AH	Authentication Header
AID	Application Identifiers
API	Application Programming Interface
CAD	Card Acceptance Device
CAP	Converted Applet
CIG	Client-Internet-Gateway
OFB	Output Feedback
DFA	Differential Fault Analysis
DOS	Denial of Service
DPA	Differential Power Analysis
ESP	Encapsulating Security Payload
FAN	Fieldbus Area Network
FIB	Focused Ion Beam
GGSN	Gateway GPRS Support Node
GSS-API	Generic Security Services API
IDE	Integrated Development Environment
IKE	Internet Key Exchange
J2SSH	Java Secure Shell toolkit
KDC	Key distribution centres
OTP	One-time Pad
PUK	Personal Unblocking Key
RID	Registered Application Provider Identifier
SA	Security Association
SCP	Secure Copy
SFTP	Secure File Transfer Protocol
SGSN	Serving GPRS Support Node
SQUID	Superconducting Quantum Interference Devices
SPA	Simple Power Analysis
WSC	Windows for Smart Card

TABLE OF CONTENTS

CHAPTER 1
RESEARCH OVERVIEW

1.1 INTRODUCTION

This chapter briefly outlines the security concerns when using the Internet or any other insecure transmission channel to transfer or exchange data. The first section takes a look at security concerns and defines security. The second section presents and then defines the problem that needs to be addressed. The next few sections provide the dissertation scope, context, objective, and contribution. Lastly, the dissertation outline is provided.

1.2 SECURITY CONCERNS

Privacy is a basic human right, but on today's computer networks, there is no guarantee of privacy. Much of the data that travels on the Internet or local network is transmitted as plain text (TCP/IP), and may be captured and viewed by anybody with a little technical know-how. The files, emails, and even the passwords typed may be readable by others. Internet use has increased tremendously in the past decade. Despite several advantages and the use of Internet technology, it is becoming very challenging and important to safeguard the use of this technology. The strong emphasis on security is unsurprising, given the large number of security breaches taking place around the world and the increased focus in organisations on risk management, corporate governance and regulatory compliance. Businesses in today's security-conscious society are all too aware of the damage that a security attack can inflict on their corporate systems and confidential data, not to mention the company's market value. Research provider Key Note, estimates security breaches in the UK last year cost businesses more than £44 billion - a figure that is likely to rise even higher in coming years. Clients who intend using Internet technology and those who are already using this technology have to take proper measures to safeguard their valuable and confidential data and must be ahead of those who intend to violate their security.

1.3 WHAT IS SECURITY?

Security is basically the protection of something valuable to ensure that it is not stolen, lost, or altered. The term "data security" governs an extremely wide range of applications and touches everyone's daily life. Concerns over data security are at an all-time high, due to the rapid advancement of technology into virtually every transaction, from parking meters to national defence. Data is created, updated, exchanged and stored via networks. A network is any computing system where users are highly interactive and interdependent and by definition, not all are in the same physical place. In any network, diversity abounds, certainly in terms of types of data, but also types of user. For these reasons, a system of security is essential to maintain computing and network functions and keep sensitive data secret. It is this aspect of security that is investigated in great depth and several findings are presented.

In order to tackle the security issues, it is best first to understand the security and the issues surrounding the security and then define a threat model from the designers' point of view. In a threat model, it is necessary to understand what resources the attacker has and what attacks the attacker can be expected to mount. Possibly, every system is vulnerable to some sort of attack, i.e. no security system is resistant to every attack [1]. It is from this perspective that this dissertation will try to tackle threats and present a security model that is realistic and also the threats that can be countered with available tools.

1.4 DEFINING THE CONCERNS

This dissertation looks at means of providing secure remote access to a fieldbus, (family of industrial computer network protocols used for real-time distributed control, now standardised as IEC 61158). A general concern is that when a remote client, using an open medium such as TCP/IP (Transport Control Protocol/Internet Protocol) over the Internet (an insecure medium), tries to gain access to a remote gateway server (a dedicated workstation for specific applications) to transfer data to or from it, or to issue some commands to perform some action on some sort of device connected to this gateway server via a fieldbus that is providing some essential services can be subject to several security threats. Figure 1.1 illustrates this scenario.

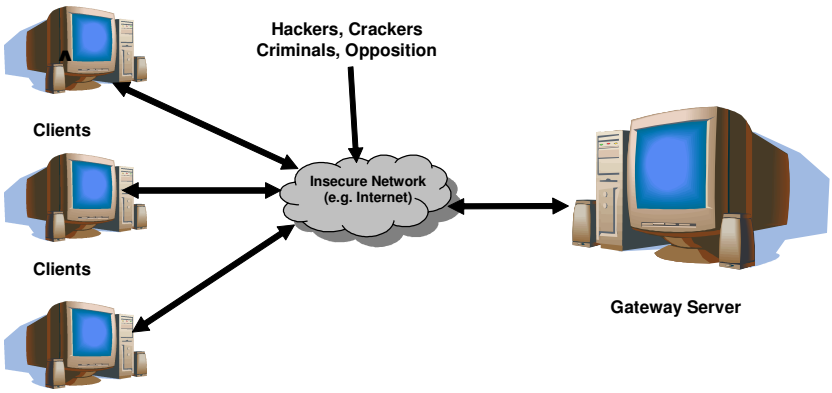

Figure 1.1 Insecure model to be considered.

Hence it is important to provide a secure connection to the fieldbus system over the Internet as it is very useful for maintenance, data capturing and control in various industrial and manufacturing processes. This dissertation investigates the security issues and presents a solution to overcome these security issues. Before pointing out the detailed research to be conducted in this dissertation, an overall outline of the system is presented next.

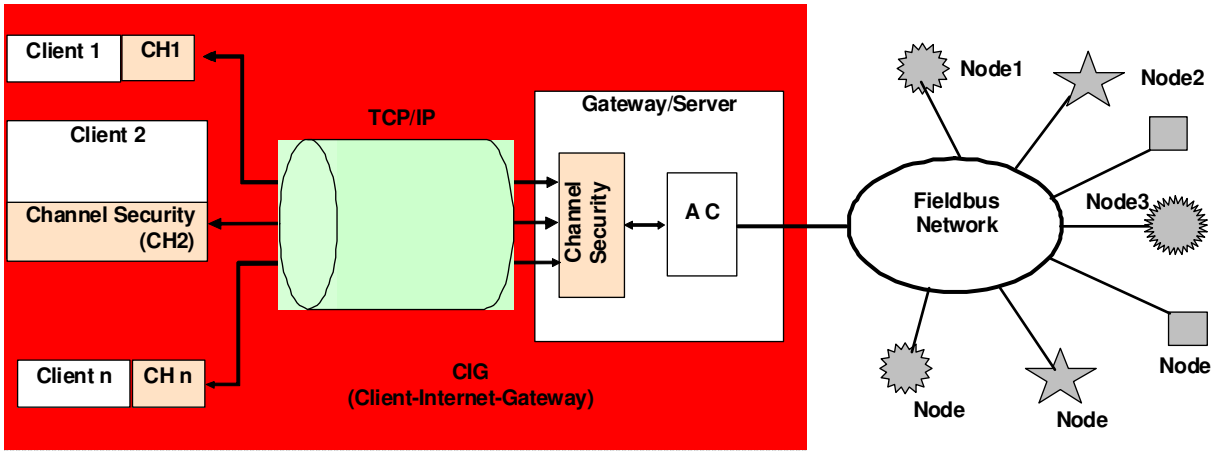

Figure 1.2 An overview of the Project.

The overall system, as shown in Figure 1.2, will provide its clients with secure remote access to the Fieldbus Network via a gateway server. The fieldbus provides accessibility to various types of data node from which measurement data can be collected. Clients who connect to the gateway, through the channel security server program, can then access these

nodes over the Internet. Security problems in Figure 1.2 can be subdivided into four security aspects. The first is to have a good security policy for the project which is presented by Burger [2]. The second issue of concern pertains to the access of fieldbus nodes, presented by Smit, et al [3]. The third security issue in the project is the access control (AC) to the fieldbus network, which is presented by Greeff, et al [4]. The final security issue is the client-end machine; the Internet; the gateway server. This dissertation focuses on the security concerns of the CIG (Client-Internet-Gateway) link.

1.5 RESEARCH CONTEXT

The fear of security breaches has been a major and prime concern for both the business world and individuals. This has not stopped the growth of Internet use by business for an effective and faster means of communication across the globe. Since the introduction of cryptography several decades ago, today, cryptography is an essential and integral part of security in every communication and information system [5].

Cryptography is finding a new dimension and greater use, with smart cards in particular achieving authentication [6]-[8] and key storage. Data security can be significantly enhanced by the use of smart cards for highly confidential information. Smart cards strongly confine keys and algorithms, and the security of these keys and algorithms relies on specific software on the smart card, sometimes referred to as secureware [1], this dissertation does not scrutinise this aspect in great detail, rather brief summarised findings will be presented. Several studies have indicated that the use of smart cards is just the start of a movement toward the bigger issues of security that can be implemented using cryptography, with a significant increase in smart-card-processing capability and a tremendous decline in cost [9]. This fact cannot be ignored, and with this in mind, smart cards with cryptographic capability are chosen to provide a secure environment between CIG.

1.6 SCOPE

This dissertation investigates the security aspect and issues related to the CIG link (Figure 1.2), the channel between two remote clients over the Internet using TCP/IP. The dissertation presents a simple, realistic and cost-effective solution towards achieving high security for data integrity, confidentiality, nonrepudiation, as well as client authentication, and data encryption using smart cards. Specifically, the dissertation focuses on applications

where lower data transmission rates and the number of clients requesting the service are limited. In order to achieve the above scope, the following will be investigated:

- Use of smart cards in providing cryptography.
- Safety measures to be considered with the use of smart cards.
- Performance of smart cards with multiple connections.
- Safe storage of keys in smart cards.
- Implementation of a secure system using available security protocols that will suit smart cards.

The dissertation does not analyse the security policy issues as they are covered in Burger [2], neither does it present any new security algorithm or protocol technologies. Rather it focuses on the existing protocols and algorithms to present a cost-effective solution for securing the communication channel and secure transmission of data between remote computers. However, the dissertation introduces the use of smart cards with cryptography alongside these existing protocols and analyses the performance of the server.

1.7 DISSERTATION OBJECTIVE

In order to meet the set requirements, a number of client programs will need a secure connection to a central server, situated on the gateway, across the Internet. First, the secure connection should provide the security services of confidentiality, integrity, nonrepudiation and authentication. The server program will manage the secure connection and provide a proxy service for the client data that it receives. A “proxy service” in this context refers to the fact that the server program will take all the data that it receives and forward the data to another server without needing to know what the data means or will be used for. Secondly, the research lists and discusses the effects on the server under different load conditions with different key and algorithm choices. Thirdly, research findings are provided to explain how to use smart card safely with security models as they form a critical part of achieving security. Several threat scenarios for smart cards are presented that need to be kept in mind. More specifically, physical and logical level smart card attacks are analysed and a countermeasure is suggested for these attack situations. With these facts in mind, smart cards will be used to provide authentication, safe storage of keys, and data encryption/decryption. Lastly, the aim of this dissertation is to develop and implement these objectives into a generic security model to support the design, and yet be able to use

most available technologies to transfer data securely, using existing secure communication protocols and smart cards for cryptography and key storage and distribution.

1.8 CONTRIBUTION

The research will contribute towards the development of a security model that enables the clients to access the resources on a remote server or gateway using the proposed secure communication models. The research will present an achievable solution to bridge the gap between the security of the data transmitted over an insecure communication channel (Internet) with acceptable performance degradation. Security implementation will offer: safekeeping of cryptographic keys; safe key distribution; maximum security for data and communication channels, while being practical and effective in their use and implementation.

Security systems mainly employ smart-card-with-user-authentication mechanisms for key storage. Their actual encryption and decryption capabilities are simply not fully used. The major shortcoming in knowledge is the lack of effective and easy implementation that can unite these technologies cost-effectively and securely [10] [11]. The dissertation presents the successful implementation of these two technologies and provides detailed findings on the types of security attack to which a smart card can be subjected. More specifically, smart cards are examined extensively in terms of two levels of attack; - logical and physical that can significantly influence the cryptographic use of smart cards in security protocols (CHAPTER 3).

The publications and reports arising from or benefited by the completion of this dissertation are [12], [13], [14].

1.9 DISSERTATION OUTLINE

This section outlines the approach taken to provide a security system that will fit its requirements. However, the security schemes presented are not completely new in their concept, but are unique on their own merits, as there are few known systems today that are actually using smart cards for client authentication, data integrity, nonrepudiation, and data confidentiality all combined.

A detailed study of cryptography is presented in CHAPTER 2. First the chapter outlines the basic cryptography. Then the chapter reviews the strengths and weaknesses of these

existing algorithms and indicates which of these could provide the security that is needed to cover the proposed threat model. Another point, that needs attention, is performance issues of these algorithms when implemented with smart cards, as some of these cryptographic algorithms demand much from the limited hardware resources. Hence, the most secure cryptographic algorithm may not be the best solution to suit the threat model. Rather, a realistic approach is implemented that is based on the easy availability of smart cards in South Africa, the types of resource these cards have, and the algorithms they can support.

CHAPTER 3 presents in great detail the architecture, standards and the operating systems used by smart cards. The chapter also supports and justifies the use of smart cards with cryptography to provide secure data transactions. Later in the chapter, various threat scenarios are presented in great detail, which provides various security safeguard measures that need to be considered when using smart cards to their full cryptographic potential.

A brief description of several commonly and widely available security protocols is presented in CHAPTER 4. Proposed models for secure communication over the Internet are presented in CHAPTER 5. CHAPTER 6 and CHAPTER 7 present the implementation of these security protocols with smart cards. In order for the proposed security schemes to be successfully implemented, the choices of software package are also very important. Some of these design approaches will have a significant impact on implementation, integration and system performance.

CHAPTER 8 presents the results obtained for the two security protocols when implemented with smart cards. Several test bench setups are described and their successful outcomes are shown. Issues such as the cost of handshaking, performance of server under load, throughput and response time of server findings are presented in detail.

Finally, CHAPTER 9 concludes the dissertation by summing up all the findings gained during the completion. Lastly, some common concerns are mentioned that will be beneficial to all future security system implementation.

CHAPTER 2
CRYPTOGRAPHY

2.1 INTRODUCTION

Cryptography has a long and fascinating history. Beginning with the work of Feistel at IBM in the early 1970s, and DES (the Data Encryption Standard), it is the best-known cryptographic mechanism in history [15]. The most striking development in the history of cryptography came in 1976 when Diffie and Hellman introduced the revolutionary concept of public-key cryptography and also provided a new and ingenious method for key exchange, the security of which is based on the intractability of the discrete-logarithm problem. In 1978 Rivest, Shamir, and Adleman discovered the first practical public-key encryption and signature scheme, now referred to as RSA. Another class of powerful and practical public-key schemes was found by ElGamal in 1985. These are based on the discrete-logarithm problem. One of the most significant contributions provided by public-key cryptography is the digital signature. It is a building block for many other services such as nonrepudiation, data origin authentication, and identification. Achieving information security in an electronic society requires a vast array of technical skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography.

Cryptography is the study of mathematical techniques related to aspects of information security such as –confidentiality; -data integrity; -entity authentication, data origin authentication; and -non-repudiation. Cryptography is not the only means of providing information security, but rather one set of techniques.

Cryptography is about the prevention and detection of cheating and other malicious activities. Cryptographic tools (primitives) are used to provide information security, examples of primitives include -encryption schemes like symmetric/public key encryption; -authentication and identification; -hash functions; and -digital signature schemes discussed later in this chapter

2.2 ENCRYPTION AND DECRYPTION

One of the main uses of encryption is to provide confidentiality. Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of transforming encrypted information so that it is intelligible again. With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a key that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Encryption techniques are typically divided into two generic types: symmetric-key and public-key.

2.2.1 Symmetric-key cryptography

Secret or symmetric key algorithms use the same key for both encryption and decryption (or one is easily derivable from the other). This is the more straightforward approach to data encryption, it is mathematically less complicated than public key cryptography, and has been used for many centuries. A two-party communication using symmetric-key encryption can be described by the Figure 2.1. One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely. This problem is referred to as the key distribution problem, discussed in section 2.7.

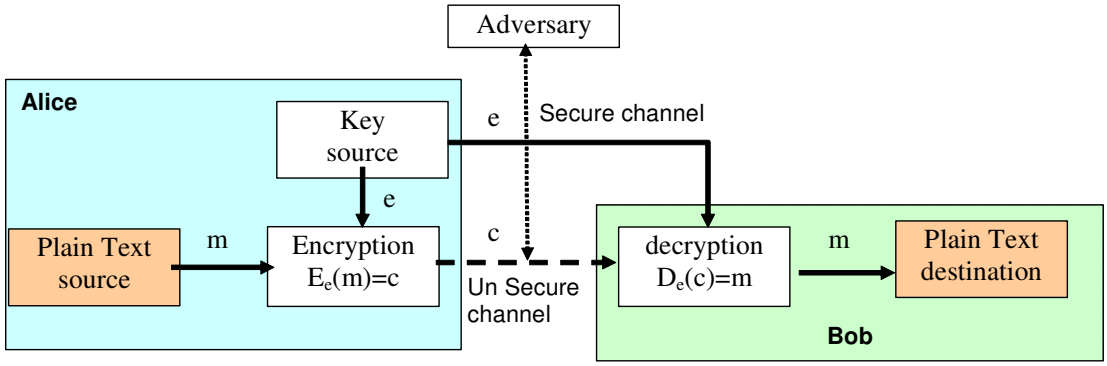

Figure 2.1: Two-party communication using encryption, with a secure channel for key exchange.

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Symmetric-key encryption is effective only if the symmetric key is kept secret by both the communicating parties involved. If anyone else discovers the key, it affects both

confidentiality and authentication. There are two classes of symmetric-key encryption schemes:

***Block cipher**:* is an encryption scheme which breaks up the plaintext messages to be transmitted into strings (called blocks) of a fixed length, and encrypts one block at a time. Typical modes of these ciphers are ECB (Electronic codebook); CBC (cipher block chaining), OFB (output feedback); and CTR (CounTeR).

***Stream ciphers*:** Stream ciphers are advantageous in situations where transmission errors are highly probable, because they have no error propagation. They can also be used when the data must be processed one symbol at a time.

2.2.2 Secret key algorithms examples

DES - Data Encryption Standard, is an algorithm developed in the mid-1970s. DES is a block cipher with 64-bit block size. It uses 56-bit keys. DES is still strong enough to keep most random hackers and individuals out, but it is easily breakable with special hardware by government, criminal organizations, or major corporations. A variant of DES, Triple-DES (also 3DES) is based on using DES three times (normally in an encrypt-decrypt-encrypt sequence with three different, unrelated keys).

***AES - Advanced Encryption Standard*.** In response to the growing feasibility of attacks against DES, NIST launched a call for proposals for an official successor that meets 21st century security needs. This successor is called the **AES**. All the five ciphers have *128* bit block size and they support *128*, *192* and *256* bit keys. AES performs very well in hardware and software across a wide range of environments in all possible modes. It has excellent key setup time and has low memory requirements, in addition its operations are easy to defend against power and timing attacks. After all analysis and received comments were considered, NIST considered Rijndael the best choice for the AES [16].

Serpent was another AES finalist. It is in many ways opposite of AES. Where AES puts emphasis of elegance and efficiency, Serpent is designed for security all the way. Serpent has a basically conservative but in many ways innovative design. It may be implemented by bit-slice (or vector) gate logic throughout. This makes it rather complicated to implement from scratch, and writing it in a non-bit-slice way involves an efficiency penalty. The *32* rounds lead to probably the highest security margin on all AES candidates, while it is still fast enough for all purposes [10]. Its biggest disadvantage is that it is about one-third the speed of AES It can be difficult to implement efficiently, as the S-

boxes have to be converted to a Boolean formula suitable for the underlying central processing unit (CPU).

Twofish uses same Fiestel structure as DES, but utilizes many different ideas. This cipher has key dependent S-boxes like Blowfish [10]. The design is highly delicate, with many alternative ways of implementation. It is cryptanalysed in much detail, by the very authoritative "extended Twofish team". It was also AES finalist, and is a compromise between AES and Serpent. It is as fast as AES, but has a large security margin. It uses a complex algebraic representation and its biggest disadvantage is that it can be rather expensive to change the encryption key, as it is best implemented with a lot of pre computation on the key.

Blowfish, Bruce Schneier designed blowfish [17]. It is a block cipher with 64-bit block size and variable length keys (448 bits). Blowfish was designed to be easy to implement and to have a high execution speed. It is very compact algorithm that requires less than 5K of memory. Blowfish utilizes the idea of randomized S-boxes: while doing key scheduling, it generates large pseudo-random look-up tables by doing several encryptions. The tables depend on the user-supplied key in a very complex way. This approach has been proven to be highly resistant against many attacks such as differential and linear cryptanalysis. Unfortunately it also means that it is not the algorithm of choice for environments where large memory space (something like than *4096* bytes) is not available. The only known attacks against Blowfish are based on its weak key classes.

IDEA - International Data Encryption Algorithm, is an algorithm developed at ETH Zurich in Switzerland by Xuejia Lai. It uses a 128-bit key, and it is generally considered to be very secure. It has been one of the best publicly known algorithms for some time (before the AES standard started its second round, see above). It has been around now for several years, and no practical attacks on it have been published despite of numerous attempts to analyze it. One of the best attacks uses the impossible differential idea of Biham, Shamir and Biryukov [18]. They can attack only *4.5* rounds of IDEA and this poses no threat to the total of *8.5* rounds used in IDEA.

MARS. This is an interesting new design (using a special type of a Feistel network), which depends heavily on the instruction sets available on modern 32-bit processors. This has the benefit that on these target machines it is efficient, but it may lead to implementation difficulties in cheaper architectures like smart cards [10].

RC6 follows the ideas of RC5 - but with many improvements. For example, it attempts to avoid some of the differential attacks against RC5's data dependent rotations. However, there are some attacks that get quite far, and it is unclear whether RC6 is well enough analyzed yet. It also uses 32 bit multiplication in the cipher.

2.2.3 Public-key cryptography

Public-key encryption, also called asymmetric encryption, involves a pair of keys: a public key and a private key, associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Figure 2.2 shows the two parties communicating using public-key.

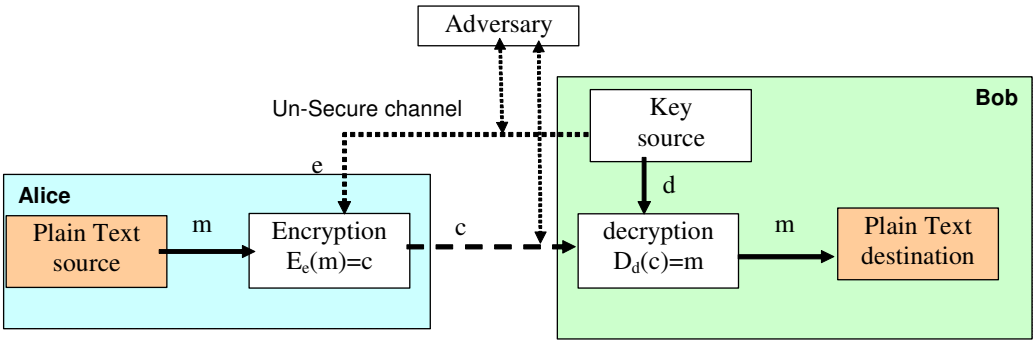

Figure 2.2 Encryption using public-key techniques

Each public key is published, and the corresponding private key is kept secret. Thus the owner of the private key would be the only one who could decrypt the messages, but anyone knowing the public key could send them in privacy. Notice how Figure 2.2 differs from Figure 2.1 for a public-key cipher. Here the encryption key is transmitted to Alice over an unsecured channel. This unsecured channel may be the same channel on which the cipher text is being transmitted. Since the encryption key need not be kept secret, it may be made public via any means. Any entity can subsequently send encrypted messages to Bob which only Bob can decrypt. The issue with public key is not the key secrecy but key distribution and key authenticity as discussed next.

2.2.3.1 The necessity of authentication in public-key systems

It would appear that public-key cryptography is an ideal system, not requiring a secure channel to pass the encryption key. This would imply that two entities could communicate over an unsecured channel without ever having met to exchange keys. Unfortunately, this is not the case. Figure 2.3 illustrates how an active adversary can defeat the system, decrypt messages intended for a second entity, without breaking the encryption system. This highlights the necessity to authenticate public keys to achieve data origin authentication of the public keys themselves. Key establishment, management, and certification looks into this problem and are discussed in section 2.7.

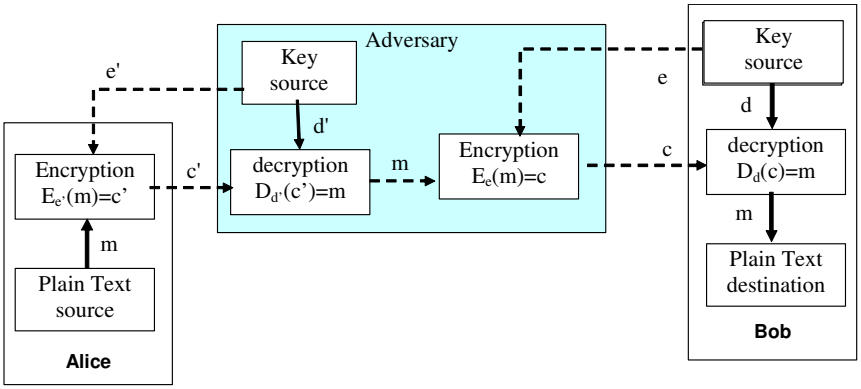

Figure 2.3: An impersonation attack on a two-party communication.

2.2.4 Public-key algorithm examples

RSA (Rivest-Shamir-Adleman) is the most commonly used public key algorithm. It can be used both for encryption and for digital signatures. The security of RSA is generally considered equivalent to factoring, although this has not been proved. The problem for the attacker is that computing the reverse *d* of *e* is assumed to be no easier than factorizing *n*. Dramatic advances in factoring large integers would make RSA vulnerable, but other attacks against specific variants are also known. RSA is vulnerable to chosen plaintext, and hardware and fault attacks. Also important attacks against very small exponents exist, as well as against partially revealed factorization of the modulus [17].

The ***Rabin*** cryptosystem may be seen as a relative of RSA, although it has a quite different decoding process. What makes it interesting is that breaking Rabin is provably equivalent to factoring. Rabin uses the exponent *2* (or any even integer) instead of odd integers like RSA. This has two consequences. First, the Rabin cryptosystem can be proven to be equivalent to factoring; second, the decryption becomes more difficult - at least in some sense.

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. RSA algorithm, are secure assuming that it is difficult to factor a large integer composed of prime factors. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of an elliptic curve element is unfeasible. The size of the elliptic curve determines the difficulty of the problem. It is believed that a smaller group can be used to obtain the same level of security as RSA-based systems. Using a small group reduces storage and transmission requirements.

2.3 SYMMETRIC KEY VS. PUBLIC KEY CRYPTOGRAPHY

Symmetric-key and public-key encryption schemes have various advantages and disadvantages, some of which are common to both. Numbers of these features are presented.

2.3.1 Advantages of symmetric-key cryptography

- Symmetric-key ciphers can be designed to have high rates of data throughput.
- Relatively short keys.
- Can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions, and computationally efficient digital signature schemes.

2.3.2 Disadvantages of symmetric-key cryptography

- In a two-party communication, the key must remain secret at both ends.
- In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted party like certification authority (CA).
- In a two-party communication key be changed frequently, and perhaps for each communication session.

- Digital signature mechanisms typically require either large key for the public verification function.

2.3.3 Advantages of public-key cryptography

- Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
- The administration of keys on a network requires the presence of only a functionally trusted authority as opposed to an unconditionally trusted authority. Depending on the mode of usage, the trusted authority might only be required in an "off-line" manner, as opposed to in real time.
- Depending on the mode of usage, a private/public key pair may remain unchanged for considerable periods of time, e.g., many sessions.
- Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
- In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

2.3.4 Disadvantages of public-key encryption

- Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes.
- Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

Symmetric-key and public-key encryption have a number of complementary advantages. Public-key encryption techniques may be used to establish a key for a symmetric-key system. In this scenario two parties can take advantage of the long term nature of the public-key scheme and the performance efficiencies of the symmetric-key scheme. Since data encryption is frequently the most time consuming part of the encryption process, the public-key scheme for key establishment is a small fraction of the total encryption process. To date, the computational performance of public-key encryption is inferior to that of symmetric-key encryption. The important points in practice are: -public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key management; and -symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.4 HASH FUNCTIONS

One of the fundamental primitives in modern cryptography is the cryptographic hash function, often informally called a one-way hash function (also called a message digest). A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values. The most common cryptographic uses of hash functions are with digital signatures and data integrity.

When used for digital signatures, a long message is usually hashed (using a publicly available hash function) and only the hash-value is signed. The party receiving the message then hashes the received message, and verifies that the received signature is correct for this hash-value. This saves both time and space compared to signing the message directly, which would typically involve splitting the message into appropriate-sized blocks and signing each block individually. Note here that the inability to find two messages with the same hash-value is a security requirement, since otherwise, the signature on one message hash-value would be the same as that on another, allowing a signer to sign one message and at a later point in time claim to have signed another.

Hash functions may be used for data integrity as follows. The hash-value corresponding to a particular input is computed at some point in time. The integrity of this hash-value is protected in some manner. At a subsequent point in time, to verify that the input data has not been altered, the hash-value is recomputed using the input at hand, and compared for equality with the original hash-value.

Some of the best known and most widely used hash functions are:

SHA, Secure Hash Algorithm and **SHS**, Secure Hash Standard. It produces a 160 bit hash value from an arbitrary length string. The NIST has issued SHA-256, SHA-384, and SHA-512 hash algorithms and are to be designed with 128, 196, and 256-bit key size of AES respectively. SHA-1 and RipeMD-160 are two examples that are still considered to be safe.

MD5 (Message Digest Algorithm 5), is a cryptographic hash algorithm developed at RSA laboratories. It can be used to hash an arbitrary length byte string into a *128* bit value. MD5's ancestor, MD4 has been broken, and there are some concerns about the safety of MD5 as well, although still in widespread use, should be considered insecure [17].

RIPEMD-160, is a hash algorithm designed to replace MD4 and MD5. It produces a digest of *20* bytes (*160* bits).

Cryptographic hash functions typically produce hash values of 128 or more bits. This number (2^{128}) is vastly larger than the number of different messages likely to ever be exchanged in the world. The reason for requiring more than 128 bits is based on the birthday paradox [15]. The birthday paradox roughly states that given a hash function mapping any message to a 128-bit hash digests, it can be expect that the same digest will be computed twice when 2^{64} randomly selected messages have been hashed. As cheaper memory chips for computers become available it may become necessary to require larger than 128 bit message digests.

2.4.1 Message authentication codes (MAC)

Hash functions as discussed above are typically publicly known and involve no secret keys. Related to these are hash functions which involve a secret key, and provide data origin authentication as well as data integrity; these are called message authentication codes (MACs). Encryption prevents message from being read but does not provide protection against tampering or manipulation, MAC does this. MACs use secret key that must be shared between the communicating parties. When Alice wants to communicate with Bob she sends the message and a MAC value computed by the MAC function. Bob then re-computes the MAC from the received message using the shared key.

HMAC, typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. HMAC is a MAC mechanism based on cryptographic hash functions, RFC 2104. HMAC can be used in combination with any iterated cryptographic hash function. MD5 and SHA-1 are examples of such hash functions. HMAC also uses a secret key for calculation and verification of the message authentication values.

2.5 DIGITAL SIGNATURES

The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of information. The process of signing entails transforming the message and some secret information held by the entity into a tag called a signature. Digital signatures are used to verify that a message really comes from the claimed sender. They can also be used to timestamp documents: a trusted party signs the document and its timestamp with his/her secret key, thus testifying that the document existed at the stated time. Digital signatures can also be used to testify (or certify) that a public key belongs to a particular person.

Encryption and decryption address the problem of eavesdropping (information remains intact, but its privacy is compromised), but by themselves, do not address the problems of tampering (integrity) and impersonation (authentication). This section describes how public-key cryptography addresses the problem of tampering.

As mentioned in Public-Key Encryption, it's possible to use your private key for encryption and your public key for decryption. Although this is not desirable when you are encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, and then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature**.** Figure 2.4 shows a simplified view of the way a digital signature can be used to validate the integrity of signed data.

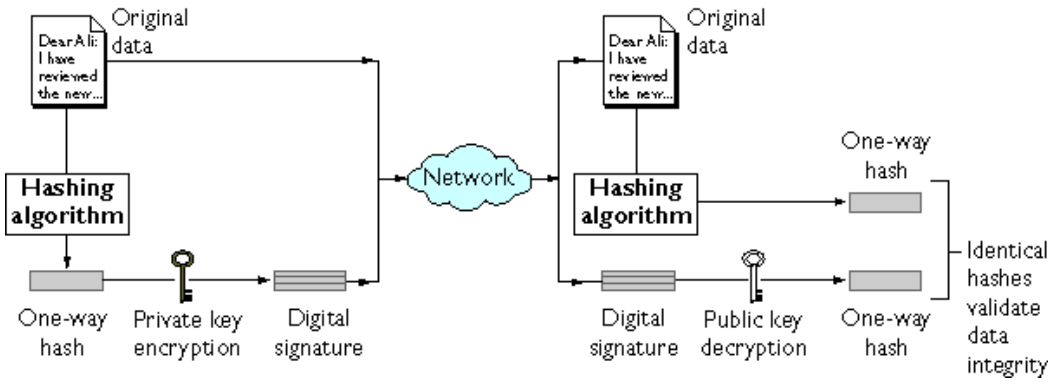

Figure 2.4: Data integrity using digital signature [17].

Figure 2.4 shows two items transferred to the recipient of some signed data: the original data and the digital signature, which is basically a one-way hash (of the original data) that has been encrypted with the signer's private key. If the two hashes match, the data has not changed since it was signed. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer. Confirming the identity of the signer, however, also requires some way of confirming that the public key really belongs to a particular person or other entity.

DSS (Digital Signature Standard), underlying algorithm DSA (Digital Signature Algorithm) is similar to the one used by ElGamal or by the Schnorr signature algorithm. Also it is fairly efficient, although not as efficient as RSA for signature verification. The standard defines DSS to use the SHA-1 hash function exclusively to compute message digests. The main problem with DSS is the fixed subgroup size (the order of the generator element), which limits the security to around only *80* bits. Hardware attacks can be a concern to some implementations of DSS. However, it is widely used and accepted as a good algorithm.

2.6 CERTIFICATES, AUTHENTICATION AND IDENTIFICATION

A certificate is an electronic document used to identify an individual, a server, a company, or some other entity and to associate that identity with a public key. Public-key cryptography uses certificates to address the problem of impersonation. Certificates work much the same way as any of these familiar forms of identification, person passport, driving licence, etc. Certificate authorities (CAs) are entities that validate identities and issue certificates. They can be either independent third parties or organizations running their own certificate-issuing server software. The methods used to validate an identity vary depending on the policies of a given CA, before issuing a certificate; the CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate. Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate. In large organizations, it may be appropriate to delegate the responsibility for issuing certificates to several different certificate authorities. For example, the number of certificates required may be too large

for a single CA to maintain; different organizational units may have different policy requirements; or it may be important for a CA to be physically located in the same geographic area as the people to whom it is issuing certificates. It's possible to delegate certificate-issuing responsibilities to subordinate CAs. The X.509 standard includes a model for setting up a hierarchy of CAs like that shown in Figure 2.5.

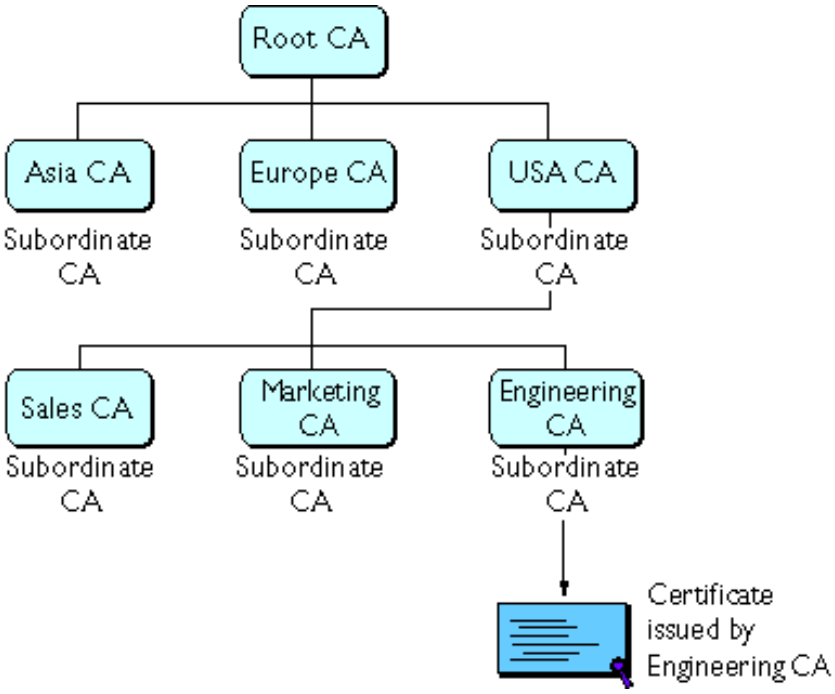

Figure 2.5 Hierarchy of certificate authorities [17].

Authentication confirms an Identity, or is the process of confirming an identity. In the context of network interactions, authentication involves the confident identification of one party by another party. Authentication over networks can take many forms. Certificates are one way of supporting authentication.

Data origin authentication or message authentication techniques provide to one party which receives a message assurance of the identity of the party which originated the message. Data origin authentication implicitly provides data integrity since, if the message was modified during transmission, sender would no longer be the originator.

Identification or entity authentication technique assures one party of both the identity of a second party involved, and that the second was active at the time the evidence was created

or acquired. Client/Server authentication refers to the confident identification of a client/server by a server/client. Two forms of authentication:

- Password-Based Authentication. The server maintains a list of names and passwords; if a particular name is on the list, and if the user types the correct password, the server grants access.
- Certificate-Based Authentication: client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server uses techniques of public-key cryptography to validate the signature and confirm the validity of the certificate.

2.7 KEY MANAGEMENT AND DISTRIBUTION

This section gives a brief introduction to methodology for ensuring the secure distribution of keys for cryptographic purposes. Key establishment is any process whereby a shared secret key becomes available to two or more wanted parties, for subsequent cryptographic use. Key management is the set of processes and mechanisms which support key establishment and the maintenance of ongoing keying relationships between parties, including replacing older keys with new keys as necessary. Key establishment can be broadly subdivided into key agreement and key transport.

Many protocols have been proposed to provide key establishment. Simple architectures are based on symmetric/public-key cryptography along with the concept of certification. A major issue when using symmetric-key techniques is the establishment of pair wise secret keys. This becomes more evident when considering a network of entities, any two of which may wish to communicate. Figure 2.6 illustrates a network consisting of 6 entities. The arrowed edges indicate the 15 possible two-party communications which could take place. Since each pair of entities wishes to communicate, this small network requires the secure exchange of $\binom{6}{2}=15$ key pairs. In a network with n entities, the number of secure key exchanges required is $\binom{n}{2}=\frac{n(n-1)}{2}$

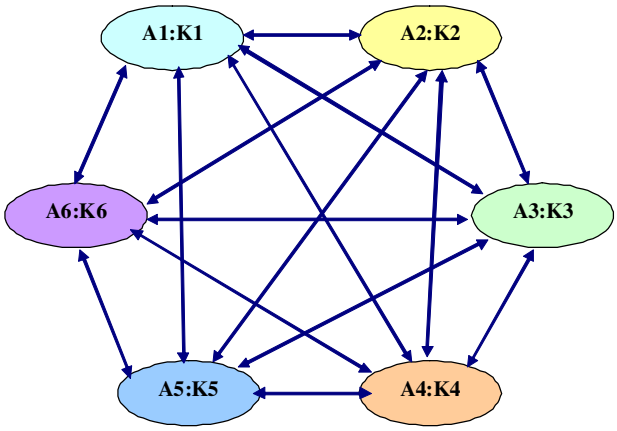

Figure 2.6 Six party keying relationships in a simple network [17].

In practice, networks are very large and the key management problem is a crucial issue. There are a number of ways to handle this problem. Two simplistic methods are discussed; one based on symmetric-key and the other on public-key techniques.

2.7.1 Key management through symmetric-key techniques

One solution which employs symmetric-key techniques involves an entity in the network which is trusted by all other entities, *trusted third party* (TTP). Each entity A_i shares a distinct symmetric key k_i with the TTP. These keys are assumed to have been distributed over a secured channel (or other means). If two entities subsequently wish to communicate, the TTP generates a key k (sometimes called a *session key*) and sends it encrypted under each of the fixed keys as depicted in Figure 2.7 for entities *A1* and *A5*.

Advantages of this approach include: easy to add and remove entities from the network, each entity needs to store only one long-term secret key. Disadvantages include: all communications require initial interaction with the TTP; the TTP must store n long-term secret keys; the TTP has the ability to read all messages; if the TTP is compromised, all communications are insecure; TTP has to be on-line all the time.

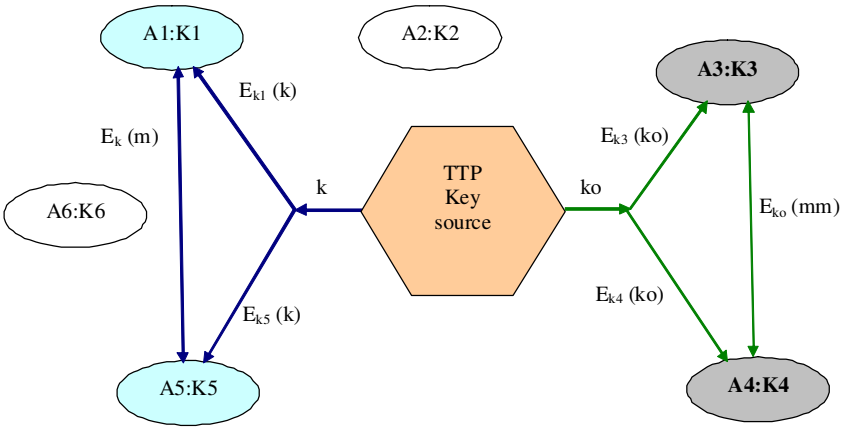

Figure 2.7 Key management using a trusted third party [17].

2.7.2 Public-key distribution

Key distribution is one of the most difficult issues in cryptographic system. One of the major roles of public-key is to address the problem of key distribution.

2.7.2.1 Using public file

Consider a simple model where each entity in the network has a public/private encryption key pair. The public key along with the identity of the entity is stored in a central repository called a *public file*. If an entity A_1 wishes to send encrypted messages to entity A_6, A_1 retrieves the public key e_6 of A_6 from the public file, encrypts the message using this key, and sends the ciphertext to A_6. Figure 2.8 depicts such a network.

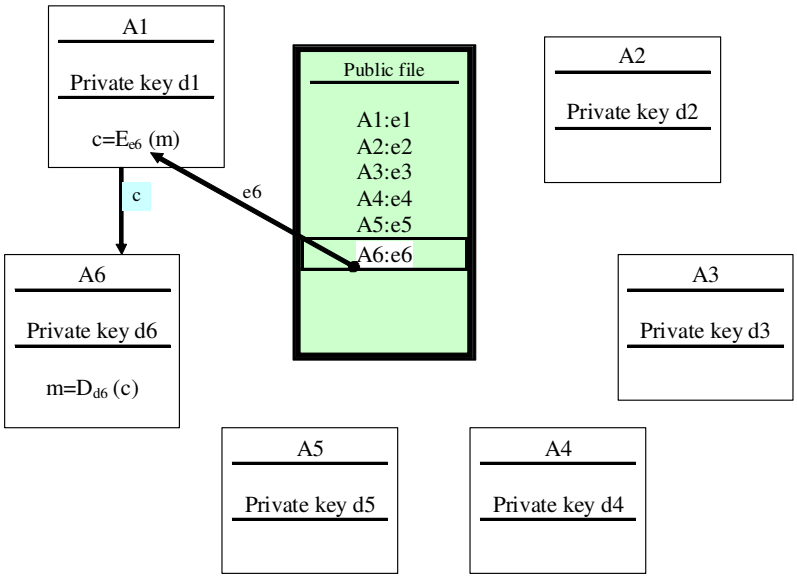

Figure 2.8: Public-key management using a public file [17].

Advantages of this approach include: no trusted third party is required; the public file could reside with each entity; only *n* public keys need to be stored to allow secure communications between any pair of entities. The key management problem becomes more difficult when one must take into account an adversary who is *active* and could compromise the key management scheme given above.

2.7.2.2 Digital certificates

The distribution of public keys is generally easier than that of symmetric keys, since secrecy is not required. However, the integrity (authenticity) of public keys is critical. A digital (public-key) certificate consists of a data part and a signature part. The data part consists of the name and ID of an entity, the public key corresponding to that entity, possibly additional relevant information (e.g., the entity's street or network address, a validity period for the public key, and various other attributes). The signature part consists of the signature of a CA over the data part. In order for an entity B to verify the authenticity of the public key of an entity A, B must have an authentic copy of the public signature verification function of the CA.

Advantages of using a CA include: prevention from an active adversary impersonation on the network. Second, the CA cannot monitor communications. Entities need to trust the CA only to bind identities to public keys properly. Third, per-communication interaction with the public file can be eliminated if entities store certificates locally. Even with a CA, some concerns still remain: i.e., if the signing key of the CA is compromised, all communications become insecure.

2.7.2.3 Diffie-Hellman (DH) key exchange

Diffie-Hellman is a commonly used protocol for key exchange. However, assume they do not initially possess any common secret and thus cannot use secret key cryptosystems. The key exchange by DH protocol remedies this situation by allowing the construction of a common secret key over an insecure communication channel. It is based on a problem related to discrete logarithms, namely the DH problem.

With symmetric encryption it is crucial to share a secret key. DH introduced that when two people communicating over an insecure line can agree on a secret key in such a way that both of them receive the same key without divulging it to someone who is listening in on their conversation. The goal of this process is for Alice and Bob to be able to agree upon a shared secret that an eavesdropper will not be able to determine. This shared secret is used by Alice and Bob to independently generate keys for symmetric encryption algorithms that will be used to encrypt the data stream between them. The "key" aspect is that neither the shared secret nor the encryption key *do not ever* travel over the network

Alice and Bob agree on two numbers "**p**" and "**g**"	"**p**" is a large prime number "**g**" is called the base or generator
Alice picks a secret number "**a**"	Alice's secret number = **a**
Bob picks a secret number "**b**"	Bob's secret number = **b**
Alice computes her public number **x** = $\mathbf{g}^{a}$ mod **p**	Alice's public number = **x**
Bob computes his public number **y** = $\mathbf{g}^{b}$ mod **p**	Bob's public number = **y**
Alice and Bob exchange their public numbers	Alice knows **p, g, a, x, y** Bob knows **p, g, b, x, y**
Alice computes **ka** = $\mathbf{y}^{a}$ mod **p**	**ka** = ($\mathbf{g}^{b}$ mod **p**)^{a} mod **p** **ka** = ($\mathbf{g}^{b}$)^{a} mod **p** **ka** = $\mathbf{g}^{ba}$ mod **p**
Bob computes **kb** = $\mathbf{x}^{b}$ mod **p**	**kb** = ($\mathbf{g}^{a}$ mod **p**)^{b} mod **p** **kb** = ($\mathbf{g}^{a}$)^{b} mod **p** **kb** = $\mathbf{g}^{ab}$ mod **p**
Fortunately for Alice and Bob, by the laws of algebra, Alice's "**ka**" is the same as Bob's "**kb**", or **ka = kb = k**	Alice and Bob both know the secret value "**k**"

Discrete logarithm algorithms can be used to attack DH, and with passive attacks. If DH is applied with usual arithmetic modulo a prime number, then it suffices to select a large

enough prime and to take some care in selecting the generator element. Many papers have been written on the problems that may occur; one of the well-known references is Carts on DH key agreement with short exponents [19]. DH key exchange does not cater for man in the middle attack. How does Alice and Bob exchange *p, g, x, and y?* If used with digital signatures then it works. Usually DH is not implemented on hardware, and thus hardware attacks are not an important threat. This may change in the future, when mobile communication becomes more widespread.

2.8 PSEUDORANDOM NUMBERS GENERATION

Random number generation is an important primitive in many cryptographic mechanisms. For example, keys for encryption transformations need to be generated in a manner which is unpredictable to an adversary. Generating a random key typically involves the selection of random numbers or bit sequences. Random number generation presents challenging issues. It is not clear what exactly it means to select at random or generate at random. Since most true sources of random sequences (if there is such a thing) come from physical means, they tend to be either costly or slow in their generation. To overcome these problems, methods have been devised to construct pseudorandom sequences in a deterministic manner from a shorter random sequence called a seed. The pseudorandom sequences appear to be generated by a truly random source to anyone not knowing the method of generation. Often the generation algorithm is known to all, but the seed is unknown except by the entity generating the sequence. Several algorithms have been developed to generate pseudorandom bit sequences of various types. In the optimal case, random numbers are based on true physical sources of randomness that cannot be predicted. Such sources may include the noise from a semiconductor device, the least significant bits of an audio input, or the intervals between device interrupts or user keystrokes [20]. The importance of the random number generator must thus be emphasized - if done badly; it will easily become the weakest point of the system.

2.9 CLASSES OF ATTACKS AND SECURITY MODELS

Over the years, many different types of attacks on cryptographic primitives and protocols have been identified. The attacks that adversaries can mount may be either passive or active. A passive attack is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data. An active attack is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel. An active attacker threatens data integrity and authentication as well as confidentiality.

2.9.1 Attacks on encryption schemes

The objective of the following attacks is to systematically recover plaintext from ciphertext, or even more drastically, to deduce the decryption key. A ciphertext-only attack is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by only observing ciphertext. A known-plaintext attack is one where the adversary has a quantity of plaintext and corresponding ciphertext. A chosen-plaintext attack is one where the adversary chooses plaintext and is then given corresponding ciphertext. Subsequently, the adversary uses any information deduced in order to recover plaintext corresponding to previously unseen ciphertext. A chosen-ciphertext attack is one where the adversary selects the ciphertext and is then given the corresponding plaintext.

Birthday attack is an attack that depends on the fact that duplicate values, also called collisions, appear much faster than one would think. Consider a system using 64-bit key, there are 2^{64} ($=18\text{x}10^{18}$) possible key values. It sounds large, not quite. After 2^{32} transactions, the attacker can expect that two transactions will use the same key, which technically reduces the key length to half.

***Man-in-the-middle attack*:** This attack is relevant for cryptographic communication and key exchange protocols. The idea is that when two parties, A and B, are exchanging keys for secure communication (e.g., using DH - Diffie-Hellman), an attacker positions itself between A and B on the communication line. The attacker then intercepts the signals that A and B send to each other, and performs a key exchange with A and B separately. The usual way to prevent the man-in-the-middle attack is to use a public key cryptosystem capable of providing digital signatures.

2.9.2 Attacks on protocols

The following is a partial list of attacks which might be mounted on various protocols. Until a protocol is proven to provide the service intended, the list of possible attacks can never be said to be complete. Known-key attack: an adversary obtains some keys used previously and then uses this information to determine new keys. Replay: an adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time. Impersonation: an adversary assumes the identity of one of the legitimate parties in a network. Dictionary: is usually an attack against passwords. Typically, a password is stored in a computer file as the image of a keyed hash function. Forward search: attack is similar in spirit to the dictionary attack and is used to decrypt messages. Interleaving attack: usually involves some form of impersonation in an authentication protocol.

2.9.3 Attack against or using the underlying hardware

In the last few years as more and smaller mobile crypto devices have come into widespread use, a new category of attacks has become relevant which aim directly at the hardware implementation of the cryptosystem. The attacks use the data from very fine measurements of the crypto device doing, say, encryption and compute key information from these measurements. The basic ideas are then closely related to those in other correlation attacks. For instance, the attacker guesses some key bits and attempts to verify the correctness of the guess by studying correlation against its measurements. Several attacks have been proposed such as using careful timings of the device, fine measurements of the power consumption, and radiation patterns. These measurements can be used to obtain the secret key or other kinds of information stored on the device. This attack is generally independent of the used cryptographical algorithms and can be applied to any device that is not explicitly protected against it [21].

Faults in cryptosystems can lead to cryptanalysis and even the discovery of the secret key. The interest in cryptographical devices leads to the discovery that some algorithms behaved very badly with the introduction of small faults in the internal computation. For example, the usual implementations of RSA private key operations are very susceptible to fault attacks. It has been shown that by causing one bit of error at a suitable point can reveal the factorization of the modulus (i.e. it reveals the private key). It is thus necessary that cryptographical devices be designed to be highly resistant against faults attacks [18].

2.10 STRENGTH OF CRYPTOGRAPHIC ALGORITHMS, AND KEY SPACE

Good cryptographic systems should always be designed so that they are as difficult to break as possible. It is possible to build systems that cannot be broken in practice (though this cannot usually be proved). The strength of encryption is related to the difficulty of discovering the key, which in turn depends on both the cipher used and the length of the key. For example, the difficulty of discovering the key for the RSA cipher most commonly used for public-key encryption depends on the difficulty of factoring large numbers, a well-known mathematical problem. Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Different ciphers may require different key lengths to achieve the same level of encryption strength. The RSA cipher used for public-key encryption, for example, can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric key encryption, can use all possible values for a key of a given length, rather than a subset of those values. Thus a 128-bit key for use with a symmetric-key encryption cipher would provide stronger encryption than a 128-bit key for use with the RSA public-key encryption cipher.

It is a great temptation to relate the security of the encryption scheme to the size of the key space. An encryption scheme can be broken by trying all possible keys to see which one the communicating parties are using. This is called an exhaustive search of the key space. It follows then that the number of keys (i.e., the size of the key space) should be large enough to make this approach computationally infeasible. Table 2.1 provides some figures of how this can be achieved.

Table 2.1 Average time required for exhaustive key search

Key size (bits)	No. of Alternative keys	Time required at 1 Decryption/μs	Time required at 10^6 Decryption/μs	Time required at 10^{12} Decryption/μs	Time required at 10^{18} Decryption/μs
32	2^{32} =4.3x 10^9	2^{31}μs=35.8minutes	2.15milli sec	2.15 nano sec	--
56	2^{56}=7.2x10^{16}	2^{55}μs =1142years	10 hours	3.6 milli sec	3.6 nano sec
128	2^{128}=3.4x10^{38}	2^{127}μs=5.4x10^{24}years	5.4x10^{18} years	5.4x10^{12}years	5.4x10^{6}years
168	2^{168}=3.7x10^{50}	2^{167}μs=5.9x10^{36}years	5.9x10^{30} years	5.9x0^{24}years	5.9x10^{18}years
256	2^{256}=1.2x10^{77}	2^{255}μs=1.8x10^{63}years	1.8x10^{57}years	1.8x10^{51}years	1.8x10^{45}years

A *32* bit key takes 2^{32} (about 10^9) steps. This is something anyone can do today on his/her home computer. The cost of the special hardware is substantial but easily within reach of organized criminals, major companies, and governments [22]. Keys with *64* bits are probably breakable now by major governments, and within reach of organized criminals, major companies, and lesser governments in few years. Keys with *128* bits will probably remain unbreakable by brute force for the foreseeable future.

The key lengths used in public-key cryptography are usually much longer than those used in symmetric ciphers. This is caused by the extra structure that is available to the cryptanalyst. Here the problem is not that of guessing the right key, but deriving the matching secret key from the public key. However, with RSA this could be done by factoring a large integer that has two large prime factors [23]. The strength of a cryptographic system is usually equal to its weakest link. No aspect of the system design should be overlooked, from the choice of algorithms to the key distribution and usage policies.

Private keys in public-key systems must be larger (e.g., 1024 bits for RSA) than secret keys in symmetric-key systems (e.g., 128 bits) because whereas the most efficient attack on symmetric key systems is an exhaustive key search, all known public-key systems are subject to “shortcut” attacks (e.g., factoring) more efficient than exhaustive search.

2.11 CHAPTER SUMMARY

In this chapter, a detailed outline of cryptography is presented. As mentioned several times throughout the chapter, cryptography alone does not provide a complete security solution. When building a secure system based on cryptography, it is necessary to make some important choices that include the choice of encryption algorithm used for data secrecy, protocols and algorithms used for channel protection, the length of keys used, how keys are generated, key storage, key distribution, and key management. Only when these choices are made properly and the system is implemented correctly, a secure and sound cryptographic security solution can be expected. The next chapter looks at the smart card structure and its use with cryptography and how it can improve some of the cryptographic concerns.

CHAPTER 3
SMART CARDS

3.1 INTRODUCTION

A smart card is a type of chip card, and very often easily mistaken for a simple plastic card. It is in fact embedded with a computer chip realised in a VLSI (Very Large Scale Integration), adhering to the ISO 7816 (International Standards Organization) standard [24], which stores and transacts data between users. A smart card contains a small microprocessor, RAM (Random Access Memory), ROM (Read Only Memory), and EEPROM (Electrically Erasable Programmable Read Only Memory) or flash memory. Today's smart cards can also have different cryptographic algorithms and protocols programmed into them and can be used to sign documents or unlock resources.

Smart cards greatly improve the convenience and security of any transaction, and provide tamper-proof storage of user and account identity as well as providing security for vital components of system security for the exchange of data throughout any type of network. They protect against a full range of security threats, from careless storage of user passwords to sophisticated system hacks. Some of the uses of smart cards are: securing information and physical assets; e-passport, e-commerce; personal finance; health care; telecommuting and corporate network security; campus access and many more.

If we look at smart cards as an entire computer system in a single integrated circuit chip, makes is quite convenient from the point that the interconnections between the components of the system are hidden and therefore are difficult to intercept the communication. This enhances the security of the system and is a very good functionality that is explored in great detail in this dissertation.

3.2 SMART CARD STRUCTURE

***CPU*:** Typically smart cards use an 8-bit processor and the instruction set is based on the Motorola 6805 or the Intel 8051 architecture. These processors have been proven in practice and fulfil the need of high reliability. However, the demand of processing power needed by the smart card operating systems is growing and the development of smart card tends to integrate 16-bit (Samsung CalmRISC16) or 32-bit (STMicroelectronics' ST22N256, and Infineons' 88 series) processor types.

***Memory System**:* Memory available on a smart card is limited and therefore it should be utilized and handled very carefully and sparingly. Smart card has three types of memory.

- ***ROM:*** is the memory area where the smart cards' operating system is stored. Common smart cards have between 32KB of ROM newer cards are having up to 384KB. Code and data are stored into ROM during the production process of the smart card and are 'hardwired' into the card.

- ***EEPROM:*** This is where user can store its data, 64KB – 128KB of EEPROM is common but some newer cards presents up to 256KB. EEPROM can store data that can be changed by an application, e.g. Cryptographic keys and certificates, PIN codes, account numbers, account balance of the wallet.

- ***RAM*:** Smart cards possess limited RAM. Therefore the RAM resources must be handled with much care. Card developer needs to be aware of the need to economize on the use of RAM. Most cards consist of 4KB – 16KB.

Input/Output System: Data transferred between the reader and smart card is byte oriented on a unidirectional serial channel at speeds up to 115 - 200bps. Although, there are several logical channels available on the smart card just one of these can be active at a time. The protocol used for communication is based on a master (card reader) and slave (smart card) relationship. The card reader sends commands to the smart card and waits for a response. Note that smart card never initiate data transfer.

3.3 TYPES OF CHIP CARDS

Smart cards are defined according to firstly, -how the card data is read and written? Secondly, -type of chip implanted within the card and its capabilities. Examples are: memory cards, microprocessor cards, and contact less cards. There is a wide range of options (Figure 3.1) to choose from when designing a system.

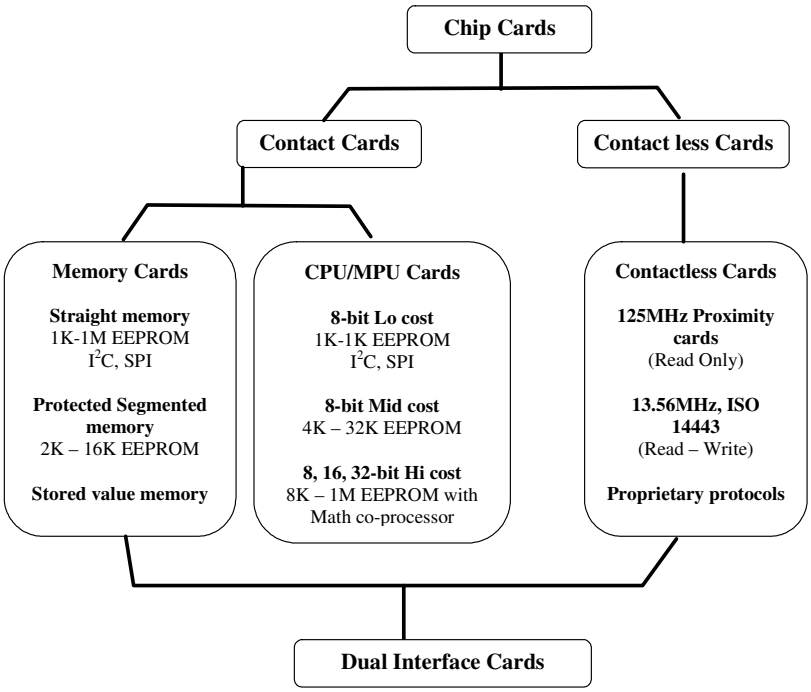

Figure 3.1 Types of chip cards [25].

3.3.1 Contact smart cards

Contact smart cards are the most common type of card available. Electrical contacts, Figure 3.2, are located on the outside of the card connect to a card reader when the card is inserted.

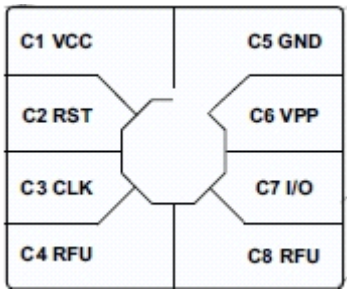

Figure 3.2 Electrical contacts of smart card [67].

3.3.1.1 Memory Cards

These were the first cards to be used in large numbers e.g. telephone cards, prepaid services, parking tokens etc. These cards have limited functionality, and can be used for

low cost applications. The data is usually stored on EEPROM, and access to memory is controlled by the security logic, which is usually write or erase. There is possibility of using advanced logic that uses encryption for safe keeping of data. Memory cards have no sophisticated processing power and cannot manage files dynamically. All memory cards communicate to readers through synchronous protocols. In all, memory cards are either read or written to a fixed address on the card. There are three primary types of memory cards: *Straight; Protected*; and *Stored Value*. Memory cards represent the bulk of the Smart cards sold primarily for pre-paid, disposable-card applications like pre-paid phone cards. These are popular as high-security alternatives to magnetic stripe cards.

3.3.1.2 Microprocessor/Multifunction Cards

These cards have on-card dynamic data processing capabilities. Multifunction smart cards allocate card memory into independent sections or files assigned to a specific function or application. Within the card is a microprocessor or microcontroller chip that manages this memory allocation and file access. This type of chip is similar to those found inside all personal computers and when implanted in a smart card, it manages data in organized file structures, via a card operating system (COS). Unlike other operating systems, this software controls access to the on-card user memory. This capability permits different and multiple functions and/or different applications to reside on the card, allowing businesses to issue and maintain a diversity of 'products' through the card. Multifunction cards benefit issuers by enabling them to market their products and services via state-of-the-art transaction and encryption technology. Specifically, the technology enables secure identification of users and permits information updates without replacement of the installed base of cards, simplifying program changes and reducing costs. For the card user, multifunction means greater convenience and security. There are many configurations of chips in this category including chips that support cryptographic PKI functions with on board math co-processors or Java virtual machine hardware blocks. As a rule of thumb - the more functions the higher the cost.

Figure 3.3 shows the microprocessor card. Apart from EEPROM and ROM, it has RAM, CPU, NPU, and an I/O (Input/Output) port. The mask ROM contains the chip's operating system. The I/O interface usually consists of only a single register, through which data is transferred serially, bit-by-bit.

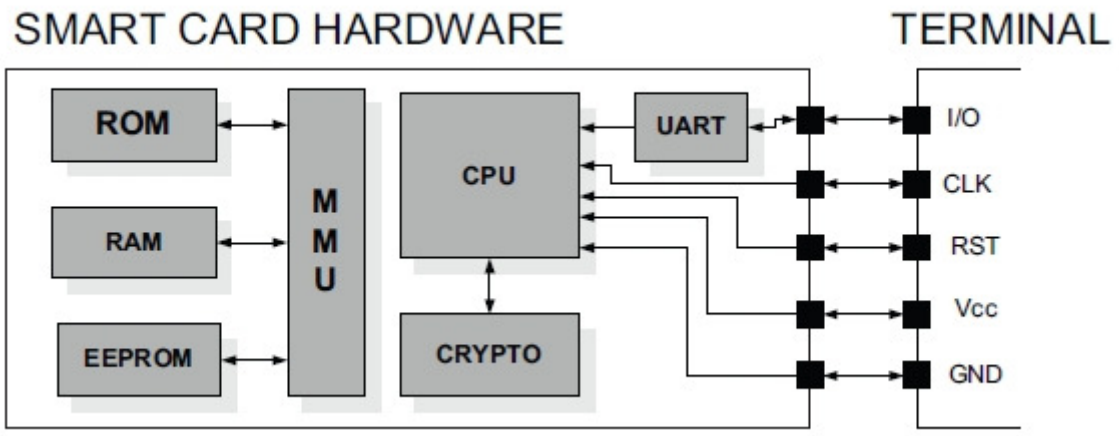

Figure 3.3 Typical Architecture of a contact-type microprocessor card [67].

Microprocessor cards are used for a variety of applications, especially those that have cryptography built in, which requires manipulation of large numbers. Very often the data processing power is used to encrypt/decrypt data, which makes this type of card very unique with cryptography. Thus, chip cards have been the main platform for cards that hold a secure digital identity. Hence they are capable of offering advanced security mechanism, local data processing, limited complex calculation and other interactive processes. Most stored-value cards integrated with identification, security and information purposes are processor cards.

3.3.2 Contactless smart cards

These types of smart cards employ a radio frequency between card and reader without physical insertion of the card. Types include proximity cards which are implemented as a read-only technology for building access. These cards function with a limited memory and communicate at 125 MHz. True read & write contactless cards were first used in transportation for quick decrementing and re-loading of fare values, where their lower security was not an issue. They communicate at 13.56 MHz, and conform to the ISO14443 standard. These cards are often straight memory types. These cards transfer data and energy without any electrical contact between the card and the terminal. Contactless card drawbacks include the limits of cryptographic functions and user memory versus microprocessor cards and the limited distance between card and reader required for operation. Both memory and processor based cards are available in contactless mode. Their working distance is usually a few centimetres away from the terminal.

3.4 SMART CARD STANDARDS

Primarily, smart card standards govern physical properties, communication characteristics, and application identifiers of the embedded chip and data. Almost all standards refer to the ISO 7816-1, 2 & 3 as a base reference. Application-specific properties are being debated with many large organizations and groups proposing their standards. Open system card interoperability should apply at several levels:

To the card itself,

The card's access terminals (readers),

The networks, and

The card issuers' own systems.

Open system card interoperability will only be achieved by conformance to international standards. The following standards and the organizations that maintain them are the most prevalent in the smart card industry:

3.4.1 ISO - International Standards Organization

This organization facilitates the creation of voluntary standards through a process that is open to all parties. ISO 7816 is the international standard for integrated-circuit cards (commonly known as smart cards) that use electrical contacts on the card, as well as cards that communicate with readers and terminals without contacts, as with radio frequency (RF/Contactless) technology.

- ***ISO 7816-1:*** Physical Characteristics, 1987; defines the physical dimensions of contact smart cards and their resistance to static electricity, electromagnetic radiation and mechanical stress. It also describes the physical location of an IC card's magnetic stripe and embossing area.
- ***ISO 7816-2*****:** Dimensions and Location of Contacts, 1988; defines the location, purpose and electrical characteristics of the card's metallic contacts.
- ***ISO 7816-3:*** Electronic Signals and Transmission Protocols, 1989; defines the voltage and current requirements for the electrical contacts.
    - asynchronous half-duplex character transmission protocol identified by the value T=0
    - asynchronous block-oriented half-duplex transmission protocol identified by the value T-1

- A block-oriented full-duplex protocol identified by the value T=2, although this node is rarely used.
- The value T=14 indicates the use of proprietary protocols, used to support applications that preceded the standard and were already planned in France and in Germany in the health field.
- The values T=3 to T=13 are reserved for future use.

- ***ISO 7816-4*:** Inter-industry Commands for Interchange; establishes a set of commands for CPU cards across all industries to provide access, security and transmission of card data. Within this basic kernel, for example, are commands to read, write and update records. Defines the local organisation of the data stored in the card and the framework for secure access to these data, in particular:
  - Cardholder authentication using password (the PIN).
  - Authentication of an external entity using a secret key that authenticates the terminal or the bank.
  - Verification of the data integrity using cryptosystem that is often a message authentication code.
  - Encryption of data
  - Commands fall in three categories: administrative, security, and communication management commands.
- ***ISO 7816-5:*** Numbering System and Registration Procedure for Application Identifiers (AID); sets standards for Application Identifiers. An AID has two parts. The first is a Registered Application Provider Identifier (RID) of five bytes that is unique to the vendor. The second part is a variable length field of up to 11 bytes that RIDs can use to identify specific applications.
- ***ISO 7816-6:*** Inter-industry data elements; physical transportation of device and transaction data, answer to reset and transmission protocols. The specifications permit two transmission protocols: character protocol (T=0) or block protocol (T=1). A card may support either but not both. (Note: Some card manufacturers adhere to neither of these protocols. The transmission protocols for such cards are described as T=14).
- ***ISO 7816-8:*** Commands for Security Operation; this document codifies internal card commands for security operations.
- ***ISO 7816-11*:** Personal verification through biometric methods; currently a draft.

3.4.2 FIPS (Federal Information Processing Standards)

Developed by the Computer Security Division within National Institute of Standards and Technology (NIST). The following FIPS standards apply to smart card technology and pertain to digital signature standards, advanced encryption standards, and security requirements for cryptographic modules.

- ***FIPS 140 (1-3):*** The security requirements contained in FIPS 140 (1-3) pertain to areas related to the secure design and implementation of a cryptographic module, specifically: cryptographic module specification; cryptographic module ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; design assurance; and mitigation of other attacks.

3.5 SC OPERATING SYSTEMS

Selection of a card OS depends on the application the card is developed for. The other defining difference is in the Encryption capabilities of the OS and the Chip. The two primary types of smart card operating systems are: Fixed File Structure, and Dynamic Application System.

Fixed File Structure type treats the card as a secure computing and storage device. Files and permissions are set in advance by the issuer. These specific parameters are ideal and economical for a fixed type of card structure and functions that will not change in the near future. An example of this kind of card is a low-cost employee multi-function badge.

Dynamic Application System type of operating system, which includes the MULTOS and JAVA card varieties, enables developers to build, test, and deploy different applications securely. Because the OS and applications are more separate, updates can easily and repeatedly be made. An example card is a SIM card for mobile GSM where updates and security are downloaded to the phone and dynamically changed.

In an ***open application*** or ***open platform environment***, smart card operating system allows third parties to load data and applications onto the smart card without involving the producer of the smart card operating system. The benefit of an open application environment is that it speeds up the development process of smart card applications. And this leads to a reduction of development costs. Furthermore, it increases the flexibility of the smart card application development process. Several open platforms for smart card are

available [27]. All of these open platforms support application design for multi application smart cards. Furthermore, these applications can be selected at the same time.

Java Card: is a smart card operating system which enables smart cards to run programs, called applets, written in Java. Strictly speaking java card is not a true operating system because of the fact that the java card specification does not include a file management. But in practice java card is considered as the prototype of an open smart card operating system [28].

MULTOS: is a multi application smart card operating system. Its origin goes back to the development of the Mondex system for electronic purses [27]. The goal was to develop an operating system which meets the requirements of electronic payment systems. Typically the code for an application is written in C and later transformed into the MEL (Multos Executable Language) using a special compiler. In order to enable a download of an application onto a smart card this application needs to be digitally signed by a licensed MULTOS certification service.

Basic Card: is a third multi application smart card operating system which allows executable program code to be downloaded by third parties. Basic card [29] is based on the programming language Basic and several versions with different features for different smart cards with various memory sizes are available. The program generation procedure is based on the traditional Basic interpreters. First the source code is translated by a compiler into P-code and in a second step this code is loaded into the memory of the smart card microcontroller. The interpreter included in the basic card operating system is furthermore responsible for the execution of the downloaded code.

Linux: Until recently smart card microcontrollers have not been powerful enough to reach the requirements of a Linux operating system. Usually linux requirements can only be provided by a 32-bit processor, several kilobytes of ROM and even more kilobytes of RAM. However, it is imaginable that newer Linux versions reduce their hardware requirements and at the same time smart cards microprocessors increase their performance with every new generation. Due to this it would be possible for Linux to be available for smart cards in the near future.

3.6 WHY SMART CARDS?

3.6.1 SC applications

Smart cards are being deployed in most sectors of the public and private marketplaces. Some popular application areas where smart cards are being used in today's world: Loyalty; Financial; Information Technology; Government; Healthcare; Telephony; Mass Transit; Identification on Internet. Some of the major applications of the smart cards, as seen around the world, are:

- Most popular and commonly used is GSM mobile telephones with smart cards, the SIM, which contain the mobile phone security and subscription information. The handset is personalized to the individual by inserting the card, which contains its phone number on the network, billing information, and frequently call numbers.
- health care card, pay phone card, dish TV satellite card, credit/debit bankcard, retail loyalty schemes and corporate staff, physical access, resort cards, mass transit, mass transit ticketing schemes, electronic toll, product tracking, national ID, drivers license,
- Other applications for smart cards include computer/internet user authentication, data integrity, confidentiality, and non-repudiation, e-passports etc.

3.6.2 Internet

The role of the Internet has developed to include the support of electronic commerce. It was designed for the free exchange of information, and as such, it is a rich supply of academic product and service information. But how does an Internet shopper go from looking at the product to actually buying it? The smart card is the ideal support for payment over the Internet, whether in cash or as credit. However, the Internet shopper needs to connect his smart payment card to his computer and through the computer to the Internet. Smart card readers are inexpensive, low-power devices which can be easily added to existing computers. The Internet is focusing the need for online identification and authentication between parties who cannot otherwise know or trust each other, and smart cards are believed to be the most efficient way of enabling the new world of e-trade. Smart cards can act as an identification card, which is used to prove the identity of the cardholder. This concept is explored in great length in this dissertation.

3.6.3 Future

The important thing about smart cards is that they are everyday objects that people can carry in their pockets, yet they have the capacity to retain and protect critical information

stored in electronic form. The “smartness” of smart cards comes from the integrated circuit embedded in the plastic card. The use of Biometrics will soon mean that his/her hand, fingerprint and the retina of the eye or the sound/ voice can reliably identify a person. Soon it will be possible to authorize the use of electronic information in smart cards by using a spoken word or the touch of a hand [30].

Sooner smart card readers will be appearing on the PC and will enable the user to pay for goods purchased over the Internet. This will be especially useful for small value purchases, which are not really appropriate for credit card transactions. The smart card will be "charged up" with money and it will then use it as one uses cash or a phone card. In the near future, the traditional magnetic strip card will be replaced and integrated together into a single card by using the multi-application smart card, which is known as an electronic purse or wallet in the smart card industry. It will be used to carry a lot of sensitive and critical data about the consumers ever more than before when compared with the magnetic strip card [30].

3.7 SC ADVANTAGES

Some advantages of the smart card:

- Proven to be more reliable than the magnetic stripe card.
- Can store up to thousands of time more information than the magnetic stripe card.
- Reduces tampering and counterfeiting through high security mechanisms such as advanced encryption and biometrics.
- Can be disposable or reusable.
- Performs multiple functions.
- Has wide range of applications (e.g., telecommunication, Networking, Cryptography, banking, transportation, healthcare...).
- Compatible with portable electronics (e.g., PCs, telephones...).
- Evolves rapidly applying semi-conductor technology.
- Smart cards can hold a large amount of personal information, from medical/health history to personal banking and personal preferences, cryptographic keys and digital certificates.
- They can carry all the necessary functions and information on the card. Therefore, they do not require access to remote databases at the time of the transaction unlike magnetic stripe cards

Gartner Dataquest, the market analysis firm, predicts the global smart card market will reach 10 billion units by 2010. This clearly indicates the future potential and capabilities one can expect from smart card as the manufactures are pushing the card with more storage, more processing, and more hardware coded cryptographic algorithms for security use and thus is very strong motivation behind this dissertation.

It is not just the sheer number of cards that are sold is sufficient to conclude the use of smart cards in data or network security but rather has some sound technical facts. Such as the self-containment of smart card makes it resistant to attack, as it does not need to depend upon potentially vulnerable external resources. Because of the security and data storage features, smart cards are rapidly being embraced as the consumer token of choice in many areas of the public sector and commercial worlds and are often used in different applications, which require strong security protection and authentication. Many of the applications of smart cards require sensitive data to be stored in the card, such as biometrics information of the card owner, personal medical history, and cryptographic keys for authentication, digital certificates, etc. Smart cards provide processing computing to business systems and present enormous benefit of portability and secure storage of data and value. At the same time, the integration of smart cards into a system introduces its own security management issues, as people access card data far and wide in a variety of applications.

Another strong point of smart cards is that they are tamper-resistant microprocessor chips. They have the ability to run application to make computations on data using programs stored in memory. Smart card provides all the five elements (the processor, RAM, ROM, EEPROM, communication devices) into single chip which makes it very attractive in running the cryptographic algorithms and communication protocols. ISO standards specify the ability of the card to withstand a given set of mechanical stresses. Chips for cards are very reliable and provide a high level of security comprising encryption using several algorithms, and distributed system of secret keys storage and generation, as well as providing authentication by executing the required algorithms within the card itself.

3.8 CRYPTOGRAPHY WITH SMART CARD

Smart cards can be very successfully used in building data security systems using cryptographic mechanism. Smart cards support: confidentiality, data integrity, non-repudiation and authentication as well as key generation and key distribution. Currently a

lot of work is being done on smart card to provide the authentication roll of cryptography as presented next.

3.8.1 Smart cards for authentication

For smart cards to be used as an authentication mechanism, some secret information is stored that could uniquely identify the user. This could either be a digital certificate, username password pair or a private-key public-key pair. A low-cost smart card would have difficulty in computing a 512-bit digital signature on the card and sending the result to the host application. But the newer cards are significantly capable of running these algorithms with some trade off.

Static authentication can be used with symmetric cryptography where the card issuer generates a digital signature on the smart card. After the card has been authenticated to the terminal through a PIN (Personal Identification Number), it sends the signature to the terminal. This requires the terminal to encrypt and decrypt data and the card only to confer the signature as a password. This means that the card does not have to do processing but it also involves relinquishing security because the password then leaves the smart card and is sent to an unsecured terminal. For the implementation to be secure, public-key cryptography will need to be implemented on the card itself, providing a real-time dynamic digital signature to the terminal or host. According to Leach [31], a smart card must implement public-key cryptography to authenticate a user with a smart card. This is not totally accurate as will be explained in the next paragraph.

The other solution, as explained in Verschuren [32], uses a combination of public-key and symmetric authentication mechanism for user authentication. This card was developed through the DESIRE project [33] with the help of IBM. A user with a smart card, a smart card reader, a PC and a web browser can authenticate him/herself to a web server. The advantage of this implementation over the public-key implementation is that all secret information is already on the smart card and no separate registration process is needed. The application uses a two-party authentication mechanism using symmetric cryptography, where the client talks directly to the server for authentication.

Urien [34] introduces the realisation of a secure and open architecture where the smart card is not tied to any specific application. This new approach will implement the smart card as an Internet card. Here the smart card is regarded as a network computer, which is able to share the resources of the host terminal to which it is connected. The smart card will not just act as a peripheral component connected to the computer, but will configure and

manage the terminal to which it is connected. In this approach the smart card is the intelligent device and the terminal is dumb. Its only function is to provide a pathway for the communication between the smart card and a server or other application.

Smart cards are mainly used as static or dynamic authentication devices [35]. To proceed, the user must first be verified to his/her own smart card by entering a PIN code. When using static authentication, the card for instance holds a signature already calculated and stored thereon. This is then sent to the terminal, and only the terminal performs cryptography. With dynamic authentication, a smart card is challenged with something from (in this case) the terminal. The smart card then provides a real-time dynamic digital signature [36] to the terminal or host. This is normally done with public-key cryptography.

RSA is used in smart cards and is very secure, but is rather used for the encryption of hashes (resulting in digital signatures), than for the encryption and decryption of data. The reason is quite simple – it takes too long, requires complex key generation and takes up too much memory space. Symmetrical encryption is, therefore, recommended for encryption other than for authentication.

According to Markantonakis [37], Java Cards are justly becoming very popular because of their interoperability. To date there exists a framework (the OCF - Open Card Framework [38]) that can link various supported Java Cards to applications - (independent of card vendors), their multi-application capability (where multiple applications can exists on one card), dynamic nature (allowing installation of new applications after issuing), and ease of development of applications. They also permit identification/authentication of users, secure storage of keys and data, cryptographic algorithms and applications like digital signatures, loading and cancelling of units of value for electronic payments and authentication of smart cards to systems Sun Microsystems, Inc. [28].

3.8.2 SC for key distribution

Today, cryptography is key and integral component of data and network security. But cryptography on its own does not provide full and complete solution. One of the serious issues with cryptography is the use of keys and specifically the key distribution or key exchange, safe storage, and good key generation. Smart cards provide a significant solution to this problem, although, it in itself is not full proof from failure. Section 3.11 presents several security attack issues that need to be considered before concluding smart cards are secure for key distribution and storage.

3.9 CHOOSING THE WRITE SMART CARD

Increased levels of processing power, flexibility and memory increases the card cost. Choosing the right type of smart card for an application can be based on by evaluating the cost versus functionality and then determining the required level of security. All of these variables should be weighted against the expected lifecycle of the card. On average the cards comprise very low (only 5 to 10 percent) of the total system cost with the infrastructure, and training making up the remaining percent. The following chart in Figure 3.4 demonstrates some general rules of thumb when considering a card.

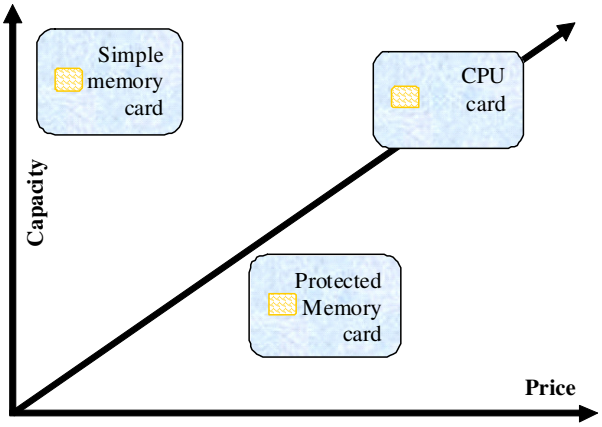

Figure 3.4 Card Function Trade-Offs

3.10 SECURITY CONCERNS WITH SMART CARD

Opting for smart cards to provide a complete secure solution is not possible and viable. There are several security concerns that needs to be scrutinised very carefully and some of these are presented next.

The processor of smart card must be as tamper resistant as possible. That is the cost of breaching the chip security mechanism must be higher than the potential gain from doing so. It should be impossible to read the secret data stored on the card, such as cryptographic keys. Or monitor processes running on the card and thus draw significant conclusion about sensitive information. Different attacks can be performed when the chip is active or inactive, hence, tamper resistance may not solve all the security issues. Security measures during the chip manufacturing are also essential. Although, each chip obtains a unique serial number, which in itself can not protect against attacks, but serves as information for deriving cryptographic keys, if the serial number of the chip is used as a seed in generating

the secret keys. These secret keys which are the core of cryptography may jeopardise the security of the model.

Most of the attacks on smart card chip are performed during the card use and these are either static or dynamic analysis of the card. Some of the principles used to minimise the static analysis include: -Opaque tamper-evidence coating to hamper direct observation, probing, or manipulation of the chip surface; -Dummy structure can be included to confuse the attacker; -Special memory design with scrambling to hide contents; -Hiding and scrambling of buses to prevent eavesdropping. Dynamic analysis can be protected using mechanism such as: -voltage watchdog that switch on/off a chip module if source voltage is unacceptable; -mechanism that set to zero any parameters representing the chip secret or private information like cryptographic keys.

3.11 ATTACKS ON SMART CARDS

The essential property of smart card is its ability to offer a secure environment for data and programmes. Protecting and securing data is one of the main advantages of smart cards in comparison to other data carriers. Consequently, the chip hardware must be designed in an optimum fashion to meet this purpose. These include the corresponding cryptographic procedures for securing the secret data. It is with this thinking attacks on smart card can be grouped into three stages: one – during the development phase, two – during the manufacturing process, and three – during the use of smart card. Wolfgang et al [27] presents detailed attack analysis of card developmental and manufacturing phase. The access to the cards' component to be attacked usually requires far less effort from attackers once the smart card has been issued and thus increases the probability of attack.

Table 3.1 lists and describes examples of attacks that can almost be called classical attacks. They provide a competent overview so that already known critical mechanisms are not be used again due to a lack of knowledge. The countermeasures specified are available for the defence of such attacks. The scenarios portrayed do not serve as manual for cracking the security of a smart card system since all of them are known and publicly available [39]. They do not constitute serious threats for the security of today's modern smart cards because the attacks specified have already been considered by the corresponding protective measures. A few years ago, however, those attacks might still have been successful.

Table 3.1 Typical attacks that had an influence on systems equipped with smart cards. [27]

Attack	Short Description of the Attack
Bugging of data transmission	The data transmission between the terminal and the card can be bugged by attaching wires to the module. The countermeasure was the introduction of Secure Messaging.
Manipulation of data transmission	The data transmission between the terminal and the card can be manipulated at will by electrically insulating the contact fields of the module and the wires attached to the module. The countermeasure was the introduction of Secure Messaging.
Setting up memory card equivalents	Both the functionality of the memory card and the secret authenticity feature can be emulated by setting up equivalents of memory cards. The countermeasure was the introduction of challenge response authentication on the memory cards.
Disconnecting the voltage supply	The retry counter of the PIN can be avoided by disconnecting the voltage supply during the PIN verification process. The countermeasure was the prophylactic increase of the retry counter prior to the PIN verification process.
Stop clock frequency	Conclusions can be drawn regarding the RAM content by halting the clock frequency and analysing the RAM with the help of electron beam testers. The countermeasure was the introduction of low frequency detectors on the microcontrollers.
Manipulation of the microcontroller by means of laser cutters	The components on the microcontroller can be manipulated with the help of laser cutters. The countermeasure was the introduction of protective layers around the microcontrollers.
Timing attack	Due to a lack of knowledge, a dependency between the key and the validity was created during the implementation of many crypto algorithms. This can be used for determining secret keys. The countermeasure was the realisation of noise free crypto algorithms.
Bugging the bus with microprobe needles	The buses on the microcontroller can be bugged with microprobe needles. The countermeasure was scrambling the buses on the microcontrollers.
DFA (Differential Fault Analysis)	Secret keys of crypto algorithms can be calculated by selectively introducing miscalculations to the processor. The countermeasure was the introduction of glitch detectors on the microcontrollers as well as the corresponding precautionary measures in the crypto algorithms.
Exhaustive key search in DES	High performance computers and computer networks can calculate DES keys within a few hours by means of a brute force attack. The countermeasure was the use of triple DES.
SPA/DPA	The data processed can be determined by the power consumption of the processor. The countermeasures were: the introduction of random waiting periods to the processor, the use of processors with steady power consumption as well as a number of precautionary measures within the software of the microcontroller.
Processor disruption	The processor can be disrupted at critical stages while processing the machine code by attacking the processor (e.g. by means of light flashes). The countermeasures were the corresponding detectors on the microcontrollers as well as a large number of precautionary measures in the software.

The next two sections present several attack scenarios during the smart card use phase. Mostly this attack scrutiny can be grouped into two level, the physical level and logical level. The next section presents various possible attacks and their possible counter measures.

3.12 PHYSICAL LEVEL ATTACKS ON SMART CARD

In order to perform manipulations in the area of semiconductors require a large amount of technical effort. Depending on the attack scenario, various tools such as microscopes, laser cutters, micromanipulators, focused ion beams, equipment for chemical removal procedures and fast computers for the analysis, the recording and evaluation of electrical procedures on the chip can be used. Very few selected specialists or organisations have these equipments and the corresponding knowledge required for their application. It in turn drastically reduces the probability of an attack on a physical level. Nevertheless, a manufacturer of cards and/or semiconductors must always assume that a potential attacker can use all the equipment required for such an attack; therefore, the corresponding security features must be included in the hardware. The following sub sections present the number of attack analysis on smart card. Static attack analysis on the smart cards microcontroller is presented followed by the dynamic attack analysis.

3.12.1 Static attack analysis

3.12.1.1 Issues relating to semiconductor technology

The chip's structures (width of conductor path, size of transistors, etc.) have reached the limits of what is technically possible today. The typical width of the structure ranges between 0.35 µm and 0.13 µm, which itself is no longer a special technological feature. The transistor density on the silicon, however, has reached the top limit of what can currently be achieved by using the typical lithographical manufacturing procedures. Only these very fine structures make it difficult to obtain information from the chip with the help of analytical procedures. Therefore, semiconductor technologies with structural sizes of one micrometer and below can be currently considered to be secure. In future, this will certainly change.

3.12.1.2 Chip design issues

The semiconductor chip design is based on so called standard cells, which, for instance, contain a processor core or certain types of memory, are often used for the design of semiconductor related components. The advantage lies in the fact that a manufacturer of semiconductors can use these standard elements to quickly produce a variety of different high quality chips. This procedure was developed for the mass production of products not taking into account security aspects. Hence, it should not be used for the production of smart card microcontrollers due to the following reasons: the structure and the mode of functioning of standard cells is publicly known, which would provide potential attackers with too much information and make their work much easier to obtain the stored information [27].

3.12.1.3 Chip bus protection issues

All with The three different types of memory, RAM, ROM, and EEPROM, are all linked using the internal buses on the chip that connects then to processor do not lead to the outside and therefore cannot be contacted. There is no possibility for an attacker to bug or influence the address, data, or control bus of the microcontroller in order to gain knowledge of the memory contents. The reason for this is that usually, the buses are integrated into the lower layers of the semiconductor, which makes it difficult to contact them directly from the surface. In addition, the buses on the chip are scrambled either statically or individually per chip or individually per session, so that the function of the individual bus circuits cannot be determined from external sources. It would be nice to have feature of constantly changing the scrambling process of the buses even during a running session in the future smart card microcontrollers. To actually achieve this feature will be significant development.

Lan Gao. et al [40] presents that the address sequence on the processor-memory bus can reveal abundant information about the control flow of a program. This can lead to critical information leakage such as encryption keys or proprietary algorithms. Addresses can be observed by attaching a hardware device on the bus that passively monitors the bus transaction. Such side-channel attacks should be given rising attention especially in a distributed computing environment, where remote servers running sensitive programs are not within the physical control of the client. Lan Gao. et al proposed a lightweight on-chip address permutation that effectively addresses most of the problems and achieves the lowest memory demands and page faults occurrence. This is obtained by permuting only

on-chip cached blocks, and launches a permutation for only those addresses that have not been remapped. This scheme incurs only a 0.88x increase in memory accesses and a close-to-base page fault rate, without compromising the security strength.

3.12.1.4 Memory design issues

Smart cards are widely known for their tamper resistance, but only contain a small amount of memory. Though very small, this memory often contains highly valuable information (identification data, cryptographic key, etc). This is why it is often subject to many attacks, as the other parts of the smart card, and thus requires appropriately chosen protections. The use of memories in smart cards induces security problems, but also other more particular ones. The main constraint is naturally the limited physical expansion and integration, but fault level, aging and power consumption are not to be discarded. Indeed, Ross et al [41] has proved that it is possible to read the content of a ROM by simply using a microscope bit by bit. Consequently, it is not very difficult to combine the bits to form bytes and to combine the bytes to form the complete ROM code. This is of course completely unacceptable because ROM contains sensitive information. The operating system and some particular applications, for example, are written in ROM. In order to avoid this analysis, the ROM should not be integrated into the uppermost layers that can easily be accessed but into the lower silicon layers. Doing this should prevent an optical analysis that can be otherwise carried out. If, however, the chip's front side is pasted on a carrier, and the chip is removed from the plastic from the backside, the contents of the ROM could be obtained. In order to avoid this, the smart card microcontrollers are exclusively equipped with ion implanted ROMs, whose data contents are not visible, neither in visual nor in IR (Infra Red) or UV (Ultra Violet) spectrum. This also protects the chip to a large extent from so called selective etching. This procedure is used to etch the semiconductor in such a way that the contents of the ROM become optically invisible.

Memory manufacturers have tried to solve these problems by increasing the complexity of the memory by adding more transistors with increase in costs, and consequently do not offer a long term solution. Efforts have also been made to reduce side channel leakage through the coupling of transistors or the use of dual rail logic. New technologies to counter these problems are appearing, as for instance FeRAM, but these do not solve the particular problem of intrinsic security. Neve et al [42] presents some of the countermeasures that can effectively be considered.

3.12.1.5 Protection using protective layers

One of the dangers lies in the analysis of electrical potentials on the chip is during its operation. Provided that the scanning frequency is high enough, this makes it possible to measure charge potentials from the chip, i.e. voltages, on very small areas of the crystal, and thus conclusions can be drawn on the data contents of the RAM during operation. This can very reliably be avoided by adding conducting metallic layers to the corresponding memory area or to the whole chip. If these metallic layers are removed chemically, the chip will no longer be functioning, as these layers are required for supplying electric voltage needed for the correct functioning of the chip. In most cases, many protective layers are arranged on top of each other, and they are permanently checked on intactness. It is also possible with semiconductor technology that allows the implementation of meander shaped current carrying structures on the complete surface of the chip or on areas that are at high risk using low frequency detectors. They can easily be monitored via measuring the resistance or they can be implemented into the function of the chip so that the chip will be switched off immediately if these structures are damaged or subjected to alterations. The security can be increased further, if the linkage of the meander shaped structures is possibly modified during the session.

3.12.1.6 Memory protection using scrambling

This is a technique that is similar to the scrambling of buses that has been implemented a long time ago. Today most chip manufactures increasingly use scrambling of memories on microcontroller chips. The security is based on keeping the scrambling pattern of the memory cells secret. Memory can be scrambled without too much complexity and this requires little additional space on the chip. An attacker without the corresponding memory scrambling information should find extremely difficult to determine the way the memory cells are addressed.

3.12.1.7 Memory protection using Encryption

Other than scrambling or swapping of data in the memory, modern smart card microcontrollers also offer the possibility of encrypting the memory and even a part of the registry of the processor on a batch or chip individual level. During this process, the corresponding data is encrypted and decrypted in real time during reading and writing. In addition to the key, some of today's chip types offer the option to include the memory address in the en/decryption process. This results in equal data having different values at different positions in the memory after the encryption process. Individual keys for each

session can especially be used for RAM areas. If an attack was successful and data could be read from the memory, the secret key would still be necessary to gather information that makes sense. This significantly increases the effort required from the attacker since he either must know at which position this key is stored or he must read all the data available on the chip, which could be very challenging.

3.12.2 Dynamic attack analysis

Dynamic attacks are very difficult to prevent and can determine what command on the card is being executed and are based on card power analysis. Power analysis attack works on the principle that different power is consumed by different command. To safeguard against is to use the command that use very similar power. Another possibility is to perform the same computation, like in cryptographic algorithm, so that each time a command is chosen randomly. An important dynamic attack is timing attack, in which time intervals needed by the card for specific computations are measured and analysed. Consider if the card encrypts data, the greater the differences in the duration of computation for different keys and data, the easier it is to reduce the set of possible keys. The safeguard against such attack would be to make the duration of cryptographic computations independent from input data. A detailed finding of timing attacks on DH, RSA, DSS and other systems is presented by Kocher et al [43], where he explains that by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements.

3.12.2.1 Protection using passivation layer

After the chip has been produced on the silicon, a passivation layer is added, which prevents oxidation, for example by atmospheric oxygen, as well as other chemical processes from taking place on the surface of the chip [39]. First, this passivation layer must always be removed in order to manipulate the chip. A sensor circuit can determine via resistance or capacitance measuring whether this passivation layer is still present. If it is no longer present or if it is damaged, either an interruption in the chip's software can be triggered or the complete chip can be disconnected from the hardware, which reliably prevents all dynamic analyses.

3.12.2.2 Voltage control analysis protection

Each smart card microcontroller is equipped with a voltage control system. This system is responsible for switching off the component in a controlled manner when the upper or lower limits of the operating voltage are exceeded. This secures the software in such a way that an operation in the limit ranges at which the chip is no longer fully functional is impossible. If no voltage control are in place, an operation in these limit ranges could e.g. lead to the programme counter of the processor not to run stable anymore, which for instance leads to uncontrolled leaps within the programme or which causes miscalculations in the processor. This abnormal behaviour can be used as a starting point for determining secret keys by means of the Differential Fault Analysis (DFA). Especially the voltage control is of high significance for the security of the microcontroller. One possible attack would be to destroy the corresponding detectors useless, e.g. by means of a FIB (Focused Ion Beam), and to start the actual attack afterwards. This is the reason why in many cases the components required for the security of the microcontroller are protected especially well, so that a manipulation will be recognised and the smart card will be deactivated automatically.

3.12.2.3 Protection issues with Scrambling of the Buses

Many smart card microcontrollers scramble the buses that are addressing the memory, which are only accessible internally on the chip. This means that the individual bus circuits are not ordered next to each other or even isolated by layers on top of each other. This is an additional obstacle for potential attackers, as they do not know which bus circuit has which function or address. Originally this scrambling of the bus circuits was only introduced in a static variant, i.e. with identical scrambling on each chip. Thus, in the medium term it would not be a real problem for an attacker to determine how the circuits are scrambled and to consider this accordingly in a bugging campaign.

There is, however, an improvement of the security by introducing scrambling processes of the buses that are individual and unique to each chip. This chip individual scrambling is not achieved by producing different exposure masks for the buses of each chip as this cannot be realised technically at the moment and it would be far too expensive. The scrambling is carried out by scramblers, which are directly located on the memory and can be controlled by the individual chip numbers. This procedure can be enhanced rather effortless with semiconductors and it makes a bugging campaign much more difficult. A

scrambling process that is individual for each chip and each session can be realised as well by using variable input values in the scramblers.

3.12.2.4 Power analysis of the CPU

Paul Kocher, Joshua Jaffe and Benjamin presented Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [44]. The principle of the simple power analysis (SPA) is rather simple. An analogue digital converter is used to measure the power consumption of a microcontroller by determining the drop in voltage at a resistor connected in series at a high temporal resolution. Due to the relatively simple structure of the CPUs of smart card microcontrollers the internal processes and the processed data lead to measurable and interpretable effects on the power consumption. To make it clearer, one can imagine that the same programme sequence with the same data leads to a certain cycle of the power consumption of the processor. If this programme is run with different data, the cycle of the power consumption differs. This deviation is used to determine the processed data.

In comparison to the simple power analysis (SPA) the differential power analysis (DPA) makes it possible to discover even smaller differences in the power consumption of the microcontroller. For this purpose the power consumption is first determined during the processing of known data and then during the processing of unknown data. The measuring is usually repeated various times and the mean value is calculated to eliminate the noise. After the measuring is terminated, the difference is determined and from the result the unknown data can be concluded.

The power analysis of smart card microcontrollers is an attack to be taken very seriously for unprepared hardware and software. The reason for this is that in some microcontrollers there can well be dependencies of the power consumption from the corresponding machine instruction and also from the data processed with this machine instruction. Furthermore, the required effort for a successful attack regarding the measuring equipment is rather small. There are, however, a number of effective countermeasures, which are based on improved hardware on the one hand and on modified software on the other hand.

The simplest hardware related solution is the installation of a fast voltage regulator on the chip that ensures power consumption independent from the machine instruction and data with the help of a shunt resistor. A technically more challenging solution is the use of a modified semiconductor design of the processor that leads to constant power consumption. However, some of these attempts increase the power consumption of the microcontrollers, which is undesirable in certain areas of application. A simple countermeasure during an

SPA/DPA critical process can also be the activation of components not required for this process such as CRC checksum generator or numerical coprocessor with random data as input values in order to produce an artificial noise of the power consumption. Another approach towards safeguarding the smartcard from DPA is presented by HoonJae at al [45].

The easiest approach, regarding the software related countermeasures, is the exclusive use of machine instructions with very similar power consumption. Machine instructions with a significant deviation from the average power consumption must not be used in the assembler code anymore. A further attempt is the introduction of different orders for the same calculations of cryptoalgorithms that are chosen randomly. This makes it much harder for the observer to recognise a convergence between known and unknown machine instructions and processed data. A similar attempt is the use of chip individual tables for the S-boxes of the (triple) DES algorithm. In order to complicate the data collection required prior to a successful power analysis, all keys should be secured by irreversible retry counters. Moreover, it is necessary to block the free access to all commands of the type internal authenticate in which any data can be sent through a cryptoalgorithms of the smart card.

3.12.2.5 Differential fault analysis

These attacks try to disturb the functioning of the card, by changing the power levels or the frequency of the external clock, or by exposing the card to different kinds of radiation. Each time the card performs symmetric or asymmetric cryptographic computations, one bit in the key is changed at some position. The result of a series of such computations, which are all different because the bit position is different in each, are analysed and used to compute the previously unknown key. The simplest mechanism is to let the card perform each cryptographic computation twice and to compare the results and compare the result. Different approach would be to append a random number to the data to be encrypted so that attacker can not analyse different results for the same plaintext. The random number generator on the smart card should ideally never repeat the random numbers at any time during the card life cycle.

3.13 ATTACKS ON SMART CARD AT LOGICAL LEVEL

Attacks on the security of a smart card on a logical level require above all an understanding of the communication and the flow of information between the terminal and the smart card. From an information technological point of view several logical level attack scenarios are presented next.

3.13.1 Dummy Smart Card

The most conceivable attack is the use of a smart card that has been self programmed and enhanced with a number of analysis and protocol functions. A few years ago this could hardly be carried out because a few companies could only purchase smart cards and the corresponding microcontrollers. By now, however, smart cards and configuration programmes can be purchased from various companies on the free market. This increases of course the number of possibilities available to an attacker. But irrespective of that with some effort and skills a functioning smart card can be build from a small plastic plate and a standard card microcontroller, or at least a card that functions like a real smart card, regarding the electric properties during data transmission. Such a dummy card can be used to record a part of the communication with the terminal, which can be evaluated later. After several attempts it may then be possible to execute a part of the communication process just as with a real smart card.

It can be argued, however, that a real advantage can be obtained from this as all professionally designed applications have a cryptographic protection for important actions. As long as one does not know the secret key the attack is over at the latest when it comes to the authentication. This attack would only be successful if the attackers know the secret key or if the complete application is running without cryptographic protection.

3.13.2 Data Transmission Bugging

In order to bug and, when required, manipulate the data during a session a slightly modified smart card can be used. An electrically insulated dummy contact can be attached to the cards I/O interface. The original I/O interface is then no longer electrically connected to it. The newly created (dummy) contact and the original I/O contact are connected to a fast computer. Depending on the programming this computer can cut out or insert any data during the communication between the terminal and the smart card. If the computer is fast enough, neither the terminal nor the smart card will be able to determine any difference from the regular data transmission during the manipulated communication.

It is clearly possible that this method can drastically influence the sequence of a session. An approved design criterion determines that the security must not be impaired by bugging, cutting out or inserting data during the communication.

3.13.3 Power supply cut off

This attack was still very successful a few years ago in many smart cards. It was simply achieved by cutting off of the power supply at a certain point of time while a command is executed. The background of this attack is the fact that in case of conventional programming all write operations on EEPROM pages are executed one after the other. If the programmer of the command has arranged the order of the write operations in an unwise fashion, the attacker can gain an advantage by cutting off the power supply at the right time. However, the designers of operating systems do know an effective countermeasure called atomic orders. They have the property of being atomic, i.e. indivisible. This means that they are either carried out completely or not at all, which is an adequate protection against the Cutting off Power Supply attack.

3.13.4 Time analysis at PIN comparison

Most programmers are always concerned that programs run as fast as possible. Generally, this is an important feature. This feature of runtime optimisation can also be used for a promising attack, though. If a smart card is in the process of verifying the PIN, the corresponding comparison routine carries out a byte-by-byte comparison of the entered PIN and the stored PIN. A programmer who does not pay attention to security will program the routine in such a way that a difference in the comparison of the two PIN codes leads to an immediate abort of the routine. This will result in minimal runtime differences that can be measured with suitable equipment (e.g. memory oscilloscope). These differences can be used by an attacker to determine the secret PIN in a relatively simple way. Few years ago the attack mentioned above was still successful. This type of attack is prevented by the comparison routines that are designed in such a way that always all digits of a PIN are compared. Thus, there is no time difference between positive and negative comparison results.

3.13.5 Noise free Cryptoalgorithm

In late 1990’s and even in the early 2000's some cryptoalgorithms were still used with major differences in runtime depending on the key and the un-coded text. With the key space being reduced that way the attacker can use a brute-force attack to search for the secret key. How long the search takes depends to a large extent on the noise of the

algorithm. The larger the time differences the smaller the key space, and the simpler and faster the key search. If the exact implementation of the corresponding cryptoalgorithm on the target computer is known, this can also be used as an additional reference for creating timetables. This type of attack known as timing attack was published by Paul Kocher et al [46], dealing especially with time dependencies at RSA and DSS. Principally a timing analysis is very dangerous for the security of a smart card. Since now its known, all of today's smart cards exclusively use noise free cryptoalgorithms, i.e. the time for encrypting and decrypting is independent from the input values.

An additional security feature in some applications is the extra retry counter of all authentication keys so that only a certain number of unsuccessful authentications can be carried out. Once the retry counter has reached its maximum value, the smart card is locked against any further authentication attempts.

3.13.6 Differential Fault Analysis (DFA) manipulation:

Dan Boneh, Richard DeMillo and Richard Lipton published a paper [47], describing a theoretical model how secret keys of asymmetrical cryptoalgorithms can be calculated by causing hardware faults. Few months later Eli Biham and Adi Shamir published an extension of the Bell-core attack with the name differential fault analysis (DFA). It now also includes symmetrical cryptoalgorithms such as the DES algorithm. As a result many smart cards were open to this new attack method and considered significantly challenging, at least theoretically.

The basic principle of both attacks is rather simple: In the first step any given un-coded text is encrypted with the key to be decoded and the encrypted text is kept. After that the smart card is interrupted during the processing of the cryptographic algorithm externally, for instance, by ionised or high frequency rays so that one individual key bit is modified at any position. The result is a key text that has not been encrypted correctly due to the corrupted bit. This process is repeated a number of times and the results are kept for the analysis. The remaining part of determining the key is pure mathematics and is specified comprehensively in the publications Dan Boneh et al [47].

The power of the attack lies in the fact that it is not even necessary to know at which position of the secret key a bit was corrupted. Biham and Shamir quote in their publication that in the case of one corrupted key bit, 200 key text blocks are enough to generate the complete secret DES key. If a real Triple DES (168 bit) is used instead of the DES, the number of required key texts does not increase significantly. Even if more than one bit is

changed, this attack is still effective. Only the number of wrongly encrypted key texts required is increased. Mark Karpovsky et al [48] presents the vulnerability of AES against side-channel attacks known as Differential Fault Analysis attacks. One of their architecture, which is efficient for faults of higher multiplicity, partitions the design into linear (XOR gates only) and nonlinear blocks and uses different protection schemes for these blocks. Linear blocks are protected with linear codes and the nonlinear with a complimentary nonlinear operation resulting in robust protection. The second architecture uses systematic nonlinear (cubic) robust error detecting codes and provides for high fault detection for faults of low and high multiplicities but has higher hardware overhead.

In practice, this type of attack is not as simple as it seems. If possible, only one bit or at least very few bits are supposed to be modified. If the complete microcontroller is subject to high frequency microwaves, so many bits are modified that the processor typically crashes eventually. Therefore, the attackers try to get the CPU to make one single wrong calculation with the help of e.g. intentionally created glitches (glitches are very short losses or increases in voltage) in the power supply or core rate generation. If the filters situated on the corresponding input lines cannot neutralise such a glitch, the desired miscalculation of the processor can occur.

A smart card, however, is not without protection against the DFA attack, if corresponding care has been applied beforehand. The simplest defence is to calculate the cryptoalgorithm in the smart card twice and to compare the two results. If the results are identical, then no attempt was made to corrupt any bits from the outside. In this context, it is assumed that an intentional implementation of faults can never modify the same bits in the smart card. This is an assumption that is very close to reality, because if a targeted modification of certain bits should ever be possible in a smart card processor, then there are much simpler and quicker attacks than a DFA.

The big disadvantage of a double calculation is the additional time required, which may cause problems. This especially concerns attacks on time consuming asymmetrical crypto procedures such as RSA or DSS. A further effective defensive measure against differentiated fault analyses can be achieved by always exclusively encrypting different uncoded texts. The simplest solution is the use of a random number in front of the uncoded text to be encrypted. Consequently, the cryptoalgorithm always encrypts different data and a DFA is no longer possible.

3.13.7 Disrupting the Processor

Similar to the use of the differentiated fault analysis (DFA) when attacking secret keys of cryptoalgorithms, it can be attempted to disrupt the processor in order to influence the sequences in the programme code. An attack known as light attack, that has been in the knowledge to the manufacturers of smart cards and smart card microcontrollers, and to some system houses was published in 2002 by Sergei Skorobogatov and Ross Anderson [49] as Optical Fault Induction Attacks. The publication describes an arrangement in which a regular flashlight is flanged to the camera adapter of a conventional light microscope. This is used to flash a very limited area of the RAM of a standard microcontroller (PIC16F84). This arrangement makes it possible to selectively set certain bits in the RAM of this microcontroller to the value 0 or 1 provided that it has no protection against this type of attacks.

In order to disrupt the processor, glitches on the supply lines, light flashes on the chip or on parts of the chip or even high frequency can be applied [50]. If at the correct point of time the jamming of the programme sequence is triggered, a query can be influenced selectively. The programme function specified has the task of transmitting the content of a send buffer whose limits are defined by a starting address and an ending address. If the attacker manages to interfere selectively with the query for the end of the send buffer, the data following the send buffer will also be transmitted to the terminal. If this memory area of the RAM were to include the secret key of a cryptoalgorithm, then this key could be obtained without permission using this method.

The defence against this attack consists of various steps. It is important that the smart card microcontroller is equipped with the corresponding sensors to detect all disruption attempts of the processor. This can be voltage sensors detecting glitches, and a large number of corresponding light sensors on the chip. The second protective layer must be realised within the software. As an additional countermeasure, the query can be carried out twice, where the timeframe between the two queries should be randomly chosen. As a result, the attacker would have to use two light flashes for manipulating the query and, moreover, would have the problem that he cannot exactly predict the point of time for the second light flash.

In addition, all confidential data should be deleted from the RAM immediately after their use or it should be temporarily encrypted. In order to further reduce the effects of this attack, it is also sensible to encrypt all secrets (e.g. PIN, key) in the EEPROM. That is to say if an attacker should manage to read parts of the EEPROM by manipulating queries, he would only receive encrypted data as a result, which are useless to him. Furthermore, modern processors can detect illegal machine code and invalid addresses and react correspondingly. This defence scenario provides a good impression of how a serious attack can be blocked off by the corresponding cooperation of protective measures of hardware and software.

3.14 CHAPTER SUMMARY

This chapter presents an overview of smart cards in great depth, especially their architecture, standards and operation. This is followed be a lengthy discussion on what smart cards are to be considered in providing a cryptographic data security system. Various concerns are outlined and issued clarified. Lastly, the chapter presents a detailed study of attacks on smart cards when in use. These attacks are analysed at the physical and logical levels. Based on these findings and a detailed attack analysis, smart cards will be used in CHAPTER 1 to build the proposed secure system by using cryptographic protocols and mechanisms that will be presented in the next chapter.

CHAPTER 4
SECURITY PROTOCOLS

4.1 INTRODUCTION

A cryptographic protocol (protocol) is a distributed algorithm defined by a sequence of steps precisely specifying the actions required from two or more entities to achieve a specific security objective. As opposed to a protocol, a mechanism is a more general term encompassing protocols, algorithms (specifying the steps followed by a single entity), and noncryptographic techniques (e.g., hardware protection and procedural controls) to achieve specific security objectives. Protocols play a major role in cryptography and are essential for meeting cryptographic goals. Encryption schemes, digital signatures, hash functions, and random number generation are among the primitives which may be used to build a protocol. This chapter presents a brief summary of various security protocols that are easily available and commonly used today. Most of these have been well studied and thoroughly scrutinised in various security aspects. The last two sections of this chapter present the detailed structure and operations of SSL and SSH protocols.

4.2 SECURE HYPERTEXT TRANSFER PROTOCOL

Secure HTTP is a secure message-oriented communications protocol designed for use in conjunction with HTTP. It is designed to coexist with HTTP's messaging model and to be easily integrated with HTTP applications. SHTTP, RFC2660, provides independently applicable security services for transaction confidentiality, authenticity/integrity and non-repudiability of origin. The protocol emphasizes maximum flexibility in choice of key management mechanisms, security policies and cryptographic algorithms by supporting option negotiation between parties for each transaction. SHTTP provides secure communication mechanisms between an HTTP client-server pair in order to enable spontaneous commercial transactions for a wide range of applications. SHTTP is a flexible protocol that supports multiple orthogonal operation modes, key management mechanisms, trust models, cryptographic algorithms and encapsulation formats through option negotiation between parties for each transaction.

Secure HTTP provides a variety of security mechanisms to HTTP clients and servers, providing the security service options appropriate to the wide range of potential end uses possible for the World-Wide Web. Several cryptographic message format standards may be incorporated into SHTTP clients and servers. SHTTP supports interoperation among a variety of implementations, and is compatible with HTTP. SHTTP aware clients can communicate with SHTTP oblivious servers and vice-versa, although such transactions obviously would not use SHTTP security features.

SHTTP does not require client-side public key certificates (or public keys), as it supports symmetric key-only operation modes. This is significant because it means that spontaneous private transactions can occur without requiring individual users to have established public key. While SHTTP is able to take advantage of ever-present certification infrastructures, its deployment does not require it.

SHTTP supports end-to-end secure transactions. It also provides full flexibility of cryptographic algorithms, modes and parameters. Option negotiation is used to allow clients and servers to agree on transaction modes (e.g., should the request be signed or encrypted or both -- similarly for the reply?); cryptographic algorithms (RSA vs. DSA for signing, DES vs. RC2 for encrypting, etc.); and certificate selection.

4.3 SECURE REMOTE PASSWORD

The Secure Remote Password (SRP) protocol, created at Stanford University, is a security protocol for authentication. SRP is not a complete security solution in itself, but rather a technology that can be a part of a security system. The design goal of SRP is to improve on the security properties of password-style authentication, while retaining its considerable practical advantages. You have to carry your private key with you on a diskette and hope that you can get the key into whatever machine you need to use. SRP provides strong two-party mutual authentication, with the client needing only to remember a short password which need not be so strongly random. With traditional password schemes, the server maintains a sensitive database that must be protected, such as the passwords themselves, or hashed versions of them. That data must be kept secret, since disclosure allows an attacker to impersonate users or discover their passwords through a dictionary attack. The design of SRP avoids such a database and allows passwords to be less random (and therefore more memorable and useful), since it prevents dictionary attacks. The server still has sensitive data that should be protected, but the consequences of its disclosure are less severe. SRP is also intentionally designed to avoid using encryption algorithms in its operation. Thus it

avoids running afoul of cryptographic export laws, which prohibits certain encryption technologies from being shared with foreign countries. The current SRP implementation includes secure clients and servers for the Telnet and FTP protocols for Unix and Windows.

4.4 THE KERBEROS NETWORK AUTHENTICATION SERVICE

The Kerberos protocol, RFC4120, provides a means of verifying the identities of principals (a named client or server entity that participates in a network communication), e.g., a workstation user or a network server on an open (unprotected) network. This is accomplished without relying on assertions by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets travelling along the network can be read, modified, and inserted at will. Kerberos performs authentication under these conditions as a trusted third-party authentication service by using conventional shared secret key cryptography.

The basic Kerberos authentication process proceeds as follows: A client sends a request to the authentication server (AS) for "credentials" for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of a "ticket" for the server and a temporary encryption key, often called a session key. The client transmits the ticket, (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication. Many applications use Kerberos' functions only upon the initiation of a stream-based network connection. Unless an application performs encryption or integrity protection for the data stream, the identity verification applies only to the initiation of the connection, and it does not guarantee that subsequent messages on the connection originate from the same principal.

The Kerberos protocol consists of several sub-protocols (or exchanges). There are two basic methods by which a client can ask a Kerberos server for credentials. In the first approach, the client sends a cleartext request for a ticket for the desired server to the AS. The reply is sent encrypted in the client's secret key. In the second method, the client sends a request to the ticket granting server (TGS). The client uses the ticket granting tag (TGT) to authenticate itself to the TGS in the same manner as if it were contacting any

other application server that requires Kerberos authentication. The reply is encrypted in the session key from the TGT. Once obtained, credentials may be used to verify the identity of the principals in a transaction, to ensure the integrity of messages exchanged between them, or to preserve privacy of the messages.

To verify the identities of the principals in a transaction, the client transmits the ticket to the application server. Because the ticket is sent "in the clear" (parts of it are encrypted) and might be intercepted and reused by an attacker, additional information is sent to prove that the message originated with the principal to whom the ticket was issued. This information (called the authenticator) is encrypted in the session key and includes a timestamp. The timestamp proves that the message was recently generated and is not a replay. Encrypting the authenticator in the session key proves that it was generated by a party possessing the session key. Since no one except the requesting principal and the server know the session key (it is never sent over the network in the clear), this guarantees the identity of the client.

The integrity of the messages exchanged between principals can also be guaranteed by using the session key (passed in the ticket and contained in the credentials). This approach provides detection of both replay attacks and message stream modification attacks. It is accomplished by generating and transmitting a collision-proof checksum (a hash or digest function) of the client's message, keyed with the session key. Privacy and integrity of the messages exchanged between principals can be secured by encrypting the data to be passed by using the session key contained in the ticket or the sub-session key found in the authenticator.

4.4.1 Security Considerations

As an authentication service, Kerberos provides a means of verifying the identity of principals on a network. By itself, Kerberos does not provide authorization. Applications should not accept the issuance of a service ticket by the Kerberos server as granting authority to use the service, since such applications may become vulnerable to the bypass of this authorization check in an environment where they inter-operate with other key distribution centres (KDC) or where other options for application authentication are provided.

Denial of service attacks are not solved with Kerberos. There are places in the protocols where an intruder can prevent an application from participating in the proper authentication steps. Because authentication is a required step for the use of many services, successful

denial of service attacks on a Kerberos server might result in the denial of other network services that rely on Kerberos for authentication. Kerberos is vulnerable to many kinds of denial of service attacks: those on the network, which would prevent clients from contacting the KDC; those on the domain name system, which could prevent a client from finding the IP address of the Kerberos server; and those by overloading the Kerberos KDC itself with repeated requests.

Authentication servers maintain a database of principals (i.e., users and servers) and their secret keys. The security of the authentication server machines is critical. The breach of security of an authentication server will compromise the security of all servers that rely upon the compromised KDC, and will compromise the authentication of any principals registered in the realm of the compromised KDC.

Password-guessing attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an off-line dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained that are encrypted under a key derived from the user's password. Because a client can request a ticket for any server principal and can attempt a brute force or dictionary attack against the server principal's key using that ticket, it is strongly encouraged that keys be randomly generated (rather than generated from passwords) for any principals that are usable as the target principal for a Ticket-Granting Service messages.

Principals must keep their secret keys secret. If an intruder somehow steals a principal's key, it will be able to masquerade as that principal or impersonate any server to the legitimate principal. Principal identifiers must not recycle on a short-term basis. A typical mode of access control will use access control lists to grant permissions to particular principals. If a stale access control lists entry remains for a deleted principal and the principal identifier is reused, the new principal will inherit rights specified in the stale access control lists entry. By not reusing principal identifiers, the danger of inadvertent access is removed.

Kerberos credentials contain clear-text information identifying the principals to which they apply. If privacy of this information is needed, this exchange should itself be encapsulated in a protocol providing for confidentiality on the exchange of these credentials.

Applications must take care to protect communications subsequent to authentication, or by applying their own confidentiality or integrity mechanisms on such communications. Applications applying confidentiality and integrity protection mechanisms other than private encrypted application message must make sure that the authentication step is appropriately linked with the protected communication channel that is established by the application.

Unless the application server provides its own suitable means to protect against replay (for example, a challenge-response sequence initiated by the server after authentication, or use of a server-generated encryption subkey), the server must utilize a replay cache to remember any authenticator presented within the allowable clock skew. All services sharing a key need to use the same replay cache. If separate replay caches are used, then an authenticator used with one such service could later be replayed to a different service with the same service principal.

Implementations of Kerberos should not use untrusted directory servers to determine the realm of a host. To allow this would allow the compromise of the directory server to enable an attacker to direct the client to accept authentication with the wrong principal i.e., one with a similar name, but in a realm with which the legitimate host was not registered.

4.5 TRANSPORT LAYER SECURITY PROTOCOL

The primary goal of the Transport layer security protocol (TLS) Protocol, RFC 4346, is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

1. The connection is private. Symmetric cryptography is used for data encryption (e.g., DES, RC4, AES, etc.) The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.

2. The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

The TLS Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

1. The peer's identity can be authenticated using asymmetric or public key, cryptography (e.g., RSA, DSS, etc.). This authentication can be made optional, but is generally required for at least one of the peers.

2. The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

3. The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that it is application independent protocol. Higher level protocols can layer on top of the TLS Protocol transparently. The TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left up to the designers of protocols which run on top of TLS.

4.6 IPSEC

IPsec, RFC2401, provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. IPsec can be used to protect one or more "paths" between a pair of hosts, between a pair of security gateways, or between a security gateway and a host. (Security gateway refer to an intermediate system that implements IPsec protocols, e.g. a router or a firewall)

The set of security services offered includes access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. Because these services are provided at the IP layer, they can be used by any higher layer protocol, e.g., TCP, UDP, ICMP, BGP, etc.

The fundamental components of the IPsec security architecture are categorised as: security Protocols -- Authentication Header (AH), RFC2404, and Encapsulating Security Payload (ESP), RFC2406; key management -- manual and automatic (The Internet Key Exchange (IKE)), RFC 2409 & 4306; algorithms for authentication and encryption, RFC4109; security associations -- what they are and how they work, how they are managed, associated processing.

When these mechanisms are correctly implemented and deployed, they ought not to adversely affect users, hosts, and other Internet components that do not employ these security mechanisms for protection of their traffic. These mechanisms also are designed to be algorithm-independent. This modularity permits selection of different sets of algorithms without affecting the other parts of the implementation.

IPsec uses two protocols to provide traffic security. IP Authentication Header (AH), provides connectionless integrity, data origin authentication, and an optional anti-replay service. The Encapsulating Security Payload (ESP) protocol may provide confidentiality (encryption), and limited traffic flow confidentiality. It also may provide connectionless integrity, data origin authentication, and an anti-replay service. (One or the other set of these security services must be applied whenever ESP is invoked.). Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

These protocols may be applied alone or in combination with each other to provide a desired set of security services in IPv4 and IPv6. Each protocol supports two modes of use: transport mode and tunnel mode. In transport mode the protocols provide protection primarily for upper layer protocols; in tunnel mode, the protocols are applied to tunnelled IP packets.

IPsec allows the user (or system administrator) to control the granularity at which a security service is offered. For example, one can create a single encrypted tunnel to carry all the traffic between two security gateways or a separate encrypted tunnel can be created for each TCP connection between each pair of hosts communicating across these gateways. IPsec management must incorporate facilities for specifying:

- which security services to use and in what combinations
- the granularity at which a given security protection should be applied
- the algorithms used to effect cryptographic-based security

Because these security services use shared secret values (cryptographic keys), IPsec relies on a separate set of mechanisms for putting these keys in place. (The keys are used for authentication/integrity and encryption services.). It specifies a specific public-key based approach (IKE) for automatic key management, but other automated key distribution techniques like Kerberos could be employed.

4.6.1 Security Associations

A Security Association (SA) is a simplex "connection" that affords security services to the traffic carried by it. Security services are afforded to an SA by the use of AH, or ESP, but not both. To secure typical, bi-directional communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

Transport mode SA is a security association between two hosts. In IPv4, a transport mode security protocol header appears immediately after the IP header and any options, and before any higher layer protocols (e.g., TCP or UDP). In IPv6, the security protocol header appears after the base IP header and extensions, but may appear before or after destination options, and before higher layer protocols. In the case of ESP, a transport mode SA provides security services only for these higher layer protocols, not for the IP header or any extension headers preceding the ESP header. In the case of AH, the protection is also extended to selected portions of the IP header, selected portions of extension headers, and selected options (contained in the IPv4 header, IPv6 Hop-by-Hop extension header, or IPv6 Destination extension headers).

Tunnel mode SA is essentially an SA applied to an IP tunnel. Whenever either end of a security association is a security gateway, the SA must be tunnel mode. Thus an SA between two security gateways is always a tunnel mode SA, as is an SA between a host and a security gateway. If AH is employed in tunnel mode, portions of the outer IP header are afforded protection, as well as the entire tunnelled IP packet. If ESP is employed, the protection is afforded only to the tunnelled packet, not to the outer header.

4.6.2 SA and Key Management

IPsec mandates support for both manual and automated SA and cryptographic key management. The IPsec protocols, AH and ESP, are largely independent of the associated SA management techniques, although the techniques involved do affect some of the security services offered by the protocols. For example, the optional anti-replay services available for AH and ESP require automated SA management. Moreover, the granularity of key distribution employed with IPsec determines the granularity of authentication provided. In general, data origin authentication in AH and ESP is limited by the extent to which secrets used with the authentication algorithm (or with a key management protocol that creates such secrets) are shared among multiple possible sources.

- Manual technique is the simplest form of management, in which a person manually configures each system with keying material and security association management data relevant to secure communication with other systems. Manual techniques are practical in small, static environments but they do not scale well. For example, a company could create a Virtual Private Network (VPN) using IPsec in security gateways at several sites. If the number of sites is small, and since all the sites come under the purview of a single administrative domain, this is likely to be a feasible context for manual management techniques. In this case, the security gateway might selectively protect traffic to and from other sites within the organization using a manually configured key, while not protecting traffic for other destinations. It also might be appropriate when only selected communications need to be secured. A similar argument might apply to use of IPsec entirely within an organization for a small number of hosts and/or gateways. Manual management techniques often employ statically configured, symmetric keys, though other options also exist.

- Automated technique is the widespread deployment and use of IPsec requires an Internet-standard, scalable, automated, SA management protocol. Such support is required to facilitate use of the anti-replay features of AH and ESP, and to accommodate on-demand creation of SAs, e.g., for user- and session-oriented keying. The default automated key management protocol selected for use with IPsec is IKE under the IPsec domain of interpretation

4.6.3 Performance Issues

The use of IPsec imposes computational performance costs on the hosts or security gateways that implement these protocols. These costs are associated with the memory needed for IPsec code and data structures, and the computation of integrity check values, encryption and decryption, and added per-packet handling. The per-packet computational costs will be manifested by increased latency and, possibly, reduced throughout. Use of SA/key management protocols, especially ones that employ public key cryptography, also adds computational performance costs to use of IPsec. These per-association computational costs will be manifested in terms of increased latency in association establishment. For many hosts, it is anticipated that software-based cryptography will not appreciably reduce throughput, but hardware may be required for security gateways (since they represent aggregation points), and for some hosts.

The use of IPsec also imposes bandwidth utilization costs on transmission, switching, and routing components of the Internet infrastructure, components not implementing IPsec. This is due to the increase in the packet size resulting from the addition of AH and/or ESP headers, AH and ESP tunnelling (which adds a second IP header), and the increased packet traffic associated with key management protocols. It is anticipated that, in most instances, this increased bandwidth demand will not noticeably affect the Internet infrastructure. However, in some instances, the effects may be significant, e.g., transmission of ESP encrypted traffic over a dialup link that otherwise would have compressed the traffic.

Note: The initial SA establishment overhead will be felt in the first packet. This delay could impact the transport layer and application. For example, it could cause TCP to retransmit the SYN before the ISAKMP (Internet Security Association and Key Management Protocol, RFC2408) exchange is done. The effect of the delay would be different on UDP than TCP because TCP shouldn't transmit anything other than the SYN until the connection is set up whereas UDP will go ahead and transmit data beyond the first packet.

4.7 GPRS RELATED MODEL

Another model that can be used is similar to GPRS (General Packet Radio Services) shown in Figure 4.1, in which MS (Mobile Station) communicates with SGSN (Serving GPRS Support Node). Algorithms A3, A8, A5 are used for authentication, cipher key generation and encryption respectively, and are network or operator specific.

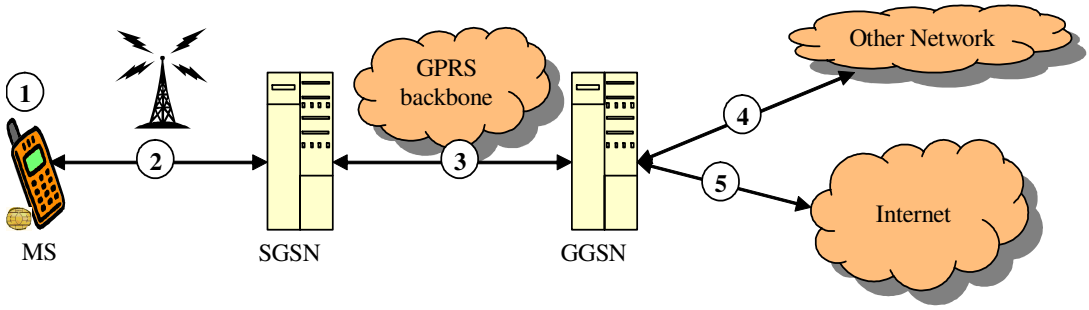

Figure 4.1 The security issues in GPRS [51]

From Figure 4.1 there are five main areas where security in the GPRS system is exposed. These five areas are:

1. Security aspects related to the mobile station (MS), mobile phone and the SIM (Subscriber Identity Module) card;
2. Security mechanics between the MS and the SGSN, which include also the air interface from the MS to the BSS.
3. The PLMNs (Public Land Mobile Network) backbone network security that mainly referrers to the traffic between the SGSN and the GGSN (Gateway GPRS Support Node), but also handles the flow of subscriber information, like triplets.
4. Security between different operators.
5. Security between GGSN and the external connected networks, like the Internet.

The data security is only provided over the radio link. Authentication is performed initially when the user switches the device (MS) on, and afterward it is network or operator specific. In this model smart card or SIM is not used for actual data encryption, but rather to store the IMSI (International Mobile Subscriber Identity), Ki, the ciphering key generating algorithm, and personal Identification Number (PIN). Hardware based cryptography is preferred instead of smart card. A large database is also required to store all the necessary parameters for each specific SIM.

To address the above security issues, different approaches can be taken and these are presented next. The first is to use an end-to-end VPN (Virtual Private Network), Figure 4.2. In this scenario, traffic is encrypted at the VPN client and decrypted at the corporate VPN server, thus the traffic goes encrypted throughout the whole connection. The operator handles authentication.

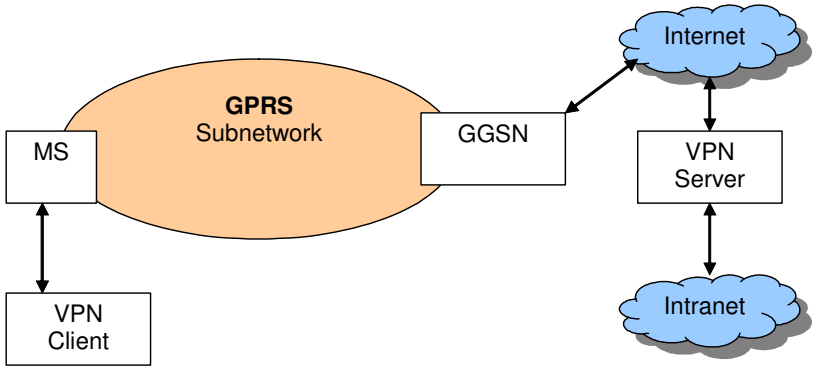

Figure 4.2 End-to-end VPN connection setting [51]

A different approach towards solving the problem is to create a VPN between the corporate Intranet to be accessed and the GGSN providing the access to the Internet, Figure 4.3. The traffic is encrypted at a VPN server or an encrypting router. Now the traffic is transported encrypted over the public Internet used to access the intranet.

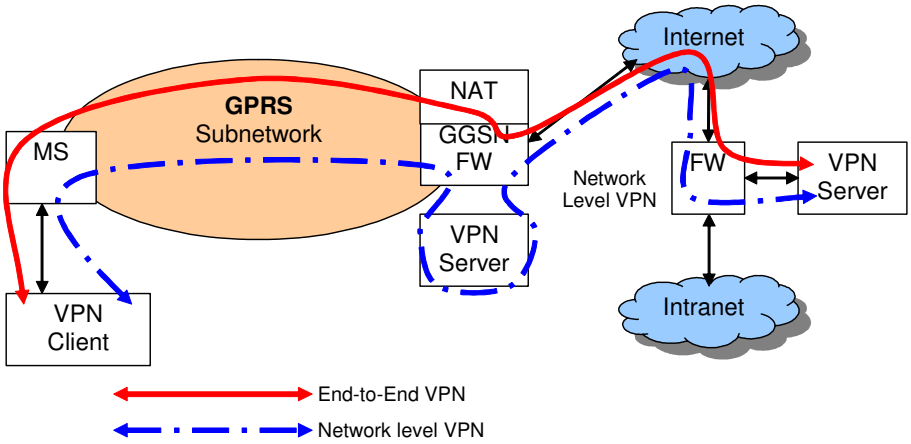

Figure 4.3 Network level VPN settings [51]

A similar VPN server at the corporate intranet gateway decrypts the traffic. The user is authenticated based on its access permission. Traffic used to connect anything else but the corporate intranet will go unencrypted.

This solution would mean, that the corporate customer would have to trust the mobile operator, or, for example, use encryption at application level. The traffic is unencrypted on the GPRS backbone, which means examining the traffic in the GPRS backbone, for example for legal interception purposes, would not be difficult.

4.8 CHAPTER SUMMARY

This chapter presents a detailed overview of several security protocols that are widely used and are easily available. Each of the protocols presented has some advantages and disadvantages when looked at and compared with others for a specific type of requirement. No one mechanism is totally secure. Some protocols are meant for providing secure entity authentication, e.g. Kerbros, and others such as SHTTP to provide secure web access. The next chapter presents the models that will be designed for the relevant security environment using smart cards.

CHAPTER 5

DESIGN OF PROPOSED MODEL

5.1 INTRODUCTION

CHAPTER 4 presents several protocols and models that can be used to provide a secure communication session. This dissertation focuses on a strategy for securing the data, communication channel, and safekeeping of keys and distribution, knowing that the attacker has more or less complete control of the communication channel (the Internet) between the two clients.

Smart card security can be done either in software or in hardware. The choice of implementation depends on the application, and on the algorithm to be implemented. Hardware implementations are expensive but yield better throughput than software implementation. Another way of looking at secure communication could be based on physical separation or segmentation, between protected private networks – or Intranets – and the public Internet. Segmentation is a major impediment to the accomplishment of the concept of a global Internet [5]. The other solution is the cryptographic security approach. This offers a viable alternative to segmentation by preserving a strongly connected global network. It is this approach that is adopted in this dissertation.

The proposed model is to provide adequate safeguards so that the remote access to the fieldbus via a gateway by a known client over the Internet is safe. The model is to provide the security concerns like authentication of client, data origin, data integrity, and data secrecy. This is to be achieved by using the cryptographic algorithms and part of the cryptography done using smart cards, as smart card technology is becoming increasingly popular and difficult to breach and greatly motivated in CHAPTER 3. The next two sections propose the models that can enhance on the exiting procedure.

5.2 SECURE SOCKET LAYER PROTOCOL

Secure socket layer (SSL) protocol is a general-purpose protocol, which is used to secure any dialogue taking place between applications communicating across the Internet through TCP (Transport Control Protocol) sockets. The Netscape Corporation developed SSL as the standard for server authentication, and it secures data exchange through encryption. SSL is one of the two protocols for secure WWW connections (the other is SHTTP). Netscape has an open protocol standard originally developed as SSL. The SSL protocol, Figure 5.1, runs above TCP/IP and below higher-level protocols such as HTTP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, and allows both machines to establish an encrypted connection.

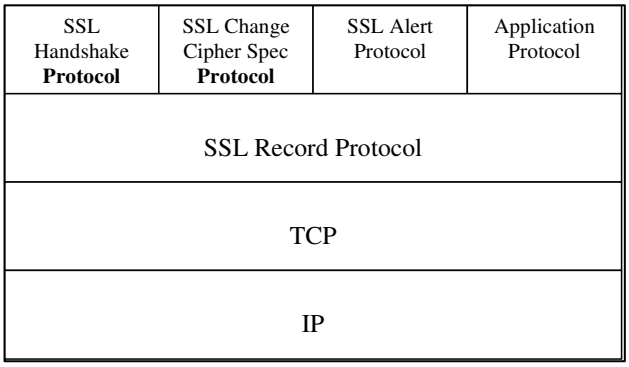

Figure 5.1 The SSL Protocol Stack [1]

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

SSL server authentication allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a CA listed in the client's list of trusted CA's. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

SSL client authentication allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID (Identification) are valid and have been issued by a CA listed in the server's list of trusted CA's. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.

An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a considerable degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected by a mechanism for detecting tampering – that is, for automatically determining whether the data has been altered in transit.

The SSL protocol includes two sub-protocols: the SSL record protocol and the SSL handshake protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL-enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

The SSL security protocol provides data encryption, server authentication, message integrity check and optional client authentication for a TCP/IP connection. SSL support the use of 40-bit and 128-bit symmetric cipher keys. In implementation, public-key systems are slower than symmetric ciphers. It is this fact that has lead to the combination of the two techniques to achieve both security and speed. SSL uses X.509 certificates for authentication, RSA or DSA as its public-key cipher and one of RC4-128, DES, Triple-DES or IDEA as its bulk symmetric cipher.

The SSL record protocol provides basic services to the various higher layer protocols for each SSL session. An SSL session is an association between a client and a server defining the cryptographic parameters for that connection. A session is started by the SSL handshake protocol. The handshake protocol allows the server and the client to authenticate each other and to negotiate an encryption algorithm, a message authentication algorithm and a set of cryptographic keys that will be used to protect data in an SSL record.

5.2.1 The SSL Handshake

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake**.** The handshake allows the server to authenticate itself to the client using public-key techniques, and then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server.

5.2.2 Server Authentication

SSL-enabled client software always requires server authentication, or cryptographic validation by a client of the server's identity. The SSL handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate to authenticate the identity, which the certificate claims to represent.

To authenticate the binding between a public key and the server identified by the certificate that contains the public key, an SSL-enabled client must receive a "yes" answer to the four questions set out in Figure 5.2. Although the fourth question is not technically part of the SSL protocol, it is the client's responsibility to support this requirement, which provides some assurance of the server's identity and thus helps protect against a form of security attack known as "man in the middle."

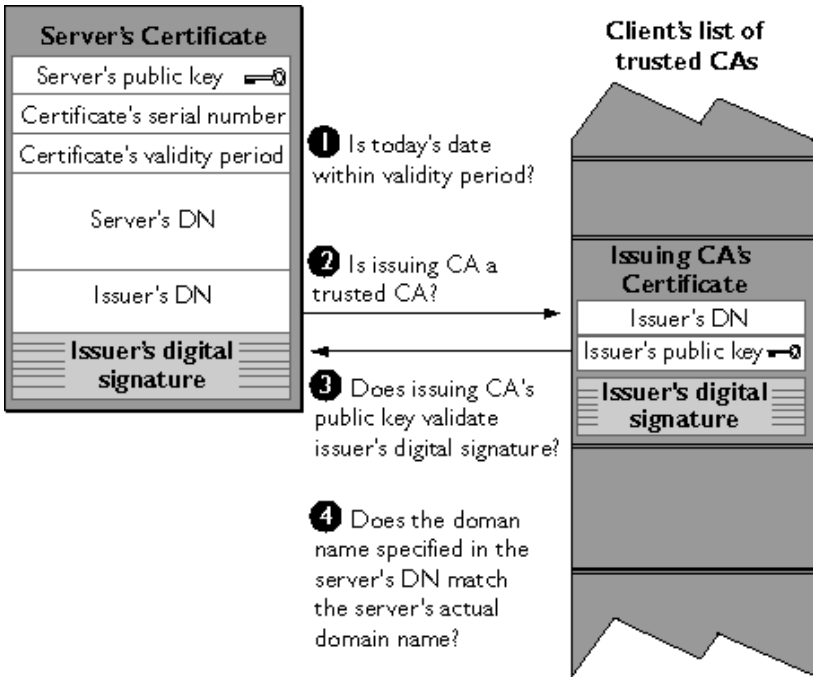

Figure 5.2 Server authenticates a client certificate [52]

After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends in Step 4 of the SSL handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

5.2.3 Client Authentication

SSL-enabled servers can be configured to require client authentication or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication, the client sends the server both a certificate and a separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one-way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

To authenticate the binding between the public key and the person or other entity identified by the certificate that contains the public key, an SSL-enabled server must receive a "yes" answer to the first four questions shown in Figure 5.3. Although the fifth question is not part of the SSL protocol, Netscape servers can be configured to support this requirement to take advantage of the user's entry in an LDAP directory as part of the authentication process.

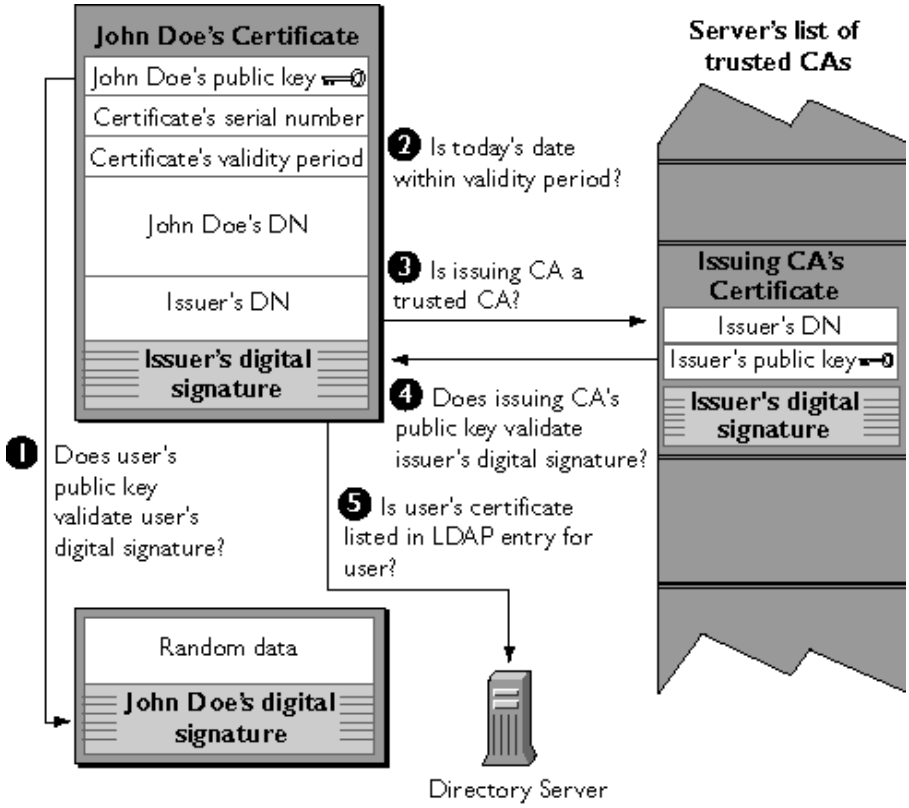

Figure 5.3 Client authenticates a server certificate [52]

This model is a mix of SSL/TLS protocol and uses a combination of certificates, for server authentication, and smart cards, for client authentication. In order to be cost effective, this model implements a simple solution to the problem, which can easily be integrated into any system. The model accomplishes this by implementing a generic solution, which is able to work with any smart card and should not incur the overhead of needing a trusted third party as a CA (Certification Authority).

In this approach the SSL is only an intermediate layer placed between TCP/IP and the application protocols. SSL will be used for the server authentication, data confidentiality, non-repudiation, and data integrity using encryption. Smart cards will provide the client authentication and store the encryption keys and server certificates safely. This will ensure a moderate level of security, as smart cards are considered fairly secure. This is to be achieved by incorporating the smart card encrypted data into the SSL protocol to provide a secure connection, by modifying the existing SSL protocol, and having a secure interface between the smart card API (Application Program Interface) and SSL. Detailed implementation of this protocol is presented in CHAPTER 6.

5.3 SECURE SHELL PROTOCOL

Secure Shell (SSH), RFC4251, is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

1. The Transport Layer Protocol server host authentication, key exchange, encryption, and integrity protection. It provides a confidential channel over an insecure network. It also derives a unique session id that may be used by higher-level protocols.
2. The User Authentication Protocol authenticates the client-side user to the server. It runs over the transport layer protocol.
3. The Connection Protocol multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

Two different trust models can be used with SSH:

1. The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure, and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.
2. The host name-to-key association is certified by a trusted certification authority (CA). The client only knows the CA root key, and can verify the validity of all host keys certified by accepted CA's.

The second alternative eases the maintenance problem, since ideally only single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible. Also, a lot of trust is placed on the central infrastructure.

The primary goal of the SSH protocol is to improve security on the Internet. It attempts to do this in a way that is easy to deploy, even at the cost of absolute security.

- All encryption, integrity, and public key algorithms used are well-known, well-established algorithms.
- All algorithms are used with cryptographically sound key sizes that are believed to provide protection against even the strongest cryptanalytic attacks for decades.
- All algorithms are negotiated, and in case some algorithm is broken, it is easy to switch to some other algorithm without modifying the base protocol

Secure Shell has seen steady improvement and increased adoption since 1995. SSHv1 was designed to replace the non-secure UNIX "rcommands" (rlogin, rsh, and rcp). SSHv2, submitted as an IETF draft in 1997, addresses some of the more serious vulnerabilities in SSHv1 and also provides an improved file transfer solution. Its increasing popularity has been fuelled by the broader availability of commercially developed and supported client and server applications for Windows, UNIX and other platforms, and by the efforts of the OpenSSH project [53] to develop an open source implementation.

Secure Shell is a popular network security protocol that defines a specification for conducting secure communication over an insecure network. It provides authentication, encryption and data integrity to secure network communications. According to Barrett and Silverman [54], client/server SSH products exist mainly in UNIX. In general, other platforms only have client parts, whilst Windows based SSH servers are only now appearing. This is not surprising, given that SSH offers a good solution to the problem of securing data sent over an insecure public network. Van Dyke [55] claims that SSH has three main capabilities.

5.3.1 Secure Command Shell

Command shells such as those available in Linux, UNIX, Windows, or the familiar DOS (Denial of Service) prompt, provide the ability to execute programs and other commands, usually with character input and output.

5.3.2 Port Forwarding

Port forwarding, as shown in Figure 5.4 allows a TCP/IP application's data to be securely transmitted over insecure channels. The application is completely unaware of the underlying forwarding mechanism and continues to send data as usual, but SSH will take all data going into the specific port and forward it to a SSH specified port on the other side, where the secure data received must be routed to the port, where the other TCP/IP application expects the data to be. It works both ways.

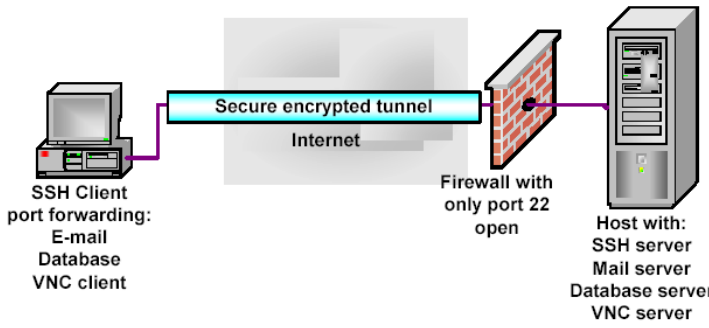

Figure 5.4 Port forwarding allowing multiple TCP/IP applications to share a single secure connection [55].

5.3.3 Secure File Transfer

SFTP (Secure File Transfer Protocol) is an interactive file transfer protocol which performs all operations over the SSH transport layer and replaces the original SCP (Secure Copy) protocol that existed in SSHv1. It is highly recommended that SFTP be used to perform file transfers in preference to the legacy FTP protocol, as in the latter, authentication details are transmitted in plain-text format and as such may be compromised through "password sniffing" attacks. The former also uses the same port as the SSH server, eliminating the need to open another port on the firewall or router.

According to Van Dyke and Barrett and Silverman, the SSH protocol provides three basic security benefits.

***Authentication – user & host*:** There are essentially three different methods of authentication – public-key, password based, and host based authentication. Of these, public-key authentication is one of the most secure methods of authentication using SSH. When the client connects to the server, it proves that it has the secret or private counterpart to the public-key on that server, and access is granted. This is shown in Figure 5.5.

The private-key never leaves the host machine, and therefore cannot be stolen or guessed like a password can. Usually the private-key has a "passphrase" associated with it, so, even if the private-key is stolen, the attacker must still guess the passphrase in order to gain access. Public-key authentication does not "trust" any information from a client or allow any access until the client can prove it has the "secret" private-key. Similarly, a server key is used by a server to prove its identity to a client and by a client to verify a "known" server. Server keys are described as persistent (they are changed infrequently) and are asymmetric. The password authentication of SSH is more secure than normal password methods because the password is sent encrypted.

Data Encryption: When a client establishes a connection with an SSH server, they must agree which cipher they will use to encrypt and decrypt data. The server generally presents a list of the ciphers it supports, and the client then selects the first cipher in its list that matches one on the server's list, as shown in Figure 5.5.

Session keys are the "shared keys" described above and are randomly generated. Both the client and the server use the same session key to encrypt and decrypt data. Session keys are generated after server authentication is successfully performed, but before user authentication, so that, if need be, usernames and passwords can be sent encrypted.

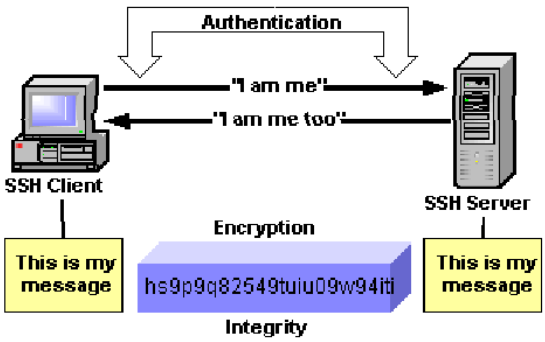

Figure 5.5 Secure Shell authentication, encryption and integrity [55].

Data Integrity: Even with SSH encryption, the data being sent over the network could still be vulnerable to someone inserting unwanted data into the message. SSHv2 uses HMAC (Hashed Message Authentication Code) algorithms to improve greatly upon SSHv1's simple 32-bit CRC data integrity checking method.

5.3.4 Major threats addressed by SSH

The following threats are those addressed by SSH:

- Eavesdropping – Combated by encryption.
- Name service and IP spoofing – Combated by server authentication.
- Connection hijacking – Combated by integrity checking.
- Man-in-the-middle-attacks – Combated by authentication.
- Insertion/replay attacks – Combated by HMAC.

5.3.5 Threats not addressed by SSH

- Password cracking – although sent securely, it can always be cracked or correctly guessed.
- Traffic analysis – cryptanalysts could monitor communications and look for patterns.
- Covert channels – unanticipated communication mechanisms.
- Carelessness of the user.

SSH enabled applications are very popular because of the security they supply for tasks carried out over a network. Some of the popular ones are Putty, SSH client for Windows [56] and VNC over SSH. As a security protocol, SSH has not been as popular as SSL, but it is gaining popularity, especially with the IETF which wants to include it as an official security protocol [53].

One of the major issues with SSH is the key exchange. The GSS-API (Generic Security Services Application Programming Interface) Authentication and Key Exchange for the SSH Protocol provides security services to callers in a mechanism-independent fashion. RFC4462 describes methods for using the GSS-API for authentication and key exchange in SSH. It defines an SSH user authentication method that uses a specified GSS-API mechanism to authenticate a user, and a family of SSH key exchange methods that use GSS-API to authenticate a Diffie-Hellman key exchange. It also defines a new host public key algorithm that can be used when no operations are needed using a host's public key, and a new user authentication method that allows an authorization name to be used in conjunction with any authentication that has already occurred as a side-effect of GSS-API-based key exchange.

This model is similar to SSL model, it further explores the use of smart cards not only for client authentication and safe keeping of keys but also to perform the actual cryptography. As can be seen in the first model, cryptography is performed on the client computer and not on the smart card. This can result in security lapses if the security of the computer is

compromised. This model uses the cryptographic capability of the smart card to encrypt and decrypt the data before it is handed to the communication channel. The secure communication channel will be implemented using SSH. In other words this model presents a scheme that is extremely challenging to break, because encryption/decryption will be performed on the data twice, firstly with smart card and secondly with the SSH protocol on the client computer.

This model uses security protocols like SSH, which are equally gaining popularity. There are, however, great possibilities in the effective and effortless integration of these two technologies in a cost effective and secure manner. This model bridges the gap between the two technologies by using a combination of the SSH protocol (to provide a secure encrypted channel for handling client and server authentication) and smart cards (user authentication to the system and data confidentiality using further encryption). The system will offer the best security due to double encryption, whilst using smart cards for authentication of users to the server. The SSH implementation is also unique. SSH is normally used to secure applications like *Telnet*, *FTP* et cetera, but in this approach it is used in a way very similar to SSL. Like SSL, it runs over TCP/IP and secures data sent between what would normally have been TCP/IP client/server applications, retaining all the security benefits of SSH. It is therefore almost a mixture of SSH port forwarding (without the linkage of different ports as in effect a genuine SSH socket is created through which data can be sent) and SSH. Detailed implementation of this model is presented in CHAPTER 7.

5.4 MODULAR BREAKDOWN OF THE PROPOSED MODEL

Figure 5.6 presents the modular block diagram of the security system that will be implemented. It uses SSL and SSH security protocols to secure the channel and then include the use of smart cards for further cryptographic capability. The server starts up with a “GUI”, and can automatically authenticate itself to its smart card. Thereafter, the server listens for connections through “Secure channel”. The client and server will be authenticated to each other through “Secure channel”. Here the type of protocol supported will be verified and selected. The user is now free to enter data, using client “GUI”, for transmission to the server and vice versa. This data may however first be sent to “Authentication, En/Decryption, Key storage”, for encryption with SC if necessary. If encryption-using SC is selected, the data is then further encrypted. This scrambled data is now transmitted over TCP/IP. The server creates a TCP/IP channel with “GUI”, through

which user resource allocation will be identified and finally Fieldbus access of data points or nodes can be established.

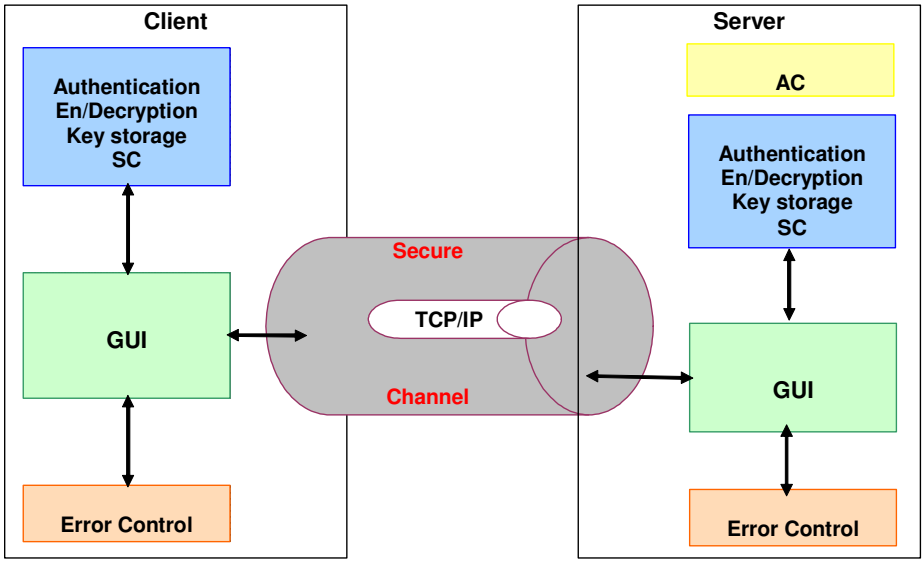

Figure 5.6 Functional diagram of the proposed models

5.5 CHAPTER SUMMARY

This chapter presents a detailed overview of secure models that will be implemented and whose behaviour will be analysed. Protocols such as like SSL and SSH allow a remote client to securely access a remote machine over the Internet, which best suits our remote access, once the requesting client is authenticated to the server and vice versa, secure communication can then take place. They provide for most aspects and goals set by cryptography and are the reason behind the choice. The next two chapters present the implementation of these protocols using smart cards.

CHAPTER 6

IMPLEMENTATION USING SSL-SC

6.1 INTRODUCTION

This chapter reviews the modular design approach adopted to implement the security model presented in CHAPTER 5. The first section presents a breakdown of the model that will be implemented using SSL. The second section presents the way in which smart cards are integrated with the server and client application.

6.2 ANALYSING SSL-SC MODEL

The following objectives are incorporated into the proposed model: a server program capable of handling concurrent client connections, as well as securing those client connections by implementing the SSLv3 or TLSv1 protocol to secure the communication between the client and the server; a client program that could connect to the server, capable of understanding the SSLv3 or the TLSv1 protocol, as well an interface with a smart card. Server authentication will be achieved by using digital certificates and client authentication using a smart card. Figure 6.1 shows the block level breakdown of the model, and explains the role of each unit.

Connection Control creates a socket, through which data will be sent and received between the client and the gateway server as well as between the gateway server and the AC server. The data packet is encapsulated within a TCP/IP packet. The TCP/IP protocol will provide error-detection, error-correction and retransmission to the communications link between the different entities. Other responsibilities are to manage, create and close the connections between the client and the server on the gateway, as well as to the AC server. All communications are secured using asynchronous I/O to the server and client program, as well by encrypting outgoing data and incoming data is verified by decryption.

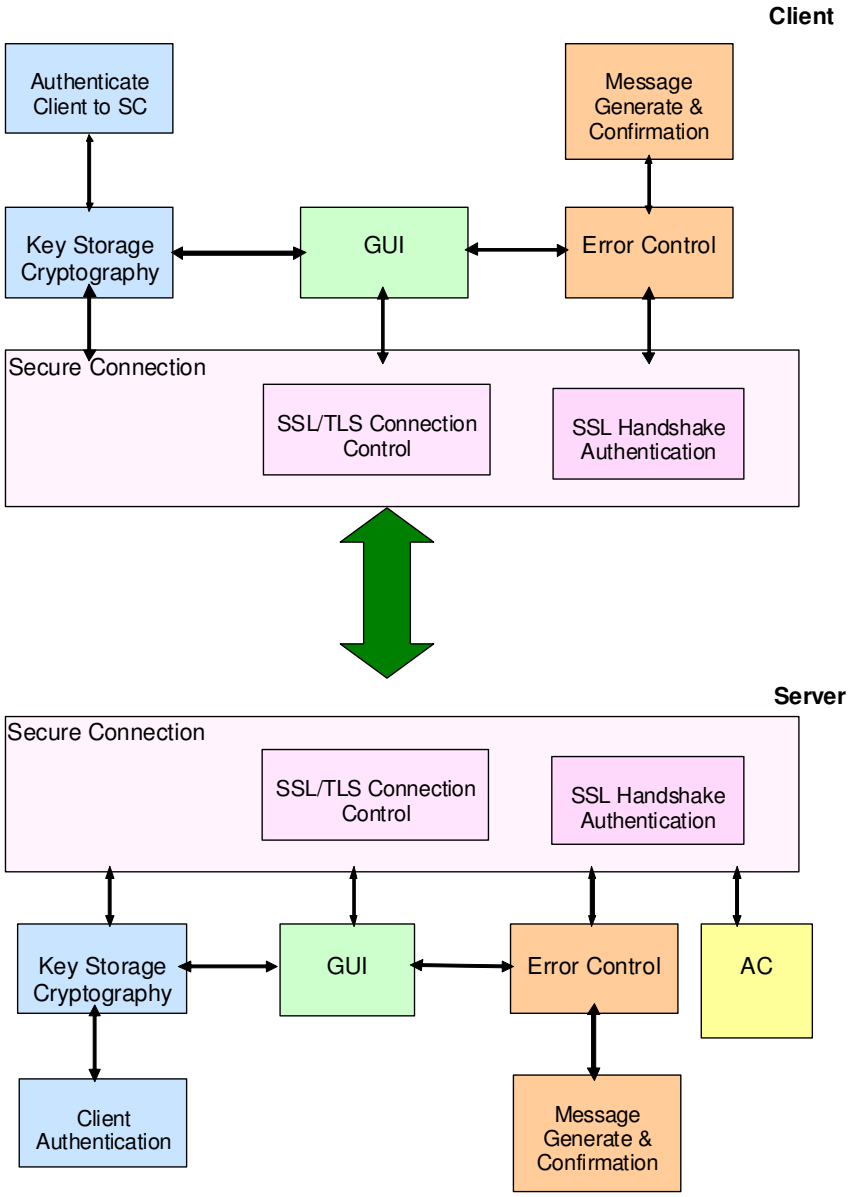

Figure 6.1 Functional diagram for the model using SSL-SC.

SSL Handshake and Authentication sets up the security parameters for the connection and authenticates the connecting client to the server. *Authenticate Client/Server to SC* authenticates the user to the smart card by using a PIN. The server is auto authenticated. *Key storage & Cryptography* keeps secret information secure and to perform cryptographic computations when requested to do so by the client or server program.

Error Control handles errors that occur during the execution of the client and server operations and/or displays error messages to the user, and in the case of the server, writes the errors in an error log file.

GUI provides a graphical interface to the user that will display received data and connection settings as well as providing interface controls to open a connection, close a connection, send data and set any other security related options for the connection to the server. Errors generated by *Error Control* as will be displayed here.

6.2.1 Model requirement

The socket interface implemented will use TCP/IP Berkley stream sockets to provide the transmission and reliable delivery of data, and secure programming principles will be used to protect the program against hacker attack. The SSLv3 or the TLSv1 protocol will be used to secure the connection between the client and the server. The program will be able to handle at least 100 concurrent client connections. Public-key cryptography for client authentication and X.509v3 certificates for server authentication will be used. A minimum key-length of 128-bits will be used in CBC mode of operation of a symmetric-key algorithm. Message authentication will be implemented using either MD5 or SHA-1 MAC. Asynchronous I/O will be provided to all connected clients. This means that data can be sent from the client to the AC server or from the AC server to the client at the same time.

DES algorithm is used to secure the communication between the smart card and the client computer. User smart card authentication by a PIN will be at least four characters in length. The smart card must be capable of implementing the RSA algorithm with a key-length of 1024-bits or more.

6.2.2 Design tools and choices

The gateway software interface program was developed using the Linux gcc and g++ compilers, as well as the OpenSSL software toolkit. The smart card used was a ZeitControl Professional BasicCard [29]. The smart card program was programmed in the basic programming language and used the BasicCard API and development environment for implementation. An issuer program will also be programmed in basic to initialise the smart cards. The smart card will be implemented using a software simulator of the Professional BasicCard provided by ZeitControl.

The client program will also be implemented using the OpenSSL cryptographic toolkit in the Visual C++ 6 development environment. The client program will interface with the smart card using a C API provided by the BasicCard development environment. A CA will be implemented using the OpenSSL command-line tools to issue client certificates. The architecture of the final product consists of the following features:

- a CA that issues server certificates;
- an issuer program that issues smart cards;
- a server program handling client connections;
- a client program capable of connecting to the server program; and
- a smart card program capable of storing data and performing cryptographic computations.

6.2.3 Design approach and alternatives

It is necessary to analyse the API's that implement SSL/TLS. There are several choices including OpenSSL, SSLeay, Cryptlib, or network security essentials (NSS) [57]. These choices offer libraries in C and are open-source implementations.

6.2.3.1 Client Authentication

Smart card is used for authentication; the choice of card and the authentication mechanism used is of utmost importance. Three possibilities are discussed below that can be used for authentication. The smart card used will be the Professional BasicCard. The Professional BasicCard has a C API and can implement DES and 1024-bit RSA.

6.2.3.2 Public-key encryption

It is important to note that the smart card, no matter which authentication scheme is used, will contain a card identification number and a username that uniquely identifies the user. It is this username that has to be sent along with each command that is transmitted from the client to the server. In this design alternative, the smart card will now contain the public-key certificate of the client, the public-key of the gateway and the private-key of the client. The protocol involves a claimant demonstrating knowledge of the private-key in one of two ways: - the claimant (client) decrypts a challenge encrypted under his/her public-key or the claimant digitally signs a challenge; - the ITU (International Telecommunications Union) X.509 two and three-way strong authentication protocols specify identification techniques based on digital signatures and, respectively, timestamps or random number challenges.

The ISO 9798-3 protocol is a one-way challenge response protocol using public-keys, as can be seen in Figure 6.2. The server computes a random number r_B with a secure random number generator which it then sends to the client. The client uses its private-key to encrypt the random value and, adding some identification data, sends the packet back to the server. The server uses the public-key of the client, which means that the server needs a secure way to obtain the client public-key as well, to decrypt the data packet and to

compare the decrypted random value to the generated random value. If these two values are the same, the client is authenticated, because the client has demonstrated knowledge of a private-key corresponding to the public-key for that client stored at the server.

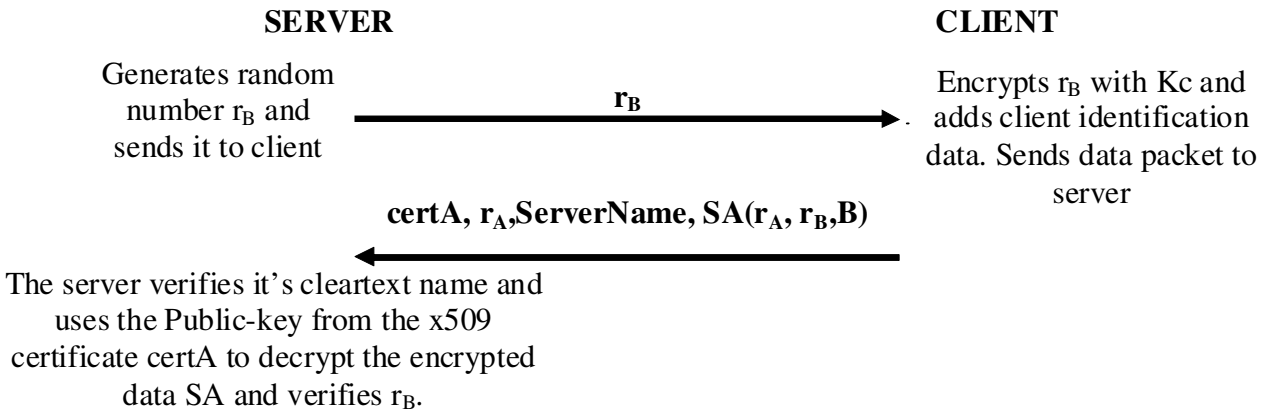

Figure 6.2 Example of an RSA public-key client authentication protocol (ISO 9798-3) [1].

6.2.3.3 Symmetric-key encryption

Both the user, and the server, through the smart card, knows the secret value to be used by symmetric key encryption. The authentication first proceeds, as a normal SSL handshake connection without client authentication, there is only server authentication. After the initial SSL handshake, an extra authentication mechanism is added that is not part of the SSL v3.0 or the TLS v1.0 specification. The authentication now proceeds using a two-way authentication structure, as discussed by Verschuren [32], where a random value is sent to the client and the value is then encrypted by the smart card using the secret key on the smart card. This encrypted value is sent to the server and verified against the known secret value of the server. Note that a username will also need to be exchanged between client and server and that each secret value will be different for each user and smart card pair. It is important to mention that the secret value never leaves the card, and thus the implementation can be considered to be more secure than a normal password implementation. Figure 6.3 is an example of this type of authentication protocol. The *Client Authentication* unit will be responsible for handling the client authentication.

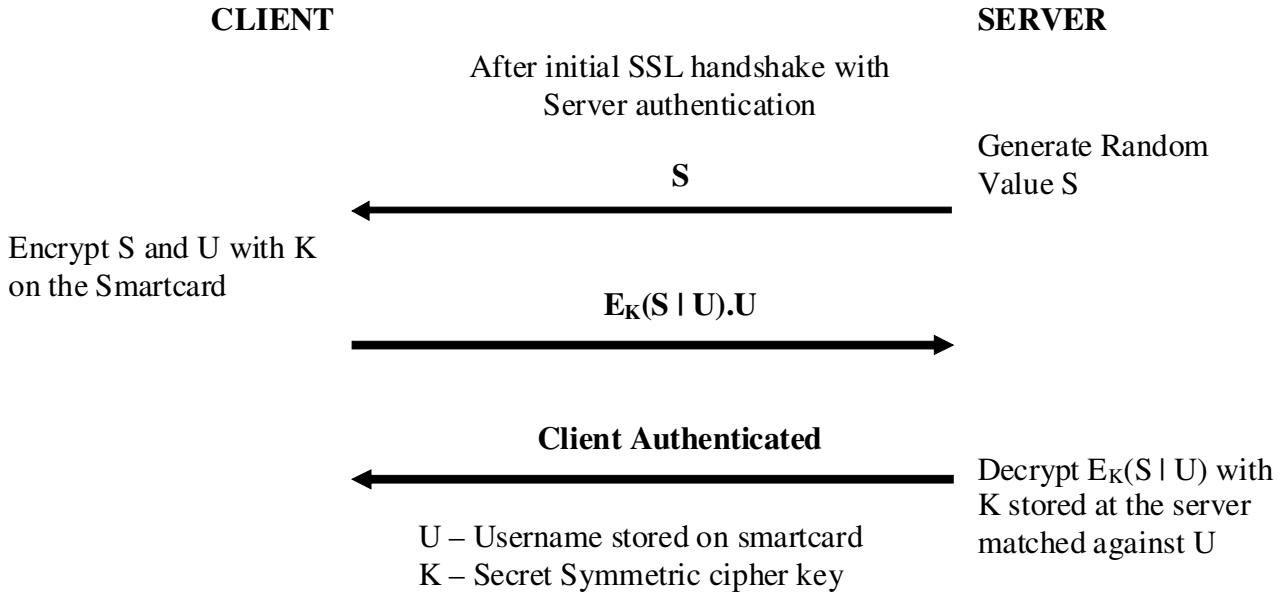

Figure 6.3 Example of a symmetric authentication protocol [1].

6.2.3.4 Username-password

This alternative design for user authentication is the most vulnerable to attack. A password and username are stored on a smart card. As with the previous implementation, after the initial SSL handshake procedure has finished, the Connection Control first asks the user for his/her username and password pair. The client implementation program asks the smart card for this information and sends it to the server.

6.2.3.5 User authentication to smart card

The user will need to authenticate him/her to the smart card, using a PIN, in order to access and use the processor and functions of the smart card. This will only be implemented on the client side of the connection.

6.2.4 Implementation choices

The most versatile SSL API appears to be the OpenSSL software library, and is used to provide a secure SSL communication channel between the client and the server. The server program will be programmed in ANSI C (American National Standards Institute) using the gcc compiler on a Linux system. The client will be programmed in C on a Windows system using the Visual C++ 6 development environment. As stated already, the smart card used will be the Professional BasicCard. The smart card system will be simulated and tested using a software simulator (debugger) of the smart card. The CA will be

implemented using the OpenSSL command-line tools to issue certificates and the BasicCard basic compiler will be used to implement a terminal issuer program to initialise and issue smart cards. Figure 6.4 and Figure 6.5 displays the conceptual flow diagram of server and client application.

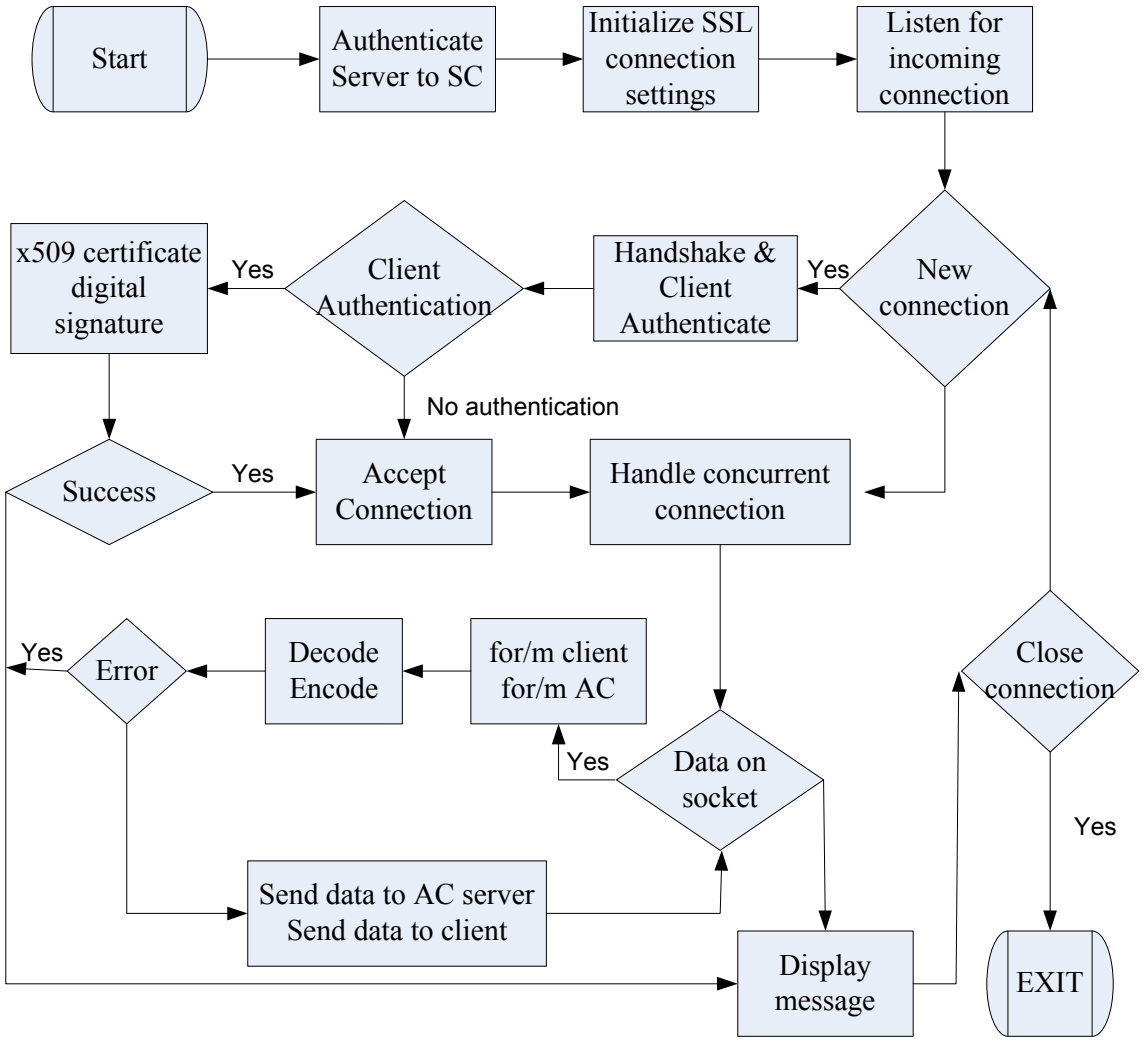

Figure 6.4 The software flow diagram of the Server application.

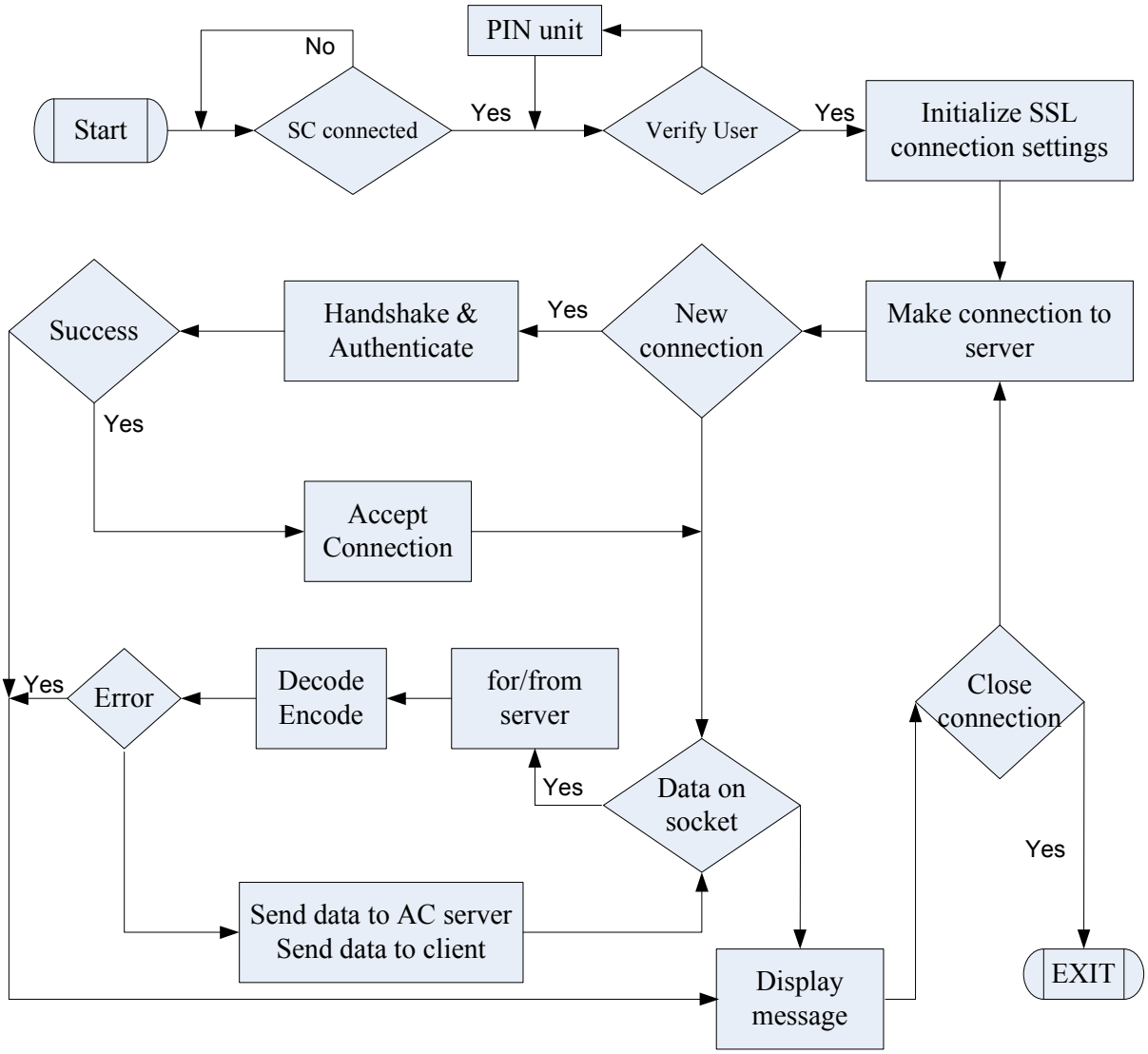

Figure 6.5 The flow diagram for the Client Application Software.

6.2.4.1 Handshake and client authentication

This unit will implement three different types of authentication methods: -no authentication; -client authentication with x509 certificates; -or client authentication using digital signatures Figure 6.2. For the client program the handshake and authentication unit will also need to implement user authentication to the smart card, by using a four digit PIN that the user will need to know in order to activate certain functions on the card.

6.2.4.2 Client authentication protocol

Public-key cryptography was chosen to implement the client authentication protocol. Symmetric-key cryptography relies on both the client and server having to share some sort of secret information before authentication can take place. It also relies on the server to store the client secret information securely in some sort of database. Here the process of registering new clients means that new client information will need to be entered for each new client in the database itself. Certain secret symmetric-keys could possibly be given to a specific group of users. The disadvantage is that, if even one of those users is compromised, all users will be compromised. The problem of users that are compromised and have to be removed has not been solved, however. This is because the system still needs a secure means for specifying which user's smart cards are no longer valid. The implementation does not take into account of compromised users and has effectively shifted the responsibility for this to the access control server. Username-password authentication is not secure enough for the system.

6.3 SSL-SC IMPLEMENTATION

The prescribed choices in earlier section are implemented as follows.

6.3.1 Certification Authority

These conditions were enforced in order to satisfy the requirements of the CA:

- an in-house CA with a self-signed X.509 root certificate;
- issue server certificates signed by the root certificate;
- issue client certificates signed by the root certificate;
- issue RSA private and public-key pairs to clients for use in the client authentication protocol; and
- digitally sign the client public-key along with a MAC value of the public-key to create an authentication string.

6.3.2 Issuing client and server certificates

Using the OpenSSL command-line tools the CA creates a self signed certificate and issues client and server certificates signed with the private-key of the CA.

6.3.3 Issuing client smart cards

The issuing of smart cards to clients occurs in two distinct phases. First, the CA generates the RSA keys that the client will use to authenticate him/herself. The second phase involves the issuer program that issues and personalises client smart cards.

6.3.4 Client Authentication Protocol

The security of the client authentication protocol, Figure 6.6, will determine the security of the client authentication process. This protocol will be implemented on top of a number of other protocols. The TCP/IP protocol will ensure the reliable delivery of data in the correct sequence. The SSLv3 or TLSv1 protocol will handle server authentication, and will establish a secure channel for client authentication to take place. It should be noted that all information sent over this channel, after the initial SSLv3 or TLSv1 handshake, will be encrypted and its integrity will be checked. The implementation will use RSA public-key encryption with 1024-bit keys for the client authentication protocol.

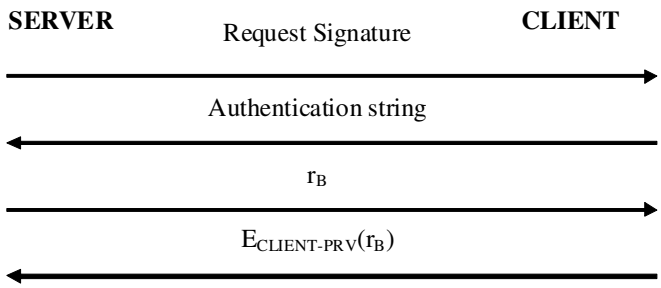

Figure 6.6 Client authentication protocol.

The client authentication protocol flow is as follows.

- Establish an in-house CA. Generate a 2048-bit RSA private public-key pair for the CA. The public-key will be saved on each of the Gateway servers. The security of the CA private-key will be guaranteed by keeping it in a secure location.
- An in-house CA, for each new user with a valid user ID, generates a 1024-bit RSA private and public-key pair.

- Concatenate the user ID, the user public-key and the SHA-1 hash of the ID and public-key to create the authentication string (Figure 6.7).

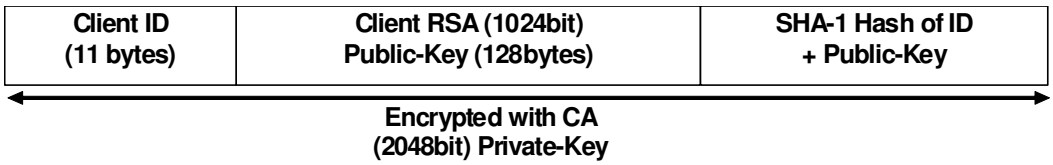

Figure 6.7 Client Authentication String.

- The authentication string is then RSA encrypted with the CA private-key.
- The authentication string is placed, along with generated client's private-key, on a smart card by the issuer program. The purpose of the authentication string is to bind a specific user ID to a specific public-key. It is important to note that the user does not need to know its user ID.
- The server will challenge each connecting client for its authentication string.
- The client's reply will be regarded as the authentication string. The string is decrypted with the CA public-key that is housed on the server. Except for the server certificate this is the only information that needs to be stored at each server. It is important that the CA public-key is not changed or compromised on the server. Thus, there needs to be some sort of physical security on the server to protect the CA public-key. The decrypted hash of the authentication string is then compared to a new computed SHA-1 hash of the decrypted public-key and user ID. This verifies that the contents of the authentication string were not changed on the client computer. It also verifies that the client is a valid client who was issued with a user ID and public-key by the in-house CA, and also that the authentication string received from the client is indeed an authentication string that was encrypted using the private-key of the in-house CA. If the CA public-key were ever to be compromised then another authentication string could be created with a different private-key that would be accepted as valid.
- A 128-byte random value is now generated on the server side using a secure pseudo random number generator and sent to the client.

- The user now authenticates itself to the smart card using a unique 4-digit PIN number. This validates the client program to use the private-key stored on the smart card. The random value received from the server is sent to the smart card and RSA encrypted with the private-key on the card. The 128-byte encrypted response is read from the card and sent to the server.
- The server now decrypts the encrypted response from the client with the public-key from the authentication string and matches the decrypted value against the random value that it generated. If the values are the same, then the server program is sure that the client is indeed in possession of the private-key corresponding to the public-key included in the authentication string that was encrypted with a 256-byte CA private-key that has a corresponding CA public-key stored on the server. The client is now authenticated and a connection is established with the AC server.

6.3.5 SSL implementation

6.3.5.1 Server application implementation

The server program was written in ANSI C using the GNU gcc compiler on the Linux system running Mandrake 8.2. The server application begins by the smart card verification, and is auto authenticated. It then loads the GUI. The SSL server flow chart diagram of Figure 6.4 is implemented. This is where the server operation, security and initialisation, can be launched. A list of some of the functions implemented appears below: Server functions: - operational, - security, - initialisation, - monitor thread, and communication function; - Client Thread Function; - Post-connection Check Function; - Smartcard Authentication Function; - Error Wrapper Function; - Handshake AC Function.

6.3.5.2 SSL client application implementation

The client program was written in the C programming language and uses the native Windows application-programming interface (API). The development environment used is Visual C++ 6. The details of the SSL client application flow can be established from Figure 6.5. The functions implemented for client are: - Message Handler; - Asynchronous Communication Function; - Connect; - Send Button; - Decode Message Function; - Authentication Request Function; - Random Value Response Function; and - Error Handling.

6.3.5.3 SSL Client GUI

The implemented GUI, Figure 6.8, displays graphical data to the user and allows the user to interface with the program.

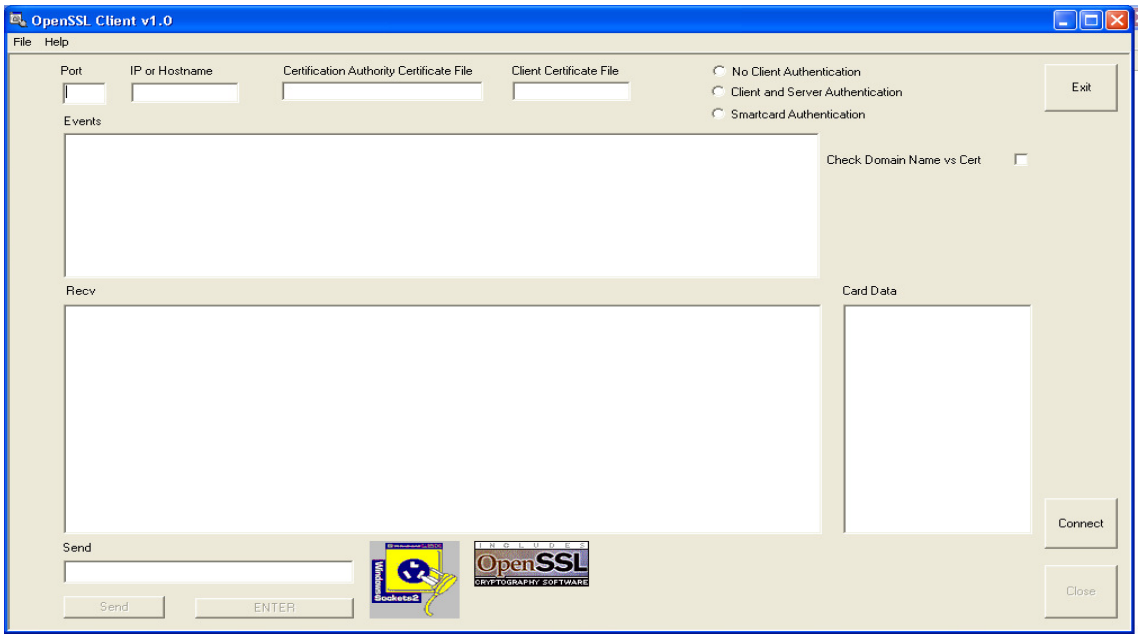

Figure 6.8 Screenshot of the client GUI.

With the GUI the client can set the IP and Port of the SSL/TLS server as well as the trusted root CA Certificate and the Client Certificate. The client can also choose the type of authentication mechanism to use: SC Authentication, Certificate Authentication or No Client Authentication. There are three list boxes on the GUI: the Events list box displays events and errors to the user; the Receive list box displays received data from the server; and the Card Data list box displays user data read from the smart card.

6.3.6 Issuing client smart card

The issuing of smart cards to clients occurs in two distinct phases. Firstly, the CA generates the RSA keys, which the client will use to authenticate itself. The second phase involves the issuer program that issues and personalises client smart cards.

6.3.6.1 Smart card Issuer Program Implementation

The smart card used in the implementation is a software simulator of the Professional BasicCard ZC4.5D. The technical specifications of the Professional BasicCard ZC4.5D can be seen in Table 6.1. The BasicCard development environment supports the use of a virtual card reader. This virtual card reader interfaces with the running virtual BasicCard and allows the smart card program to function as a physical smart card [29].

Table 6.1: Functional specification of the Professional BasicCard [29].

Version	PK Algorithm	EEPROM	RAM	Protocol	Encryption	MAC	File system
ZC4.5D	RSA	30K	1K	T=0;T=1	DES	SHA-1	Yes

The issuer program is a terminal program that acts as the smart card issuing authority. The program uses the private and public-key pairs as generated by the CA, and a username and PIN to personalise a smart card. The issuer program was programmed in the basic programming language using the BasicCard IDE. The execution of the issuer program proceeds accordingly:

- Select the user for which the smart card will be issued.
- Read the authentication string from a file.
- Read the private-key from a file.
- DES encrypts the connection between the client program and the smart card.
- Verify the issuer master key.
- Set the username.
- Set the user PIN.
- Set the authentication string on the card.
- Set the private-key on the card.
- Verify the encryption on the card and exit.

After the issuing authority has finished issuing a smart card, the card is personalised and can be used for client authentication to the TLS server

6.3.6.2 Smart card Security States

There are three security states defined on the smart card. These are listed below.

- ***Master state:*** the connection between the client program and the smart card is DES encrypted and the client program must pose the Master key for authentication. The Master key is a six-digit key that is hard coded into the smart card program.
- ***Client state:*** the connection between the client program and the smart card is DES encrypted and the client program must presume the user PIN for authentication. A program that can pose the Master key for authentication can change the user PIN.
- ***User state:*** the connection between the client program and the smart card must be DES encrypted, but no authentication is required.

6.3.6.3 SC Functions

All the smart card functions are programmed using the basic programming language and the BasicCard IDE. In order to change between different security states, an authentication key is required as shown in Figure 6.9.

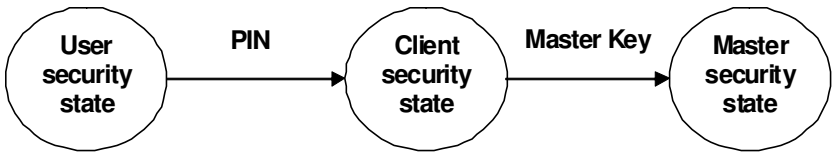

Figure 6.9 Authentication required moving between different security states of the smart card program.

6.4 CHAPTER SUMMARY

This chapter presents the detailed design approach adopted to achieve the model set out in CHAPTER 5. The SSL-SC model was successfully implemented, where SSL was used to provide the channel security based on the SSL protocol and smart cards were integrated to provide the client and server authentication using the pin. Moreover, smart cards in this model were mainly used to store the keys and certificates. The next chapter will present a similar model implementation where the full cryptographic potential of smart cards is used for actual encryption.

CHAPTER 7

IMPLEMENTATION USING SSH-SC

7.1 INTRODUCTION

This chapter presents the implementation of the security protocol proposed in CHAPTER 5. In this design implementation, a secure-shell protocol is used to provide the channel security and smart cards will further be used to provide the cryptographic goals, more specifically, encryption and key issues. The first section presents the functional analysis of the model and presents the design approach and implementation choices available. The second section presents the actual implemented model using a secure shell and smart cards.

7.2 ANALYSING SSH-SC MODEL

In this design approach SSH is to provide a secure communication link between a client and the server application. The smart card will be used for: data encryption/decryption; storage of secret session keys; client authentication to the server; client and server PIN verification to their smart card. Figure 7.1 shows the modular break down of the system that is implemented.

GUI is where the user interacts with the system. It receives data inputs from user and displays received data from server as well as other session information. The system can also be configured using this function. The client is authenticated to smart card using *Authenticate Client to SC. Secure Connection* is responsible for creating, managing and closing a secure connection between a client program and the server program. This unit also performs encryption and decryption of data over the secure channel. All the errors that occur during the transmission of data are managed by *Error Control. Key Storage, Cryptography* should first generate and store the keys on the smart card. Thereafter, this unit will be called upon to encrypt outgoing data and to decrypt incoming data. It will auto authenticates server to its smart card. *Client Authentication* authenticates the client to the server using client's smart card. *Message Generate & Confirmation* generates and sends a confirmation message acknowledging the received data, or any type of error.

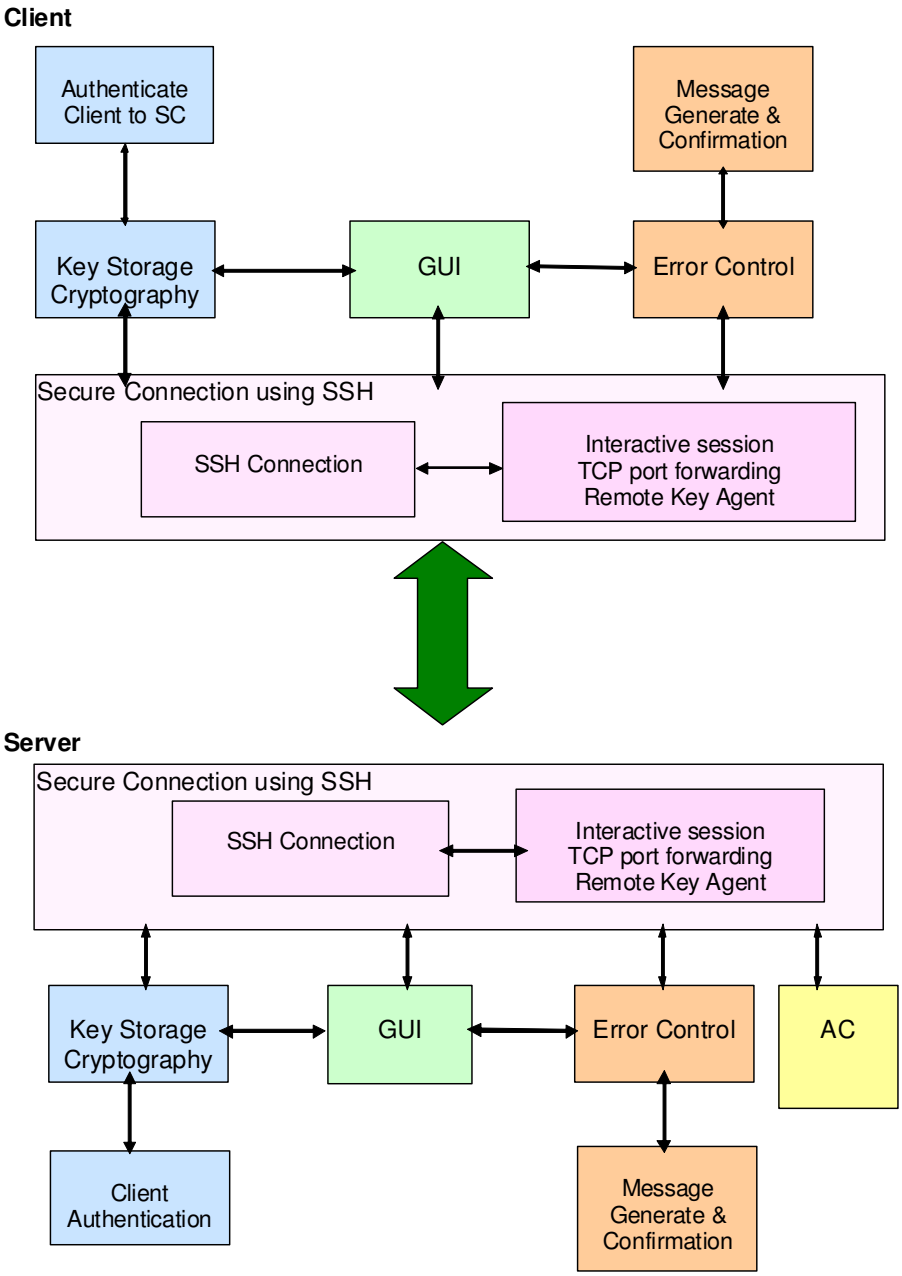

Figure 7.1 Functional diagram of the proposed system using SSH and SC.

7.2.1 Model requirement

Communication channel will be TCP/IP stream sockets. On top of this, there must be a SSH connection established to secure the channel. A symmetric block cipher, key length of 128-bits, will be used between the client and the server for encryption and decryption. A once off secret key of 128-bit is generated and will be stored on smart card. Smart card authentication for client is by manual entry of four digits PIN, and can be automated for server authentication. Client authentication by server must use 128bit symmetrical cryptography on the smart card for client authentication to the server. Other requirements are those already mentioned in CHAPTER 5.

7.2.2 Design tools and choices

Client and server application were developed in Borland JBuilder 9 Personal Edition IDE (Integrated Development Environment) for Windows [58]. The J2SSH (Java Secure Shell toolkit) toolkit [54] was integrated. This allowed the SSH connection. OpenCard 1.2 [38] was used for communications between the Card Acceptance Device (CAD) and the application. The java source code was compiled with Java 2 SDK (Software Development Kit) 1.4.1. The Java Card applet are designed using Sm@rtCafé Professional Personal Edition V2.0b [59], IDE was used to develop the applets residing on the Java Cards. The applet source code was compiled with Java 2 SDK 1.3.1, using libraries from Java Card Toolkit 2.1 [28]. Giesecke & Devrient Sm@rtCafé Test Card. This is a Java multi-application open platform smart card, allowing programming with Java in an easy and secure way and dynamic downloading of applets after card issue. The smart card reader used will be TOWITOKO CHIPDRIVE micro 100. The main features of the Java Card include:

- implementation according to the Java Card 2.1.1 Standard;
- convenient object-orientated programming in high-level language Java;
- dynamic downloading of applets;
- improved virtual machine security model;
- security space management and encapsulated applets;
- support for transport protocols T=0 or T=1 via Protocol Type Selection;
- card individual, without the ability to seed, random number generator;
- scalable crypto API with DES/DES3 encryption;
- hash algorithm SHA-1; and
- Open Platform Card Specification, version 2.0.1.

7.2.3 Design approach and alternatives

SSH can be implemented using, OpenSSH, J2SSH or Cryptlib software libraries to implement the protocol, since they offer open source libraries in C and Java. The following smart card choices are available: Basic cards, Starcos cards or Sm@rtCafé Java Cards.

7.2.4 Client authentication

Smart card based authentication is achieved by: -public-key authentication; -symmetric-key authentication; or -username-password pair authentication. These are already presented in CHAPTER 5, section 6.2.3.

7.2.5 Implementation choices

The J2SSH SSH toolkit was chosen because it is freely available open source SSH toolkit. This implies that both the client and server will also be Java applications, implementing Java 2 SDK 1.4.1. With SSH, client authentication, server authentication, encryption and message authentication will be implemented. The ciphers used for symmetric encryption are 3DES and Blowfish.

Sm@rtCafé Java Cards are selected and they runs Java Card toolkit 2.1. Java 2 SDK 1.3.1 will be used for compiling the applet source code. Java 2 SDK 1.3.1 and SDK 1.4.1 are compatible, but, because there is no direct interaction between the application and applet code, there are no compatibility issues. Public-key cryptography would have been used for the Java Card based user authentication, but, because the available Java Cards can not handle RSA, symmetric encryption was used on the Java Card for user authentication to the system. The TOWITOKO CHIPDRIVE micro 100 smart card readers were chosen because it is supported by the available CAD Java libraries. Figure 7.2, and Figure 7.4 shows the details of the conceptual design software flow diagrams.

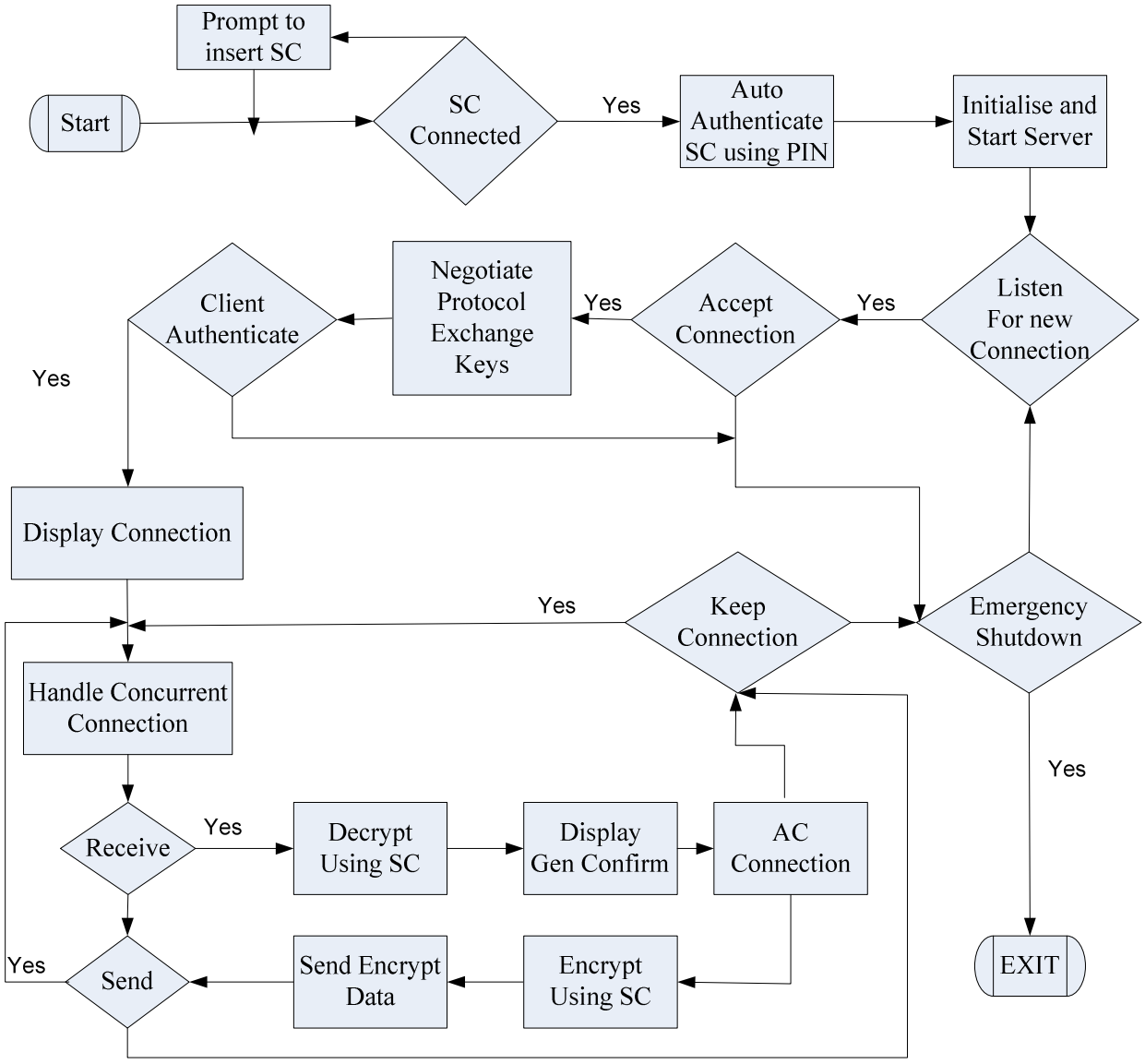

Figure 7.2 The conceptual software flow diagram for the server application.

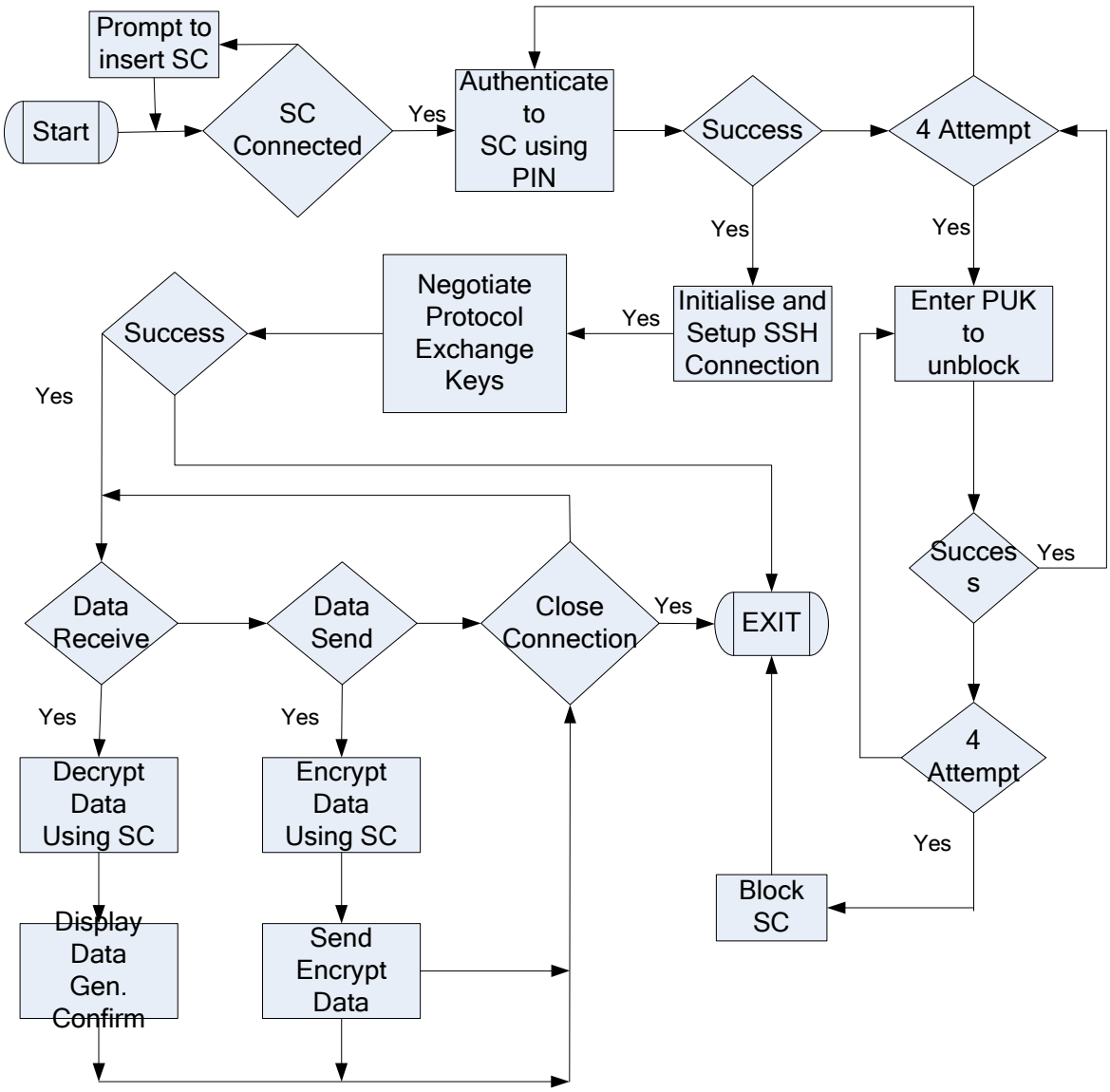

Figure 7.3 The conceptual software flow diagram for the client application.

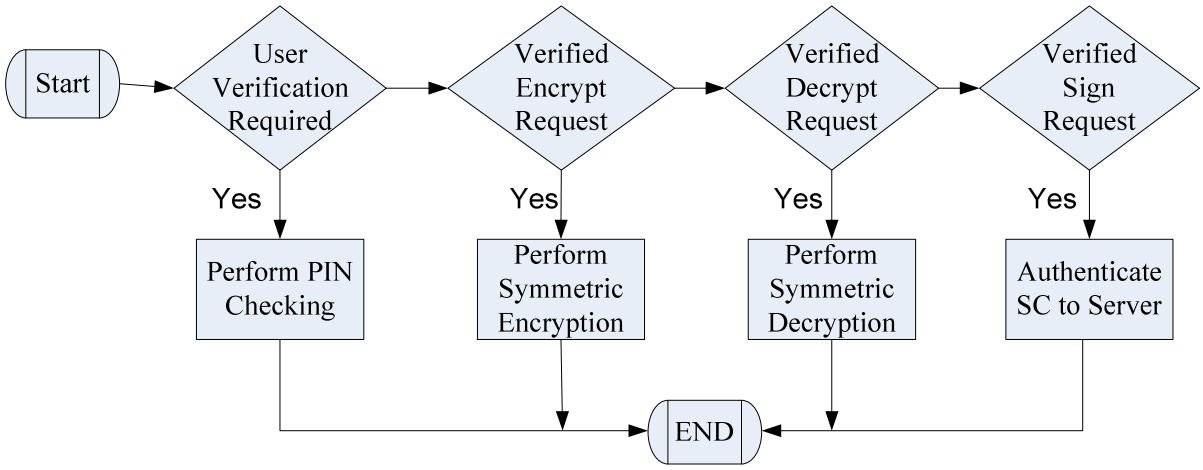

Figure 7.4 Software flow diagram of the Java Card software.

The Java Card will perform these tasks when called by other parts of the system.

7.3 SSH-SC IMPLEMENTATION

7.3.1 SSH server

The server creates the channel and is handled at the client end, so that there is something to be requested by the client. To create the channel, a channel factory class had to be coded. It returns a channel object. The available functions that the channel object has are specified in another class – *sshChannel*, which were developed. Figure 7.5 demonstrates how the communication channel is established.

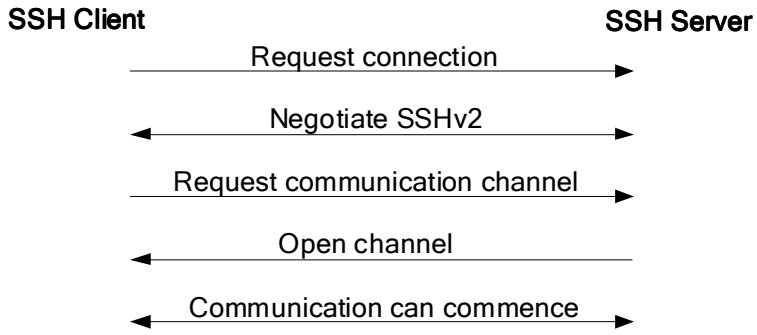

Figure 7.5 Establishment of a channel for allowing SSHv2 secured streaming sockets.

The Server application starts by server verification to its smart card. The server only proceeds forward if a suitable smart card is present. If not, the operator will be prompted continually to insert one, until one is inserted. The server is then verified to its smart card through an automatically entered 4-digit PIN code. If verification through this hard coded PIN fails, it is assumed that the smart card is a fake and the application is halted immediately.

7.3.2 SSH Server GUI

The operator can start the SSH server by pressing the **Start SSH Server** button, Figure 7.6. It then creates an SSH object that will handle the SSH connections. The server will then listen for connections coming from SSH client applications. All information regarding any accepted connections is displayed in the Current SSH Connection Information text area. All data received is displayed in the SSH secured data received text area. Also displayed are the number of clients currently connected to the SSH server and the status of SC. Verbose mode can be selected in the Display menu. This will display encrypted data received, the decrypted version thereof, as well as the encrypted data sent back to the client.

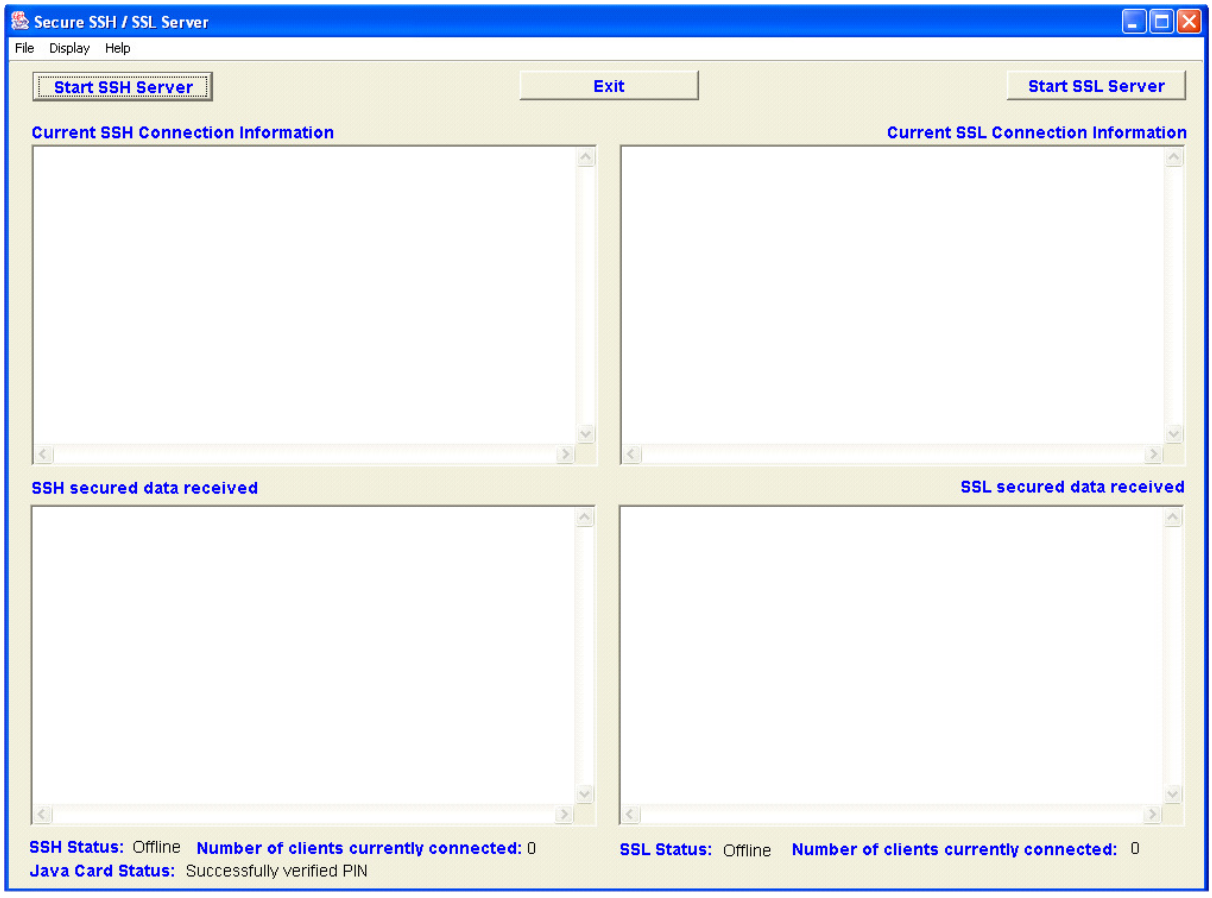

Figure 7.6 Secure shell Server GUI

7.3.3 SSH client

The toolkit provides an SSH client object that can be used to connect and perform authentication with a SSH server. To enable the system to use SSH to protect a communication channel, a channel object had to be created. Creating such a channel is complex. Seen from the client side, a channel client class had to be developed to extend an abstract class – *IOChannel* with functions that can be modified to suit the needs. This enabled the client SSH object to handle SSH secured stream sockets.

The process of setting up an SSHv2 secured communication channel is described below.

- Configure the encryption algorithms for use from client to server and from server to client.
- Configure the hash algorithms used in both directions.
- Use this properties object (containing the configuration) and use this as parameter for establishing a SSH connection and subsequently returning a handle on the connection by means of an object.
- Use the toolkit's password authentication methods.
- Instantiate the channel class.
- Use the channel object as a parameter for the *openChannel* method (of SSH connection object) supplied by the toolkit.

The client application can only be run when a smart card is inserted. If a suitable smart card is found, the user is prompted to enter an IP address, port number, PIN number and to indicate whether or not database support is needed. The user is verified to its smart card only if the correct PIN code is supplied within a maximum of four attempts. If not, the card is temporarily blocked and a PUK (Personal Unblocking Key) will have to be entered. If this can't be done within four attempts, the smart card will be irreparably blocked. However, if the correct PUK is entered, the user will still have to enter the PIN code correctly within four tries, before access is granted and the user is successfully verified to his/her card. Addendum A4 shows the detailed explanation.

7.3.4 SSH Client GUI

The user's smart card must be connected at all times for the client application to work. As with the server, the GUI forms the core of the client application. Since this is where all events take place, this was the preferred solution. It also makes it easier to display results and other information, such as the connection status, smart card status, and client authentication results. Figure 7.7 shows the screen shot of the client GUI.

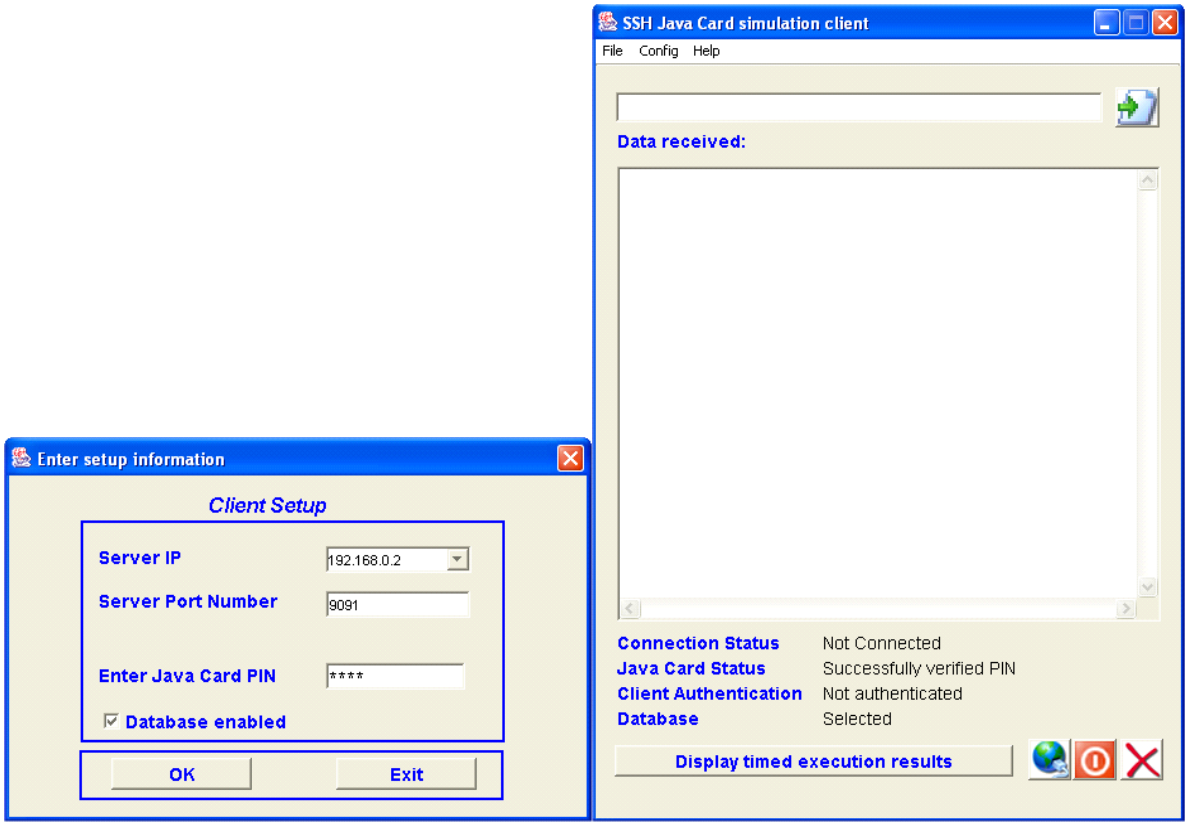

Figure 7.7 SSH Client application GUI.

7.3.5 SSH connection

The J2SSH toolkit is an effort to develop a robust, open source SSH Java toolkit implementing SSHv2. The toolkit, however, does not supply a stream socket implementation. The toolkit is normally used by developers in applications that securely transmit commands for shell based programs running on already designed SSH servers - like the ones found under Linux. Thus, this had to be developed. The toolkit does, however, supply numerous abstract classes, and this was explored to fit the required need.

7.3.6 SSH Dataflow between Client and Server

After the establishment of a secure communication channel and the successful completion of the smart card based user authentication, the client must still inform the server whether database support will be needed. The SSH transmissions between the client and the server are done in bytes, not strings. This is preferred, because the encrypted bytes the smart card returns do not have to be converted to a string. This resulted in some problems, however, when sending data from the server to the client – the channel created for communications has to know the length of the buffer for the incoming data the system has to handle. To bypass this, the protocol indicated in Figure 7.8 was developed.

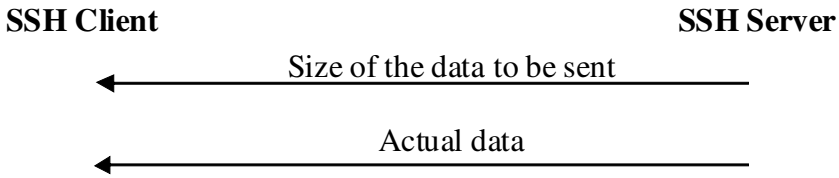

Figure 7.8 Protocol for sending data from server to client

7.3.7 SC based authentication

SSHv2 can and does provide server and client authentication. This is performed by public-key authentication in the case of server authentication and through password-based authentication in the case of the client. This, however, only provides proof to the client application that the server is in fact the server it claims to be and to the server that a legitimate client application is requesting service. But to increase security, the server should also be able to verify the authenticity of the user operating the client application.

To achieve this objective, a smart card based user authentication protocol was developed. After SSHv2 has been negotiated and the key exchange has taken place, the following happens.

- The server generates and sends a 20 bit secure random number to the client.
- The client forwards the random number to its Java Card.
- The Java Card hashes this number with SHA-1, after which it is encrypted with 3DES and sent back to the client.

A digital signature is thus created and this is graphically displayed in Figure 7.9.

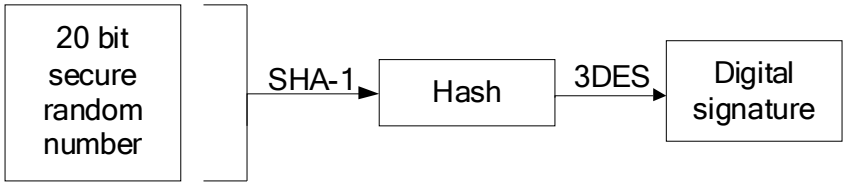

Figure 7.9 Digital signature creation.

This created digital signature is then returned to the server where it is compared to one created in the same manner on the server's own smart card. If they match, the user is authenticated and confirmation thereof is sent to the client. Figure 7.10 explains the protocol data flow between the client and server application.

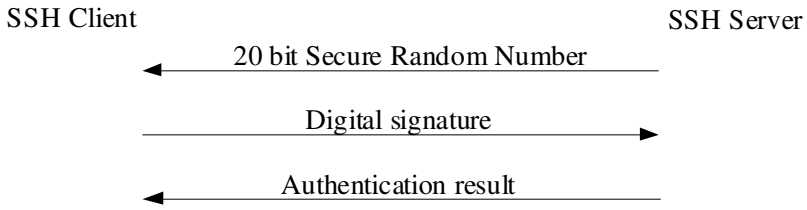

Figure 7.10 SC based user authentication and database indication protocol

7.4 CHAPTER SUMMARY

In this chapter, a detailed secure-shell implementation of the model proposed in CHAPTER 5 is presented, where the SSH protocol secures the channel and then the smart card is used for further data confidentiality and security issues, in particular data encryption and decryption. This scheme is useful where additional data confidentiality is required by the client. The next chapter gives the results generated by the two security protocols that are implemented and provides the necessary results from both the implemented models in CHAPTER 6 and CHAPTER 7.

CHAPTER 8
RESULTS AND DISCUSSION

8.1 INTRODUCTION

This chapter presents the various tests performed on the security models presented in CHAPTER 6 and CHAPTER 7. The observation made during the tests is presented in great detail, as well as a discussion on performance of the models under different conditions under which the server operates and the factors that influence its performance are highlighted. First, the cost of handshake is discussed in detail. The second and third sections present findings of the server under different load conditions and its throughput. Sections 4 and 5 present the connection security and channel confidentiality issues. Lastly, authentication and cryptography operations tests based on the smart card are presented and discussed.

8.2 COST OF HANDSHAKING

A very common complaint from many network administrators is that SSL and SSH is very slow. The primary reason for this is that cryptography, particularly public-key cryptography is extremely CPU-intensive. There are two phases in a connection: the handshake phase and the data transfer phase. The handshake phase only takes place once during a connection and is, compared with the data transfer stage, expensive. The cipher suite used for encryption, authentication and message integrity for each of the connections is made up of 168-bit 3DES, 1024-bit RSA authentication and a SHA-1 MAC.

The cost of the handshake can be significantly improved without client authentication. With no client authentication, only the server is authenticated through its certificate.

If client certificate authentication is opted then server will request the client for its certificate that will be used by the server for authentication; this adds significant time and thus delays the handshake process.

The client authentication protocol exchanges (records 10 to 13 in Figure 8.1) between server and client occur after the initial handshake with no certificate request by the server.

```
New TCP connection #1: b121pc126.up.AC.za(37290) <-> b121pc126.up.AC.za(10002)
1 1  0.0155 (0.0155)  C>S SSLv2 compatible client hello
1 2  0.0177 (0.0021)  S>CV3.1(74)   ServerHello
1 3  0.0177 (0.0000)  S>CV3.1(3482) Certificate
1 4  0.0177 (0.0000)  S>CV3.1(4)    ServerHelloDone
1 5  0.0304 (0.0126)  C>SV3.1(134)  ClientKeyExchange
1 6  0.0304 (0.0000)  C>SV3.1(1)    ChangeCipherSpec
1 7  0.0304 (0.0000)  C>SV3.1(40)   Finished
1 8  0.0519 (0.0214)  S>CV3.1(1)    ChangeCipherSpec
1 9  0.0519 (0.0000)  S>CV3.1(40)   Finished
1 10 0.0921 (0.0402)  S>CV3.1(24)   Smartcard Authentication Request
1 11 0.0932 (0.0010)  C>SV3.1(280)  Authentication String
1 12 0.0952 (0.0020)  S>CV3.1(152)  Random Value
1 13 0.1054 (0.0101)  C>SV3.1(152)  Encrypted Response

FORMAT
Connection No, Record No, Time since connection started, Time since last record,
Direction (Ex. Server > Client), SSL version, Record length, Message Type
```

Figure 8.1 The *ssldump* printout of the SSL handshaking process with client authentication using a smart card and the client authentication protocol

For the server, more than half of the CPU time is spent on the RSA decryption operation with the private key. The RSA decryption operation performed by the server takes place between the time when the client sends the *Finished* message and the time the server replies with the *Change Cipher Spec* message. The client spends most of its time waiting for the server. The most expensive operation the client needs to perform is verifying the server certificate and encrypting the *premaster-secret*. If client authentication is used, the client will perform an RSA private-key operation equivalent to the operation performed by the server. This RSA operation will therefore become the dominant performance cost operation that the client needs to perform. Using client authentication with certificates adds about 0.0060 (0.0142-0.0083) seconds' additional time to the authentication process. This time addition only indicates the delay with one client request. Authenticating several concurrent client connections will entail a significantly great authentication delay.

Table 8.1 Digital Signature performance [1].

Algorithm	Key size	Signs/s	Verifies/s
RSA	512	342	3287
DSA	512	331	273
RSA	1024	62	1078
DSA	1024	112	94
RSA	2048	10	320
DSA	2048	34	27

As can be seen from Table 8.1, the performance of public-key algorithms declines with key size. An RSA operation with a 1024-bit key is almost four to six times slower than an RSA with a 512-bit key size. Therefore, using RSA with a key size of 512 bits will decrease the time needed for handshaking at a cost of a weaker key size. The results of client authentication with a smart card provide an indication of the minimum time that would be required for handshaking and authentication when using the client authentication protocol. The result does not take network latency or the smart card interface with the client computer into account. The client authentication protocol used with a smart card almost doubles the time needed. The server program takes almost 40 ms to set the connection information, create a thread to handle the new client and decrypt the authentication string. The server spends another 2 ms to generate a random value. The client spends 10,2 ms to RSA private encrypt the random value. The handshake time is doubled because the server now has two extra RSA operations to perform. One operation is to decrypt the authentication string from the client with its CA public key and the other is to decrypt the encrypted response from the client with the public key from the authentication string. The client performance is now dominated by the RSA private-key encryption operation performed on the smart card.

Nagle's algorithm [60] is intended to reduce small TCP packets. This algorithm requires the network interface to not immediately send out TCP data, but buffers it and only sends the data when the algorithm times out or receives an ACK. The Berkley socket option TCP_NODELAY is used to disable the algorithm. Results with a disabled Nagle algorithm have reduced the time needed for handshaking by an average of 10,1 ms for ten simulated clients.

Too much time is needed to perform client authentication using a smart card. If a server were ever bombarded with a large number of clients, the handshaking and authentication would create a huge bottleneck and limit the server performance in terms of throughput and response time. There are several ways to improve performance. The first method would be to use smaller keys, but this would limit the security of the authentication protocol. The second system should always disable Nagle's algorithm to increase throughput and prevent temporary deadlock between the server and the client. The third method to increase performance is to decrease the time needed to perform an RSA operation. This can be accomplished by using dedicated RSA-processing hardware or cryptographic accelerators.

8.3 PERFORMANCE OF SERVER UNDER LOAD

SSL/SSH can very easily become a huge part of the processing overhead of an application, especially a server application. The cryptographic primitives used in the protocol can consume a great deal of processing time and resources and the result is that an application running at its SSL/SSH-processing capacity can drag a system to its knees. Servers running SSL/SSH are therefore at the mercy of the demands made on them. These two crucial questions are asked:

- How much traffic can a given server sustain if it is saturated to capacity?
- If a given simulation of "client use" is thrown at the server, how does it behave?

When simulating SSL/SSH client(s) a swamp simulator [61] may be used so that the speed, capacity, throughput and behaviour of the server can be tested. Each run of the swamp program lasts for 60 seconds. The results of the load generator can be seen in Table 8.2. The cipher suite used for generating the results was 3DES in CBC mode, with SHA MAC and RSA as authentication algorithm.

Table 8.2 Server performance result under load conditions.

	Lab Environment (LE)				External Environment (EE)			
No. of concurrent client connections	**Total connections**	**No. of successful connections**	**No. of failed connections**	**% of connections that failed**	**Total connections**	**No. of successful connections**	**No. of failed connections**	**% of connections that failed**
1	1169	1169	0	0%	549	548	1	0.18%
10	1274	1272	2	0.16%	560	551	9	1.61%
20	918	896	22	2.40%	558	535	23	4.12%
30	908	856	52	5.73%	544	497	47	8.64%
40	918	828	90	9.80%	556	487	69	12.41%
50	932	812	120	12.88%	553	463	90	16.27%
60	1299	1118	181	13.93%	596	489	107	17.95%
70	1173	996	177	15.09%	681	563	118	17.33%
80	1174	1002	172	14.65%	701	576	125	17.83%
90	1097	929	168	15.31%	656	536	120	18.29%
100	1202	1022	180	14.98%	696	566	130	18.68%

The results in Table 8.2 are generated by saving the values of the number of connecting clients from the swamp load generator and the number of accepted clients in a file. A connecting client is a client that has not yet been authenticated. It indicates the time taken by the server to accept a client or place the client in a queue.

Table 8.3 Impact of concurrent client connection on the server under load conditions.

Concurrent Clients				
Time (s)	Lab Environment (LE)		External Environment (EE)	
	Queued Clients	Accepted Clients	Queued Clients	Accepted Clients
1	40	40	29	28
2	62	60	42	37
3	149	113	63	58
4	169	125	115	106
5	173	119	135	97
6	180	100	127	71
7	167	20	95	41
8	165	10	59	21
9	152	8	33	14
10	75	8	19	9
11	38	5	7	7
12	5	5	7	7

Data in Table 8.2 and Table 8.3 indicates that as the number of concurrent client connections increases, so the percentage of failed connections also increases. The connection failures are caused by handshaking errors during the client and server handshaking phase. Under heavy load conditions, the cryptographic computations necessary for SSL become a bottleneck to server performance. When the number of client connections is relatively low, the server will serve each request as it comes in. As the concurrent client connections increase in number, the server will have to queue connecting clients while it is still busy handshaking with others (Figure 8.2). Thus the server is being saturated by the SSL handshakes that it is busy performing. The result is that the server is keeping a number of clients in a queue. As the new connections decrease in number, the server is able to serve the clients in its queue. For the laboratory environment, the average connection time for a client is between one and two seconds, and the optimal number of clients that can connect to the server without being queued is below 90 clients. The results

for the external environment (EE) are between two to three seconds, and the optimal number of clients that can connect to the server is lower than 60.

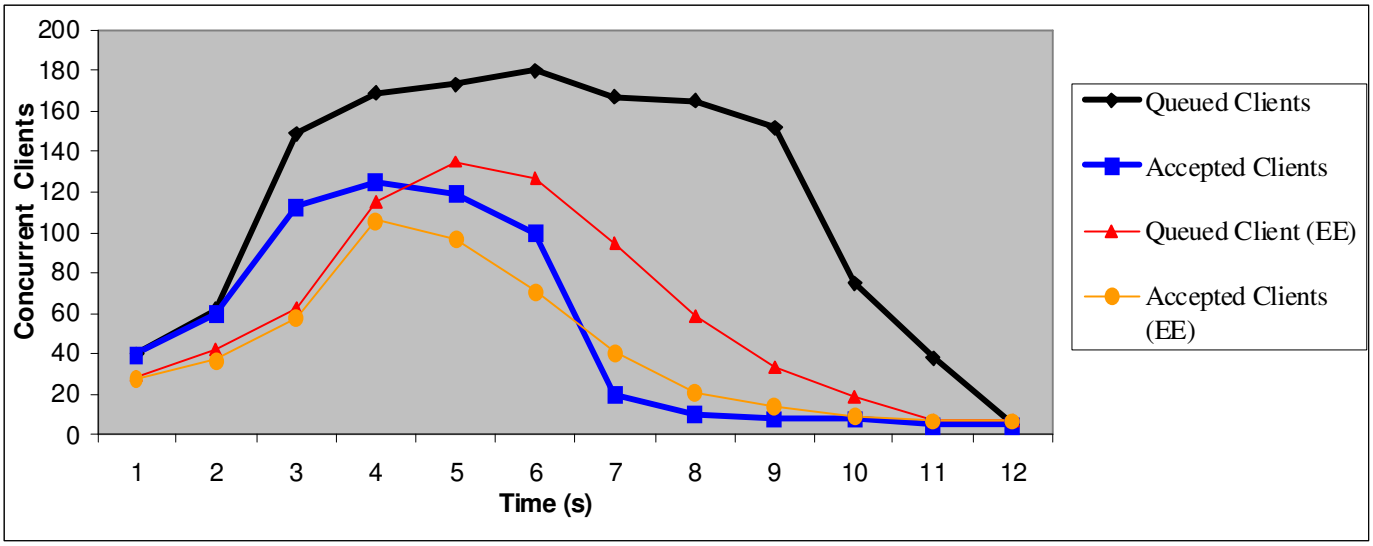

Figure 8.2 Server performance under load.

8.4 THROUGHPUT AND RESPONSE TIME OF SERVER

Latency is the time taken between making a request and seeing a result. Throughput is the number of total transactions that can be maintained over a time. The results of the experiment to determine the throughput and response time for the server using the siege load generator program [62] are set out in Table 8.4. The simulation was repeated ten times for a certain number of concurrent client connections. The average HTML page size sent to the client by the web server was 370 bytes. The average over the ten runs was taken as the final result. Network latency between the client (computer 1 - 137.215.121.126) and the web server (computer 2 – 137.215.121.142) will have an effect on the results. Running the *traceroute* command between computer 1 and computer 2 yields the following latency result:

1. b121pc142.up.ac.za (137.215.121.142-server) 0.962ms 0.617ms 0.156ms

Table 8.4 Throughput and response time results for the server under certain load conditions (TCP maximum segment size = 4096 bytes)

Number of concurrent clients	HTTP Throughput (Bytes/sec.)	HTTP Response Time (m sec.)	SSL Throughput (Bytes/sec.)	SSL Response Time (m sec,)
1	3765.41	5	4637.50	60
10	4457.44	7	5523.32	78
20	6451.85	8	6745.45	95
30	8975.61	9	8772.23	106
40	12517.01	10	9222.21	122
50	13757.01	12	9919.79	139
60	13886.79	15	10357.14	156
70	14553.67	18	9988.46	172
80	14360.98	20	10393.26	185
90	13301.20	22	10701.92	197
100	12390.57	23	9003.48	210

The throughput of the normal HTTP transaction between the client and the web server increases as the numbers of concurrent connected clients increase. The throughput reaches a maximum when there are between 60 and 80 connected clients. The throughput for the SSLv3 connections reaches a maximum value at around 60 concurrent clients. The throughput is initially higher than for the normal HTTP connections, but as soon as more than 20 concurrent clients connect to the TLS implementation server, the throughput remains lower than the HTTP throughput. The response time of the SSLv3 connections is much higher than for the HTTP connections and increases as the number of concurrent client connections increases. This is due to the cryptographic computations that the SSL server needs to perform for each client connection.

In the data transfer phase of an SSL connection, there are two relevant cryptographic operations. These two operations are the record encryption and decryption with a symmetric cipher, and the MAC calculations and verification operation. The choice of a MAC and encryption algorithm affects the performance of the data transfer phase. As can be seen from Table 8.5, RC4 with MD5 is almost eight times faster than 3DES with SHA. The results in Table 8.4 were obtained using 3DES-SHA1.

Table 8.5 SSL record processing speed for a certain symmetric cipher and a certain MAC algorithm [1].

Cipher-MAC algorithm	Kilo Bytes/second
RC4-MD5	15034
RC4-SHA	10831
DES-CBC-SHA	4758
3DES-CBC-SHA	2068

Using the RC4 symmetric algorithm along with the MD5 MAC algorithm can decrease the response times. Using dedicated cryptographic accelerators to perform all cryptographic calculations will also decrease response time.

8.4.1 Connection security verification

The OpenSSL library allows the programmer to set the cipher suite that will be used by the SSL server during handshaking with the client. The cipher suites are listed in Table 8.6.

Table 8.6 Cipher suites supported by the server.

Version	Public-key algorithm	Key-exchange algorithm	Symmetric encryption algorithm	MAC algorithm	Symmetric algorithm key size
TLS v1	RSA	DH	3DES	SHA	168-bits
TLS v1	RSA	RSA	3DES	SHA	168-bits
TLS v1	RSA	RSA	RC4	SHA	128-bits
TLS v1	RSA	RSA	RC4	MD5	128-bits
TLS v1	RSA	RSA	IDEA	SHA	128-bits

The *premaster-secret,* along with the server and client random values, gathered during the handshake phase, is used to compute the various secret keys that are used for the symmetric encryption algorithms. The client authentication will use RSA digital signatures with a key size of 1024-bits. As seen in Table 8.6, the supported symmetric encryption ciphers all have key sizes greater than or equal to 128-bits. The *premaster-secret*, is used to compute the secret key values. These secret key values are used as the key for the 3DES

functions provided by the *OpenSSL* toolkit. The same string value is then encrypted with 3DES and compared to the results gathered with *ssldump*. The results are the same. This means that, for this connection, the client and server did indeed implement the 3DES algorithm using a same length key (168-bit) to encrypt the data. The RSA keys used by the client authentication algorithm with the smart card do indeed use 1024-bit keys.

The server does not allow the use of the anonymous Diffie-Hellman algorithm for key exchange. The anonymous Diffie-Hellman algorithm is vulnerable to man-in-the-middle attack. Preventing the use of the anonymous Diffie-Hellman algorithm eliminates the threat of man-in-the-middle attack. Moreover, the system is not vulnerable to modification attack because of the MAC algorithms computed for all sent data. The addition of the server and client random numbers, in conjunction with the *premaster-secret* used to compute the secret keys for the encryption algorithm, prevents the use of a reply attack against the implementation. Key sizes of 128 bits and larger prevent the use of a brute-force attack. Indeed, some experts estimate that it could take as long as 5.4×10^{24} years to attack an algorithm successfully using a brute-force attack [63].

The greatest threat to the TLS server implementation program is a DOS attack. The *tcpkill* program, provided by the *dsniff* library, is used to kill all TCP connections that try to connect to the server. As stated in Wheeler [64], a buffer overflow is a very serious security flaw in most server implementations. Whenever the server deals with received client data, the length of the data is checked and unsecure functions such as *printf*, *gets*, *sprintf,* etc., are not used to manipulate the data.

8.4.2 Channel confidentiality

This experiment verifies whether the data is transmitted between a client and the server is encrypted, protecting the system against passive attacks such as wire-tapping. Ethereal and WinPcap were installed on the client and the server applications.

```
Frame 80 (154 bytes on wire, 154 bytes captured)
SSH Protocol
    SSH Version 2
        Encrypted Packet: FB092BDC1A3F1FBFCAE0D486BCDCB2D6...

Frame 84 (114 bytes on wire, 114 bytes captured)
SSH Protocol
    SSH Version 2
        Encrypted Packet: CC43681B80407BD9E9DF4BF40607A2D7...
```

Figure 8.3 Confidentiality provided by encrypting transmissions.

The bold italic outputs shown in Figure 8.3 depict the SSH secured data sent. Frame 80 is the packet that was sent from the client to the server for the doubly encrypted request. Frame 84 is the packet containing the doubly encrypted response to that request. It confirms that the request and the response were encrypted before being transmitted in both directions. This assures data confidentiality in the system.

8.4.3 Capturing packets transmitted between client and server

Ethereal is used to capture the packets exchanged between client and server. Texts to be transmitted are typed and then sent to the server. The capturing of packets is ceased and the captured packets are saved in a file for analysis. The SSH protocol is negotiated (the encryption ciphers that are supported are requested and also the message digests that will be used). Then the DH algorithm is used to carry out the key exchange. Hereafter, the application-specific settings and protocol can be negotiated – protected by SSH.

8.5 TESTS ON SMART CARD

Three types of test are performed on the smart card. The aim and objectives of these tests are set out in CHAPTER 3. The first test is the PIN verification on the smart card, the second test achieves smart card authentication and the last test is to show that the actual cryptography is performed by the smart card.

8.5.1 SC PIN verification

In order to verify the smart card PIN, the smart card Test Suite application is initiated and a wrong PIN is typed. This is repeated four times. Enter the wrong PUK and click on the Send PIN button. Enter the correct PIN and click on the Send PIN button. An attempt is made to create a digital signature and this is rejected with a request to enter the correct PIN.

```
PIN entered: 12345
Status: PIN verification Failed. Please try again : 3 tries remaining
PIN entered: 123
Status: PIN verification Failed. Please try again : 2 tries remaining
PIN entered: 123
Status: PIN verification Failed. Please try again : 1 tries remaining
PIN entered: 45646749687
Status: Card blocked – Enter PUK : 4 tries remaining
PUK entered: 012345678
Status: Card blocked – Enter PUK : 3 tries remaining
PUK entered: 0123456789
Status: Card Unblocked – Please enter PIN code
Try encrypting or decrypting data
Status: First enter PIN
Try creating digital signature
Status: First enter PIN!! – No match
PIN entered: 1234
Status: Successfully verified PIN
```

Figure 8.4 PIN verification results

The results, Figure 8.4, indicate what happens when a partially correct, shorter and longer PIN is entered. It also shows what happens when a totally different PIN is entered. A PUK is needed to unblock the card. No functionality was available before successful verification. The results indicate that a smart card can only perform encryption, decryption, and digital signature creation after successful verification. Figure 8.4 also indicates that after four wrong attempts at guessing the PIN, the smart card is blocked. It can, however, be unblocked by entering the correct PUK within four tries. The user is, however, not

verified for the smart card when the correct PUK is entered; the user must still enter the correct PIN number before the smart card can provide any functions.

8.5.2 Authenticating a smart card

This test ensures the user authentication for the smart card and this is accomplished by starting the Java Card Test Suite application. When prompted for the smart card, it is inserted and user the PIN/PUK verification is completed. It uses a 20-bit secure random number resulting in a different digital signature each and every time. The result is shown in Figure 8.5.

```
PC generated digital signature
     05 CA F1 0D AB F5 cj. 1F CP DF F3 C9 0wL2 D4 D8 DE A0 BE

Java Card generated digital signature
     05 CA F1 0D AB F5 cj. 1F CP DF F3 C9 0wL2 D4 D8 DE A0 BE
```

Figure 8.5 Signature results.

The first hexadecimal byte stream in Figure 8.5 is the digital signature created on the PC. The second hexadecimal byte stream is the digital signature created by the smart card. These digital signatures are created in the same way by first obtaining the SHA-1 hash of the 20-bit secure random number followed by encrypting the hash with 3DES. The experiment confirms that a simulated card creating a digital signature matches a digital signature created on the smart card. This means that the smart card is capable of creating the correct digital signature when a 20-bit secure random number is received. Thus, in the case of a smart-card-based user authentication, the client smart card will be able to generate a digital signature and the server smart card will also be able to generate a digital signature. If the two digital signatures match, it is known that these two are using the same key for the encryption part and the same 20-bit secure random number is used.

8.5.3 Smart card encryption

The results of this experiment verify whether the smart card does encrypt the data sent to it and returns the encrypted data to the calling application. These experimental parameters were used:

Text to be encrypted: Hallo World
Expected encrypted text: 5 D4 D1 9A Rj F7 AB Iq 27 86 07 w 26 06

```
No interference
Encrypted text: 5 D4 D1 9A Rj F7 AB Iq 27 86 07 w 26 06
Decrypted text: Hallo World

With interference
Modified encrypted text: 5 D5 D1 9A Rj F7 AB Iq 27 86 07 w 26 06
Decrypted text: 0F 11 1F 84 91 22 96 09 rld
```

Figure 8.6 Smart card encryption results.

The results show the response obtained from the smart card when requested to encrypt legitimate data, and data that has been interfered with (illegitimate data). The byte that was interfered with is indicated in bold in Figure 8.6. Results confirm that the smart card is capable of encrypting plain text. The smart card can also decrypt cipher text sent to it and return the clear text. It also shows that, if the data is interfered with, the decryption will also be affected.

8.6 CHAPTER SUMMARY

This chapter presents several tests on the model implemented in CHAPTER 6 and CHAPTER 7. These tests are necessary to verify the thoughts presented in earlier chapters. The results presented are analysed and discussed in great detail. All the tests prove that the concept of the security model presented in CHAPTER 5 is successfully implemented in providing a secure environment when smart cards are being used. The next chapter presents the final conclusion of the dissertation.

CHAPTER 9
CONCLUSION

One of the goals of this dissertation is to see the impact of using smart cards with cryptographic functionality in providing secure access to a remote server. Of course, it is virtually impossible to design a complete system or even one single smart card in a perfectly secure way that cannot be breached by anything or anybody. In the end, one only has to put in a sufficiently great effort for an attack in order to be able to infiltrate or manipulate any system. However, each potential attacker will always make some sort of benefit analysis for himself and for his targets. After all, the result he will achieve by breaching a system must be worth the work, the money and the time he is investing in the operation. If the result - whether monetary value or reputation among experts and in the world - does not justify the effort, nobody will put too much energy into breaching a system or a smart card. This thought should be one of the main criteria that should be used when designing a secure system for smart cards.

The proposed models provide adequate security for a remote client to access the remote gateway using the Internet. The models are successfully implemented and provide channel security by using existing protocols such as SSL and SSH for server authentication and nonrepudiation, and use the cryptographic capability of the smart cards for key storage, encryption/decryption, and client authentication. The model presented is generic for new security protocols which should be easily incorporated.

SSL and SSH provide channel-level security and prevent most known attacks. This simply means that the data being transmitted is being kept secret and it has not been tampered with. Both the server and client can be authenticated to each other before any data exchange takes place. Secure generation and storage of the keys are very important to achieve the above. The security of the protocol depends on the secrecy of the master secret, if it is compromised, then the session is completely vulnerable to attack. In simple terms, if an attacker has the master secret, then the data might as well be transmitted in the clear.

The master secret is acquired form the server's private key, and the secrecy of the server's private key is also maintained. This is achieved by the use of smart cards. Compromise of the master secret can lead to confidentiality and integrity attacks. The second key issue of concern is that all cryptographic protocols require strong random numbers to operate securely. Poor random numbers can compromise the protocol completely. Both client and server must be able to generate strong random numbers. If the server generates weak random numbers, then the attacker can guess the server's private key, leading to compromise of the premaster secret. If the client generates weak random numbers, then the attacker can guess the premaster secret directly. There are two ways in which these random numbers can be generated: either hardware-based random number generators or pseudo-random number generators. Both have their weaknesses and strengths. This dissertation used the latter option. All number generators need to be seeded with a certain amount of secret data provided by the client. Another key security factor is algorithm selection, and there are varieties of cipher suits available for connection. These algorithms vary in strength from very weak to very strong ciphers.

The performance of the model is a key concern and has already been discussed in CHAPTER 8. A good mix is used to maintain the performance and security of the model. Performance improvement could translate into direct operational costs. Unfortunately, some performance degradation is unavoidable. In particular, cryptography is computationally expensive. Specifically, the large number of operations required for public key cryptography is extremely CPU intensive. Handshake is comparatively expensive, as it happens only once per connection. On the other hand, each individual data record is comparatively within the acceptable range, but for large amounts of data, it can dominate the handshake. The server performance is limited due to the handshake process from both the client and the server and lowers the number of connections the system can handle. This situation could be changed by using client authentication only. The client could perform an RSA private key operation more or less equivalent to that performed by the server and this would become the dominant performance cost for the client. In the data transfer phase, there are two operations: record encryption and the record MAC. A fast algorithm could improve the time over MACing, thus encrypting and MACing would share the cost somewhat evenly. Record assembly and disassembly account for an insignificant amount of time. Hence the choice of MAC and encryption algorithms affects the performance of the system.

The client authentication protocol effectively doubles the handshaking time needed between a client and the server, before both are authenticated and a secure connection is then set up. The server is capable of handling a moderate client load, but performance will decrease dramatically with an increasing number of client connections. As already mentioned in CHAPTER 8, cryptographic computations necessary for SSL becomes the bottleneck for server performance. Avoiding unnecessary handshake or re-handshake can slightly improve this bottleneck and, as in this case, limited client connections are required, it should not be a very serious concern.

In CHAPTER 3, detailed findings regarding security issues with smart cards are presented. The main focus is on the physical level attack and the logical level of attacks. Several types of smart card attack are discussed and various findings are presented. In most cases, safeguards and guidelines are presented for both the manufacturer of the card and the user of the card used in the security model.

With these supporting findings presented in CHAPTER 3 and CHAPTER 8, it can be concluded that the dissertation has analysed in great detail the behaviour and performance of smart cards in providing secure connection to and data communication with remote computers over the insecure Internet. Finally, it may be concluded that the outcomes set out in CHAPTER 1 have been convincingly met. The security attack scenarios presented in CHAPTER 3 and the results presented in CHAPTER 8 indicate that the system is capable of creating and managing a secure communication channel that can protect the user's data from unauthorised, unauthenticated users and provide a secure connection.

Cryptography, algorithms and protocols are not new, but their implementation and use with smart cards definitely introduces a degree of newness to the security measures and will become more widely accepted as the smart card technology matures and newer smart cards with more processing capability and faster communication standards are introduced.

9.1 SUGGESTIONS FOR FUTURE WORK

An interesting area of future research is the use of server-side cryptographic accelerators to speed up cryptographic operations to decrease the handshake time. A second consideration is the issue of PKI (public-key infrastructure), and the dynamic key distribution of secret keys between all the servers of different gateways. Thirdly, file transfer instead of merely text data, can lead to exciting application exercises, although this would require significantly large bandwidth between the smart card interface and the machine providing the connection.

Another study would be to investigate the algorithms based on elliptic curve cryptosystems (ECC) as they provide a much better level of security per bit of key size when compared with public-key cryptosystems. ECC can deliver higher strength per bit when compared to public-key systems because of the difficulty of the hard problem upon which it is based and up to six times better strength is obtainable when compared with RSA/DSA [65]. A more recent study indicates that accessing a smart card based on ECDSA (Elliptic Curve Digital Signature Algorithm) shows much better performance for encryption, decryption and signature generation and verification when compared with RSA[66]. Based on these results, it will be interesting to see the effect on performance.

The major constraint with the smart card is its serial communication bus that is the interface between the card and the host computer. In view of this, it will be very much appropriate and interesting to investigate the use of more advanced serial bus technologies such as USB (Universal Serial Bus), IEEE 1394, etc.; in particular, USB2.0 where the theoretical maximum data rate is 480 Mbit/s (60 MB/s) per controller and is shared amongst all attached devices. The most challenging issue would be to redefine the ISO7816 standard to incorporate these new bus specifications.

REFERENCES

[1] E. Rescorla. "SSL and TLS: Designing and Building Secure Networks", Indianapolis: Addison-Weasley, 2001.

[2] Chris R. Burger. "Security policy: IGUANA", University of Pretoria, Department of Computer Engineering, M. Eng, 2004, pp 1-15.

[3] Smit J., and Hancke G.P. "The design and implementation of a general-purpose, secure, measurement and control network incorporating Internet-based access", Instrumentation and Measurement Technology Conference, 2003. IMTC '03. Proceedings of the 20th IEEE, Vol. 2, 20-22 May 2003, pp1643 - 1647.

[4] Greeff P.G., Hancke, G.P., and van der Merwe D.L. "Design of an access control module for an instrumentation gateway", Instrumentation and Measurement, IEEE Transactions, Vol. 55, Issue 1, Feb. 2006, pp140 - 149.

[5] Refik Molva. "Internet Security Architecture", Computer Networks, Vol. 31, Issue 8, pp 787-804, 1999.

[6] Min-Shiang Hwang, and Li-Hua Li. "A new remote user authentication scheme using smart cards", Consumer Electronics, IEEE Trans. Vol. 46, Issue 1, Feb. 2000, pp 28 - 30.

[7] Wen-Sheng Jaung. "Efficient three-party key exchange using smart cards", Consumer Electronics, IEEE Trans., Vol. 50, Issue: 2, May 2004, pp 619 - 624.

[8] Chou-Chen Yang, and Ren-Chiun Wang. "Cryptanalysis of a user friendly remote authentication scheme with smart cards", Computers & Security, Vol. 23, Issue 5, July 2004, pp 425-427.

[9] Card Technology Today. Science Direct, “Smart card sales to reach 3 billion by 2008”, Vol. 16, Issue 9, September 2004, pp 4.

[10] L. C. Guillou, M. Ugon, and J. J. Quisquater. "Cryptographic authentication protocol for smart cards", Computer Networks, Vol. 36, Issue 4, pp 435 - 451, 2001.

[11] John Borst, Bart Preneel, and Vincent Rijmen. "Cryptography on smart cards", Computer Networks, Vol. 36, Issue 4, pp 423-435, 2001.

[12] Bhatt D.V., Schulze S., Hancke G.P., and Horvath L. “Secure Internet access to gateway using secure socket layer”, IEEE International Symposium on Virtual

Environments, Human-Computer Interfaces and Measurement Systems, 2003. VECIMS '03. 27-29 July 2003 Page(s):157 – 162.

[13] Bhatt D.V., Blignaut J.F., and Hancke G.P. "Securing a transmission channel between two remote computers with secure shell and implementing cryptography on smart card",. 7th AFRICON Conference in Africa, Volume 1, 2004 Page(s):377 - 382 Vol.1.

[14] Bhatt D.V., Schulze S., and Hancke G.P. "Secure Internet access to gateway using secure socket layer", IEEE Transactions on Instrumentation and Measurement, Volume 55, Issue 3, June 2006 Page(s):793 – 800.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. "Handbook of Applied Cryptography", CRC Press, 1996. http://www.cacr.math.uwaterloo.ca/hac/

[16] G. Keathing. "Performance analysis of AES candidates on the 6805 CPU cores", Proceedings on 2nd Advanced Encryption Standard Candidate Conference, 1999.

[17] Bruce Schneier. "Applied Cryptography", second edition, John Wiley & Sons, 1996. [Online]. Available from http://www.schneier.com/ [Updated: Jan 2010].

[18] E. Biham, and A. Shamir. "Differential fault analysis of secret key cryptosystems", Advances in cryptosystems, Lecture notes in computer science, Vol.1294, Springer, Berlin, 1997.

[19] D. A. Carts. "A review of the DH algorithm and its use in secure internet protocols"; Information security reading room. SANS Institute, Nov 2001.

[20] Neil Koblitz. "A Course in Number Theory and Cryptography", second edition, Springer, 1994.

[21] M. E. Hellman. "Cryptography Research Inc.", [Online]. Available from http://www.cryptography.com [Updated: Jan 2010].

[22] Electronic Frontier Foundation. "Cracking DES", [Online] http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html [Updated: Jan 2010].

[23] Hans Riesel. "Prime Numbers and Computer Methods for Factorization", Birkhauser, 1994.

[24] ISO 7816. Information Technology – Identification cards – Integrated circuit(s) cards with contacts, 1999.

[25] R. Bright. "Smart Cards: Principles, Practice, Application", Ellis Horwood Ltd., 1988.

[26] W. Rankl, and W. Effing. "Smart Card Handbook", 2nd Edition, John Wiley & Sons. Ltd., West Sussex, England, 2000.

[27] Wolfgang Rankl, and Wolfgang Effing. “Smart Card Handbook”. John Wiley and Sons, third edition, February 2004. Information available online http://www.wrankl.de/SCH/SCH.html

[28] Sun Developer network (SDN). "Java Card Platform Security", [Online]. Available from http://java.sun.com/products/javacard [Updated: Feb 2010].

[29] BasicCard. "The ZeitControl BasicCard Family", [Online]. Available from www.basiccard.com [Updated: Feb 2010].

[30] Iain Chidgey. “Making security more SIMple”, [Online]. Available from http://www.smartcardbasics.com/index.html, [Updated: Feb 2010].

[31] J. Leach. "Dynamic Authentication for Smartcards", Computers & Security, 15, pp 385-389, 1995.

[32] T. Verschuren. "Smart access: strong authentication on the Web", Computer Networks and IDSN Systems 20, pp 1511-1519, 1998.

[33] DESIRE. "The DESIRE project", [Online]. Available from http://www.desire.org [Updated: Mar 2010].

[34] P. Urien. "Internet card, a smart card as a true internet node", Computer Communications, volume 23, issue 17, pp 1655-1666, Oct 2000.

[35] T. Sauter, and C. Schwaiger. "Achievement of secure Internet access to fieldbus systems", Microprocessors and Microsystems, pp 331-339, 2002.

[36] C. Chong. "Digital Signatures-II", Computer Law & Security, pp 322-331, 1998.

[37] C. Markantonakis. "Java Card Technology and Security", Information Security Technical Report, pp 82-89, 1998.

[38] OCF. "OpenCard Framework 1.2", [Online]. Available from http://www.openscdp.org/ocf/ Updated: Mar 2010].

[39] Oliver Kömmerling, and Markus G. Kuhn. “Design Principles for Tamper Resistant Smart cards Processors”, USENIX Workshop on Smart cards Technology, Chicago USA, 10–11 Mai 1999.

[40] Lan Gao, Jun Yang, Marek Chrobak, Youtao Zhang, San Nguyen, and Hsien-Hsin S. Lee. “A Low-cost Memory Remapping Scheme for Address Bus Protection”, *PACT'06,* September 16.20, 2006, Seattle, Washington, USA. [Online]. Available from

http://www.cs.virginia.edu/~pact2006/program/pact2006/pact135-gao1.pdf [Accessed: Mar 2010].

[41] Ross J. Anderson, and Markus Khun." *Low Cost Attacks on Tamper Resistant Devices"*, in proc. of 5th Security Protocols Workshop, LNCS 1361, pp. 125-136, Springer, 1997.

[42] Neve M., Peeters E., Samyde D., and Quisquater J. J. "Memories: A Survey of Their Secure Uses in Smart Cards", Security in Storage Workshop. SISW '03. Proceedings of the Second IEEE International 31-31 Oct. 2003 Page(s):62 – 62, 2003.

[43] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", [Online]. Available from http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf [Accessed: Mar 2010].

[44] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Introduction to Differential Power Analysis and Related Attacks", Internet, 1998.

[45] HoonJae Lee, ManKi Ahn, SeonGan Lim, and SangJae Moon. "A Study on Smart Card Security Evaluation Criteria for Side Channel Attacks**",** [Online]. Available from http://kowon.dongseo.ac.kr/~hjlee/ [Accessed: Apr 2010].

[46] Paul C. Kocher. "Timing Attacks on Implementations of Diffie Hellmann, RSA, DSS, and Other Systems", Internet, 1995.

[47] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the importance of checking cryptographic protocols for faults", Journal of Cryptology, Springer-Verlag, Vol. 14, No. 2, pp. 101--119, 2001, Internet, http://crypto.stanford.edu/~dabo/papers/

[48] Mark Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. "Differential fault analysis attack resistant architectures for the advanced encryption standard**",** [Online]. Available from http://mark.bu.edu/papers/175.pdf [Accessed: Apr 2010].

[49] Sergei Skorobogatov, and Ross Anderson. "Optical Fault Induction Attacks", Internet, Mai 2002.

[50] Michael Lamla. "Hardware Attacks on Smart Cards Overview", Eurosmart Security Conference, Marseille, 13–15 Jun 2000.

[51] Jussi Rautpalo. "GPRS Security - Secure Remote Connections over GPRS", Helsinki University of Technology, Department of Computer Science, pp 11-14, 1998.

[52] Netscape [Online]. Available from http://docs.sun.com/source/816-6156-10/contents.htm [Updated: Apr 2010].

[53] OpenSSH. "The OpenSSH project", [Online]. Available from http://www.openssh.org [Updated: Apr 2010].

[54] D. J. Barret, and R. E. Silverman. "SSH, The Secure Shell: The Definitive Guide", Sebastopol, O'Reilly & Associates, Inc, pp 1-106, 2001.

[55] Van Dyke. "VShell® 3.5 Server for Windows and UNIX". [Online]. Available from http://www.vandyke.com/ [Updated: May 2010].

[56] SSH. "Secure Shell client for Windows", [Online]. Available from http://www.ssh.com [Updated: May 2010].

[57] Network security essentials (NSS). "The mozilla NSS page", [Online]. Available form http://www.mozilla.org/projects/security/pki/nss/index.html [Updated: May 2010].

[58] Borland. "Borland JBuilder8 Personal Edition", [Online]. Available from http://www.borland.com [Updated: Jun 2010].

[59] Giesecke & Devrient. "Sm@rtCafé Professional Personal Edition", [Online]. Available from http://www.smartcafe.gieseckedevrient.com [Updated: Jul 2010].

[60] J. Nagel. "Congestion control in IP/TCP internetworks", ACM SIGCOMM Computer Communication Review, Vol. 25, Issue 1, January 1995, pp 61 - 65.

[61] Swamp. "swamp", [Online]. Available from http://www.geoffthorpe.net/crypto/swamp/ [Updated: Jul 2010].

[62] Siege. "siege", [Online]. Available from http://www.joedog.org/siege/ [Updated: Jul 2010].

[63] W. Stallings. "Network Security Essentials: Applications and Standards", New York: Prentice-Hall, 1999.

[64] D. A. Wheeler. "Secure programming for Linux and Unix HOWTO", [Online]. Available from http://www.dwheeler.com/secure-programs [Updated: Jul 2010].

[65] Mohammed Emarah, El-Shennawy. "Elliptic curve cryptosystems on smart cards", IEEE 35th International Carnahan Conference on Security Technology, 2001.

[66] Chatterjee, K.; Gupta, D. "Secure Access of Smart Cards Using Elliptic Curve Cryptosystems", IEEE 5th International Conference on WiCom '09, Wireless Communications, Networking and Mobile Computing, 2009.

[67] Selimis, G.; Fournaris, A.; Kostopoulos, G.; Koufopavlou, O., “Software and Hardware Issues in Smart Card Technology ”, Communications Surveys & Tutorials, IEEE Volume: 11 , Issue: 3 , 2009, Page(s): 143 - 152