INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A PSEUDO ONE-TIME-PAD BASED SECURITY SYSTEM

JOHN VELISSARIOS

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AT

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

JUNE 1997

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-40222-3

Canada

Abstract

A PSEUDO ONE-TIME-PAD BASED
SECURITY SYSTEM

JOHN VELISSARIOS

Cryptography is playing an increasingly important role in the computing and telecommunications industry. The cryptographic schemes currently employed, however, are theoretically breakable given enough time and/or computing resources. Nevertheless, there exists one scheme, the One-Time-Pad, that is theoretically unbreakable given that it satisfies certain requirements. The One-Time-Pad, however, is challenged by serious constraints due to inherent deployment problems. The purpose of this thesis is to outline a pseudo One-Time-Pad-based security system whereby deployment requirements are satisfied while leveraging the inherent strengths of the One-Time-Pad encryption scheme.

The One-Time-Pad, also known as the Vernam Cipher, provides unconditional security regardless of the computational resources available. This state of unconditional security is also known as *perfect secrecy*, and the Vernam Cipher is the only scheme capable of making that claim. There are, however, several strict requirements that must be satisfied in order to guarantee perfect secrecy. The Vernam Cipher requires a truly random source of bits each of which undergoes an exclusive-or Boolean operation with the plaintext message. The problem of distributing a truly random One-Time-Pad to a remote location has severely limited the applicability of the Vernam Cipher for industrial, commercial, and personal use. A pseudo One-Time-Pad-based security system is one wherein a pseudo-random bit sequence is transformed into a cryptographically secure pseudo-random bit sequence, possessing all of the characteristics of the highly desirable truly random bit sequence. This proposed scheme is used in our distributed and secure implementation of the One-Time-Pad encryption scheme.

Acknowledgments

I would like to acknowledge Dr. Bipin C. Desai for all his patience and guidance throughout my graduate studies. His supervision and support proved to be instrumental in helping me complete what at times seemed to be an insurmountable task.

I would also like to thank my family and friends for their patience, support, encouragement, and understanding throughout my academic endeavors. Without them this thesis would have been impossible.

I would also like to thank Mike Christoulas for his help in editing this thesis. His editorial skills and ability to clarify the most obscure explanations proved to be invaluable.

Table of Contents

List of Figures

List of Tables

Glossary

Terminology	
Plaintext	Original message to be transmitted.
Encryption	Process of encrypting the plaintext message.
Ciphertext	Original message in encrypted form.
Decryption	Process of decrypting the plaintext message.
Original Plaintext	Original message received by recipient.
Cryptanalysis	The science of breaking ciphertext.

Acronyms	
OTP	One-Time-Pad
TROTP	Truly Random OTP
PROTP	Pseudo-Random OTP
CSPROTP	Cryptographically Secure PROTP
TRBG	Truly Random Bit Generator
PRBG	Pseudo-Random Bit Generator
CSPRBG	Crytpographically Secure PRBG
LCG	Linear Congruential Generator
LFSR	Linear Feedback Shift Register
POTPSS	Pseudo One-Time-Pad-based Security Scheme
NIST	National Institute of Standards and Technology

NSA	National Security Agency
SHS	Secure Hash Standard
SHA	Secure Hash Algorithm
NBS	National Bureau of Standards
DES	Data Encryption Standard
DES_3	Triple DES
CKR	Cryptographic Key Ring
XOR, $\oplus$	Bit-Wise Exclusive Or Operation
RSA	Rivest Shamir Adleman Encryption Algorithm
PGP	Pretty Good Privacy
IEEE	Institute of Electrical and Electronics Engineers
ASCII	American Standard Code for Information Interchange
SET	Secure Electronic Transaction
MIPS	Million Instructions Per Second
MIME	Multipurpose Internet Mail Extension

Conventions

Alice	Client A in any information exchange.
Bob	Client B in any information exchange.
$C = E_k(P)$	The encryption function *E* (with key *k*) operating on plaintext *P* to produce ciphertext *C*.
$P = D_k(C)$	The decryption function *D* (with key *k*) operating on ciphertext *C* to produce plaintext *P*.

Secret Key	The key used to encode and decode messages in symmetric algorithms (i.e. DES).
Public/Private Keys	The keys used to encode and decode messages in asymmetric algorithms (i.e. RSA).
$M_{xy}n$	Message client *x* sends to client *y*. *n* equals the sequence number of message *x* to *y* or *y* to *x*.
PW_{ab}	Negotiated password for communications between client *a* and client *b*.
$K_{xy}n$	CKR hash value for message between client *x* and *y*. n equals the sequence number.
$C_{xy}n$	Ciphertext client *x* sends to client *y*. *n* equals the sequence number of message *x* to *y* or *y* to *x*.
T_{ab}	One-time throw-away password to bootstrap communications between client *a* and client *b*.

Chapter 1

Introduction

1.1 Overview

Cryptography is the science of hiding information. This is done by effectively transforming an original plaintext message into an encrypted ciphertext message such that, even if intercepted, is unreadable without access to the secret key(s) used to encrypt the original message. Cryptography may be defined as the process of expressing information in symbols which convey this information only to those individuals who can successfully interpret these symbols.

Cryptography has existed for thousands of years, and until recently has been a science practiced primarily by secret agents, spies, and covert government agencies [1]. In the last twenty years or so, cryptography has taken on a renewed interest as we evolve towards a society where information becomes the most sought after commodity. This evolution has both influenced and been influenced by advances in telecommunications, the computer hardware and software industry.

It is now possible at the push of a button to exchange messages, video conference, or even collaborate with peers on a project from almost anywhere in the world using global networks such as the Internet.

Security and privacy are major issues that are raised whenever individuals and corporations alike exchange vital and confidential information over *unsecure*

networks such as the Internet [2]. As information becomes the currency of this new economy, protecting it has never been more imperative [3]. Cryptography, both in the form of algorithms and protocols, is the only means available to protect information assets from being intercepted and used against individuals and corporations.

1.2 The Problem

There exist many cryptographic algorithms that are deemed sufficiently secure provided that it is computationally infeasible to perform a *brute force* attack on them. A brute force attack attempts to extract the secret key(s) that protect a message by trying every possible secret key permutation. A strong cryptographic algorithm, therefore, is one where the only means of extracting the key(s) is by brute force, thereby discouraging such attacks by making it computationally infeasible to attempt them. More specifically, the effective strength of these algorithms is based on the length of the key(s). The longer the key(s), the more time it would take to attack the algorithm successfully.

An attack that is infeasible to some, however, may not be infeasible to others. Governments and large corporations have the necessary computational and financial resources to perform the systematic calculations to attack strong encryption schemes effectively. Whether they choose to do so remains unanswered. However, one should keep in mind the motive behind the International Traffic in Arms Control Regulations (ITAR) classification of encryption software as military arms, when asking that question [4]. Once a key has been extracted, it can be used to decipher all past, current, and future messages without the sender's knowledge, creating a false sense of security. Also, key lengths and cryptographic algorithms that provide sufficient security today may be trivial to attack in the future.

Moreover, there are situations, such as in diplomatic contexts [5], where the security provided even by strong algorithms is still insufficient; for instance, those cases where a secret must remain secret forever, regardless of advances made in computer science or mathematics, or where the interception and successful decipherment of a single message must never reveal enough information to decipher previous and/or future messages. In other words, an attack only reveals just enough information to decipher a single message once and does not jeopardize the security of previous and/or future messages. Situations such as these are not addressed by the majority of current encryption algorithms.

1.3 The Goal of this Thesis

This thesis presents a Pseudo One-Time-Pad-based Security System (referred to hereafter as POTPSS) which attempts to provide almost *perfect secrecy* by the use of a cryptographically strengthened One-Time-Pad (OTP). POTPSS is a symmetric protocol whereby both communicating parties share the same secret key(s) required to both encrypt and decrypt a message. POTPSS also uses previous message spaces to generate message digests that are used as seeds for the encryption of future messages. This ensures that message spaces are effectively inter-linked. This also ensures that successful encryption and decryption of the current message space is based on the successful encryption and decryption of previous message spaces. Any break in the links is seen as an attack on the POTPSS scheme. In addition, the POTPSS scheme never stores the secret key(s) negotiated between two parties in the system or in any of the messages.

The generation of a truly random sequence via computational means is subject to suspicion as Knuth [6] quoted John Von Neumann in saying "Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin." Although there is no way to deterministically generate Truly Random

One-Time-Pads (TROTP) on two separate machines, as would be required in this scheme in order to guarantee perfect secrecy, the POTPSS scheme generates a Pseudo-Random One-Time-Pad (PROTP) with all the required properties of a TROTP except unpredictability. However, using techniques described in detail in section 4.1.1 of this thesis, the undesirable predictability property is effectively removed by cryptographically strengthening the PROTP such that it is capable of resisting attacks that exploit this predictability weakness.

1.4 The Thesis Outline

Chapter 2 describes the state of the art in encryption technology. Secret key and public key cryptography are covered in detail followed by encryption protocols that have been proposed recently in order to ensure secure electronic commerce over the Internet. Lastly, the subjects of perfect secrecy and the Vernam OTP are covered in detail in order to better understand the simple yet highly effective security that this scheme provides. Chapter 3 describes the problem domain and the various categories of encryption schemes, their strengths and weaknesses. Chapter 4 introduces the POTPSS scheme and discusses in detail its major components, their inter-operation, as well as the POTPSS protocol. Chapter 5 conducts a cryptanalysis of POTPSS, after a discussion of cryptanalysis methods. A cryptanalysis of the POTPSS protocol and a comparison of the results with those achieved by other encryption schemes are also presented. Chapter 6 discusses the results achieved by the prototype itself and how these results compare with other encryption schemes. Chapter 7 identifies areas where the POTPSS scheme can be applied. Finally, chapter 8 concludes the thesis and discusses the implications for future work in the field.

Chapter 2

State of the Art Survey

2.1 Encryption Algorithms

This section discusses the state of the art in encryption algorithms. The algorithms selected for discussion here in no way represent an exhaustive list, but have been selected because they represent de-facto standards in symmetric and asymmetric encryption algorithms. This is significant because these algorithms have been open to scrutiny by academia and industry for quite some time.

Encryption algorithms operate directly on plaintext messages and secret keys in order to generate ciphertext messages. The encryption function E (with key k) operates on plaintext P to produce ciphertext C as follows:

$$C = E_k(P)$$

Consequently, the decryption function D (with key k) operates on ciphertext C to produce plaintext P as follows:

$$P = D_k(C)$$

Each encryption and decryption algorithm determines the exact functionality that E and D will have. The secret key, k, will also be determined by the algorithm used. This will be discussed shortly.

2.1.1 DES

The Data Encryption Standard (DES) was developed in 1975 at IBM, as a derivative of their LUCIFER system, in response to a National Bureau of Standards (NBS) solicitation for cryptosystems. It was adopted as a standard for "unclassified" applications on January 15, 1977 and has been reviewed by the National Institute of Standards and Technology (NIST) approximately every five years since its adoption. Its most recent renewal was in January 1993, when it was renewed until 1998. However, it is anticipated that it will not remain a standard past 1998 [5].

DES is a block cipher; encrypting data in 64-bit blocks [7]. When a plaintext or ciphertext is introduced to the encryption or decryption algorithm, which happen to be the same except for differences in the key schedule, encryption, or decryption is performed in 64-bit blocks, generating the ciphertext or plaintext respectively.

DES requires a 56-bit key, usually expressed as a 64-bit number with every eighth bit used for parity checking. This key can be changed at any time as it is not stored in the algorithm. There are a handful of weak keys, but they can easily be avoided. The security of DES lies entirely within the key; therefore, the only feasible method of attack is by brute force [7].

DES provides confusion and diffusion through complex encipherment transformations of large blocks of data. Confusion involves substitutions that make the relationships between the key and the ciphertext as complex as possible, while diffusion involves transformations that dissipate the statistical properties of the plaintext across the ciphertext via the key [7]. The fundamental building block of DES is a *round*, which is essentially a substitution followed by a permutation on

the plaintext P or ciphertext C based on the key k. DES performs the same round on the plaintext P or ciphertext C 16 times.

When the DES algorithm was designed, there was a basic requirement that it be easily implementable in 1970s hardware technology. As a result, DES uses only standard arithmetic and logical operations on blocks of at most 64-bits in length [5].

Each round in DES takes the 64-bit plaintext block, breaks it up into a left and right half, each 32-bits long. The key is then combined with the data via a round. This operation is performed 16 times. A final permutation is performed, which is the inverse of the initial permutation and the right and left halves are joined.

Each round in DES sees the key bits shifted and 48-bits selected out of the 56-bit key. The 32-bit right half of the data is expanded to 48-bits via an expansion permutation. The result is then XORed with the 48-bit shifted and permuted key. It is then substituted with 32 new bits using a substitution algorithm, and permuted once again. The output of the operations described above becomes the new left half, the old left half becomes the new right half. These operations are repeated 16 times, each time making 1 round for a total of 16 rounds [7].

One of the operations typical in DES is the S-Box substitution which is performed after the compressed key is XORed with the expanded block. This S-Box substitution essentially hard wires the substitution of the bits in the block according to a pre-determined S-Box schedule. This substitution is non-linear as opposed to all other operations which are easy to analyze. This S-Box substitution essentially gives DES its security [8].

The security of DES has long been a source of controversy [5]. The key length, the number of iterations or rounds, and the design of the mysterious S-Boxes

which are used to wire the substitution of the bits in the message have always aroused suspicion, because a particular selection of these parameters is seen as arbitrary.

Cryptographers have argued for longer key lengths than the 56-bit key that DES requires. "This key length requires on average 2^{55} permutations in order to recover the secret key and hence decrypt the message" [9]. Biham and Shamir successfully performed differential cryptanalysis on DES [10]. However, the consensus remains that DES is still secure against differential cryptanalysis when implemented properly. There is essentially no better method of attack than by brute force [9].

The number of rounds that DES performs is vital to the security of the algorithm. Alan Konheim [11] showed that a minimum of eight iterations was required in order to ensure that the ciphertext was essentially a random function of every plaintext bit and every key bit. DES, with any number of rounds fewer than 16, could be broken with a known-plaintext attack more efficiently than by brute force as demonstrated by Biham and Shamir's differential cryptanalysis method [10].

There are ways to increase the security of DES; specifically by performing multiple encryptions which effectively increase the key length and, as a result, strengthen the algorithm. If performing multiple encryptions effectively reduces the security of an algorithm, it is then known as a *group*. If an algorithm is known as a group, then it has highly undesirable properties which effectively weaken the ciphertext, thus making it more susceptible to attack. Triple DES (DES_3) doubles the key length from 56-bits to 112-bits. This is due to the way the keys are used in the DES_3 scheme. This can only be possible if performing multiple encryptions does not effectively reduce the security of the algorithm. There has been

overwhelming evidence that has proven that DES is not a group [12]. This confirms that DES_3 is in fact secure and the only effective method of attack is by brute force.

2.1.2 RSA

RSA, introduced in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman has withstood years of extensive cryptanalysis. "Although the cryptanalysis neither proved nor disproved RSA's security, it does suggest a confidence level in the theoretical underpinnings of the algorithm" [9].

RSA's security is based on the inherent difficulty of factoring large prime numbers. RSA has two keys: a public key and a private key. The public key is published and used to decrypt messages signed with a user's private key. The private key is kept secret and is used to encrypt messages. There exist many variations on the use of the keys with RSA. Discussion will be focused on the encryption and decryption process with authentication. Each key is a function of a pair of large prime numbers. These numbers are typically 100 to 200 digits long but can be even longer. Recovering the plaintext message from one of the keys and the ciphertext message is considered to be equivalent to factoring the product of two primes [13].

The RSA algorithm requires that every party must have two keys: a public key and a private key. The two keys are mathematically related in such a way that data encrypted with one key (private key) can only be decrypted with the other key (public key) and vice versa. The generation of keys will be discussed in the next section. In this section we illustrate the message encryption and decryption process. Example: Alice (client A) wants to send Bob (client B) a message. Alice has two keys: $Public_{Alice}$ and $Private_{Alice}$. Bob also has two keys: $Public_{Bob}$ and $Private_{Bob}$.

As we mentioned earlier, the encryption function E (with key k) operates on plaintext P to produce ciphertext C as follows:

$$C = E_k(P)$$

For public key cryptosystems, however, the encryption function E is a two step process. The encryption function E (with keys: $Private_{Alice}$ and $Public_{Bob}$) operates on plaintext P to produce ciphertext C as follows:

$$C = E_{PublicBob}\ (\ E_{PrivateAlice}\ (P)\ )$$

Alice takes her private key ($Private_{Alice}$) and encrypts plaintext P. She then takes the result and encrypts it again with Bob's public key ($Public_{Bob}$). This generates ciphertext C which is transmitted over an unsecure channel.

The decryption function D (with key k) typically operates on ciphertext C to produce plaintext P as follows:

$$P = D_k(C)$$

For public key cryptosystems, the decryption function D is also a two step process. The decryption function D (with keys: $Private_{Bob}$ and $Public_{Alice}$) operates on ciphertext C to produce plaintext P as follows:

$$P = D_{PublicAlice}\ (\ D_{PrivateBob}\ (C)\ )$$

Bob receives the ciphertext C and decrypts it first with his private key ($Private_{Bob}$) and then with Alice's public key ($Public_{Alice}$). This ensures that only Bob can decrypt the ciphertext C with his private key ($Private_{Bob}$) and only Alice could have encrypted the plaintext P with her private key ($Private_{Alice}$). This guarantees that the message could only be read by Bob and could only have been sent by Alice.

Next, the process required in order to generate a public and private key pair [13], will be discussed in detail.

Compute the product *(n)* of two large prime numbers, *p* and *q*:

$$n = p * q$$

Randomly select an encryption key, *e*, such that: *e* and *(p-1) * (q-1)* are relatively prime.

Using Euclid's algorithm, compute the decryption key, *d*, such that:

$$d = e^{-1} \, (mod \, (p-1) * (q-1))$$

Note: *d* and *n* are relatively prime.

The key *e* and the product *n* are the public key; while *d* is the private key. The two primes, *p* and *q*, are no longer needed and should be discarded and never revealed.

The ciphertext *c* is generated from plaintext *m* as follows:

$$c_i = m_i^e \, (mod \, n)$$

The plaintext *m* is retrieved from ciphertext *c* as follows:

$$m_i = c_i^d \, (mod \, n)$$

While the security of RSA is based on the inherent difficulty of factoring large prime numbers, there exists no mathematical proof that states that *n* needs to be factored in order to calculate *m* from *c* and *e*. There is, therefore, always the slight

possibility that an entirely new and different way to cryptanalyze RSA may be discovered [13].

The most obvious means of attack for RSA is to factor n given the public key e and the modulo n. In order to find the decryption key d, n must be factored. The length of n should be much larger than a 120-bit number, since these are anticipated to be factored within a short period of time. Ultra-sensitive applications may require the use of 1024-bit, 1280-bit and 2048-bit n's.

Unsecure implementations of RSA exist. Breaking these implementations should not be mistaken for "breaking RSA", as it is the implementation that is unsecure and not the RSA algorithm [13].

Implementations of RSA should ensure that:

1. The modulus (key) should be as large as possible.

2. The choice of primes should be of approximately equal length.

3. The random numbers used to select the two random primes should themselves be determined by an outside source (i.e. elapsed time between keystrokes).

4. The private key management techniques should ensure limited access to the private keys used to encrypt the message.

5. Keys should have an expiration date to prevent long-term factoring attacks.

2.2 Encryption Protocols

An encryption protocol is essentially composed of a series of encryption algorithms. POTPSS is an encryption protocol, similar to those discussed in this section, in that it uses cryptographic algorithms at a lower level in order to achieve a higher level of functionality and security. An encryption protocol is a series of pre-defined steps that each party agrees to follow in order to successfully send and receive messages. An encryption protocol is effectively a method of engagement.

The Secure Electronic Transaction (SET) protocol has been chosen for discussion, because it represents the collective effort of several major financial, software, and hardware organizations that have contributed to the specification of this protocol which will enable electronic transactions to be securely effectuated over an unsecure network such as the Internet. The SET protocol, however, resembles the POTPSS scheme proposed in this thesis on many levels and as such warrants a detailed description.

NetBill has also been selected for discussion because it represents an effort to enable a trusted third party server to authenticate and validate transactions between two parties securely over an unsecure network such as the Internet. NetBill is an academic effort undertaken at Carnegie Mellon University's Information Networking Institute.

2.2.1 SET

Visa and MasterCard have jointly developed the SET protocol as a method to secure bankcard transactions over open networks [14]. This specification is available for application to any payment service and may be used by software vendors to develop applications [15]. Through the collaboration of organizations

such as GTE, IBM, Microsoft, SAIC, Terisa and Verisign, the SET protocol will be implemented in various hardware and software solutions; therefore enabling secure electronic transactions over open networks.

The SET protocol uses lower level cryptographic algorithms such as RSA, DES, and the Secure Hash Algorithm (SHA) in order to address the following requirements [15]:

- *Confidentiality of information* is ensured by the use of message encryption.
- *Integrity of data* is ensured by the use of digital signatures.
- *Cardholder account authentication* is ensured by the use of digital signatures and cardholder certificates.
- *Merchant authentication* is ensured by the use of digital signatures and merchant certificates.
- *Inter-operability* is ensured by the use of specific protocols and message formats.

The SET encryption process is illustrated with an example which demonstrates the required procedure when Alice wishes to send Bob an electronic document, also known as a property description [15].

Encryption:

1. Alice runs the property description through a one-way algorithm to produce a unique value known as the message digest. This is a kind of digital fingerprint of the property description and will be used later to test the integrity of the message.

2. She then encrypts the message digest with her private signature key to produce the digital signature.

3. Next, she generates a random symmetric key. This key, generated at random by Alice, must be transmitted to Bob in order for him to decrypt the message, hence the symmetry. Alice uses it to encrypt the property description, her signature, and a copy of her certificate which contains her public signature key. In order to decrypt the property description, Bob will require a secure copy of this random signature key.

4. Bob's certificate, which Alice must have obtained prior to initiating secure communication with him, contains a copy of his public key. To ensure secure transmission of the symmetric key, Alice encrypts the certificate using Bob's public key. The encrypted key, referred to as the digital envelope is sent to Bob with the encrypted message itself.

5. Finally, Alice sends a message to Bob consisting of the following: the symmetrically encrypted property description, signature and certificate, as well as the asymmetrically encrypted symmetric key (the digital envelope).

Decryption:

6. Bob receives the message from Alice and decrypts the digital envelope with his private key to retrieve the symmetric key.

7. He uses the symmetric key to decrypt the property description, Alice's signature, and her certificate.

8. He decrypts Alice's digital signature with her public signature key which he acquires from her certificate. This recovers the original message digest of the property description.

9. He runs the property description through the same one-way algorithm used by Alice and produces a new message digest of the decrypted property description.

10. Finally, he compares his message digest to the one obtained from Alice's digital signature. If they are exactly the same, he confirms that the message content has not been altered during transmission and that it was signed using Alice's private signature key. If they are not the same, then the message either originated somewhere else or was altered after it was signed. In that case, Bob takes some appropriate action such as notifying Alice or discarding the message.

Message digest generation in SET is done using SHA. The public and private key generation is done using RSA. This is used for message signing and authentication. DES is used in SET in order to perform symmetric encryption. The exchange of information in SET is done via Multipurpose Internet Mail Extension (MIME) formatted messages[16].

SET is not intended to be used in the secure transfer of messages other than those related to financial transactions. Applications of SET restrict the encryption process only to financial information. In fact, its international acceptance by foreign governments is contingent upon the requirement that it cannot be used to transfer any information other than information required for financial transactions.

2.2.2 NetBill

NetBill is an ongoing research project at Carnegie Mellon University's Information Networking Institute. Its purpose is to support electronic commerce over computer networks. NetBill serves as a mediator between a merchant and consumer who wish to engage in a business transaction with one another. The NetBill protocol provides the ability to transfer electronic goods (i.e. documents, publications, etc..) from a merchant to a consumer as well as the ability to transfer funds between these two parties [17].

The NetBill protocol supports price negotiations, goods delivery, and payment. Innovations that the NetBill protocol provides include [18]:

- A method of (atomic) certified delivery such that a customer pays if and only if she receives her information goods intact.
- A system for allowing access control to be out-sourced such that different users may use different access control servers.
- A credential mechanism for allowing users to easily prove membership in groups to qualify for discounts or for other purposes.
- A structure for easily constructing pseudonyms so that buyers of information can protect their identities.

The transaction protocol supports the following eight steps [18]:

1. Customer ⇒ Merchant - Price Request
2. Merchant ⇒ Customer - Price Quote

3. Customer ⇒ Merchant - Goods Request

4. Merchant ⇒ Customer - Goods, Encrypted With a Key K

5. Customer ⇒ Merchant - Signed Electronic Payment Order

6. Merchant ⇒ NetBill - Endorsed Electronic Payment Order (Including K)

7. NetBill ⇒ Merchant - Signed Result (Including K)

8. Merchant ⇒ Customer - Signed Result (Including K)

The NetBill protocol manages the exchange of messages, thereby enabling transactions such as the one illustrated above to take place. Cryptographic checksums on the message are done via the SHA hash function [18]. Message encryption and signing are done via RSA's public key cryptographic algorithm [18]. The exchange of information in NetBill is done via MIME formatted messages.

2.3 Perfect Secrecy

2.3.1 Vernam's OTP

Unconditional security differs from computational security in that it guarantees perfect secrecy regardless of the computational resources available. Perfect secrecy can only be achieved by one encryption scheme; namely, the Vernam OTP. Gilbert Vernam developed the Vernam OTP in 1917 for use in the automatic encryption and decryption of telegraph messages [5]. Shannon formally developed the concept of perfect secrecy 30 years later, thereby proving that the Vernam OTP was perfectly secure against any possible attack [5].

Perfect secrecy is defined by the condition where the interception of additional ciphertext messages gives the cryptanalyst no extra information in order to decipher one or any of the messages. Perfect secrecy requires that the number of keys must be at least as great as the number of possible messages [7]. The key must never be re-used, and the generation of the key must be perfectly mathematically random and of equal length to the message space [19]. Pseudo-random sequences in the keys will not generate a perfectly secret message as these generators often have non-random properties.

Essentially, the TROTP is XORed with the plaintext message to produce the ciphertext message. Once the ciphertext message is received by the recipient, the same TROTP is XORed with the ciphertext message to produce the plaintext message. Extending the TROTP to satisfy bit-wise encryption is necessary for the encryption of binary data.

Assume Alice wants to send Bob a message *(m)* using a common TROTP (*o*).

Message:

m_i *where* $0 <= i <=$ *length of message (m) in bits*

TROTP:

o_i *where* $0 <= i <=$ *length of message (m) in bits*

Encryption:

Alice: For every i: $m_i \oplus o_i = c_i$, *where* c_i *is the ciphertext message that is transmitted to Bob.*

Decryption:

> *Bob: For every i:* $c_i \oplus o_i = m_i$, *where* m_i *is the plaintext message that was encrypted by Alice.*

Assuming the secret TROTP is never revealed - in fact, it should be destroyed after every transaction - the message will remain unconditionally secure. Since the TROTP contains perfectly random properties, the ciphertext c contains no information that can be statistically cryptanalyzed.

There are several serious drawbacks to the Vernam OTP that have limited its widespread use. Most of the problems revolve around the key management facilities. Furthermore, the generation of a TROTP is not a trivial task. Attacks against this scheme are focused on the method used in the generation of the key sequence [1]. Also, a TROTP of sufficient length must be generated in order to satisfy all the messages that will be sent prior to a subsequent TROTP re-generation. The distribution of the TROTP imposes other serious constraints. It is necessary that the TROTP be transmitted via secure means. Once the TROTP is used up, it must be discarded securely; otherwise, anyone who may have intercepted previous messages will be able to decode all of them with the aid of the intercepted TROTP! If one element (bit or byte) in the message or the TROTP is lost, the message will effectively be rendered useless. Conversely, if one element (bit or byte) is transmitted incorrectly, only that element will be corrupted.

Chapter 3

Problem Domain

3.1 The Need for Secure Communications

The explosive growth of personal and commercial digital communications has resulted in a vast amount of personal and commercially sensitive information being stored and transmitted via computer networks. Personal privacy and commercial security are now more vulnerable than ever as a result of the widespread use and acceptance of unsecure communications media such as the Internet. Cryptography is the only means by which information can be protected while being sent over public unsecure networks.

There are essentially two categories of encryption algorithms: secret key cryptosystems and public key cryptosystems, each of which will be discussed in detail in the following sections. Each of these categories has strengths and weaknesses that will be highlighted in the subsequent sections without going into cryptanalytic details.

Cryptosystems have different levels of security, depending on how difficult they are to break. Computational security is defined as the inability to break an algorithm with any number of current or future resources [5].

The encryption scheme (POTPSS) proposed in this thesis will attempt to address several weaknesses and offer a solution that is superior in situations where:

1. It is feasible to occasionally exchange secret keys between users.

2. Encryption can be performed with the POTPSS scheme in a specific state. In other words, the user not only must have access to the protocol scheme, but must also be able to access a Cryptographic Key Ring (CKR) in a state that is necessary in order to communicate with another party.

3. The security of the message space is more important than the actual delivery of the message. In other words, if there is an attempt to attack the scheme in the form of interception as opposed to impersonation, the scheme would be rendered useless to both parties, until it is bootstrapped once again, thus losing any messages in transit. Although, the security of the message would not be compromised, the message would be rendered useless, even to the intended recipient.

4. Messages should arrive and be sent in sequential order; otherwise, they should be serialized for the scheme.

3.2 Problems with Existing Encryption Algorithms and Protocols

Secret and public key cryptosystems have one fundamental flaw: once a secret or private key is extracted, by brute force or otherwise, all past, present, and future messages can be easily decrypted.

The ability to dynamically change the secret, public, and private key after every transaction is prohibitively expensive in either of these algorithms. Secret, public, and private keys are intended to possess a lifecycle of sufficient duration to ensure security while minimizing the cost of re-negotiating keys.

SET has narrowly-defined application areas and is not intended for general cryptographic use. As a result, SET may possess highly desirable properties for certain types of applications. However, SET was developed explicitly for satisfying the requirements of secure electronic transactions (namely, credit card purchases) over the Internet [15].

Existing encryption algorithms and protocols cannot provide the security required in situations where:

1. A secret must remain secret forever, regardless of advances in computer science or mathematics.

2. Keys used to encrypt and decrypt messages must be generated dynamically.

3. Information being transmitted is more voluminous than simple financial data.

In high security diplomatic contexts, communications must never fall victim to systematic attacks. In these contexts, adversaries do have the computational resources to coordinate and carry out systematic attacks in order to intercept and recover extremely valuable messages.

It is also extremely cumbersome and unsecure to constantly change the keys used to encrypt and decrypt messages. This must be an automated feature of any encryption scheme used in environments such as these.

The volume of information exchanged is more than simple account numbers and amounts. The volume could potentially be data-intensive applications such as voice and video communications.

The inadequacy of many existing schemes to address all of the above requirements resulted in the development of the POTPSS scheme. Although

many schemes were designed to address most of the requirements above, few have managed to address them all.

Chapter 4

The Proposed Approach - POTPSS

4.1 Major Component Overview

This section discusses the major components that make up the POTPSS protocol. We begin our discussion with the Cryptographically Secure Pseudo-Random One-Time-Pad (CSPROTP) followed by DES, SHA, and then the CKR. Each component will be discussed in the context of the POTPSS protocol. The security of each component will also be discussed. The intent is to show how these components stand on their own in the context of the POTPSS protocol. Since "a chain is only as strong as its weakest link", no major component can have cryptographic weaknesses that can be readily exploited by a cryptanalyst. Therefore, it is imperative that each component have a minimum level of security in order to assure the overall security of the POTPSS scheme. This will be shown in detail in this section.

4.1.1 Cryptographically Secure Pseudo Random OTP

4.1.1.1 Overview

PROTPs are used in the POTPSS scheme in order to satisfy distributed key generation requirements. PROTPs, however, also possess certain properties which effectively render them useless for cryptographic applications [21]. PROTPs are, therefore, transformed into CSPROTPs which effectively give them the same properties as TROTPS; namely, randomness and unpredictability

properties. Before we begin our discussion on the cryptographic strengthening of PROTPs, we justify the need to use PROTPs and the transformation which they must undergo.

Algorithm	Pad	Security	Comments
PRBG	PROTP	Unsecure	Next Bit Predictable
CSPRBG	CSPROTP	Computational	Perfect For POTPSS
TRBG	TROTP	Unconditional	Not Usable In POTPSS

Table 1: Bit Generators and their Use in POTPSS

Table 1 shows the three types of random bit generation algorithms and their potential use in POTPSS. The highlighted fields indicate the path which the POTPSS scheme follows in order to generate a CSPROTP. We see that the Truly Random Bit Generator (TRBG) generating a TROTP is equivalent to the Vernam OTP. It is not usable in POTPSS because, as desirable as unconditional security may be, there is no way to deterministically generate two identical TROTPs. We see from the table that the minimum security that could be used in the POTPSS scheme is available from CSPROTPs. The POTPSS scheme takes a PROTP generated by a Pseudo-Random Bit Generator (PRBG) and cryptographically strengthens it so that is becomes a CSPROTP.

The CSPROTP is essentially XORed with the plaintext message in order to generate a ciphertext message that is then transmitted to the recipient. The recipient then generates the identical CSPROTP, XORs it with the ciphertext message, generating the plaintext message.

As long as the CSPROTP remains secret and is handled in the same manner as the TROTP after the transaction (i.e., is destroyed), the message will remain cryptographically secure.

In order to generate a CSPROTP that is usable by the POTPSS scheme, we must be able to:

1. Generate a PROTP equal in length to the plaintext message.

2. Seed the generation of future PROTPs with keys that are shared by both communicating parties.

3. Ensure that once the PROTP is strengthened into a CSPROTP, it is capable of resisting cryptographic attacks.

The first item above is satisfied by using a PRBG that is capable of accepting a length as an input parameter and producing a PROTP equal in length to the plaintext message. The length of the message is always known, as the ciphertext is always equal in length to the plaintext message.

The POTPSS protocol (which will be described in detail later) must have the ability to generate identical PROTPs at both the sending side (to encrypt the message) and at the receiving side (to decrypt the message). The PRBG must be able to accept a seed that is available to both parties, thus satisfying the second requirement above.

PRBGs generating PROTPs are cryptographically weak in that they posses properties which render them useless for cryptographic applications [22]. In order to generate a CSPROTP (thus satisfying the last item above), therefore, the CSPROTPs must possess the following properties/feature:

Property A. Period *(p)* of PRBG or Cryptographically Secure PRBG (CSPRBG) must be longer than the length *(l)* of the plaintext message. This ensures that there is no cycle in the ciphertext as can happen when $p < l$.

Feature B. The keys used to seed the PRBG must produce a unique PROTP and not a PROTP generated by selecting an offset from a larger PROTP of period *(p)* indexed by that seed.

Property C. Given a sequence of bits $(m_{0..i})$ generated by the PRBG in the PROTP, bit m_{i+1} must remain unpredictable for all i.

Most PRBGs are capable of producing sequences equal to the word length of the computer being used. For a typical system with a 32-bit word, the effective period is 2^{31} (the last bit is usually thrown out). This by far exceeds the typical message length that would be exchanged via the POTPSS scheme and as such satisfies the first (Property A) requirement above.

PRBGs use initialization registers where the seeds are entered. The seed values are then reduced via a hashing function to an initial value and then the bit generation commences. These initial values are not pointers to a master PROTP but initial values that are used to actually generate the PROTP. This satisfies the second (Feature B) requirement above.

The third (Property C) requirement and its resolution will be the focus of the remainder of this section. Predictability is the characteristic that must be cryptographically minimized in order to produce a CSPROTP from a PROTP. CSPROTPs are perfect for stream ciphers such as POTPSS. The output of these generators (CSPROTPs) is indistinguishable (at least, in any reasonable amount of time) from TROTPs [23]. PRBGs such as Linear Congruential Generators (LCG) [22] have been cryptographically attacked [23]. These attacks are not based on the period of the pseudo-random sequences but rather on the next-bit predictability of the PROTP, making PRBGs unsuitable for cryptographic applications.

In the POTPSS scheme, we take the PROTP generated from a PRBG and cryptographically strengthen it in order to generate a CSPROTP. It has been proven ([24], [25], [26], [27]) that it is possible to generate a CSPROTP using DES. POTPSS implements this approach as follows:

1. The length of the plaintext message ($M_{xy}n$) is determined and fed into the PRBG along with the seed values PW_{ab} and $K_{xy}n$. This produces a PROTP equal in length to the plaintext message.

2. DES_3 encryption is invoked with a password equal to the concatenation of PW_{ab} and $K_{xy}n$. The input to the DES encryption is the PROTP generated in step 1 above. The output is a CSPROTP.

3. The plaintext message ($M_{xy}n$) is then cryptographically XORed with the output generated in step 2 above, and the output of this process is the ciphertext message ($C_{xy}n$).

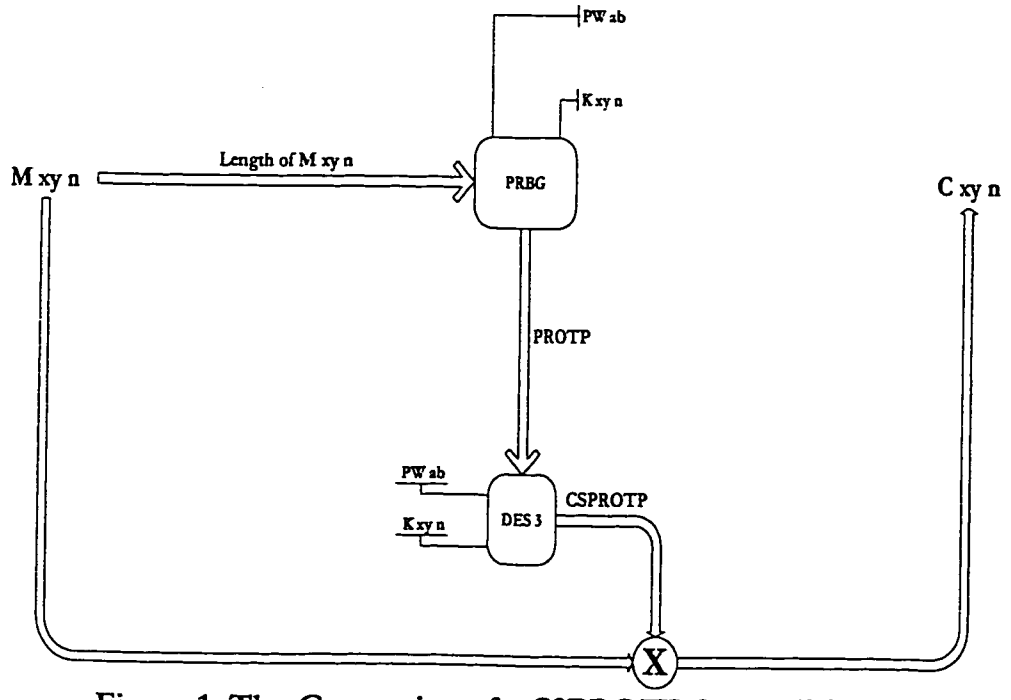


Figure 1: The Generation of a CSPROTP from a PRBG

4.1.1.2 Security

This section discusses the method used in order to generate a CSPROTP from a PROTP generated by a PRBG. In brief, this is done using multiple calls to the DES algorithm with secret keys and a PROTP.

Sadeghiyan and Pieprzyk [27] discuss how PRBGs "are deterministic algorithms which generate binary strings that look truly random". These PRBGs pass a given statistical test if the results are similar to TRBGs. If the statistical results are not similar, then it is said that PRBGs fail some statistical test. "However, even if a generator passes all known statistical tests, it is possible to predict the next bit knowing some previous ones" [27]. PRBGs are indistinguishable from TRBGs, "if and only if, knowing the first s bits of the string, predicting the $s + 1$ bit is difficult, in other words the next bit is unpredictable" [27].

Sadeghiyan and Pieprzyk [27] discuss the possibility of constructing a super pseudo-random permutation from a single pseudo-random function, where six rounds of DES-like permutations and six references to the pseudo-random function are required. This would translate to six calls to the PRBG algorithm (LCG) and six calls to the DES encryption algorithm (effectively two DES_3 calls), thus generating a CSPROTP from a PRBG. Lee, Kim, and Choi [28] also argue that it is possible to construct a super pseudorandom permutation generator even by applying a minimum of five rounds of DES.

A PRBG with three rounds of DES-like permutations and three different random functions is deemed secure against a chosen plaintext attack, when a cryptanalyst can ask for a polynomial number of plaintexts [27].

In the POTPSS scheme, we use a PRBG provided by J. Walker [29]. This PRBG generates four pseudo-random sequences based on the LCG and then uses the lower order bit in a Linear Feedback Shift Register (LFSR). Although LCGs have been known to be cryptographically weak [22], this organization of the LCG and LFSR satisfies the requirements of the three different random functions as described by Sadeghiyan and Pieprzyk [27]. The POTPSS scheme then performs three rounds of DES encryptions as depicted in the diagram above using DES_3 encryption. This also satisfies the other half of the requirements for security against a chosen plaintext attack as described by Sadeghiyan and Pieprzyk [27].

The POTPSS scheme addresses and satisfies the requirements of super pseudo-randomness by effectively performing the entire encryption process twice. As can be seen in Figures 2A - 2D, the entire CSPROTP is generated twice, effectively doubling the number of DES encryptions and the number of references to PRBG.

The next-bit predictability property of the CSPROTP has, therefore, been effectively removed by using multiple calls to the DES_3 algorithm with a set of keys and a PROTP.

Given algorithm:

> *DES_3(Key, Message), where: Key is the encryption key and Message is the message to encrypt.*

and

> *PRBG(l($M_{xy}n$), Key), where: l($M_{xy}n$) is the length of $M_{xy}n$ and Key is the encryption key.*

Message $M_{xy}n$ is encrypted into $C_{2xy}n$ as follows:

Intermediate (1st round) encryption results in ciphertext $C_{1xy}n$:

> *For every i where: 0 <= i <= l($M_{xy}n$), $C_{1xy}n_i$ = DES_3($PW_{ab} \bullet K_{xy}n$, PRBG(l($M_{xy}n$), $PW_{ab} \bullet K_{xy}n$))$_i \oplus M_{xy}n_i$*

Final (2nd round) encryption results in ciphertext $C_{2xy}n$:

> *For every i where: 0 <= i <= l($M_{xy}n$), $C_{2xy}n_i$ = DES_3($K_{xy}n \bullet PW_{ab}$, PRBG(l($M_{xy}n$), $K_{xy}n \bullet PW_{ab}$))$_i \oplus C_{1xy}n_i$*

$C_{2xy}n$ is the ciphertext that is then transmitted to the recipient.

DES_3 is a cryptographically strong algorithm. The output of the DES_3 algorithm is essentially a random function of every plaintext bit and every key bit. Just as any message cannot be computationally decrypted using DES_3, so too the output of

the PRBG. It too cannot be decrypted and, as a result, predicted. The randomness property of DES_3 ensures that the CSPROTP remains pseudo-random.

Since PW_{ab} and $K_{xy}n$ are the only variables in the equations above, these two parameters are the only means to attack this scheme. This is more than sufficient for the POTPSS scheme because it would represent a brute force attack by attempting all possible permutations of PW_{ab} and $K_{xy}n$. As a result, the CSPROTP leaves no other method of attack available other than by brute force. To increase the security against brute force attacks further, we could increase the length of PW_{ab} and $K_{xy}n$, thereby increasing the level of security of the CSPROTP.

4.1.1.3 Conclusion

We have shown that it is possible to take a simple, cryptographically weak PRBG and produce a cryptographically secure super pseudo-random sequence of bits (in the form of a CSPROTP) for use in the POTPSS stream cipher. We have also shown that the CSPROTP is secure against chosen plaintext and chosen ciphertext attacks.

A CSPROTP, therefore, enables us to approach the state of perfect secrecy explained earlier. Our CSPROTP provides a possible way to solve this problem as follows: assuming that two communicating parties agree to share the seed, the two parties could both compute the same CSPROTP for use in the encryption and decryption process. The seed functions as a key, and the PRBG can be thought of as a keystream generator for the stream cipher required in the POTPSS scheme [5].

4.1.2 Data Encryption Standard

4.1.2.1 Overview

The purpose of the DES algorithm in the POTPSS scheme is to cryptographically strengthen PROTPs into CSPROTPs by encrypting the PROTP with two secret keys, thus generating a CSPROTP. As can be seen in Figure 1, the output of the PROTP is fed into the DES_3 algorithm in order to generate the CSPROTP that is then XORed with the message $M_{xy}n$, thus generating the ciphertext $C_{xy}n$.

The DES_3 algorithm is only used to encrypt and is never used to decrypt the PROTP. It simply randomizes the PROTP, thereby removing the next bit predictability property. This process must also be repeated at the recipient's side in order to generate an identical CSPROTP required to XOR with the ciphertext $C_{xy}n$ in order to generate the message $M_{xy}n$. This is illustrated in Figures 2A - 2D. This section will be a brief discussion of how the DES_3 encryption scheme generates the CSPROTP.

4.1.2.2 Security

Since DES is not a group [12], multiple encryptions effectively strengthen the scheme. DES_3 requires 2^{112} attempts, while simple DES requires 2^{56} attempts in order to effectively break it. The output is a random function of every plaintext bit and every bit in the key [7].

In the POTPSS scheme we feed two keys into the DES_3 algorithm: PW_{ab} and $K_{xy}n$. The DES_3 algorithm requires two keys in order to perform the three necessary steps to successfully encrypt the PROTP. DES_3 works as follows:

1. The sender encrypts with the first key (k_1).

2. The sender then decrypts with the second key (k_2).

3. The sender then encrypts with the first key (k_1).

This is summarized as follows:

$$C = E_{k1}\,(\,D_{k2}\,(\,E_{k1}\,(\,P\,)\,)\,),\ \textit{where: (C)iphertext, (P)laintext}$$

The POTPSS scheme generates two CSPROTPs: $CSPROTP_1$ and $CSPROTP_2$. Both of these are used to encrypt the plaintext message $M_{xy}n$. Message $M_{xy}n$ is encrypted into $C_{2xy}n$ as follows:

Intermediate (1st round) encryption results in ciphertext $C_{1xy}n$:

$$C_{1xy}n = M_{xy}n \oplus CSPROTP_1$$

Final (2nd round) encryption results in ciphertext $C_{2xy}n$:

$$C_{2xy}n = C_{1xy}n \oplus CSPROTP_2$$

$C_{2xy}n$ is the ciphertext that is then transmitted to the recipient where:

$$CSPROTP_1 = E_{PWab}\,(\,D_{KxyN}\,(\,E_{PWab}\,(\,PROTP_1\,)\,)\,)\ \textit{and}$$

$$CSPROTP_2 = E_{KxyN}\,(\,D_{PWab}\,(\,E_{KxyN}\,(\,PROTP_2\,)\,)\,)$$

Note: $PROTP_1$ and $PROTP_2$ represent the first and second generation of OTPs respectively from the PRBG as can be seen in Figure 1. E and D represent the DES encryption and DES decryption functions respectively.

DES_3 has two keys, k_1 and k_2, each one 56-bits in length for an effective key length of 112-bits. Increasing the key length in POTPSS would not increase the

effective key length of DES_3 as only 56-bits of the key (k_1 and k_2) are used and the remainder is discarded. As a result, the maximum security that DES_3 provides is equal to the maximum security provided by a 112-bit key. This equates to 2^{112} attempts in order to recover the original message (in this case the PROTP) from the ciphertext message (in this case the CSPROTP).

4.1.2.3 Conclusion

In this section we have shown the process of generating a CSPROTP from a PROTP. Using DES_3 encryption and a pair of keys (PW_{ab} and $K_{xy}n$), we cryptographically strengthen the PROTP, thus making it a CSPROTP. This is done by removing the next bit predictability liability from the PROTP. The recovery of the PROTP from the CSPROTP without the given keys would effectively require 2^{112} attempts. This represents a computationally infeasible task which ensures that the PROTP cannot be derived from the CSPROTP.

4.1.3 Secure Hash Algorithm

4.1.3.1 Overview

SHA was designed by the NIST along with the National Security Agency (NSA) as part of the Secure Hash Standard (SHS). SHA generates a 160-bit message digest from an input message. The message digest possesses properties that make it ideal for cryptographic applications. These properties and how they are used in POTPSS are discussed in this section.

SHA is used in the POTPSS scheme to generate a message digest, or signature, of a message that is exchanged between two communicating parties. The message digest is stored in the CKR. In the POTPSS scheme, the number of sub-digests

that we request from SHA is determined by the length of the message itself and, as a result, cannot be readily guessed by the cryptanalyst.

4.1.3.2 Security

SHA accepts a message of any length less than 2^{64} bits as its input and produces a 160-bit message digest as its output. The POTPSS scheme determines how many of those bits will be used. The original plaintext message is analyzed, and the POTPSS selects a number of sub-digests and inserts these selections into the CKR. The entries in the CKR serve as keys (K_{xyn}) to both encrypt and decrypt plaintext messages and ciphertext messages respectively (see Figures 2A - 2D for more details). SHA requires that a common password (PW_{ab}) be used in the generation of the message digests. This ensures that even though someone may know the contents of the message, generating the message digest (which would generate the keys required to encrypt and decrypt future messages) becomes a computationally infeasible task.

SHA is deemed secure because it is designed to be computationally infeasible to recover a message from its corresponding message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with a very high probability, result in a different message digest, and the signature will fail to verify [30]. The 160-bit hash produced by SHA is longer than other available hash functions, making it considerably more resistant to brute force attacks, including birthday attacks. The most important aspect of SHA is that "there are no known cryptographic attacks against SHA" [9].

SHA is used in the POTPSS scheme to append message digests to the CKR as follows:

$$CKR_{d1\text{-}n} \mathrel{+}= f(\, M_{xy}n,\ SHA(\, PW_{ab},\ M_{xy}n\,)\,)$$

where:

$d_{1\text{-}n}$: *digest(s) (1-n) generated by function f appended to CKR*

f: function determining number of message digests to select

$M_{xy}n$: *message exchanged between communicating parties*

PW_{ab}: *password negotiated between communicating parties*

SHA: SHA generating digests based on PW_{ab} *and* $M_{xy}n$

Since there are no known methods of attack other than by brute force, the 160-bit hash represents an effective level of computational security in the order of 2^{160}. This is considerably higher than other hashing standards and, as a result, is used in the POTPSS scheme to generate keys that are added to the CKR. The keys are in turn used as keys ($K_{xy}n$) in future transactions.

4.1.3.3 Conclusion

SHA generates the CKR in order to constantly generate the cryptographic keys used in the POTPSS scheme. The actual keys in the CKR are not directly generated by the individuals using POTPSS but indirectly from the message digests between two communicating parties. SHA enables the linkage property of the POTPSS scheme (discussed shortly) by generating cryptographically strong signatures used as part of the key for future transactions.

4.1.4 Cryptographic Key Ring

4.1.4.1 Overview

The CKR, as described earlier, stores components of the message digests generated by SHA. The CKR is used to form the keys used to seed the PRBG that will then generate a PROTP. To generate a PROTP, entries are inserted into the CKR whenever a message is transmitted or received. The number of entries inserted per message is determined by the message itself. This was done in order to ensure:

1. confusion in the selection of message digests from the CKR for any party except those possessing the original plaintext message $M_{xy}n$.

2. that there is no runaway key condition where more keys are generated than needed, therefore overloading the CKR.

3. that there are enough available keys for future transactions without having to wait for a response from the other party.

4.1.4.2 Security

The CKR stores entries as follows:

user: bob atob: af4jq10z btoa: ljq1inx0

Assuming we are looking at Alice's CKR, we see that she has registered user *bob* and she possesses two keys. These two keys are typical values generated from SHA. The *atob* key will enable Alice to encode (send) a message to Bob with $K_{xy}n$ equal to "af4jq10z" concatenated with PW_{ab}. The *btoa* key will enable Alice to

decode (receive) a message from Bob with $K_{xy}n$ equal to "ljq1inx0" concatenated with PW_{ab}. Typical key lengths are 64-bits.

To attack the key $K_{xy}n$ stored in the CKR without knowledge of the original plaintext message, a cryptanalyst would have to resort to brute force. This represents an attack in the order of 2^{64} attempts. However, this method of attack would have to be performed for every single message, making it computationally infeasible for more than a trivial number of messages.

4.1.4.3 Conclusion

The CKR stores the message digests generated by SHA that are used as seeds to the PRBG to generate the PROTPs which are in turn cryptographically strengthened into CSPROTPs. The choice of an indeterminate number of sub-digests is done deliberately in order to confuse the selection. It is computationally infeasible to attempt to guess the message digest in the CKR for every transaction as would be required in POTPSS.

4.2 The POTPSS Protocol

4.2.1 Description of Protocol

4.2.1.1 Overview

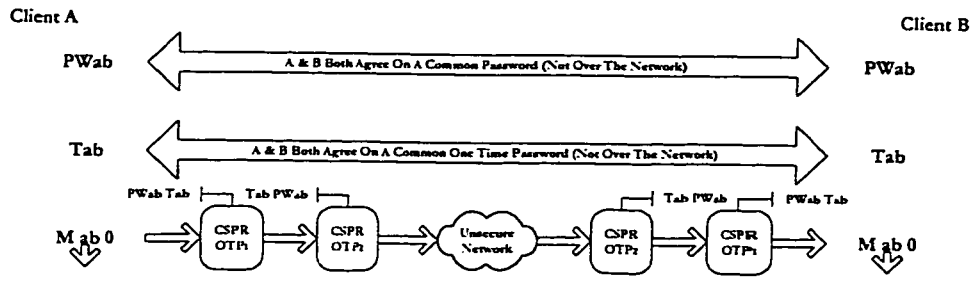
The POTPSS scheme is a protocol utilizing lower level encryption algorithms in order to achieve a higher level of security. Focus is placed on the cryptographic strengthening of the PROTP into a CSPROTP, as this is the pivotal component of the POTPSS scheme.

The POTPSS scheme is a stream cipher that reads a plaintext message and converts it to ciphertext one bit at a time. This is different from block ciphers such as DES which operate on 64-bit blocks at a time. The scheme or protocol incorporates lower level algorithms in order to ensure:

- **Integrity**: A ciphertext message could not have been modified in transit.
- **Authentication of Origin**: A message claimed to have been sent by a particular user could in fact only have been sent by that user and nobody else.
- **Confidentiality**: If in fact a ciphertext message was intercepted in transit, the eavesdropper could not have recovered the message.
- **Linkage**: Message exchanges between any two communicating parties are inter-linked such that message y is linked with some component of message x.

These characteristics of the POTPSS scheme represent core capabilities. They are in turn guaranteed by employing algorithms and components such as SHA, CKR, DES, and CSPROTPs.

4.2.1.2 Protocol Operation



Mxyn	Message x sent to y; n = sequence number of message x to y or y to x.
PWab	Negotiated password for communications between client a and client b.
Kxyn	CKR hash value for message betwen x and y; n = sequence number.
Tab	One time throw away password to bootstrap communications between a and b.
Note:	Tab and PWab are not permitted to be the same.

Figure 2A: Diagram of the POTPSS Protocol - Bootstrap Phase

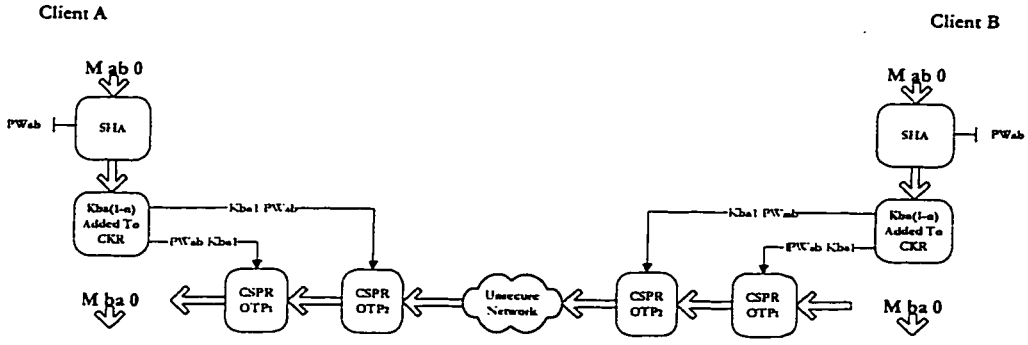


Figure 2B: Diagram of the POTPSS Protocol - Initial Response Phase

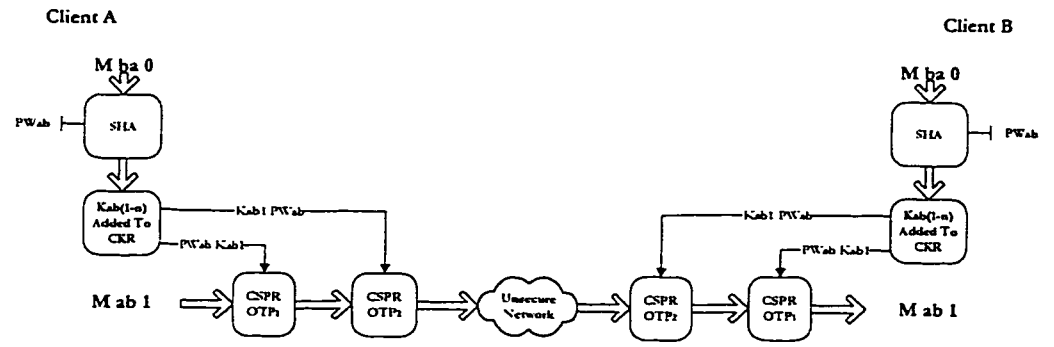


Figure 2C: Diagram of the POTPSS Protocol - First Post-Bootstrap Message

Figure 2D: Diagram of the POTPSS Protocol - Subsequent Messages

Figures 2A - 2D demonstrate the major POTPSS components and the overall functionality of the POTPSS protocol.

The POTPSS protocol requires an initial bootstrapping for communication between two parties (client *a* and client *b*) by registering T_{ab} into the CKR as follows:

$$CKR_{d0} = T_{ab}$$

Once this is done, client *a* can send client *b* a message as follows:

$$C_{ab}0 = M_{ab}0 \oplus CSPROTP_1 \oplus CSPROTP_2$$

where:

$C_{ab}0$ *is the ciphertext transmitted to client b from client a*

$M_{ab}0$ *is original plaintext message client a sends to client b*

$CSPROTP_1 = DES_3(PW_{ab} \bullet T_{ab}0, PRBG(l(M_{ab}0), PW_{ab} \bullet T_{ab}0))$

$CSPROTP_2 = DES3(T_{ab}0 \bullet PW_{ab}, PRBG(l(M_{ab}0), T_{ab}0 \bullet PW_{ab}))$

PW_{ab} *password negotiated for communication*

At the same time, client *a* generates keys in order to add them to the CKR as follows:

$$CKR_{d1\text{-}n} \mathrel{+}= f(M_{ab}0, SHA(PW_{ab}, M_{ab}0))$$

where:

$d_{1\text{-}n}$: *digest(s) (1-n) generated by function f appended to CKR*

f: function determining the number of message digests to select based on message $M_{xy}n$

The CKR now contains keys K_{ba}(1-n) that will be used to seed the PRBG in generating a CSPROTP capable of decrypting the next message from *b* to *a*.

At the receiving side, client *b* is capable of decrypting the received ciphertext as follows:

$$M_{ab}0 = C_{ab}0 \oplus CSPROTP_2 \oplus CSPROTP_1$$

After having decrypted $M_{ab}0$, client *b* also generates keys to add to the CKR in order to encrypt future messages for client *a*. Client *b* does this as follows:

$$CKR_{d1\text{-}n} \mathrel{+}= f(M_{ab}0, SHA(PW_{ab}, M_{ab}0))$$

Once T_{ab} is used up in the initial transaction, there will be sufficient keys contained in the CKR to satisfy transmissions from *a* to *b* and from *b* to *a*. T_{ab} was used to bootstrap the initial message exchange and enable the generation of keys in the CKR for future messages.

4.2.1.3 Typical Protocol Execution

Alice and Bob need to exchange a secret password, PW_{ab}, over a secure channel of communication. This channel could be in person, a secure telephone call, or some other medium. At the same time, Alice and Bob could agree on T_{ab}. This one-time throw-away password will be used to bootstrap the POTPSS protocol and will be discarded immediately afterwards. T_{ab} must not be equal to PW_{ab}. If this happens, it will have the net effect of canceling the CSPROTP encryption after the second round, which effectively sends the original plaintext message.

Once these passwords have been agreed upon, Alice would then send Bob a message. Alice invokes POTPSS with message $M_{ab}0$. $M_{ab}0$ is fed into the $CSPROTP_1$. $CSPROTP_1$ is seeded with PW_{ab} and T_{ab}. A ciphertext is generated and then fed into $CSPROTP_2$ which is seeded with T_{ab} and PW_{ab} presented in an inverted order. The ciphertext that is generated, $C_{ab}0$, is then transmitted to Bob.

At the same time that $M_{ab}0$ is submitted to $CSPROTP_1$ for encryption, it is also submitted to SHA in order to generate a series of keys (determined by the content of the message) to be added to the CKR. PW_{ab} is used as the key to the SHA function. This ensures that on Bob's side the same key must be used in order to generate the ciphertext from $CSPROTP_1$ and $CSPROTP_2$ for messages that Bob will send Alice. SHA will generate a number of hashes of $M_{ab}0$. The protocol selects the number that will be entered into the CKR. From Alice's point of view these keys are added to the list of keys for communication with Bob. The keys stored in the CKR at this point represent the keys that will be used when Bob sends a message to Alice. At this point, there are insufficient keys for Alice to send Bob another message, without Bob responding to Alice's first message.

When Bob receives the ciphertext message $C_{ab}0$ from Alice, Bob knows the PW_{ab} and T_{ab} required in order to decipher $C_{ab}0$ into $M_{ab}0$. The following steps are performed by the protocol: T_{ab} and PW_{ab}, in that order, are fed into $CSPROTP_2$, and the ciphertext is then fed into $CSPROTP_1$ with PW_{ab} and T_{ab} as seeds. The plaintext message $M_{ab}0$ is then recovered at Bob's end.

The process is not finished, though, because the protocol still needs to generate the keys for the CKR that will be used for messages Bob will wish to send Alice. Therefore, $M_{ab}0$ is fed into SHA along with PW_{ab}. SHA will generate a number of hashes of $M_{ab}0$ and the protocol selects the number that will be entered into the CKR. This CKR will then be used to encode $M_{ba}0$ in the next round. The same number of keys is selected, since the number is a function of $M_{ab}0$ which both Alice and Bob have. The keys found in both Alice's and Bob's CKR at this point are identical.

Once this initial step is complete, T_{ab} is no longer required and can be discarded. There are now enough keys in the CKR generated from $M_{ab}0$ that can be used to encrypt messages Bob may wish to send to Alice.

If Bob wants to send Alice a message $M_{ba}0$, he takes a key out of the CKR and along with PW_{ab} feeds it into $CSPROTP_1$ and $CSPROTP_2$. The ciphertext that is generated, $C_{ba}0$, is transmitted to Alice. At the same time he ensures that there will be enough keys for future communications. Therefore, SHA is called with $M_{ba}0$ and PW_{ab} and, as a result, a new series of keys are added to Bob's CKR.

When Alice receives Bob's encrypted message $C_{ba}0$, she removes a key from the CKR and uses it along with PW_{ab} to seed $CSPROTP_1$ and $CSPROTP_2$. The plaintext message $M_{ba}0$ is then recovered. At the same time, Alice generates a series of keys to add to her CKR. These keys, which Bob already has, will be used by Alice to encrypt $M_{ab}1$ in the next message Alice will send Bob.

This process continues until either Alice or Bob has run out of keys in his/her CKR or until there is a breach of security. If Alice runs out of keys (probably due to the fact that she has sent many more messages than she has received), she will have to wait until Bob sends her a message which will in turn create a series of keys that will be added to her CKR. The same scenario applies to Bob. If, on the other hand, there is a breach in security, there will be no way for Alice or Bob to exchange any messages until the POTPSS scheme is bootstrapped once again.

4.2.1.4 Security of the POTPSS Protocol

The objective of developing the POTPSS protocol was to ensure that the only method of successful cryptographic attack would be by brute force in order to discover the keys used to encrypt and decrypt messages between two parties.

We begin our discussion of the security of the POTPSS protocol with an analysis of an attempt to breach authentication. This will not be a detailed cryptanalytic attack but will illustrate the properties of the POTPSS protocol which thwart such attacks.

Authentication is a property which ensures that a message could not have been sent by anyone else except the person claiming to have sent it. Assume there exists a third person (client *c*) who wishes to send a message to client *b* and claim it is from client *a*. Client *c* claims to be *a* and sends a message to *b* as follows:

$$C_{cb}n = E(M_{cb}n, PW_{ab}, K_{ab}n)$$

If *c* attempts to guess both PW_{ab} and $K_{ab}n$ and transmit $C_{cb}n$ based on the selection, *b* will receive $C_{cb}n$ and attempt to decrypt it in order to generate $M_{ab}n$ (remember, *b* thinks he's receiving messages from *a*). However, if either PW_{ab} or $K_{ab}n$ is guessed incorrectly, *b* will recover an incorrectly decrypted message and will signal a breach in security. If, however, both PW_{ab} and $K_{ab}n$ were guessed correctly (a highly improbable event) client *a* and client *b* will no longer share identical entries in their CKRs and the next message *b* will send *a* will fail to decrypt properly.

Client *c* will have a single chance to guess both PW_{ab} and $K_{ab}n$ correctly. With typical key lengths for PW_{ab} equal to eight characters (used here for simplicity, in fact it could be much longer) and for $K_{ab}n$ also equal to eight characters, there exist 2^{128} possible permutations for a brute force attack. This number represents a very small chance to select the correct keys. As a result, authentication of origin is computationally guaranteed in the POTPSS scheme.

Integrity is the property that ensures that a message could not have been modified in transit. Assume that client *c* wishes to modify client *a*'s transmission to client *b* after having intercepted the message. This would be done as follows:

$$M_{ab}n = D(C_{ab}n, PW_{ab}, K_{ab}n)$$

There are two methods of attacking the integrity of a message. The first is to decrypt the message, replace it, and then re-encrypt the message and send it off to its intended recipient. The second is simply to modify bits in the ciphertext $C_{ab}n$, without knowing the effect the changes will have. The first attack on integrity represents a formidable task given the time constraint (before *a* suspects something is going on, since there is no reply from *b*) and the fact that the key space is identical to the one discussed previously. Even if this were possible, client *c* would have to find another message that would generate the identical hash value of the message transmitted (this was discussed in detail previously where it was observed that there have been no known attacks of this kind against SHA). If this is not done, then the next message sent by *b* to *a* will simply not decrypt with the key stored in the CKR. The second method of attack is a futile attempt to simply corrupt a message in transit which will inevitably be detected by client *b* when the message is decrypted into nonsensical characters. This is known as a denial of service attack. Also, since a message digest is generated from the nonsensical message, it will conflict with the message digest generated by client *a*. The conflicting entries in client *a*'s and client *b*'s CKRs will result in the next message failing to decrypt properly. This would represent a breach in security and be detected immediately.

Confidentiality represents the security of the message even if it is intercepted. In other words, client *c* cannot read communications between client *a* and *b*. Assume that client *a*'s message to client *b* at the time of interception is as follows:

$$M_{ab}n = D(C_{ab}n, PW_{ab}, K_{ab}n)$$

Again, trying to recover $M_{ab}n$ from $C_{ab}n$ represents a formidable task as discussed previously. Therefore, we can say that recovering $M_{xy}n$ from $C_{xy}n$ is computationally infeasible as long as PW_{ab} and $K_{xy}n$ remain secret. In the event PW_{ab} is known, $K_{xy}n$ will still remain secret for every message, since it is unique for each transaction. This implies that the only method of attack in this case is once again by brute force, representing an attack against a 64-bit key that would involve 2^{64} operations. The scheme is still cryptographically strong given that such an attack would have to be repeated for every message transmitted. Also, this method of attack still assumes that PW_{ab} is known, a problematic assumption.

Linkage is a property that requires that message y be linked with message x assuming that message y is transmitted after message x. In POTPSS the message digest generated by message x is stored in the CKR and is used as a seed to generate the CSPROTPs that encrypt message y. As such, message z would also be linked to message y assuming message z is transmitted after message y. The linkage property in the POTPSS scheme ensures that there is always one unknown key in the generation of the CSPROTPs. Dynamically changing keys in every transaction severely limits the effectiveness of a brute force attack, since the keys generated from a brute force attack are never all the same for every new message.

4.2.1.5 Conclusion

By ensuring message integrity, authentication of origin, confidentiality, and linkage, the POTPSS scheme provides a level of security that leaves no method of attack other than by brute force in order to recover and transmit messages as an impostor. This scheme is therefore computationally secure because it is unforgiving and would abort any transmission deemed an attack. The POTPSS

protocol is susceptible to denial of service attacks in that any attempt to alter a message in transit would require re-negotiation of bootstrap passwords between any two communicating parties.

Chapter 5

Cryptanalysis of POTPSS

5.1 Methods of Attack

This section discusses the methods which a cryptanalyst has at his / her disposal in order to attack any cryptographic scheme including the POTPSS scheme. The application of these methods to an actual attack on the POTPSS scheme will be covered in the subsequent section.

5.1.1 Ciphertext-Only Attack

This method of attack assumes that the cryptanalyst has intercepted the ciphertext of several messages. The ciphertext must have been generated using the same encryption algorithm for each plaintext message. The cryptanalyst must recover as many of the original plaintext messages as possible or ideally deduce the key used in the encryption of the original plaintext message(s) into the intercepted ciphertext message(s). If the key is recovered, then the cryptanalyst can decrypt other future messages encrypted with the same key [9].

Given:

$$C_1 = E_k(P_1),\ C_2 = E_k(P_2),\ \ldots..\ C_n = E_k(P_n)$$

Where:

C_i = *intercepted ciphertext i*

E_k = *encryption algorithm with key k*

P_i = *original plaintext i*

Deduce:

$P_1, P_2,...P_i$, *k, or an algorithm to infer* P_{i+1} *from* $C_{i+1} = E_k(P_{i+1})$

In the POTPSS scheme, the ciphertext is the only part of the message that is conveyed in an unsecure fashion across an unsecure network. As a result, this is the most obvious way to attack the scheme as messages can be readily intercepted.

5.1.2 Known-Plaintext Attack

This method of attack assumes that the cryptanalyst has not only intercepted the ciphertext of several messages but also has the plaintext to accompany those messages. The cryptanalyst must deduce the key used in the encryption of the messages in order to decrypt future messages or must deduce an algorithm to decrypt any new messages encrypted with the same key [9].

Given:

$P_1, C_1 = E_k(P_1), P_2, C_2 = E_k(P_2), ... P_n, C_n = E_k(P_n)$

Deduce:

$$k, \text{ or any algorithm to infer } P_{i+1} \text{ from } C_{i+1} = E_k(P_{i+1})$$

In the POTPSS scheme, the cryptanalyst must not only intercept the transmitted ciphertext message(s) but also the accompanying original plaintext message(s). This requires a more systematic attack than merely snooping unsecure packets over an unsecure network. It requires some form of keystroke intercept or system-level access in order to intercept the message prior to encryption and then following decryption in the case of a received message. Although possible, it is highly improbable due to the level of trust required in order to have such access to system resources. Since the interceptor in this type of attack already has access to the contents of the original messages (i.e., since the plaintext is known), the objective would be to acquire the keys used in order to impersonate one of the two communicating parties. In the POTPSS scheme we prevent this type of attack via message linking.

5.1.3 Dictionary Attack

A dictionary attack is an attack whereby a reduced keyspace is used in order to break a given cryptosystem. This type of attack is based on the observation that when people choose their own keys, they generally choose poor ones [9]. English words, names, and trivial extensions to English words are commonly used as keys. A dictionary attack is a brute force attack whereby a dictionary of common keys is used in order to break a given cryptosystem.

The POTPSS scheme is susceptible to this type of attack, but only through the common password that is agreed upon (PW_{ab}) by the two communicating parties. The keys used to generate the CSPROTP require two keys. The first is PW_{ab}, and the second is $K_{xy}n$ which is extracted from the CKR. The second key, $K_{xy}n$, is

automatically generated and is not one that would be found in a dictionary of common keys.

5.1.4 Chosen-Plaintext Attack

This method of attack requires that the cryptanalayst not only has the ciphertext and associated plaintext of several messages, but can also selectively determine the encrypted plaintext. This method is more powerful than the known-plaintext attack, because cryptanalysts can choose specific plaintext blocks to encrypt, ones that might yield more information about the key [9]. The cryptanalyst must deduce the key used to encrypt the messages or use some algorithm that will enable the decryption of any new messages with the same key.

In the POTPSS scheme, the cryptanalyst must not only intercept the transmitted ciphertext message(s) but must also selectively intercept the accompanying original plaintext message(s). This method is more difficult to implement than the known-plaintext attack, as it requires the ability to select the message(s) that is (are) encrypted.

5.1.5 Adaptive-Chosen-Plaintext Attack

This method of attack assumes that the cryptanalyst can not only choose the plaintext that is encrypted but can also modify the choice based on the results of the previous encryption [9]. The objective of this method of attack is to allow the cryptanalayst to selectively encrypt a smaller block of plaintext and then choose another based on the results of the first. This is a special case of the chosen-plaintext attack.

In the POTPSS scheme, the cryptanalyst must not only intercept the transmitted ciphertext message(s) but must also selectively intercept the accompanying

original plaintext message(s) and be able to selectively encrypt smaller blocks of the plaintext. The cryptanalyst can iteratively choose to select other blocks based on the results of the first. This method is more difficult to implement than the known-plaintext attack, as it requires the ability to select the message(s) that is (are) encrypted.

5.1.6 Birthday Attack

This attack is applicable to one-way hash functions. Basically, this attack entails finding two messages that hash to the same hash value. This is more subtle than the other attack on one-way hash functions which entails finding a message M_1 that would hash to a value equivalent to message M_2.

5.2 POTPSS Component Cryptanalysis

This section discusses each key component in the POTPSS scheme and which suitable method of attack can be applied in order to cryptanalyse it.

5.2.1 Cryptographically Secure Pseudo Random OTP

The CSPROTP is the key component of the POTPSS scheme and as such is discussed in detail in this section. The objective of this section, therefore, is to outline how to effectively break the CSPROTP. Since the DES_3 algorithm is used in the generation of the CSPROTP, it too is briefly discussed in this section. However, it will be covered in greater depth later on. There are three major types of attacks that can be performed against the CSPROTP. Each of these will be discussed followed by a summary.

5.2.1.1 Ciphertext-Only Attack

In the ciphertext-only attack the cryptanalyst only has the intercepted ciphertext in his/her possession. The objective is to derive the plaintext message, extract the key used to encrypt the message or to define an algorithm which will enable him/her to decrypt future messages.

Given:

$C_{xy}n$ = *intercepted ciphertext*

Deduce:

$M_{xy}n$ = *original plaintext message*

Assuming that the DES_3 generates a CSPROTP, there is no effective means to deduce $M_{xy}n$ other than by iterating through all the possible keys that could be used for PW_{ab} and $K_{xy}n$.

5.2.1.2 Known-Plaintext Attack

In the known-plaintext attack the cryptanalyst only has the intercepted ciphertext and plaintext message(s) in his/her possession. The objective is to extract the key(s) used to encrypt the message(s) or to define an algorithm which will enable him/her to decrypt future messages.

Given:

$C_{xy}1$, $C_{xy}2$, ... $C_{xy}n$ *intercepted ciphertext(s)*

$M_{xy}1$, $M_{xy}2$, ... $M_{xy}n$ *original plaintext message(s)*

Such that:

$$M_{xy}n = D(\ C_{xy}n,\ PW_{ab},\ K_{xy}n\ )$$

$$D = decryption\ function\ (POTPSS\ scheme)$$

$$PW_{ab} = password\ negotiated\ for\ secure\ communication$$

$$K_{xy}n = key\ retrieved\ from\ CKR$$

Deduce:

$$M_{xy}n+1 = subsequent\ original\ plaintext\ message$$

Assuming that the DES_3 generates a CSPROTP, there is no effective means to deduce the key(s) used to encrypt $M_{xy}n$ into $C_{xy}n$ other than by iterating through all the possible keys that could be used for PW_{ab} and $K_{xy}n$.

5.2.1.3 Dictionary Attack

A dictionary attack is one whereby a known lexicon of English words and their variations are used as input into an algorithm which will attempt to break the secret key by trying every word in the lexicon or dictionary. In the POTPSS scheme the dictionary attack is applicable to the CSPROTP when used to attack PW_{ab}. Since this is a user selectable password, it can be chosen from an entry in the dictionary. Poor selection of PW_{ab} will increase the probability that a dictionary attack will break PW_{ab}.

5.2.1.4 Summary

The security of the CSPROTP is derived from the cryptographic strengthening that the DES_3 algorithm provides when a PROTP is converted into a CSPROTP.

5.2.2 Data Encryption Standard

The DES_3 algorithm is responsible for generating the cryptographic security in the CSPROTP. It accepts as input parameters a PROTP with a period that far exceeds the message space and performs a DES_3 encryption on it with two keys, PW_{ab} and $K_{xy}n$. The result is a CSPROTP.

5.2.2.1 Ciphertext-Only Attack

The DES_3 algorithm does not encrypt the actual message, but in fact encrypts a PROTP into a CSPROTP. Consequently, performing a ciphertext-only attack on the DES_3 algorithm will require an analysis of the output which in turn itself does not reveal any information except next-bit predictability. The only effective means of attack on the DES_3 algorithm in a ciphertext-only way is by brute force; in other words, trying every possible combination of the two keys, PW_{ab} and $K_{xy}n$.

5.2.2.2 Known-Plaintext Attack

In a known-plaintext attack the task of guessing the PROTP from the CSPROTP is again assumed by the cryptanalyst. The maximum key length in this situation is 112-bits. The maximum number of bits required to recover the PROTP from the CSPROTP is 112-bits. There are no effective attacks except via brute force. However, this is only for the first round of the CSPROTP. This operation has to be performed twice in order to generate the second CSPROTP. Furthermore, even if the PROTP was recovered, the task of generating the seeds to the PROTP still remains. This too can be done by brute force, but yields results that are no

better than those obtained by attacking the entire scheme by brute force. The only effective means of attack on the DES_3 algorithm in a known-plaintext way is by brute force; in other words, trying every possible combination of the two keys, PW_{ab} and $K_{xy}n$.

5.2.2.3 Dictionary Attack

A dictionary attack can be performed on the DES_3 algorithm. However, it provides results that are no better than those obtained by a dictionary attack on the CSPROTP.

5.2.2.4 Summary

The only known effective means of attacking the DES_3 algorithm is by performing a brute force attack on PW_{ab} and $K_{xy}n$. This, however, provides cryptographic results that are no better than those obtained by applying the same brute force attack on the CSPROTP.

5.2.3 Secure Hash Algorithm

SHA generates a message digest given a plaintext message and a password (PW_{ab}) for subsequent messages. There are two types of attacks that could apply to the POTPSS scheme concerning SHA. The first is a brute force attack, and the second is the Birthday attack. Both are discussed in detail.

5.2.3.1 Brute Force Attack

A brute force attack on the POTPSS scheme would attempt to find a message M_2 that would hash to the same value as message M_1. The purpose of this type of attack would be for a cryptanalyst to intercept a message between two parties, and replace it with a message that would hash to the same value as the intercepted

message and then forward the message to the intended recipient. This would be done in order to inject a message in the correspondence between two individuals, as follows:

Given:

$C_{xy}n$ = *intercepted ciphertext*

Deduce:

$M_{zy}n$ = *plaintext message to be transmitted*

Such that:

$SHA(\ M_{zy}n\ ) = SHA(\ M_{xy}n\ )$ *and* $length(\ M_{zy}n\ ) = length(\ M_{xy}n\ )$

In order for the cryptanalyst to perform such an attack, he/she would have to first decode the message, itself a computationally infeasible task. Second, the cryptanalyst would then have to find a message of length equal to the intended message to ensure hashing to the same hash value as the original message. This is required in order to generate the same CSPROTP. This again is a computationally infeasible task [9].

5.2.3.2 Birthday Attack

The Birthday attack attempts to find two messages that hash to the same hash value as opposed to already having a message and trying to find another message which would generate the same hash value as described in the previous section. In the POTPSS scheme, this is not a practical form of attack because it would mean that messages being attacked are the same ones transmitted between two parties, rather than attacked by a third party since the third party has no control over

sending a message on behalf of either one of the two original parties. This still remains a computationally infeasible attack and in the POTPSS scheme requires extremely special circumstances to implement [9].

5.2.3.3 Summary

There are no known cryptographic attacks against SHA and, as a result, it does not introduce any vulnerability into the POTPSS scheme.

5.3 Comparison with Other Encryption Schemes

5.3.1 Comparison of Results

The purpose of this section is to compare the POTPSS scheme with other encryption schemes from the point of view of cryptographic strength. We begin our discussion with a brief introduction to computational complexity, keyspace considerations, and the time needed to attack a scheme with a given keyspace.

The computational complexity of an encryption scheme is determined by the size of the key used to encrypt a plaintext message into a ciphertext message. In our discussion, the key varies in length from 7-bytes (56-bits) in DES to 17-bytes (136-bits) in the 9 character PW_{ab} version of POTPSS. If there is no other suitable form of attack other than by brute force, every possible permutation of the key must be attempted in order to successfully attack the cryptosystem. The longer the key, the more computations are required in order to successfully break the cryptosystem [7]. Certain encryption algorithms (such as DES) have a fixed size key and it cannot be increased without altering the algorithm and without having adverse effects. These schemes have been optimized to operate with certain pre-defined key lengths and their overall security could be potentially weakened by modifying the algorithm to allow larger key lengths [5]. The idea behind the

POTPSS scheme is to be able to add more security instantly simply by selecting a longer key without having to alter any algorithm. This is done by two users simply re-exchanging PW_{ab} with a longer key length at any time. This can be done at any time as long as there are no outstanding messages between the two users that need to be decoded.

Passwords (secret keys) are selected in order to be easy to remember. These keys usually correlate with digram and trigram distribution charts, and rarely make use of all the available character set. As such, the keyspace, or the total set of characters that could be used in selecting a password (key), seldom makes full use of the American Standard Code for Information Interchange (ASCII) character set. Consequently, this reduces the computational resources needed to successfully attack a cryptosystem.

As we can see from Table 2, the keyspace limitations effectively reduce the total amount of time required to attack an encryption scheme. Table 2 identifies the keyspace and the various encryption schemes chosen for consideration. The first column considers only lower case characters (26). The second column considers lower case characters and digits (36). The third column considers all alphanumeric characters (62). The fourth column considers all printable characters (95). The fifth column considers the 7-bit ASCII character set (128) and finally the sixth column considers the 8-bit ASCII character set (256). The first row considers the POTPSS scheme with an 8 character password. The second row considers the POTPSS scheme with a 9 character password. The third row considers the DES scheme with a 56-bit key. The last row considers the DES scheme with a 112-bit key.

The time taken to effectively attack a given scheme with a given keyspace is calculated as follows:

$$Time\ to\ attack = N^x / (1\ MIPS * 365.25 * 24 * 60 * 60)$$

Where:

N = *number of keys in keyspace*

x = *number of characters in key*

Note: The calculation assumes that a computer with 1 Million Instructions per Second (MIPS) is being used and that each successive attempt to attack consists of 1 instruction per second [9]. A MIPS year is the total number of instructions which a 1 MIPS computer can perform within a given year. Assuming each attack can be reduced to one instruction per second, Table 2 demonstrates the amount of time required to perform attacks on the various schemes with varying key lengths. Obviously, as the amount of time increases, so does the strength of the scheme. The first entry in a row of the table represents the search space required in order to iterate through the given key length. The second and third entries represent the amount of time given 1 MIPS year at your disposal.

	Lowercase Letters (26)	Lowercase Letters & Digits (36)	Alphanumeric Characters (62)	Printable Characters (95)	ASCII (128)	8-bit ASCII (256)
POTPSS	26^{16}	36^{16}	62^{16}	95^{16}	128^{16}	256^{16}
PW_{ab} 8 Char.-Key	$1.38*10^{9}$ years	$2.52*10^{11}$ years	$1.51*10^{15}$ years	$1.39*10^{18}$ years	$1.65*10^{20}$ years	$1.08*10^{25}$ years
POTPSS	26^{17}	36^{17}	62^{17}	95^{17}	128^{17}	256^{17}
PW_{ab} 9 Char.-Key	$3.59*10^{10}$ years	$9.08*10^{12}$ years	$9.37*10^{16}$ years	$1.32*10^{19}$ years	$2.11*10^{21}$ years	$2.76*10^{27}$ years
DES	26^{7} 133.86 minutes	36^{7} 21.77 hours	62^{7} 40.76 days	95^{7} 2.21 years	128^{7} 17.84 years	256^{7} 2283.37 years
DES_3	26^{14} $2.04*10^{6}$ years	36^{14} $1.95*10^{8}$ years	62^{14} $3.93*10^{11}$ years	95^{14} $1.55*10^{14}$ years	128^{14} $1.00*10^{16}$ years	256^{14} $1.65*10^{20}$ years

Table 2: Keyspace Comparison Against Other Schemes

We see from the results above that even under a limited keyspace environment, the POTPSS scheme fairs quite well in ensuring the security of plaintext messages. DES was developed in order to be easily implementable in hardware and can achieve incredibly fast speeds in both the encryption and decryption processes. The POTPSS scheme is considerably slower due to the intensive computations required to generate the PROTP and then to cryptographically strengthen it (six DES encryptions). Therefore, this effectively increases the computation time and as a result would take much longer to attack than indicated in Table 2 above. However, it is intended to serve as a measure against other encryption schemes and of the impact of a limited keyspace.

Chapter 6

Experimental Results of the POTPSS Prototype

6.1 Implementation Details

6.1.1 Cryptographically Secure Pseudo Random OTP

The CSPROTP is generated by taking the output of a PRBG (i.e., the PROTP) and cryptographically strengthening it using DES_3 encryption. The implementation of DES will be discussed in the subsequent section. This section highlights the implementation details of the PROTP generation via the PRBG.

The PRBG used was the OTP generator made available by J. Walker [29] on the World Wide Web. From the source code comments, it incorporates four LCGs in a LFSR fashion in order to generate a PROTP with a period equal to the word size of the machine employed (32-bits). The PRBG is capable of accepting several input parameters and is also capable of formatting the output in order to satisfy the requirements of POTPSS. Input parameters of interest to us include the length of the PROTP that will be generated, as well as a seed which will enable the generation of the identical PROTP on the recipient side. Output formatting of interest to us includes the ability to produce a PROTP in a single stream as opposed to a stream broken down into blocks, which would then have to be filtered.

The POTPSS implementation opens in binary format the message to be transmitted, counts the number of ASCII characters, and then closes the file. With this information, the PRBG is called upon to generate a PROTP of length equal to the message file just opened. It is also seeded with the two necessary seeds by simply taking the second seed and appending it to the first. A PROTP is generated and piped to a file. The file pointer is then passed to DES_3 along with the two seeds (PW_{ab} and K_{syn}). The PROTP file is thereby cryptographically strengthened into a CSPROTP.

Once this process is complete, the CSPROTP file pointer and the original message file pointer are passed to a routine which performs a bit-wise XOR of every ASCII character in the message along with the CSPROTP. The CSPROTP is 8 bytes longer than the message file at this point because DES_3 introduces characters as part of the encryption process; however, these characters are ignored. All intermediate files are deleted in order to remove any traces of information which could potentially be used to re-generate the PROTP and compromise security.

This entire process is performed twice in order to satisfy the requirements of the POTPSS protocol. However, the seeds are inverted in order to generate unique PROTPs and to encrypt with a different DES_3 key. This is necessary in order to generate different PROTPs.

6.1.2 Data Encryption Standard

The DES implementation used in the POTPSS protocol is a DES_3 implementation of the Data Encryption Standard. Its purpose in POTPSS is to accept a PROTP and cryptographically strengthen it by performing a DES_3 encryption given two secret keys which are appended to one another.

The DES_3 encryption is invoked from the command line with arguments prepared in the POTPSS application. Identical to the PRBG generating the PROTP, they are not implemented as application program interfaces (APIs), but rather are invoked from a command line interface. The output generated is piped to a file which is then used by the POTPSS application.

The input file to the DES_3 application is immediately destroyed, once the CSPROTP is generated, in order to eliminate intermediate files. The DES application is used to perform encryption in a single direction. In other words, it is never used to decrypt text. Rather, it is used only to encrypt PROTPs into CSPROTPs.

6.1.3 Secure Hash Algorithm

SHA used in POTPSS was provided by Bruce Schneier in the source code that accompanies his book [9]. The purpose of SHA is to generate a message digest up to 160-bits given an input message and a password that acts as the key to SHA. The message digests that are generated are then inserted into the CKR for use in future transactions.

SHA is implemented as a module in the POTPSS source code and is called directly (rather than as an external system call like the one employed in DES and the PRBG).

SHA generates a message digest of 160-bits in length given an input message. The number of sub-digests selected is an arbitrary value between one and five. The reason why the entire message digest is not selected was discussed previously. The number selected in that range is determined by the length of the message. Other techniques can easily be employed such as the checksum of the message, etc.. The

length of the message was chosen simply because this information was readily available.

6.1.4 Cryptographic Key Ring

The CKR stores the sub-digest generated from SHA. Its purpose is to ensure that the keys are inserted under the appropriate user and appropriate category. There are essentially two types of keys: ATOB keys and BTOA keys. ATOB keys are used to encode messages A wants to send B. BTOA keys are used to decode messages B wants to send to A. The CKR is loaded into memory at the beginning of the POTPSS application and is written to disk once POTPSS has completed the generation of SHA sub-digests, at which point it reflects any and all changes made to the CKR.

6.2 Analysis of Ciphertext

The objective of all encryption schemes is to generate a ciphertext message that provides no information that can help the cryptanalyst break a given encryption scheme. In order to evaluate the effectiveness of the POTPSS scheme, a series of tests has been drawn up in which the ciphertext will be carefully analyzed. The entire basis of the TROTP is that a message XORed with it will generate a truly random ciphertext. The statistical tests which the POTPSS scheme (as well as other schemes) will be subjected to will help determine the effective randomness of the ciphertext message. According to Knuth [6] "we apply about half a dozen kinds of statistical tests to a sequence, and if it passes these satisfactorily we consider it to be effectively random".

The tests which were selected include entropy, optimum compression, arithmetic mean, serial correlation, Monte Carlo value for Π and chi square distribution.

These test programs were developed by J. Walker [31] in order to evaluate pseudorandom number generators for encryption and statistical sampling applications. The tests were performed on three encryption schemes (algorithms and protocols): DES, PGP and POTPSS. The data (i.e. message space) used in order to generate the random ciphertext message include the following:

Message Containing
100 pseudo-random characters
1,000 pseudo-random characters
10,000 pseudo-random characters
100,000 pseudo-random characters
1,000,000 pseudo-random characters
1,000 consecutive "A"s
270,336 byte binary file
578 character ASCII English message

Table 3: Message Space Characteristics

The pseudo-random characters were generated using the PRBG identical to the one used in the POTPSS scheme. These messages were each encrypted with the aforementioned encryption schemes, and the results are tabulated in Tables 4 through 9. Each test is discussed in its respective section, followed by a presentation of the results and a detailed discussion of the meaning behind the results. The objective of these tests is to show that the POTPSS scheme is superior in effectively randomizing the ciphertext message, thus making cryptanalysis more difficult.

6.2.1 Entropy

Maximum entropy signifies the number of bits required to represent an ASCII character (i.e., 8). It is essentially the information density of the contents of a file

expressed as the number of bits per character [32]. The entropy test represents the number of bits that would be required in order to represent the sequence of characters being analyzed. The range in the entropy test would be from 0 to 8 bits, 0 implying that 0 bits would be required to represent the sequence to a maximum of 8, where the sequence is perfectly random. It is therefore highly desirable for pseudorandom generators or encrypted files or messages to approach 8, as is seemingly more random and less susceptible to cryptographic attack. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	4.71553	4.72220	4.70448	4.70056	4.70008	0.00000	5.67421	4.24514
DES	6.45459	7.80273	7.98238	7.99829	7.99981	7.79086	7.99938	7.66543
PGP	7.20480	7.72759	7.97098	7.99678	7.99967	7.00066	7.99836	7.58185
POTPSS	6.34807	7.82052	7.97838	7.99839	7.99986	7.81060	7.99921	7.67572

Table 4: Analysis of Ciphertext - Entropy

We see from Table 4 that the plaintext message, no matter what form it takes, scored poorly. This, of course, was expected. 1000 A's had an entropy of 0, also as expected. The POTPSS scheme scored higher in entropy of the ciphertext in 5 of the 8 tests performed. We see weaknesses when the message length is small (i.e. 100 characters in length).

6.2.2 Optimum Compression

If a message is perfectly random, it cannot be compressed because there are no patterns that can be effectively reduced [33]. This test attempts to optimally compress text in order to determine its randomness. The total compression possible (measured in percentage) is displayed followed by the number of characters in the message that were used to compress the text. When the message

is small in size, we see some variations in the effective compression such as those in the 100 character PGP compression. This is due to the fact that when the message is small, PGP effectively expands the message size.

For all ciphertext messages, the total compression ideally should be 0%, representing a truly random sequence of characters that have no repetitious patterns. In this test, the first entry in the row represents the percentage compression while the second entry represents the number of characters in the original message. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	41 %	40 %	41 %	41 %	41 %	100 %	29 %	46 %
	106	1,006	10,006	100,006	1,000,006	1,000	270,336	578
DES	19 %	2 %	0 %	0 %	0 %	2 %	0 %	4 %
	112	1,008	10,008	100,008	1,000,008	1,008	270,344	584
PGP	9 %	3 %	0 %	0 %	0 %	12 %	0 %	5 %
	260	813	6,429	62,657	624,503	188	114,051	491
POTPSS	20 %	2 %	0 %	0 %	0 %	2 %	0 %	4 %
	106	1,006	10,006	100,006	1,000,006	1,000	270,336	578

Table 5: Analysis of Ciphertext - Optimum Compression

We see from the results above that the plaintext message scored higher in terms of percentage of compression, again as expected. The POTPSS scheme always scored amongst the lowest in terms of percentage compression in comparison with the other encryption schemes except with a small message (100 characters).

6.2.3 Arithmetic Mean

The purpose of the arithmetic mean test is to find the median of all characters encoded into a ciphertext. This is simply the result of the summation of all the bytes in the file divided by the file length [31]. The optimal arithmetic mean would be 128 as it is the midpoint between 0 and 255. It shows that the characters in the ciphertext message are more random and less subject to biases than if the

arithmetic mean were higher or lower than 128. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	104.72	109.32	109.44	109.49	109.51	65.00	65.64	93.49
DES	131.36	129.64	127.53	127.60	127.46	125.89	127.35	129.90
PGP	114.44	122.23	127.43	127.30	127.71	128.15	127.30	127.70
POTPSS	127.53	126.57	126.91	127.56	127.53	125.93	127.44	128.43

Table 6: Analysis of Ciphertext - Arithmetic Mean

The POTPSS scheme performed favorably in this test, especially for very small and very large message spaces. However, in the remainder of the tests it was very close to the median and only off by a small amount.

As a result, the POTPSS scheme generates ciphertext messages that are sufficiently random in comparison with other encryption schemes and for theoretical statistical experiments.

6.2.4 Serial Correlation

The serial correlation test indicates the correlation of a character in the ciphertext message with the subsequent or previous character in the ciphertext message. This quantity measures the extent to which each byte in the file depends upon the previous byte [6]. Ideally, in a perfectly random series of characters, serial correlation would be 0.0. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	0.6816	0.3250	0.0535	0.0043	0.0001	N/A	0.2507	-0.0868
DES	0.0232	0.0224	0.0100	0.0006	0.0018	-0.0602	-0.0021	0.0405
PGP	-0.0440	-0.0443	-0.0083	-0.0009	-0.0013	0.1111	0.0009	-0.0635
POTPSS	-0.0531	0.0190	0.0065	0.0005	-0.0009	0.0024	-0.0041	0.0540

Table 7: Analysis of Ciphertext - Serial Correlation

An interesting observation is that the serial correlation for 1000 sequential A's is N/A. This means that they are perfectly correlated as expected.

The POTPSS scheme performs favorably in this test, as it has a serial correlation of the ciphertext bits that is lower than any other scheme. This implies that the POTPSS scheme's ciphertext is more random than that of any other encryption scheme.

6.2.5 Monte Carlo Value for Π

This test determines the value for Π with the smallest deviation possible. Ideally, the Monte Carlo value for Π should be as close as possible to Π with as small an error as possible. Each successive sequence of four bytes is used as 16 bit X and Y co-ordinates within a square. If the distance of the randomly-generated point is less than the radius of a circle inscribed within the square, the four-byte sequence is considered a "hit". The percentage of hits can be used to calculate the value of Π. For very large streams (this approximation converges slowly), the value will approach the correct value for Π if the sequence is close to random [31]. This test determines the value for Π (first entry in a row) and the error (second entry in the row) that was generated. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	3.7309	4.0000
	27.32	27.32	27.32	27.32	27.32	27.32	18.76	27.32
DES	3.1429	3.1429	3.1559	3.1391	3.1438	3.1905	3.1454	3.0411
	0.00	0.00	0.45	0.08	0.02	1.56	0.12	3.20
PGP	3.5077	3.2512	3.1786	3.1320	3.1358	2.9787	3.1818	3.0820
	11.65	3.49	1.18	0.30	0.19	5.18	0.01	1.90
POTPSS	3.0769	3.2191	3.1555	3.1424	3.1449	3.2480	3.1328	3.2778
	2.06	2.47	0.44	0.03	0.10	3.39	0.28	4.33

Table 8: Analysis of Ciphertext - Monte Carlo Value For Π

We see from the results that the POTPSS scheme compared favorably to the other encryption schemes. On two occasions, it generated a value for Π that was quite close to the original value, i.e. the error was quite small. On several occasions, it had small error rates and values for Π that were sufficiently in the range of acceptable values.

6.2.6 Chi Square Distribution

The Chi Square distribution test for a given ciphertext message yields optimum values when the given samples result in a value randomly exceeding a given value 50% of the time. Values close to 0% or 100% should be subject to suspicion, since these figures do not represent random samplings. The chi-square distribution is calculated for the stream of bytes in the file and expressed as an absolute number (first entry in row) and a percentage (second entry in row) which indicates how frequently a truly random sequence would exceed the value calculated [31]. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
Plaintext	1044	8972	88603	885418	8852352	255000	6297497	9836
	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
DES	240	261	245	237	258	272	233	248
	50.00	50.00	50.00	75.00	50.00	25.00	75.00	50.00
PGP	250	273	255	280	286	231	250	249
	50.00	25.00	50.00	25.00	10.00	75.00	50.00	50.00
POTPSS	256	224	298	223	199	264	297	240
	50.00	90.00	5.00	90.00	99.50	50.00	5.00	50.00

Table 9: Analysis of Ciphertext - Chi Square Distribution

As we can see, the POTPSS protocol again faired very reasonably in several tests, especially for small messages, 1000 sequential A's, and text messages. The test, however, returned extreme values in those cases where the message length was extremely large (subject to suspicion).

6.3 Prototype Performance

6.3.1 Benchmarks

The POTPSS scheme was compared with other schemes in order to determine the time required to generate the ciphertext message. The UNIX System V [34] system command time was used in order to generate the user time (first entry in row) and system time (second entry in row). Both are tabulated in Table 10 below. Please refer to Table 3 for a detailed description of the sample text used in this experiment. The results of the test are as follows:

Refer to Table 3	100	1,000	10,000	100,000	1,000,000	1000 A's	Binary	Text
DES	0.010 u	0.050 u	0.080 u	0.550 u	5.260 u	0.010 u	1.430 u	0.020 u
	0.040 s	0.010 s	0.030 s	0.070 s	0.280 s	0.060 s	0.120 s	0.030 s
PGP	0.060 u	0.120 u	0.200 u	1.020 u	8.990 u	0.080 u	2.610 u	0.120 u
	0.140 s	0.150 s	0.120 s	0.270 s	1.200 s	0.130 s	0.360 s	0.100 s
POTPSS	0.200 u	0.190 u	0.270 u	0.190 u	0.140 u	0.330 u	19.44 u	0.300 u
	0.540 s	0.580 s	0.550 s	0.590 s	0.620 s	0.570 s	1.120 s	0.510 s

Table 10: Benchmark Results - Time to Encrypt

We see that the POTPSS scheme is slower than the other encryption schemes. With reasonable messages sizes (1,000 and 10,000), we see that the POTPSS scheme is not too far off the other schemes where user time is concerned. Although the time to encrypt is considerably longer in the POTPSS scheme than in other schemes, it is acceptable for certain applications as will be discussed in the next section.

Chapter 7

Areas of Application

7.1 Electronic Messaging

The POTPSS scheme is ideal for e-mail security in that it provides point-to-point encryption of messages where the message space is not extremely large (due to time required to encrypt) and where the volume of transactions is not excessive.

The ability to invoke the POTPSS scheme from a standard mail application, encrypt the message and then transmit the message makes this "plug-in" ideal for sensitive e-mail transmission requirements.

Although this scheme can be implemented from an end user point of view, it can also be implemented from an organization point of view, where two large organizations that communicate between each other can use point-to-point encryption schemes such as POTPSS between their mail hubs. This eliminates the need for individual users to administer the token registers and the list of users registered to use the POTPSS scheme. A scheme such as the POTPSS scheme can be set up between two large organizations with existing facilities and still make use of unsecure networks in order to transmit sensitive information.

This scheme does not deter individual users from employing the scheme even if it is being used on a more global scale. It does not analyze the plaintext message in any way and, as such, it can be employed repeatedly to achieve the required level of security for individuals and organizations alike.

7.1.1 Implementation Details

In this section we will discuss an e-mail application that has been developed based on the POTPSS scheme. The application is not intended to replace common e-mail applications, but to complement them in order to provide a heightened level of security. The POTPSS e-mail application works in conjunction with standard e-mail applications when messages are received. However, for e-mail transmission, the POTPSS scheme can work stand-alone.

The POTPSS e-mail application is composed of two separate interfaces (one for sending; known as the POTPSS Send interface, and one for receiving; known as the POTPSS Receive interface) which act as a graphical user interface to the core encryption routines. The application also performs all the required validation and all the necessary commands to both encode and decode e-mail messages.

The command line arguments given at invocation time determine which components are assembled and presented to the user in the graphical user interface. When a user wishes to send a message he/she would simply have to invoke the POTPSS Send interface by typing "POTPSS Send". Conversely, when a user wishes to receive a message he/she would simply invoke the POTPSS Receive interface by typing "POTPSS Receive".

7.1.1.1 The Send and Receive Common Interface Components

The POTPSS Send and Receive interfaces are composed of several common areas:

E-Mail Address

The e-mail address area allows the user to enter the destination address of the intended recipient of the e-mail message. There are two ways in which this could

be done. If a recipient has already been added to the CKR, then the e-mail address push-button will display the recipient's complete e-mail address. A recipient can at that point be selected directly from the list. Otherwise, the recipient must be added to the list through the text field. This requires the user to enter the recipients complete e-mail address. When a recipient is entered for the first time, the bootstrap password and re-typed bootstrap password fields will remain active. If an existing user is selected or typed, these fields become insensitive indicating that they cannot be entered. This indicates to the user that the recipient's address has already been bootstrapped into the system. An e-mail address must always be entered in any send or receive session and cannot contain any spaces and/or special characters. Also, the entry of multiple recipients is not permitted. In the event that the e-mail address is left blank, the user will be notified.

Negotiated Password

The negotiated password must be agreed upon by two communicating parties in order to successfully exchange secure messages. The user would enter the password negotiated with the recipient of the message in this field. The characters typed in this field are hidden by "*" characters in order to avoid inadvertent exposure. The negotiated password must be re-typed to avoid any typing errors which may occur. The user is prohibited from copying to or from any other fields. In the event that these passwords do not match, the user will be notified.

Bootstrap Password

The bootstrap password must also be agreed upon by two communicating parties in order to successfully exchange secure messages. This field is used only to initiate the first secure message exchange. Afterwards, sufficient keys are generated by the actual messages themselves, not to require this field to be

entered. If the field is sensitive that means that the user has not been recognized from the existing list of users and therefore requires a bootstrap password to initiate the first secure message exchange. This field behaves similar to the negotiated password such that it is hidden and cannot be copied to or from other fields. The bootstrap password must also not be identical to the negotiated password. The bootstrap password must be re-typed in the field directly below. In the event that these passwords do not match, the user will be notified.

Encrypt / Decrypt File

One of the features of the POTPSS e-mail application is the ability to encrypt and decrypt local files. The encrypt/decrypt feature allows the user to encrypt/decrypt a local file which is specified in the file selection field using DES encryption and a user-specified password. When a file is selected, the user simply has to click on the Encrypt file push-button and the user will be prompted to enter a secret password and will also be asked to re-type the same password for verification. Once the file is encrypted, it can also be sent using the POTPSS scheme or can be saved locally for future reference. The file can be decrypted by specifying the same file name and selecting the Decrypt file push-button and entering the password.

7.1.1.2 POTPSS Send Fields

Subject

The subject field allows the user to enter the subject of the message which is being transmitted to the recipient. Special care must be taken to not reveal too much information in the subject field as this is transmitted in the clear. This field can be empty if not required.

Input File

The input file field allows the user to specify an input file to transmit to the recipient. An input file can be entered directly in the field or can be selected by clicking on the push-button and then selecting a file via the file selector dialog. If a file is entered, and if for any reason, it is unreadable (due to system error, etc..) then a dialog will warn the user of the problem. The input file field is used in conjunction with the encrypt file push-button. When a file is selected, the encrypt file and mail push-button allows the POTPSS scheme to encrypt the specified file and transmit it to the recipient.

Message

The message field allows the user to compose a brief message to the intended recipient. This field is used in conjunction with the encrypt text push-button. When a message is entered, the encrypt text and mail push-button allows the POTPSS scheme to encrypt the composed message and transmit it to the recipient.

7.1.1.3 POTPSS Receive Fields

Input File

The input file field allows the user to specify the file which is to be decrypted. It is important to note that the original file will be overwritten by the newly decrypted file. This is done because, the original file, even if preserved, can no longer be decrypted due to the consumption of the keys in the CKR. Also, the file transformation is transparent to the e-mail application since the filename is identical. This field must be entered in the POTPSS Receive interface. This field

is also validated in order to ensure that a correct and valid filename has been entered. In the event that there is an error, the user will be notified.

7.1.1.4 New POTPSS Exchange Session

A new secure exchange session requires the transmitting user to enter the negotiated and bootstrap password in the POTPSS Send interface. The receiving user must also enter the negotiated and bootstrap password in the POTPSS Receive interface. The transmitting user must enter the intended recipients complete e-mail address in the e-mail address field of the POTPSS Send interface. The receiving user must also enter the originators complete e-mail address in the e-mail address field of the POTPSS Receive interface. A message is composed or a file is selected by the transmitting user. The message is then encrypted and e-mailed to the recipient. The recipient must then select the input file to decrypt.

After this initial exchange is performed, both parties can select each others e-mail address from the e-mail address push-button list. This makes future exchanges trivial, as the only field that must be entered is the negotiated password and the input file or message. Key management for the CKR is performed automatically by the POTPSS scheme.

7.1.1.5 The POTPSS Send (File) Interface

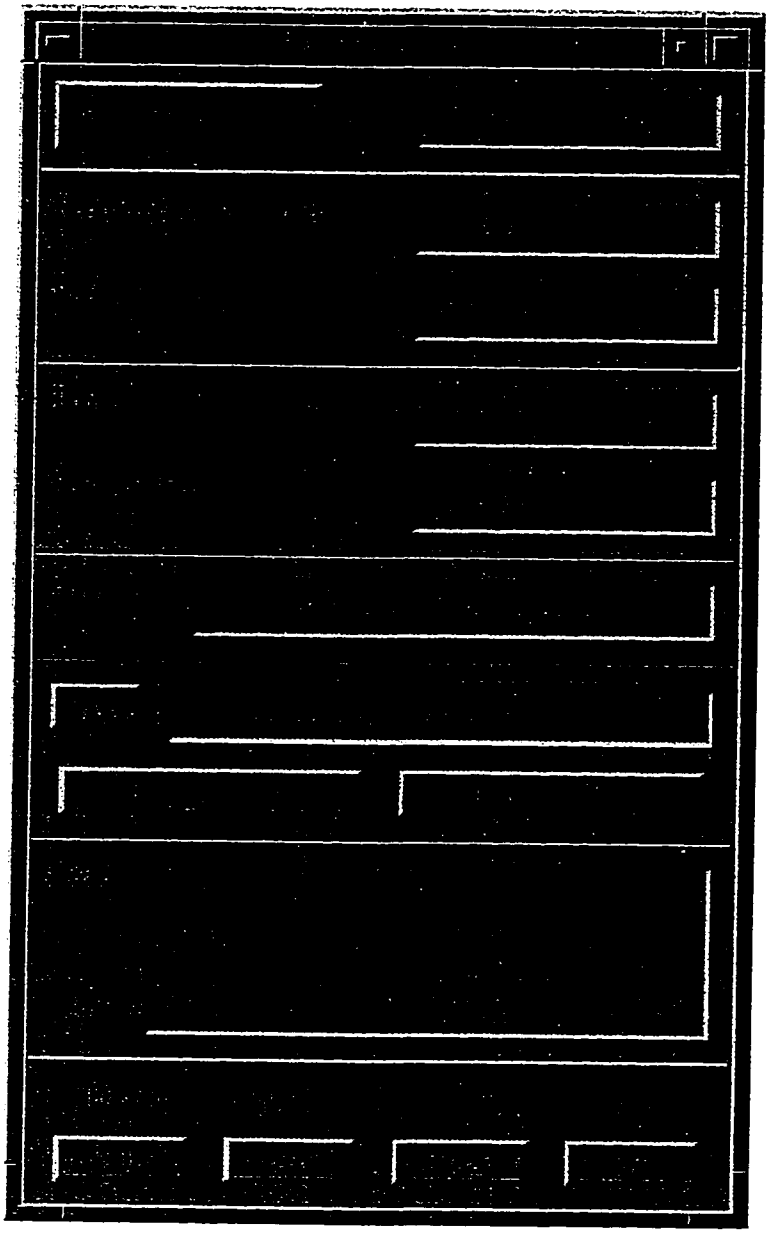


Figure 3: The POTPSS Send (File) Interface

7.1.1.6 The POTPSS Send (Message) Interface

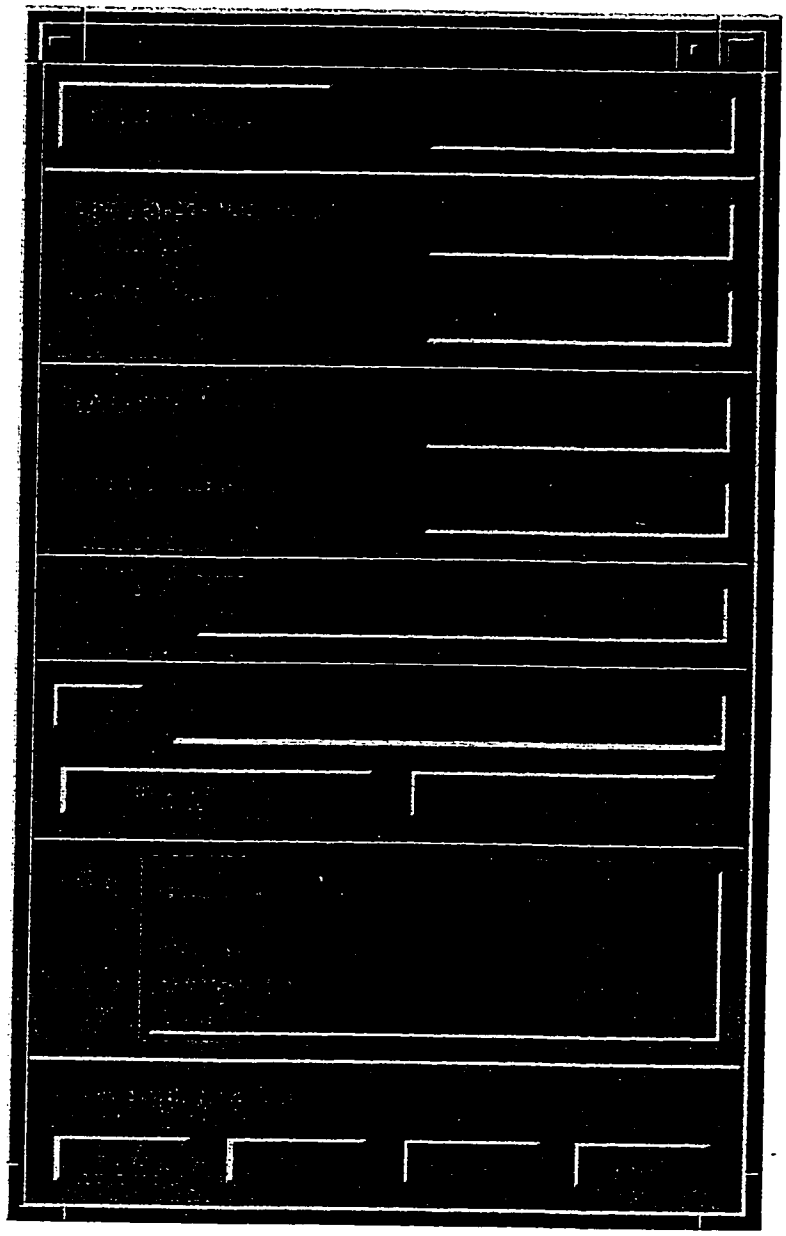


Figure 4: The POTPSS Send (Message) Interface

7.1.1.7 The POTPSS Receive Interface

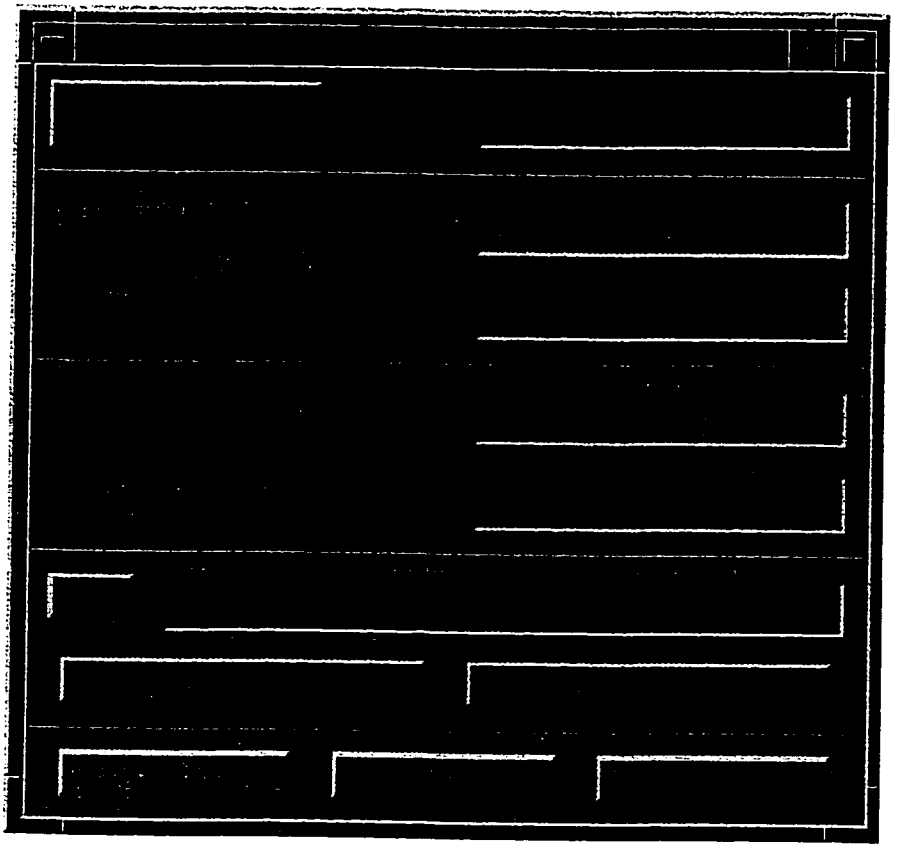


Figure 5: The POTPSS Receive Interface

7.1.1.8 The POTPSS Encrypt File Interface

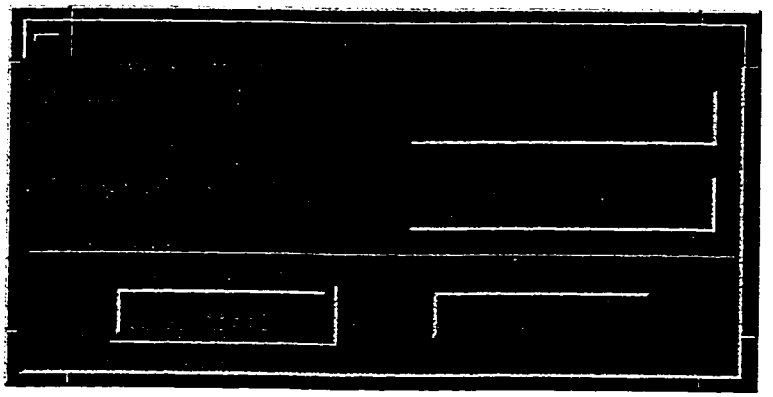


Figure 6: The POTPSS Encrypt File Interface

7.1.1.9 The POTPSS Decrypt File Interface

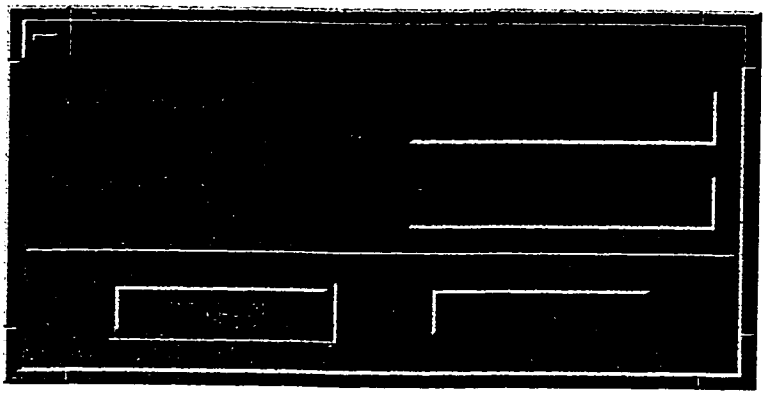


Figure 7: The POTPSS Decrypt File Interface

7.2 Wireless Telephony

With the advent of cellular communications technology, there is a need for enhanced communications security. Telephone communication privacy is increasingly threatened by individuals possessing the technology to listen in on wireless communications.

The POTPSS scheme would have to be employed directly in the communications device via firmware or a hardware upgrade to the phone. Once installed, the POTPSS scheme can be configured by the individual by entering the common password directly in the phone along with the phone number. It is then used to generate a series of CSPROTPs in order to encrypt telephone conversations between two individuals. Since the telephone conversations are synchronous, SHA is only required to generate a single key with every message, requiring less storage space in order to implement this scheme.

The POTPSS scheme would add a level of enhanced privacy to the personal communications systems being used today with encryption determined by the individual user and not the network. As a result, even network administrators cannot listen in on the conversation without the valid key.

7.3 Electronic Commerce

In virtual on-line catalogue applications, the POTPSS scheme can be employed in order to encrypt transactions between the client and the merchant. The client provides method of payment information while the merchant provides the electronic merchandise. The POTPSS scheme can be used by the merchant to keep in the key registry the information about a given client. The client can then in turn keep in his/her key registry a list of merchants whom he/she deals with.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Cryptographic schemes, such as the one presented in this thesis, are only a part of a complete computer security solution. Computer security can be broken down into several categories [35]:

1. Administrative and Organizational Security
2. Personnel Security
3. Physical and Environmental Security
4. Hardware Security
5. Software Security
6. Operations Security
7. Communications Security
8. Transmission Security
9. Cryptographic Security
10. Emission Security

Each category imposes a set of requirements which is dependent upon the required level of security. Cryptographic security is only one aspect in a complete security solution. When developing a secure solution, it is imperative that the other aspects also be addressed.

The Pseudo One-Time-Pad-based Security Scheme presented in this document and discussed in [36], provides the security inherent in the Vernam OTP while enabling distributed generation of the OTPs. By virtue of the cryptographic strengthening of a PROTP into a CSPROTP, the POTPSS scheme is capable of addressing the requirements of cryptographic systems in order to provide a heightened level of security.

The implementation of this scheme shows promising results from the statistical tests performed. Although it is still in a prototype stage, this scheme can be incorporated into various application areas, especially in the communications industry.

8.2 Future Work

The ability to generate a PROTP more efficiently remains an open issue in this research. This would permit increased performance in the implementation of the scheme. Moreover, automatic re-synchronization of the CKRs is not addressed in this scheme. Also, the initial distribution of bootstrap keys can be implemented via a scheme similar to a certificate authority. Finally, a more efficient implementation of the way the DES encryption is performed in the POTPSS scheme can also enhance performance.

References

1. D. Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Macmillan, 1967.
2. R. Kalakota, A. Whinston, *Frontiers of Electronic Commerce*, Reading, Massachusetts: Addison-Wesley Publishing Co. Inc., 1996.
3. P. Wayner, *Digital Cash: Commerce on the Net*, Boston, Massachusetts: Academic Press Professional Inc., 1996.
4. *Answers to Frequently Asked Questions About Cryptography Export Laws*, Available on WWW: http://www.rsa.com/, RSA Data Security, Inc., Redwood City, California, 1996.
5. D. Stinson, *Cryptography: Theory and Practice*, Boca Raton, Florida: CRC Press Inc., 1995.
6. D. Knuth, *The Art of Computer Programming: Volume 2 - Seminumerical Algorithms*, Reading, Massachusetts: Addison-Wesley Publishing Co. Inc., 1969.
7. D. Denning, *Cryptography and Data Security*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.
8. C. Meyer, S. Matyas, *Cryptography: A New Dimension in Computer Data Security*, Kingston, New York: John Wiley & Sons , 1982.
9. B. Schneier, *Applied Cryptography : Protocols, Algorithms, and Source Code in C*, New York, New York: John Wiley & Sons, Inc., 1994.
10. E. Biham, A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Advances in Cryptology - CRYPTO '90 Proceedings, Berlin: Springer-Verlag, 1991, pp. 2-21.
11. A. Konheim, *Cryptography: A Primer*, New York: John Wiley & Sons, 1981.
12. K. Campbell, M. Wiener, *Proof That DES Is Not a Group*, Advances in Cryptology - CRYPTO '92 Proceedings, Berlin: Springer-Verlag, 1993, pp. 518-526.
13. *Answers to Frequently Asked Questions About Today's Cryptography - Version 3.0*, Available on WWW: http://www.rsa.com/, RSA Data Security, Inc., Redwood City, California, 1996.
14. Mastercard Corporation, *Secure Electronic Transaction (SET) Specification, Book 3: Formal Protocol Definition*, Available on WWW: http://www.mastercard.com/, Specification Prepared by VISA & Mastercard Corporations, June 21, 1996.
15. Mastercard Corporation, *Secure Electronic Transaction (SET) Specification, Book 1: Business Description*, Available on WWW: http://www.mastercard.com/, Specification Prepared by VISA & Mastercard Corporations, June 17, 1996.

16. Mastercard Corporation, *Secure Electronic Transaction (SET) Specification, Book 2: Programmer's Guide*, Available on WWW: http://www.mastercard.com/, Specification Prepared by VISA & Mastercard Corporations, June 21, 1996.
17. V. Goradia, et. al. *NetBill: 1994 Prototype*, Carnegie Mellon University, Information Networking Institute, Master of Science in Information Networking Thesis.
18. B. Cox, J. Tygar, M. Sirbu *NetBill Security and Transaction Protocol*, Carnegie Mellon University, Information Networking Institute.
19. G. Simmons, ed., *Contemporary Cryptology: The Science of Information Integrity*, Piscatoway, New Jersey: IEEE Press, 1992.
20. P. Zimmermann, *PGP User's Guide*, 4 Dec 1992.
21. M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton, New Jersey: Princeton University Press, 1996.
22. H. Krawczyk, *How to Predict Congruential Generators*, Advances in Cryptology - CRYPTO '89 Proceedings, New York: Springer-Verlag, 1990, pp. 138-153.
23. T. Beth, Z. Dai, *On the Complexity of Pseudo-Random Sequences - or: If You Can Describe a Sequence It Can't be Random*, Advances in Cryptology - EUROCRYPT '89 Proceedings, Berlin: Springer-Verlag, 1990, pp. 533-543.
24. J. Pieprzyk, *How To Construct Pseudorandom Permutations From Single Pseudorandom Functions*, Advances in Cryptology - EUROCRYPT '90 Proceedings, Berlin: Springer-Verlag, 1991, pp. 140-150.
25. J. Patarin, *New Results on Pseudorandom Permutation Generators Based on the DES Scheme*, Advances in Cryptology - CRYPTO '91 Proceedings, New York: Springer-Verlag, 1992, pp. 301-312.
26. J. Patarin, *How To Construct Pseudorandom and Super Pseudorandom Permutations From One Single Pseudorandom Function*, Advances in Cryptology - EUROCRYPT '92 Proceedings, Berlin: Springer-Verlag, 1993, pp. 256-266.
27. B. Sadeghiyan, J. Pieprzyk, *A Construction for Super Pseudorandom Permutations from A Single Pseudorandom Function*, Advances in Cryptology - EUROCRYPT '92 Proceedings, Berlin: Springer-Verlag, 1993, pp. 267-284.
28. E. Lee, K. Kim, U. Choi, *A Construction of the Simplest Super Pseudorandom Permutation Generator*, Computers and Mathematic Applications, Vol. 29, No. 8, Great Britain: Permagon Press, 1995, pp. 19-25.
29. J. Walker, *One-Time-Pad Generator*, Available on WWW: http://www.fourmilab.ch.
30. *Proposed Federal Information Processing Standard for Secure Hash Standard*, Federal Register, v. 57, n. 21, 31 Jan 1992, pp. 3747-3749.
31. J. Walker, *ENT – A Pseudorandom Number Sequence Test Program*, September 24, 1992, Available on WWW: http://www.fourmilab.ch.
32. R. Hamming, Coding and Information Theory, Englewood Cliffs, NJ: Prentice Hall, 1980.

33. T. Bell, J. Cleary, I. Witten, *Text Compression*, Englewood Cliffs, New Jersey: Prentice Hall, 1990.
34. *Sun Release 4.1 UNIX System V System Command - Time*, 19 September, 1989.
35. Communications Security Establishment (CSE), *Guide to Risk Assessment and Safeguard Selection for Information Technology Systems*, Available on WWW: http://www.cse.dnd.ca/, Specification Prepared by Communications Security Establishment, 1996.
36. B. C. Desai, J. Velissarios, *A Pseudo One-Time-Pad Based Security System*, Ninth Annual Canadian Information Technology Security Symposium Proceedings, Ottawa: Communications Security Establishment, 1997.