CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE COMPUTACIÓN

The Analysis and Implementation of a Practical Crypto-System in the Limited Access Model

Tesis que presenta

René Ernesto Henríquez García

Para obtener el grado de

Maestro en Ciencias

en Computación

Departamento de Computación

Director de la Tesis: **Dr. Debrup Chakraborty**

México, D. F. Octubre 2010

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
COMPUTER SCIENCE DEPARTMENT

The Analysis and Implementation of a Practical Crypto-System in the Limited Access Model

by

René Ernesto Henríquez García

In partial fulfillment of the requirements for the degree of

Master of

Computer Science

Computer Science Department

Thesis Advisor: **Dr. Debrup Chakraborty**

México, D. F. October 2010

Esta tesis se la dedico Al Ser Supremo que siempre ha guiado mis pasos;
a mi bella madre Sonia García, quien es la máxima inspiración y adoración en mi vida y a quien debo cada bit de mi existencia;
a mi ejemplar padre Carlos Henríquez, por ser mi gran referente inculcándome los mejores valores y a quien admiro profundamente;
y a mi país El Salvador a quien represento y llevo con mucho orgullo en mi corazón.

Agradecimientos

Deseo comenzar agradeciendo a mis hermanos Carlos y Eduardo por su inagotable apoyo y cariño, con los que fue un honor haber crecido y que siempre cuidaron de mí. A mi sobrinita María José por su ternura e inocencia que siempre alegró mi corazón.

A mi gran familia presente en varios países y a la que me siento orgulloso de pertenecer, quienes siempre han creído en mí y me han animado a luchar para concretizar cada uno de mis sueños.

A mi prima Lil María que me ha regalado lo mejor de ella en cada momento, apoyándome sin descanso y quien es indiscutiblemente gran responsable en la consecución de mis éxitos hasta el momento. A mi prima Ana María por su tiempo y sus palabras de aliento, recordándome que con dedicación y esmero es posible triunfar en la vida.

A mi primo Francisco Rodríguez-Henríquez, quien siempre fue un ícono para lograr mi superación académico-científica que me he trazado en la vida, y a su apreciable esposa Nareli Cruz por su afecto y cariño incondicionales.

Agradezco con especial cariño a Zayrita Perla, quien me motivó a continuar en momentos de debilidad y de quien aprendí tanto para mejorar como ser humano a través de sus bellas palabras.

A mis amigos en México, quienes en forma desinteresada me ofrecieron su amistad y con los cuales compartí momentos gratos e inolvidables, con especial mención a Jorge González y Francisco Quintanar.

A mis amigos de El Salvador, quienes ciegamente me impulsaron a conquistar mis objetivos y a los que adeudo la fortaleza que me inspiró para alcanzar mis ideales.

A mi asesor el Dr. Debrup Chakraborty quien pacientemente me ha transmitido la calidad de sus conocimientos, influenciándome siempre a dar lo mejor de mí y brindándome su valiosa amistad.

A mis revisores de tesis y profesores durante mi maestría, por su ejemplar enseñanza.

A la Organización de Estados Americanos (OEA), con especial agradecimiento por haberme regalado la invaluable oportunidad de cumplir mi gran deseo de superarme estudiando en el extranjero.

A México, por haberse convertido en mi casa durante este tiempo de estudio abrigándome material, cultural y espiritualmente.

Al CINVESTAV modelo y paradigma de enseñanza superior, por haberse convertido en mi *alma mater*, guía y conductora de mi formación científica.

A Sofía Reza por su nobleza e inigualable espíritu de servicio.

Contents

List of Figures

List of Tables

Resumen

La confidencialidad de la información es un requisito crucial en muchas aplicaciones, donde los datos viajan a través de medios inseguros como el Internet. Hoy en día muchos esquemas de cifrado utilizados en estas aplicaciones, basan su seguridad en suposiciones no demostrables. Dichas suposiciones se refieren a la dificultad computacional de ciertos problemas matemáticos, para los cuales no se conocen algoritmos eficientes que los resuelvan; por tanto, son utilizadas para garantizar la seguridad de los esquemas de cifrado. Debido a que el nivel de conocimiento actual no nos permite aseverar la validez de tales suposiciones, no hay una razón sólida para creer que estos esquemas permanecerán seguros en el futuro. Además, la seguridad brindada por los esquemas actuales es llamada *computacional*, pues sólo se consideran adversarios con poder de cómputo "razonable" que pueden vulnerar su seguridad. Por tanto, la seguridad computacional no considera adversarios con capacidad de cómputo ilimitado.

La alternativa a la seguridad computacional es la denominada *seguridad teórica de la información*. En sistemas de este tipo, la seguridad es garantizada sin suposiciones sobre la dificultad computacional de cualquier problema matemático o la capacidad de cómputo del adversario. Por tanto, son conocidos como sistemas perfectamente seguros. A pesar que estos esquemas proveen el nivel más alto de seguridad, no son prácticos. Esto se debe a que el tamaño de la llave en bits debe ser igual al tamaño en bits del mensaje, y además se debe utilizar una nueva llave por cada mensaje.

El *modelo de acceso limitado* propuesto por Michael Rabin, describe un cripto-sistema *demostrablemente seguro* sin suposición alguna en la capacidad de cómputo del adversario. El objetivo del modelo es superar las limitantes de los esquemas perfectamente seguros. El modelo supone la existencia de una fuente de aleatoriedad distribuída la cual es una red de miles de computadoras dispersas alrededor del mundo, a la que el adversario posee un accesso limitado. Cada computadora genera, almacena y actualiza páginas aleatorias, las cuales son llamadas *Page Server Nodes* (PSNs). Así, dos usuarios que desean comunicarse, hacen uso de una llave inicial k y seleccionan aleatoriamente los mismos PSNs descargando el mismo conjunto de páginas de los servidores. Dado que algunas páginas descargadas pueden ser alteradas, los usuarios utilizan un mecanismo de reconciliación para determinar cuales páginas mantienen en común. Posteriormente, esta aleatoriedad es empleada para comunicarse mediante *one-time pads*.

En esta tesis, describimos de manera precisa una implementación práctica de un esquema de cifrado en el modelo de acceso limitado. En ella mostramos modificaciones sobre una implementación previa de tal esquema, las cuales proveen mejoras en componentes del modelo como el protocolo de reconciliación de páginas y la construcción de la llave inicial k. Además, se sugieren los valores para varios parámetros a fin de lograr una implementación práctica y se discute la seguridad del esquema. Asimismo, presentamos el diseño de un generador de números aleatorios requerido por el modelo y finalmente presentamos los detalles de nuestra implementación.

Abstract

Secrecy of data is an important requirement in many application areas where data travels through insecure channels such as the Internet. Now-a-days, most of the existing encryption schemes used for practical applications rely on un-proven assumptions. These assumptions refer to the computational hardness of some mathematical problems for which there is no knowledge of efficient algorithms to solve them. These assumptions are used to derive security. But given our current state of knowledge nobody is able to prove the validity of these assumptions. Thus, there is no firm reason to believe that these encryption schemes which are based on un-proven assumptions will remain secure forever. Moreover, the security that modern encryption schemes guarantees is computational in nature, i.e., they only consider adversaries with "reasonable" computational power for breaking the security of these crypto-systems. Thus, computational security does not rule out the success of an adversary in breaking a system if (s)he is given unlimited computational ability.

The alternative to computational security is what is known as *information theoretic security*. In an information theoretically secure system, security is guaranteed without any assumption regarding computational hardness of any mathematical problem or the computational power of an adversary. Thus, they are rightly called perfectly secret systems. Even though these schemes provide the highest level of security, they are not practical. This is because the number of necessary key bits is the same as the number of bits in the message, and also one needs to use a different key for each message.

The *limited access model* proposed by Michael Rabin, describes a crypto-system which is *provably unbreakable* without any assumption on the computational power of an adversary. The aim of this model is to overcome the inherent drawbacks encountered in perfectly-secret schemes. But this model makes assumption on the inaccessibility of a distributed source of randomness which is a network of tens of thousands of computers distributed across the globe. Each of these computers generates, stores and updates random pages acting as Page Server Nodes (PSNs). Two communicating users who initially shares a key k uses this key to randomly select the same PSNs downloading the same set of pages from them. Since some pages may be different between users, they also employ a reconciliation mechanism for assuring the pages they have in common. Then this randomness is used to communicate among themselves using one-time pads.

In this thesis we precisely describe an encryption scheme in the limited access model for a practical prototypical implementation. We describe some modifications over a previous attempt, and provide improvements in some components of the model like the page reconciliation protocol and also in the construction of the initial key k. Moreover, we suggest various choices of parameters for a practical implementation and also argue about the security of the scheme. Furthermore, we describe the design of a physical random number generator required for the model and finally provide some implementational details of a prototype which we implemented.

Chapter 1

Introduction

Simplicity is the ultimate sophistication.

Leonardo da Vinci

We live in a connected world. An important basis of the prosperity that we have achieved in the modern days is the ease of communication that current technology offers us. The technology of communication has made the world smaller. Nowadays, it is perfectly normal that an employee in Tokyo maintains regular communication and receives instructions from his boss working at Montreal. The cost of the communication technology has been also reduced a lot over the years and thus, it has become accessible to the common man. A day without internet, emails, mobile phone has become inconceivable to the modern man.

The ease of communication has brought certain new concerns. An important concern being the privacy of the data that gets transmitted through the various communication channels which are mostly public. The connected world has become more dangerous as the users are vulnerable to new kinds of dangers. Now we are concerned to keep confidential the messages that we send or receive and even our identity. An adversary can do much harm if (s)he can intercept the messages that go through the communication channels every day. As an average user also transmits and receives sensitive information like information about his bank accounts etc, thus, privacy in communication which was a concern to the military or the administrators fifty years ago, is a house-hold requirement of today.

There have been tremendous improvements in development of means for secure communication. Cryptography which is the age-old science of secret communication has gained a considerable importance in the modern society, and this science has flourished in the past few decades with the discovery of elegant techniques, which offer solutions to the problem of privacy in communication. In cryptography, the type of security offered by any system can be broadly classified into two categories namely, information theoretic and computational. Information theoretic security is the highest form of security achievable as it does not depend on any assumption regarding the computational power of an adversary. But achieving such kind of security has many bottlenecks and thus it is difficult to have a practically usable system which provides information theoretic security. This thesis develops over some existing techniques to build a practical system which can provide information theoretic security. In the rest of the chapter we discuss some basic issues of cryptography and discuss in detail the scope and contributions of this thesis.

1.1 Modern Cryptography

Historically, the main goal of cryptography is to provide privacy, i.e., two entities could communicate with each other over an insecure channel, in such a way that an intruder would not be able to learn anything meaningful about the content of the messages sent between the entities. However privacy it is not the only service that cryptography can offer. There exist cryptographic schemes which can offer other security services like integrity, authentication, non-repudiation etc. Cryptography is an age-old science, it has been in use for hundreds of years to enable secret communication between entities. Modern cryptography is mathematical in nature, i.e., it involves strict definitions and assumptions and relies upon a broad spectrum of mathematical tools to develop systems to provide security services. Security provided by a cryptographic system largely depends on the assumptions that it makes on the adversary. An adversary is the entity who is supposed to break the system by intercepting the ciphered information. An adversary can perform computations, thus, formally it can be viewed as a computational procedure or an algorithm. The most natural parameter involved with an adversary is its running time, which translates to its computational power, i.e., an adversary with less running time is considered to be more powerful. The security provided by crypto-systems can be classified according to the assumption on the adversary, a crypto-system which assumes nothing about the computatuional power of the adversary, i.e., is secure even if the adversary is allowed to have un-limited computational power is called *information theoretically secure*. On the other hand, most practical crypto-systems makes some assumption regarding the computational power of the adversary and are only secure if it is assumed that the adversary does not have an un-limited running time (for example a cryptosystem may provide the guarantee that it cannot be broken by an adversary in more than say one million years), such cryptosystems are said to provide computational security. Computational security is thus weaker than information theoretic security.

The security of computationally secure systems also depends on some hardness assumptions. A hardness assumption is an assumption involving the computational hardness of certain mathematical problems. It is believed that there are no efficient computational procedure to factorize large integers or to find the discrete logarithm of elements in certain groups. Crypto-systems are developed based on these assumptions. It is important to note that these are assumptions, we do not yet know of any efficient procedure to factorize large integers but there do exist a possibility that in the near future we would have such a method. Assuming something about the running time of an adversary also amounts to some kind of assumption regarding the technology of computation, as enhanced computation technologies can reduce running time of adversaries drastically. Thus, computational security stands on the shoulders of shaky assumptions regarding human ignorance (of efficient computational procedures for certain problems) and limitations of the computational technologies. In contrast, information theoretic security does not depend on any kind of assumptions.

So, it seems clear from the above discussion, that information theoretic security is desirable. But, there are certain practical limitations which prevents practical deployment of information-theoretic schemes. The basic limitation being the size of the secret key and issues regarding reuse of a particular key (these issues are discussed in detail in Chapter 2). For practical deployment we have to be satisfied with computational security and this kind of security works well and seems to be secure for a short period of time.

There have been attempts to develop systems whose security does not depend on assumptions or depends on assumptions very different from the computational power of the adversary. There have been two notable attempts towards this direction: the bounded storage model and the limited access model. The bounded storage model makes an assumption on the storage capacity of an adversary but allows it to have unlimited computational power. The limited access model assumes that an adversary would be unable to gain total access to a huge distributed source of randomness. With these assumptions the mentioned models achieve information theoretic security.

1.2 Scope and Contributions of the Thesis

In this thesis we develop a cryptosystem secure in the limited access model. The *Limited Access Model* introduced by Michael O. Rabin in [1] is an interesting approach that proposes the possibility of a practical *provably unbreakable crypto-system*. It is based on the functionality of a voluntary network of computers, each storing and updating random pages, called as Page Server Nodes (PSNs). The PSNs provide randomness to users who in turn use this randomness to communicate among themselves using the one-time pad encryption (which is known to be information-theoretically secure). The security of the system depends on the assumption that an adversary can monitor only a part of the total number of PSNs which are involved in providing the randomness necessary for the functioning of the crypto-system. The theoretical aspects of the limited access model were proposed in [1]. A practical implementation of the system was first attempted in [2]. There were several issues not addressed in [2], one of them being the generation of the randomness. We propose a better method to generate randomness. Also, we propose some modifications of the protocols involved to improve efficiency and reduce key sizes. Next we describe briefly the contents of the rest of the thesis.

In Chapter 2 we discuss the background of our work. Specifically, we define information theoretically secure systems, and provide the characterization of such systems, from those characterizations we conclude why such a system would not be practical. Then, we discuss some previous works in the bounded storage model and the limited access model. As discussed before, those models do not make any assumptions regarding the computational power of the adversary but they make certain other assumptions.

In Chapter 3 we discuss in details the limited access model. All the protocols involved in the model are explained keeping in mind the ease of implementation. The model described in this chapter deviates a bit from the original description in [1]. In particular, we do some modification in the page reconciliation protocol which helps us to achieve drastic reduction in key sizes. Also we add a message authentication code which enhances security in a certain sense.

In Chapter 4 we discuss the security properties of the proposed scheme. We were unable to provide a strict security definition and the corresponding security proof, but we carefully point out the various possible vulnerabilities of the proposed system and discuss how the system is protected against those vulnerabilities. These arguments about security are also our contribution as to our knowledge no such study has so far appeared in the literature.

In Chapter 5 we discuss about the design of a physical random number generator. As stated earlier, in the limited access model the randomness required for encryption is provided by a network of computers each of which is called a page server node (PSN). Thus, there must be a mechanism to generate high quality random numbers within each PSNs. We propose to use a physical random number generator. This is also a modification over [2], as the work in [2] does not consider physical random number generation. In Chapter 5 we discuss some general issues of random number generation and then describe our design and some experimental results to judge the quality of randomness generated by our generator.

In Chapter 6 we describe in detail a prototypical implementation of the whole scheme and in Chapter 7 we conclude the thesis and provide some directions for future work.

Chapter 2

Background and Previous Work

Never think you've seen the last of anything.

Eudora Welty

During the last two decades, research toward designing a *provably unbreakable* scheme has received an increasing attention from the cryptographic community. Nevertheless, the only way to assure *perfect secrecy* is by using an *information theoretically secure* scheme. In 1949 Claude E. Shannon demonstrated that the so-called *one-time pad* encryption scheme is information theoretically secure [3].

As a result, the one-time pad encryption scheme assures confidentiality even against adversaries without any limitation in their computational capabilities. But this alternative imposes a cruel penalty in the overhead of the scheme, because of the unbounded size of the keys needed to encrypt/decrypt messages. Thus, historically the one-time pad has been considered impractical for all general purposes.

In this regard, there has been a lot of research toward the design of models in which it could be possible to overcome the natural drawbacks of an *information theoretically secure* scheme. All these attempts try to avoid computational assumptions of an adversary, defining the security in terms of some other types of assumptions. Some of these works can be seen in [4, 5, 6, 7, 8, 9, 10].

In this chapter we discuss about the types of security and their characteristics. Then we explicitly define what perfect secrecy means and the one-time pad encryption scheme which is fundamental for our research. We also describe in detail some of the previous models proposed to achieve provably unbreakable crypto-systems, based on assumptions other than the computational ability of an adversary.

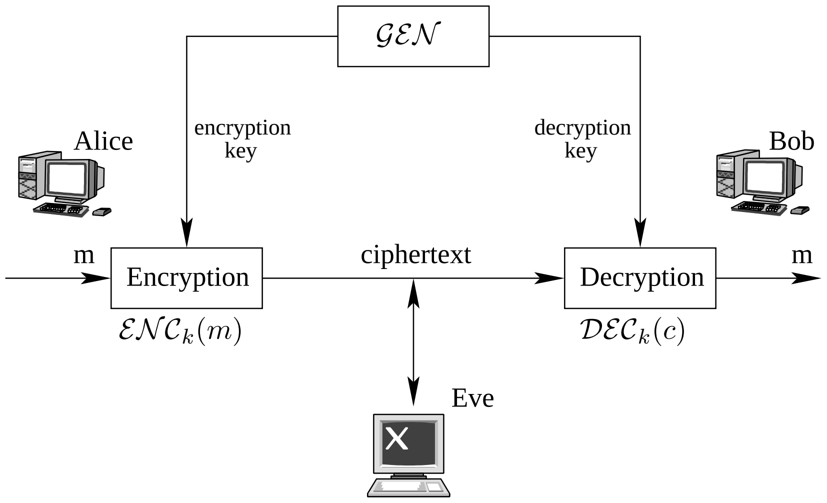


Figure 2.1: The basic setting for a symmetric encryption scheme.

2.1 Types of security

One important contribution (out of many) of modern cryptography is the realization of formal definitions of security which are fundamental for the design, usage and study of any cryptographic construction [11]. A definition of security essentially provides a mathematical formulation of a real-world problem, so special care must be taken for considering every part of the model in order to deliver the "real" security according to the given definition.

The power of the adversary is an essential part of the model that needs to be considered. This is closely related to the type of security offered by an encryption scheme. The security guarantees that modern crypto-systems provide are computational in nature, which means that an adversary with "reasonable" computational power would be able to break a system with only a very low probability. Thus, computational security makes assumptions on the computational power of the adversary and it does not rule out the success of an adversary in breaking a system given unlimited computational ability. Moreover, most cryptographic procedures depend on other mathematical assumptions. Like in symmetric key cryptography one assumes the existence of pseudorandom generators and pseudorandom functions. Stream ciphers and block-ciphers are the most used symmetric cryptographic primitives and they are used to build other procedures providing different functionalities like encryption schemes, authentication schemes etc. The security of these primitives (like block-ciphers and stream ciphers) cannot be mathematically proved, so one assumes that stream ciphers are pseudorandom generators or block-ciphers are pseudorandom functions to derive security of most symmetric encryption schemes. In asymmetric (public) key cryptography one assumes the computational hardness of certain problems like integer factoring or the computation of the discrete logarithm in certain groups. Though it is "socially believed" that these assumptions are true but still one cannot rule out the possibility that we are building systems based on wrong assumptions.

On the other hand, "information theoretic security" stands in stark contrast to "computational security". An information theoretically secure system neither depends on any un-proven assumption regarding computational hardness of any mathematical problem, nor makes any assumption regarding the computational power of an adversary, thus they are rightly called perfectly secret systems. Information theoretically secure systems were a starting point of the rigorous study of cryptography which was initiated by Shannon [3]. An example of such a secure system is the famous one-time-pad encryption scheme which was proposed by Gilbert Vernam and proved to be information theoretically secure by Shannon [3][1]. Even though information theoretically secure schemes provide the highest level of security, they are not practical. As for the one-time pad, the number of necessary key bits is the same as the number of bits in the message, also one needs to use a different key for each message. These requirements make the scheme practically infeasible.

2.1.1 Perfect secrecy and the one-time pad scheme

The basic and the most used cryptographic primitive is an encryption scheme. A symmetric encryption scheme is pictorially depicted in Fig. 2.1. We define an encryption scheme as a tuple consisting of three algorithms: $\pi = (\mathcal{GEN}, \mathcal{ENC}, \mathcal{DEC})$. These algorithms are described as follows:

- $\mathcal{GEN}$ is the key-generation algorithm that outputs a key k chosen according to some distribution determined by the algorithm. It is a probabilistic algorithm.
- $\mathcal{ENC}$ is the encryption algorithm, that takes as input a key k and a plaintext message m and outputs a ciphertext c. $\mathcal{ENC}_k(m)$ denotes the encryption of the plaintext m using the key k.
- $\mathcal{DEC}$ is the decryption algorithm, that takes as input a key k and a ciphertext c and outputs a plaintext m. $\mathcal{DEC}_k(c)$ denotes the decryption of a ciphertext c using the key k.

The basic correctness requirement of any encryption scheme is that for every key k output by $\mathcal{GEN}$ and every plaintext message $m \in \mathcal{M}$, it holds that

$$\mathcal{DEC}_k(\mathcal{ENC}_k(m)) = m$$

[1]It is said that Vladimir Kotelnikov also independently proved the perfect secrecy of one time pad [12]

This means that decrypting a ciphertext (using the appropriate key) yields the original message that was encrypted.

The *key space*, denoted by $\mathcal{K}$ is defined as the set of all possible keys output by the key-generation algorithm. Without loss of generality, we assume that $\mathcal{GEN}$ chooses a key uniformly at random from the key space. The *plaintext or message space*, is denoted by $\mathcal{M}$, which is the set of all legal messages supported by the encryption algorithm. Finally, the sets $\mathcal{K}$ and $\mathcal{M}$ together with the encryption algorithm define the set of all possible ciphertexts denoted by $\mathcal{C}$.

Now, we define the security of a crypto-system. To define security we assume certain probability distributions over the finite sets $\mathcal{K}, \mathcal{M}$ and $\mathcal{C}$. The distribution over $\mathcal{K}$ is defined by the key generation algorithm $\mathcal{GEN}$. So, $\Pr[K = k]$ denotes the probability that the key output by $\mathcal{GEN}$ is equal to k. Also, for $m \in \mathcal{M}$, let $\Pr[M = m]$ denote the probability that the message is equal to m. So in this way, we are assuming a probability distribution over the message space $\mathcal{M}$. Such a probability distribution over $\mathcal{M}$ is natural, as different messages may have different probabilities of being sent. The key and the message are chosen independently (the key is chosen and fixed before the message is known), and $\Pr[C = c]$ denotes the probability that the ciphertext is c. Once we fix the encryption algorithm $\mathcal{ENC}$, the distribution over $\mathcal{C}$ is fully determined by the distributions over $\mathcal{K}$ and $\mathcal{M}$.

Now, we proceed to define the notion of perfect secrecy. We consider an adversary who knows the probability distribution over $\mathcal{M}$. The adversary eavesdrops some ciphertexts sent by a honest party to another. Then, ideally this ciphertext should not make any difference in the knowledge of the adversary. In other words, the *a posteriori* probability that some message m was sent should not be different from the *a priori* probability that m would be sent. This should hold for any $m \in \mathcal{M}$ and even if the adversary has unbounded computational power. This is stated formally as:

DEFINITION 1. *An encryption scheme $\pi = (\mathcal{GEN}, \mathcal{ENC}, \mathcal{DEC})$ over a message space $\mathcal{M}$ is perfectly secret if for every probability distribution over $\mathcal{M}$, every message $m \in \mathcal{M}$ and every ciphertext $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$:*

$$\Pr[M = m | C = c] = \Pr[M = m]$$

In short, the definition also means that a scheme is perfectly secret if the distributions over the messages and ciphertexts are *independent*. The requirement of $\Pr[C = c] > 0$ is needed to prevent conditioning on a zero-probability event.

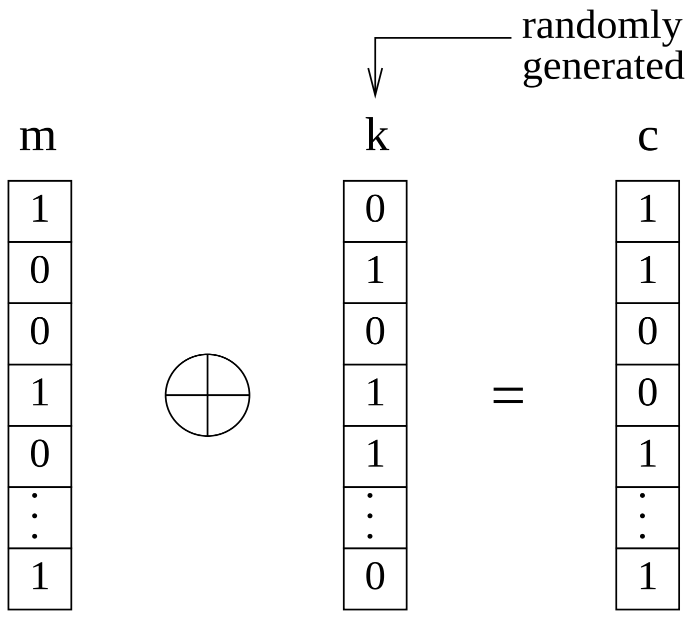


Figure 2.2: The one-time pad encryption scheme

The crypto-system called *One-Time Pad* (Vernam's cipher, created at 1917), is an example of a perfectly-secret scheme as Claude Shannon showed in [3]. The one-time pad encryption scheme is defined as follows:

- The message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ are all equal to $\{0,1\}^\ell$ where $\ell > 0$ is an integer.
- The key-generation algorithm $\mathcal{GEN}$ works by randomly choosing a string from $\mathcal{K} = \{0,1\}^\ell$ according to the uniform distribution. Thus each key has a probability of $2^{-\ell}$ in getting selected.
- The encryption algorithm $\mathcal{ENC}$ works like this: given a key $k \in \{0,1\}^\ell$ and a message $m \in \{0,1\}^\ell$, output $c = k \oplus m$.
- The decryption algorithm $\mathcal{DEC}$ works like this: given a key $k \in \{0,1\}^\ell$ and a ciphertext $c \in \{0,1\}^\ell$, output $m = k \oplus c$.
- Note that for every invocation of the encryption algorithm a new key needs to be used.

The encryption scheme is pictorially depicted in Figure 2.2. The one-time pad is perfectly secret because given a ciphertext c, there is no way an adversary can know which plaintext m it originated from. Notice that for every possible m there exists a key k such that $c = \mathcal{ENC}_k(m)$. Moreover, each key is chosen uniformly at random and no key is more likely than any other. Finally, it is possible to say that c reveals nothing whatsoever about which plaintext m was encrypted, since every plaintext is equally likely to have been encrypted. This can be formally written as:

Theorem 1 (Shanon)**.** *The one-time pad encryption scheme is perfectly-secret.*

Proof Fix some distribution over $\mathcal{M}$ and fix an arbitrary $m \in \mathcal{M}$ and $c \in \mathcal{C}$. The key observation is that for the one-time pad,

$$\Pr[C = c|M = m] = \Pr[M \oplus K = c|M = m] = \Pr[m \oplus K = c] = \Pr[K = m \oplus c] = \tfrac{1}{2^\ell}$$

Since this holds for all distributions and all m, we have that for every probability distribution over $\mathcal{M}$, every $m_0, m_1 \in \mathcal{M}$ and every $c \in \mathcal{C}$,

$$\Pr[C = c|M = m_0] = \tfrac{1}{2^\ell} = \Pr[C = c|M = m_1]$$

□

Perfect secrecy in the aforementioned encryption scheme is attainable but unfortunately has a number of drawbacks. The most prominent is that the number of key bits necessary for one-time pad encryption is same as the number of plaintext bits. But the problem of the unbounded key length is not only specific to the one-time pad; it is inherent to any scheme achieving perfect secrecy. In the following theorem we now prove that any perfectly-secret encryption scheme must have a key space that is at least as large as the message space.

Theorem 2. *Let $(\mathcal{GEN}, \mathcal{ENC}, \mathcal{DEC})$ be a perfectly-secret encryption scheme over a message space $\mathcal{M}$, and let $\mathcal{K}$ be the key space as determined by $\mathcal{GEN}$. Then $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof We show that if $|\mathcal{K}| < |\mathcal{M}|$ then the scheme is not perfectly secret. Assume $|\mathcal{K}| < |\mathcal{M}|$. Consider the uniform distribution over $\mathcal{M}$ and let $c \in \mathcal{C}$ be a ciphertext that occurs with non-zero probability. Let $\mathcal{M}(c)$ be the set of all possible messages that can be decrypted to c for a given key k; that is

$$\mathcal{M}(c) \overset{\text{def}}{=} \{\hat{m} | \hat{m} = \mathcal{DEC}_{\hat{k}}(c) \text{ for some } \hat{k} \in \mathcal{K}\}.$$

Clearly $|\mathcal{M}(c)| \leq |\mathcal{K}|$ since for each message $\hat{m} \in \mathcal{M}(c)$ we can identify at least one key $\hat{k} \in \mathcal{K}$ for which $\hat{m} = \mathcal{DEC}_{\hat{k}}(c)$. (Recall that we assume $\mathcal{DEC}$ is deterministic.) Under the assumption that $|\mathcal{K}| < |\mathcal{M}|$, this means that there is some $m' \in \mathcal{M}$ such that $m' \notin \mathcal{M}(c)$. But then

$$\Pr[M = m' | C = c] = 0 \neq \Pr[M = m'],$$

and so the scheme is not perfectly secret.

□

2.2 Bounded Storage Model

The bounded storage model was introduced by U. Maurer in his seminal work [4] in 1992. It is based on the assumption that the adversary's storage capacity is bounded, say by s bits, but the adversary is allowed to have unlimited computational power. This model assumes that a random string R of size u bits where $s \ll u$ is publicly available (e.g. the signal of a space radio source with a physical random number generator, beaming down a stream of random bits at a very high rate). We shall sometimes call this random source R as the randomizer. The adversary is able to store only a small portion of R (arbitrary s bits of information about R), and it is assumed that it is technically or financially infeasible for the adversary to store the whole random string R. Then, legitimate parties simply collect a relatively small number of common bits from R according to a shared secret key k so they can communicate through one-time-pads created from the randomness in R.

This model has been criticized to be unrealistic as having a public random source with high throughput as demanded by the model may be beyond what today's technology can offer us suggesting that it could be impractical, and also it has been said that with the current pace of storage technology it is not feasible to make a long term assumption regarding the storage bound of an adversary.

Despite the criticisms many attempts have been made at the beginning of this decade, modifying some components of the original model proposed in [4]. Based on this model, it was shown by Aumann, Ding and Rabin in [6] that a perfectly secret encryption scheme can be designed. They also showed a nice characteristic called "everlasting security" which means that the security of the messages sent in the past is still guaranteed even if the secret key is revealed in the future. What is more, Ding and Rabin in [13] presented extensions of previous works with a major new result regarding that the shared secret key employed by the honest users can be re-used an exponential amount of times to encrypt messages, against strong adaptive attacks. By strong adaptive attacks we understand the *active* attacks where the adversary can adaptively ask for encryptions and/or decryptions of its choice. Examples of this type of attacks are the so-called *chosen-plaintext attack*[2] and *chosen-ciphertext attack*[3].

Later, Chi-Jen Lu also studied the problem of an information-theoretically secure encryption in the bounded storage model showing that an encryption scheme with the appealing characteristics mentioned above can be derived immediately from a strong randomness extractor [10]. An extractor is a function that extracts randomness from a weak random source, where its output and its seed together are almost random. Since an efficient encryption scheme in this scenario calls for a strong extractor that can be evaluated in an on-line and efficient way, then it is interesting to notice that not every strong extractor is suitable for an encryption scheme in the setting considered here. The author proved the security of the scheme and described a construction with an efficient on-line extractor.

The aforementioned attempts also dealt with the size of the initial secret key k, essential to derive the security of the schemes. In some cases the initial key had to be longer than the derived one-time pad X, or the size u of the randomizer R had to be extremely large ($u \gg s$). In any case, the results in the bounded storage model were partial or far from optimal. Dziembowski and Maurer [14] introduced a new security proof for which k is short, the pad X needed by the users is very long and u needs to be moderately larger than s (the ratio $\frac{s}{u}$ called the *randomness efficiency* is close to 1). The authors proved security also for the case when R is not uniformly random, provided that the min-entropy of R is sufficiently greater than s.

In all previous works, it was always assumed that Alice and Bob initially share the secret key k without considering how it could be obtained. Dziembowski and Maurer addressed in [15] the problem of generating the initial key k focussing also on how the key generation process relates to the security proof of the scheme. They argued about the case in which a key agreement in the bounded storage model is performed, requiring for the users to have very high storage capacity and also investigate the *hybrid model* where k is generated by a computationally secure key agreement protocol, showing that this scenario is questionable.

In 2005, Savas and Sunar [7] made a first attempt in constructing a secure communication protocol in the bounded storage model defining means for successfully establishing and carrying out an encryption session. In their work, novel methods were provided for aspects like the synchronization between users at the time they tap into the random string R, the handling of transmission errors and the one time pad generation mechanism. Also, an analysis regarding the parameters values involved in the

[2]In this attack, the adversary has the ability to obtain the encryption of plaintexts of its choice. It then attempts to determine the plaintext corresponding to a ciphertext which it has not yet seen.

[3]In this attack, the adversary is even given the capability to obtain the decryption of ciphertexts of its choice. The adversary's aim, is to determine the plaintext corresponding to a ciphertext whose decryption it has not yet asked for.

model was made which is of practical interest for an implementation.

In 2006 Harnik and Naor [9] considered the idea of a *hybrid bounded storage model* were computational limitations on the eavesdropper are assumed up until the time that the transmission of R has ended. In this regard, they studied the possibility of achieving everlasting security with low memory requirements in the hybrid bounded storage model. One interesting proposal was to incorporate the use of a random oracle RO apart from the big random stream R in order to form the shared key needed by the model. In this manner, they would form the indices to read in R by querying first this random oracle RO that is accessible to all the parties. Furthermore, they argued about the possibility of not letting the adversary read the entire string R, perhaps by dividing that big string into many sub-channels so an adversary would not manage to record them all.

Nevertheless, serious practical issues prevent this model from being employed in real life communications. Some of them are: the publicly accessible random source at a very high rate, the authentication of this random stream, the syncronization of the honest parties, among others.

2.3 Limited Access Model

The limited access model introduced by Michael O. Rabin in [1] at 2005 is based on the functionality of a voluntary network of computers, each generating, storing and updating random pages, called Page Server Nodes (PSNs). The PSNs provide randomness to users who in turn use this randomness to communicate among themselves using one-time pads. An important property of the PSNs is that they serve each random page at most twice preventing an adversary from having common pages with the honest parties. Initially, two communicating parties share a short key k, and asynchronously in time randomly select the same PSNs and download the same set of random pages based on k. Since these communications could be compromised by an adversary, the users run a page reconciliation protocol to verify which pages they have in common. These common pages are employed to form one-time pads common to the parties. The security of such a system is not based on any assumption on the computational power of the adversary but on the inability of an adversary to monitor a huge number of PSNs distributed across the globe while the parties are downloading pages of random bytes from them. Under this assumption an encryption scheme in this model can be proved to be information-theoretically secure.

The limited access model can be seen as an improvement over the bounded storage model, in the sense that it is proposed a *provably unbreakable* encryption scheme where the single source of publicly available random bits are replaced by a network distributed source that supplies the required randomness to the clients.

An important characteristic of the limited access model is the fact that the initial key k is continually extended using the common one-time pads, playing an essential role in the security of the scheme. As a result, a scheme in the limited access model provides **Everlasting Secrecy**. This means that even if the initial key k or its later extensions were revealed after their use for collecting common random pages from the PSNs, the enchipered messages will still be secure. This is because the pages used to construct the one-time pad key are not available any more at the servers as a result of the serve only twice policy. After the second request or after a specified short amount of time, the pages are destroyed.

In the literature there is a single work related to the field of interest of this thesis project as reported in [2], developed by Jason K. Juang at the M.I.T. in may 2009. Some of the components involved in the model were addressed leaving some gaps with many interesting parts left open for future work.

Chapter 3

The Limited Access Model Description

Whoever wishes to keep a secret must hide the fact that he possesses one.

Johann Wolfgang Von Goethe

The *Limited Access Model* introduced by Michael O. Rabin in [1] enables the realization of a practical provably unbreakable encryption scheme. As mentioned before, the security of an encryption scheme in this model is based on the existence of a huge network of computers called the Page Server Node (PSN) network who are the ones that provide pages of random data to the users. Initially, two communicating parties share a short secret key k, and use this key to randomly select the same PSNs and download the same set of random pages based on k. Since some pages may not be received properly by the users for different situations (i.e. modifications during transit, network communication problems, etc.) they run a page reconciliation protocol to verify which pages they have in common. These common pages are then employed to form a long random string, a part of which is used to encrypt the messages using one-time pad and a part used to update the initial key.

An important property of the PSNs crucial for the security of the system is that each random page can only be served at most twice. Moreover, as already mentioned, the initial short secret key k is continuously updated by using part of the created random string. So a new round for downloading new pages can start again with a new key. We remark that the security of such a system is not based on any assumption on the computational power of the adversary but on the inability of an adversary to monitor a huge number of PSNs distributed across the globe. Under this assumption an encryption scheme in this model can be proved to be information-theoretically secure.

In this chapter we provide a detailed description of each entity and algorithm involved in a scheme in the limited access model, in a precise manner so that they would be amenable to a practical implementation. We also analyze deeply some modifications regarding the page reconciliation protocol, which results in improvements in the size of the initial shared key needed by the model and in the security of the scheme. Fi-

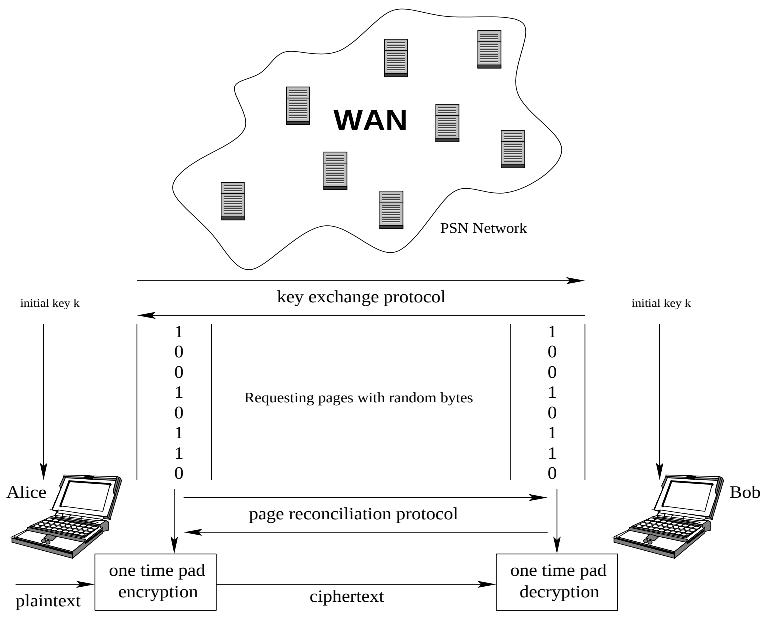


Figure 3.1: The limited access model with protocols

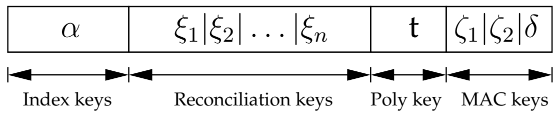


Figure 3.2: Various parts of the key: The key is a binary string containing four distinct parts named the index key (α), the reconciliation keys ($\xi_1, \ldots, \xi_n$), the Poly key (t) and the MAC keys (ζ_1, ζ_2, δ).

nally, we outline a secure communication protocol which defines means for securely establishing an encryption session in the limited access model.

3.1 The description of the scheme

An encryption scheme under the limited access model can be seen as a system comprised of the following entities and algorithms: the Page Server Node network (PSN network), the page reconciliation protocol, the initial key-exchange protocol, the encryption algorithm and the decryption algorithm. The basic scheme is shown in Fig. 3.1.

The scheme assumes that two legitimate users Alice and Bob share a secret key k. The key k is divided into four distinct fields as shown in Fig. 3.2. Thus the key consists of four basic parts: the index keys, reconciliation keys, the Poly key and the MAC keys. Also it assumes the existence of a huge network of PSNs which serves random bytes when requested. A part of the secret key k called the index key (α) acts as an index to a PSN and the pages stored within a PSN.

If Bob wants to communicate securely with Alice both of them request random pages from multiple PSNs, they choose the PSNs based on their index key. Once the selected PSNs have served their requests, Alice and Bob run a page reconciliation protocol to verify whether they own the same pages. The reconciliation protocol uses the reconciliation keys, the Poly key and the MAC keys (refer to Fig. 3.2). After running the page reconciliation protocol, Alice and Bob will know which pages they have in common. Using a part of these common pages Alice and Bob construct one-time-pads for secure communication and the other part is used to update their initial key which they would use in future if they require more randomness for secure communication. We describe in details each of the entities involved in the subsequent subsections.

Notation: In what follows, by $a||b$ we shall mean concatenation of the binary strings a and b, and $\mathsf{bin}_m(x)$ would mean the m bit binary representation of x. We shall denote $\log_2 n$ by $\lg n$.

3.1.1 The Page Server Node Network

The PSN network is a collection of server nodes as shown in Fig. 3.1 which consist of N PSNs distributed around the globe (N needs to be huge, say there are tens of thousands of computers maintaining random pages). Each of these computers acting as a PSN has an identity associated with it (it may be the server name or an IP address), so a PSN named j will be referred as PSN_j. Moreover, each PSN has a "true" random number generator ($TRNG_j$) built into it. This random source of bits is the most important component of a PSN. We assume that each PSN_j maintains a table $PageTable$ of r pages, thus for any natural number i ($i \leq r$), $PageTable[i]$ represents a page, which has been generated by the $TRNG_j$. Also, two algorithms run in a PSN, the algorithms are shown in Fig. 3.3.

The algorithm $\mathsf{getNewRandPage}()$ creates a random page using the TRNG. Each random page is ℓ bytes long. We assume that the procedure $\mathsf{GetTrueRandomness}()$ generates the required ℓ bytes of randomness. In our implementation the procedure $\mathsf{GetTrueRandomness}()$ is realized by a hardware based random number generator, which is described in Chapter 5. The algorithm $\mathsf{sendPage}(id)$ returns a page of the desired id if $\mathsf{PageTable}[id]$ contains a valid page, and later destroys this page (by assigning a NULL value to $\mathsf{PageTable}[id]$). We remark that placing a NULL value to the memory location may not ensure complete erasure of the data from the memory. For security one needs to totally destroy a used page preventing further analysis using forensics or other techniques to recover the page. If $\mathsf{PageTable}[id]$ does not contain a page (i.e., contains a NULL) then the algorithm creates a page using the routine $\mathsf{getNewRandPage}()$ and sends the same. The interesting thing to note is that the way the algorithm $\mathsf{sendPage}(id)$ is designed, it

```
**Algorithm** getNewRandPage()
1. $P \leftarrow \mathsf{GetTrueRandomness}()$;
2. return $P$;
```

```
Algorithm sendPage(id)
1. $P \leftarrow PageTable[id]$;
2. if $P = \text{NULL}$
    2.1. $PageTable[id] \leftarrow \mathsf{getNewRandPage}()$;
    2.2. $P \leftarrow PageTable[id]$;
3. else
    3.1. $PageTable[id] \leftarrow \text{NULL}$
4. return $P$;
```

Figure 3.3: The algorithms needed for the internal functionality of each PSN: GetTrueRandomness returns a random string of ℓ bytes on each invocation.

```
Algorithm downloadPage(k)
1. $(i_1, j_1), (i_2, j_2), ..., (i_n, j_n) \leftarrow k$;
2. for $l = 1$ to $n$
    2.1. $P_l \leftarrow PSN_{i_l}.\mathsf{sendPage}(j_l)$;
3. end for
```

Figure 3.4: The algorithm needed by the users to download pages from the PSNs: $PSN_{i_l}.\mathsf{SendPage}(j_l)$ represents the running of $\mathsf{SendPage}(j_l)$ at PSN_{i_l}

is guaranteed that a single page is served at most twice.

There is another important functionality of the PSNs which is not depicted by the algorithms described. As is obvious from the algorithm $\mathsf{sendPage}(id)$, a random page is generated only when it is first requested, and it is stored until the second request, after the second request has been served the page is destroyed. But a PSN does not store a page in-definitely. We assume that when a new page is created by $\mathsf{getNewRandPage}()$, the page is marked with a time stamp which keeps the information regarding the time at which it was created (this would be the same as the time when the page was first requested). The PSN keeps track of the time stamps of all the pages currently stored, and after a specified amount of time t_{max} after a certain page was created if a second request is not received by the PSN then that specific page is destroyed. A reasonable value for the parameter t_{max} can be considered to be 24 hours.

The honest users employ a mechanism to request and download a page of random bytes produced by a PSN. As discussed earlier the key consists of four parts, the first part which is called the index key is used for the purpose of selecting PSNs and the pages within a PSN. We assume that when a user requires some randomness it downloads n pages at a time from different PSNs. The index key α is a string of length $n(\lg N + \lg r)$ bits, which can be parsed as $i_1||j_1||\ldots||i_n||j_n$, where each i_l and j_l are $\lg N$ and $\lg r$ bits long respectively. i_l acts as an index to a PSN and j_l is the id of a page within PSN_{i_l}. Thus, the index key helps a user to address the PSNs and the pages within them.

The algorithm for getting pages called downloadPage() is shown in Fig. 3.4. Since the key has been chosen randomly, we say that this PSN selection is also random. It is assumed that the users share a list of the available PSNs in the system.

Regarding the PSN internal specification, since it can be seen as a database (in memory) of random pages generated by the TRNG built into it then there are few requirements needed for a node of this type. We will describe in Chapter 6 the implementational details of our work, but for now it is clear that a PSN just only requires an interface connection to the TRNG (i.e. USB-Serial or Ethernet), a necessary amount of principal memory to hold the random data, an eviction policy for staled pages and a communication protocol for letting users contact it (i.e. TCP sockets) every time a new page is needed.

3.1.2 The page reconciliation protocol

For proper functioning of the system it is required that Alice and Bob download the same pages, as these pages would be the source of randomness from which both would construct their keys for encryption. There exist a possibility that Alice and Bob end up downloading different pages. The possible reasons for this can be as follows. Since in the PSNs one random page is served at most twice and the honest users request them asynchronously, it is possible that Alice made a first request and downloaded a specific page P, and then an adversary was served with the same page before Bob made his request. Moreover, it is possible that an adversary manage to subvert or control internally a PSN providing malicious data to the honest parties or even modifies the content of a page while it is in transit.

There must be a mechanism for Alice and Bob to verify which pages they have in common. The page reconciliation protocol provides this mechanism. The reconciliation protocol uses two special functions $Poly_t^\lambda()$ and $MAC_{\delta,\zeta}()$. $Poly_t^\lambda()$ outputs a string of λ bits and $MAC_{\delta,\zeta}()$ outputs a string of γ bits. The parameter t associated with $Poly_t^\lambda()$ is a string of λ bits which is the Poly key of the initial key. The parameters δ and ζ associated with $MAC_{\delta,\zeta}()$ are γ bits long and they are part of the MAC keys of the initial key. Note that, the MAC key part of the initial key contains three strings ζ_1, ζ_2 and δ, as we shall see soon that Alice and Bob will respectively use ζ_1 and ζ_2 to compute the function $MAC()$. The function $Poly_t^\lambda()$ is used to compute a foot-print of the pages, whereas $MAC_{\delta,\zeta}()$ acts as a message authentication code. The details of these functions would be described in the next section. In addition to the functions and pa-

```
Algorithm Alice.Reconciliation($P_1, ..., P_n$)
1. for $i = 1$ to $n$
2.     $\tau_i \leftarrow Poly_t^\lambda(P_i)$
3.     $c_i \leftarrow \tau_i \oplus \xi_i$
4. end for
5. $C \leftarrow c_1||c_2||...||c_n$
6. $\tau \leftarrow MAC_{\delta,\zeta_1}(C)$
7. return $(C, \tau)$ to Bob
```

```
Algorithm Bob.Reconciliation($C, \tau$, $P_1, ..., P_n$)
1. if $MAC_{\delta,\zeta_1}(C) \neq \tau$
1.1.   return INVALID
2. $c_1||c_2||...||c_n \leftarrow C$
3. $L \leftarrow \epsilon$ ($\epsilon$ is the null string)
3. for $i = 1$ to $n$
4.     $\tau_{bi} \leftarrow Poly_t^\lambda(P_i)$
5.     $\tau_{ri} \leftarrow c_i \oplus \xi_i$
6.     if $\tau_{bi} = \tau_{ri}$, $L \leftarrow L||\mathsf{bin}_{\lg n}(i)$
7.     else $L \leftarrow L||\mathsf{bin}_{\lg n}(0)$
7. end for
8  return $L, MAC_{\delta,\zeta_2}(L)$ to Alice
```

Figure 3.5: Page Reconciliation Algorithms

rameters mentioned the reconciliation protocol requires n strings ξ_i $(1 \leq i \leq n)$, where each ξ_i is λ bits long. These ξ_is are called the reconciliation keys which are part of the key k.

Assuming that Alice initiates the reconciliation protocol, the procedures followed by both Alice and Bob are depicted in Fig. 3.5. Suppose Alice has downloaded the pages $P_1, P_2, \ldots, P_n$, she computes a string $C = \xi_1 \oplus Poly_t^\lambda(P_1)|| \ldots ||\xi_n \oplus Poly_t^\lambda(P_n)$ and sends $(C, MAC_{t,\zeta_1}(C))$ to Bob. As Bob shares t, ζ_1 and $\xi_i(1 \leq i \leq n)$ with Alice he can compute the foot prints of $Poly_t^\lambda()$ on his downloaded pages and thus verify which of the pages he has in common with Alice. Finally, he prepares as a response a list of indexes of the common pages that they have downloaded, and sends this list to Alice (for details see Fig. 3.5) along with the corresponding MAC tag computed over the list by using the key ζ_2. Thus, in a single round Alice transmits $\lambda n + \gamma$ bits to Bob which contains the footprint of the n pages downloaded by Alice along with a authentication tag of γ bits of these footprints. Bob transmits $n \lg n + \gamma$ bits to Alice, where each $\lg n$ bits of the the list is either an index of a page that Bob has in common with Alice or is a 0 along with a γ bit authentication tag of the list.

After the reconciliation protocol both Alice and Bob would have a list of indices of the pages that they have in common. From this list both selects the first μ ($\mu < n$) entries, and thus get μ pages. They perform an XOR of these μ pages to get ℓ bytes, we call these ℓ bytes as the *final* page. A part of the final page is used to change the key k to a new key and the rest is available to use as key for the one-time-pads, by which Alice and Bob would have secure communication. The encryption / decryption algorithm is just one-time pad as explained in Section 2.1.1 using the bits in the final page as key.

It is clear that the protocol will fail if users cannot own at least μ common pages of random data during a page reconciliation phase. Then it is crucial for the security of the encryption scheme to define in a precise manner the parameters involved. In the next chapter we will analyze the choice of values for the different parameters involved and necessary for a practical and secure implementation. Moreover, even though it is necessary to define the length in bits of the foot prints output by $Poly_t^{\lambda}()$, in terms of security this is not a problem, since these foot prints are encrypted with one-time pad using the corresponding part of the initial key ξ_i ($1 \leq i \leq n$). Then absolute secrecy is guaranteed during the interchange of messages phase in the Page Reconciliation Protocol. The use of the functions $Poly_t^{\lambda}()$ and $MAC_{\delta,\zeta}()$ is an improvement over the page reconciliation protocol described in [2], in terms of security and also for decreasing the initial shared key k as will be shown later.

Finally, the users share an initial key k, and after each download session followed by reconciliation the key is updated using the randomness they have downloaded. The initial key exchange is not part of the protocol. One can use computationally secure key exchange schemes like the Diffie Hellman scheme without much implications on security, as this key is very short lived. This is true since we can assume that the adversary though is unbounded in his computational capabilities, still will take some days to finally break the computationally secure key exchange scheme. Moreover, this scheme needs only to be performed once; the system itself refresh this key providing secure communication in perpetuity.

3.2 The description of the Poly and MAC

Before we describe the construction of the functions $Poly_t()$ and $MAC_{\delta,\zeta}()$, we would discuss a few important things about the finite field $GF(2^n)$. n bit strings can be viewed as polynomials of degree less than n with coefficients in $GF(2)$, thus one could view n bit strings as elements of $GF(2^n)$. For n-bit strings x, y, $x \oplus y$ will denote the addition in the field, which is a bitwise xor of the strings, and xy will denote multiplication in the field which can be realized by ordinary polynomial multiplication modulo the irreducible polynomial representing the field.

The function $Poly_t^{\lambda}(X)$ takes as input a arbitrary length string X and a string t of length

λ bits. It partitions X into blocks of λ bits as

$$X = x_1||x_2||...||x_m,$$

where each x_i, $1 \leq i \leq m-1$ is λ bits long, if the last block x_m is less than λ bits then it pads it with appropriate number of zeros to make it λ bits. The partition and padding thus ensures that each x_i is an element of $GF(2^\lambda)$. Then the function $Poly_t^\lambda()$ is defined as

$$Poly_t^\lambda(X) = x_1 t \oplus x_2 t^2 \oplus ... \oplus x_m t^m,$$

where all the operations are in the finite field $GF(2^\lambda)$.

The function $MAC_{\delta,\zeta}(X)$ outputs a string of length γ bits and takes in as input two strings δ and ζ each of length γ bits and an arbitrary long string X. The function is defined as

$$MAC_{\delta,\zeta_i}(X) = Poly_\delta^\gamma(X) \oplus \zeta.$$

Here all operations are in the field $GF(2^\gamma)$.

3.3 Outline of a secure communication protocol

So far we have described precisely every entity and algorithm that comprises an encryption scheme in the limited access model. Later, in Chapter 4 we will provide guidelines for the choice of the parameters involved in a practical implementation. In this section, we now describe a protocol which defines means for efficiently establishing and carrying out an encryption session in the limited access model.

The description in the following section is general and does not refer to any specific technological tools. The specific implementational details would be explained later in Chapter 6.

3.3.1 The Environment Architecture and Required Assumptions

We now describe the general architecture that would enable users who wish to primarily transfer sensitive data over an insecure network such as the Intenet. We assume the existence of an environment where the publicly available PSNs are spread all over across the globe. More to the point, we explicitly define the required and reasonable assumptions needed by the system to perform correctly. The important entities part of the whole environment are described as follows:

- **A True Random Number Generator**. The existence of true randomness is a complex question that even has become a rather philosophical matter. Since the one-time pad encryption calls for true randomness (as pseudo-random numbers are unsuitable in the presence of an unbounded computationally adversary), many researchers think that it can only be extracted from physical phenomena (i.e. atmospheric noise, white noise or radioactive decay).

- **A PSN**. A PSN is capable of serve pages of ℓ random bytes (reading the TRNG) efficiently to any users who contacts it. There is no privacy channel between the user and the PSN during the communication, and we can assume that the transmission of data takes place by using a TCP socket. It also maintains an internal database (in the principal memory) of the pages served just once. No other special equipment is required.

- **PSN Network**. Every single PSN that is part of the whole network is independent of the others, in the sense that it has its own random generator built into it. These PSNs can be installed in any place of the world (better if they are very scattered across) as long as it has a public IP address in order to contact it without inconveniences.

- **The honest clients**. These represent all the honest users (i.e. Alice and Bob) using the system who wish to communicate securely between them making use of the distributed randomness of the PSN network. In this regard, an efficient client needs only to access the PSNs selected with the key k asynchronously in time with the other honest party. Also, these clients are intended to manage in a secure manner the randomness downloaded.

- **The adversary**. We model the protocol in the presence of an adversary without limitations in his computing power, but on his inaccessibility to monitor all the PSN's connections. Specifically, we make the assumption that an adversary can compromise no more than one-fifth of the total number of PSNs connections. Also, the adversary may disrupt the system in a variety of ways like, reading confidential information, tampering messages, preventing user to communicate, etc.

3.3.2 The Protocol

We now proceed to describe the steps that will take place per every encryption session in the protocol proposed for a practical implementation. In what follows, assume that a special field or header will mark every message exchanged between the users so they will know it is a message of the encryption system and also, Alice will always start the communication with Bob. Basically, the procedures that the client users will perform are as follows:

- **Key-exchange Request**. If Alice and Bob are not sharing yet an initial key k, then they will make use of a computationally secure key exchange scheme, i.e. Diffie-

Hellman. Alice starts the protocol by computing her corresponding part of the scheme, and sends to Bob the corresponding message for initialization.

- **Key-exchange settlement**. Bob receives the message from Alice and computes his corresponding part of the key exchange scheme. Also he send to Alice the message for letting her to finally compute the initial shared k.
- **Synchronization hello and reply**. Alice wish to send sensitive data securely to Bob. Thus, she issues a communication request which includes her identity and the length of the message to be send. Bob receives the request from Alice and store the length of the message. In this manner, they will exactly know how much random bytes they need for encryption / decryption with one-time pad. He confirms reception to Alice. This transaction is required to be authenticated and there must be a secure way for Bob and Alice to identify each other. This is not a part of the encryption protocol under the limited access model.
- **Request of pages to the PSNs**. After the synchronization messages confirmed, both Alice and Bob asynchronously start to download n pages from the PSNs according to the key k.
- **Page Reconciliation initialization**. After the request of pages phase, Alice issues the algorithm *Alice.Reconciliation* presented in subsection 3.1.2 for the first part of the protocol, and send to Bob the corresponding tuple as was described above (see Fig. 3.5 for details).
- **Page Reconciliation reply**. Bob receives the tuple from Alice and performs the algorithm *Bob.Reconciliation* presented in subsection 3.1.2 for constructing the list of common pages own with Alice. He sends this list authenticated to Alice as depicted in the algorithm (see Fig. 3.5 for details). If more randomness is required, they will re-run the last three steps until they have the necessary randomness for communicating.
- **One-time pad encrypt**. With the required randomness harvested, Alice will encrypt the original sensitive message with the one-time pad scheme. She also authenticates the ciphertext before sending the information through the insecure channel to Bob.
- **One-time pad decryption**. Bob receives the ciphertext and proceeds to decrypt it with the random bytes downloaded from the PSNs and reconciliated with Alice.
- **Dispose of the used key**. After the encryption / decryption phase, Alice and Bob proceed to dispose the key used for the secure communication, ensuring that it is employed just once and thus maintaining security.

Chapter 4

The Security of the Scheme

There is no security on this earth. Only opportunity.

General Douglas MacArthur

The security of the one-time pad encryption scheme can be mathematically proved as it was shown in Chapter 2. Nevertheless, because of its inherent drawbacks we have introduced the *limited access model* in the chapter before, turning our research toward the design and implementation of an *information-theoretically secure* encryption scheme in that model.

But a crypto-system designed in this model introduces new assumptions and components (i.e. entities and protocols) besides of the one-time pad encryption block, and they need to be analyzed carefully. This is true since before encrypting with one-time pad, they are playing an important role during the time for progressively construct the long key needed (which has the same size as the plaintext desired to be sent between the honest users).

In this chapter we discuss the security aspects of the different entities and algorithms involved in the limited access model. First we start this chapter by providing values for many important parameters involved in the system and then we argue about the security of the whole scheme and provide some justification regarding the choice of the parameters. Then, we make reference to the *initial key-exchange* protocol, analyzing its impact to the model. Moreover we continue our discussion by computing and arguing about the probabilities of success for an adversary in breaking the page reconciliation protocol and also the system, based on the limited access assumption and the paremeters suggested by the section before. Finally, we also analyze the probabilities regarding the collisions that could be encountered by using the unconditionally secure MAC algorithm, which is an important block inside the system.

4.1 Choice of Parameters

There are multiple parameters involved in the system. The parameters whose values required to be fixed are listed below:

N	The number of PSNs.
r	The number of pages stored in each PSN.
ℓ	The size of each page in bytes.
n	The number of pages downloaded in each session.
λ	The size in bits of the Poly output.
γ	The size in bits of the MAC output
μ	The number of pages used for constructing the final page.

The values of these parameters are crucial for the security of the system and also the values would dictate whether an efficient implementation would be practically feasible. Next, we provide guidelines for the choice of the parameters and also argue regarding the suitability of our choices.

The first important parameter is N. The security of the model depends on the fact that the access of an adversary is limited, which translates to the fact that an adversary must not be able to manipulate or control more than a small fraction of the total number of PSNs. In [1] it was stated that an assumption on the inability of an adversary to manipulate less than one-fifth of the total PSNs is reasonable. We fix $N = 2^{16}$, which is reasonable, and given the current state of technology it may be possible to maintain these many page server nodes. Also we assume that each PSN has the capability of storing 2^{16} pages each of size 4096 bytes, i.e., $r = 2^{16}$ and $\ell = 4096$. We propose $\ell = 4096$ bytes because it is an adequate amount of bytes that can be feasibly transmitted through TCP sockets nowadays without much overhead.

As stated earlier, to construct the final page Alice and Bob XORs μ pages out of all the common pages they downloaded in a session. So, after a reconciliation protocol if the parties fail to have μ common pages the protocol will fail, and they have to start from the beginning. As they download n pages in a single session, it is necessary that μ be less than n, also μ should be sufficiently large such that there is an effect of multiple PSNs on the final page. In [1] it was proposed that $\mu = 30$ is a reasonable value and in our scheme also we stick to this choice. Based on this value of μ we choose n to be equal to 64, which provides lot of flexibility and even if the adversary manages to manipulate $\frac{1}{5}$ of the downloaded pages, the protocol will not be affected.

The Poly used in the reconciliation protocol acts as a foot print of the pages downloaded, so a small value can be sent which represents a page instead of sending the whole page. Moreover, the MAC function employed in the reconciliation protocol is used to authenticate the lists that are being exchanged by the parties and performs calls to the Poly function for computing tags, as was described in Chapter 3. The best choice would be an information theoretically secure message authentication code. Such codes can be realized through universal hash functions, which can be constructed by simple polynomial evaluations but are not suitable in the sense that the key also require huge amount of key material. We assume that the Poly key t is of 32 bit long. Then all the operations are performed in the finite field $GF(2^{32})$ (refer to Section 3.2 for details). Thus, every foot print is of 32 bits long making $\lambda = 32$. For computing the MAC tags, we need two keys which are known as the MAC keys. The first key is called δ and is of 128 bits long. As we showed in the description of the MAC in Chapter 3, we are calling the Poly function but this time all the operations are performed in the finite field $GF(2^{128})$. Since the random string ζ_i is XORed with the output of the Poly function, then we define that ζ_1, ζ_2 are of 128 bits long each. Moreover, we also assume that the MAC output γ is of 128 bit long.

With these parameters fixed, we can comment on the size of the key. As stated earlier, the key consist of four parts the index keys, the reconciliation keys, the Poly key and MAC keys. Thus the total size of the key would be $n(\lg N + \lg r) + n\lambda + \lambda + 3\gamma$. Using the values suggested above we have the size of the key as 564 bytes, which is a big improvement over the key length suggested in [2], which suggested a key length of 2880 bytes. The huge key size in [2] was due to the fact that they used more specialized finger-printing methods instead of the Poly that we use.

As $\ell = 4096$ bytes is the size of the final page, so out of this 4096 bytes 564 bytes would be used to form the new key and 3532 bytes would be available for forming the one-time-pads. Thus at the end of the protocol the users would be able to send a message less than or equal to 3532 bytes. If they intend to send bigger messages they have to re-run the protocol to obtain more randomness.

4.2 Possible vulnerabilities in the system

In this work we are not able to give a precise security definition and a security proof of the system described. However, we provide some heuristic arguments regarding security of the whole scheme by pointing out possible vulnerabilities and how our system can protect itself against those vulnerabilities. This analysis also helps us to justify the various choice of parameters made in the previous section. To begin with, as already stated, we have two basic assumptions which are:

1. The adversary cannot control more than one fifth of the total Page Server Nodes.

2. The randomness provided by the Page Server Nodes is truly random.

Now we list down the possible vulnerabilities and pitfalls in the system that can be exploited by an adversary:

- *The initial key-exchange protocol*: One can implement an exponential key exchange Diffie-Hellman which is an anonymous key agreement protocol. But this protocol does not provide authentication of the parties, and consequently is vulnerable to man-in-the-middle attacks. These type of attacks refer to a form of active eavesdropping where the oponent makes independent connections with the honest users, causing them to transmit messages between them so they believe that they are talking directly to each other. But actually, all the conversation is controlled by an atacker.
- *Total decryption*: The adversary manages to decrypt the ciphertext containing the foot prints of every downloaded page from the PSNs. This implies that the adversary will discover the encryption key.
- *Intercepting the servers*: An adversary can compromise some PSNs and the connection between the users and these servers. This has serious implications in the security of the system since an adversary can prevent users the possibility of having at least μ common pages, which are required by the scheme in order to construct the *final page*. Also if the adversary can monitor all the servers which are used by the users in a single session then it can easily decrypt the encrypted communication and get hold of the key that would be used in the future.
- *Finding collisions in the Poly and MAC functions*: An adversary can find a collision in the Poly function used by the page reconciliation protocol, providing two different pages say p, p' with the same foot print to the users. In this case, they will reconciliate but they will produce different final pages. Moreover, this can also be a pitfall when computing tags with the MAC function, where an adversary modifies a tag associated to a ciphertext which are then sent to the other communicating user.

4.3 How the system defends against the vulnerabilities

We now point out how the system can face and defend against the pitfalls described in the previous section.

4.3.1 The initial key-exchange protocol

As we pointed out before, pitfalls like man-in-the-middle attacks can be exploited by an adversary when issuing an initial key-exchange protocol. But these vulnerabilities can be defended by using authentication mechanisms. For example, one can think in public key infrastructure solutions and certificate authorities, or the option of password-authenticated key agreement. Moreover, there is a possibility of still perform anonymous key-agreement protocols by using the *Interlock Protocol* proposed by Shamir and Rivest.

Nevertheless, we remark that it is outside of the scope for this work a practical and formally secure implementation of a key exchange protocol, since the initial key exchange is not actually part of the system in the limited access model. We just implemented the exponential key exchange Diffie-Hellman protocol for a proof of concept in our work.

4.3.2 Total decryption

The encryption scheme used by our system is the one-time pad. There are many messages that are sent using one-time pad between the users, like the messages which are part of the page reconciliation protocol (when a user sends the foot prints of the pages enciphered) or the encryption of the original plaintext. Then, one possible attack from an adversary would be try to decrypt all these enciphered strings. However, based on the assumption that the key is truly random (also has the same size as the message to be encrypted) and that it is only shared between the honest users, then we know that the only opportunity for an adversary to totally decrypt is just guess, as was demonstrated in Chapter 2. Thus, the probability that an adversary can achieve total decryption is very low.

4.3.3 Intercepting the servers

By intercepting the servers an adversary can attempt to do two things:

1. In a single session, the users download n pages from n servers. Then they run the page reconciliation protocol and select the first μ pages which both the users have in common to form the final page. If an adversary is able to intercept the specific μ servers whose pages were used to form the final page then the adversary would have knowledge of the one-time-pad key along with the new key which would be used for the future sessions. Note that if the adversary fails to know even one of the μ pages which would be used to form the final page, the adversary would have no useful knowledge, as all the μ pages are randomly generated.

2. Out of the n servers that the users access in a single session, if the adversary have access to any of the $n-\mu+1$ or more servers, then he can tamper the pages sent by the server and thus prevent the users from reconciliating the pages and making the protocol fail.

Considering the adversary has no knowledge of the index keys being used by the users it can at best randomly intercept the servers. In this scenario we shall analyze the probability of success of the adversary in both scenarios.

Let us consider a single session where the users choose n servers randomly (as dictated by the randomly generated index key) from the existing N servers. If there is no adversarial tampering of the random pages, or no other technical problems like loss of network packets or accidental alteration during transit, then the users will reconciliate with the first μ downloaded pages, and using those pages they would form the final page. If the adversary has gained access to these designated μ pages then he himself would be able to form the final page. Suppose the adversary has access to m servers, then the probability of success of the adversary would be same as the probability that the μ pages used for computing the final page are all among the m pages controlled by the adversary. If Succ1 be the event that the adversary succeeds in the first scenario, we have

$$\Pr[\mathsf{Succ1}] = \frac{\binom{m}{\mu}}{\binom{N}{\mu}}. \tag{4.1}$$

Now, consider the second scenario which can be described using an urn model. Consider an urn with N balls out of which m are white and the rest are black. n balls are randomly chosen without replacement from the urn, we are interested in the probability that k of the drawn balls are white. Let X be the random variable which denotes the number of white balls drawn, then it is easy to see that

$$\Pr[X = k] = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}. \tag{4.2}$$

The random variable X follows the so called Hypergeometric distribution [16]. The relation of our scheme with this urn model is that the balls represents the servers and the white balls represents the compromised servers. When the users select n servers randomly, we are trying to see the probability that k many of them would be the compromised servers.

With the choice of our parameters and assumptions, we can say the following:

1. The number of servers $N = 2^{16}$.
2. The number of compromised servers $m \leq \lceil N/5 \rceil$.
3. The number of servers selected in a single session $n = 64$.
4. The number of pages used to create the final page $\mu = 30$
5. Finally we are interested in $k \geq n - \mu + 1 = 35$, as if the adversary can tamper more than k pages then the users will not be able to reconciliate.

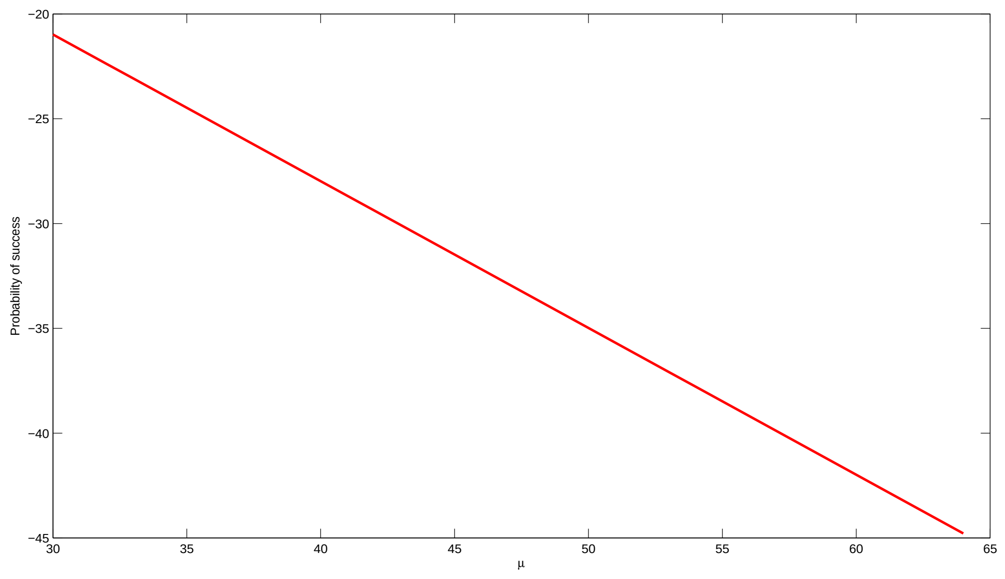


Figure 4.1: Graph for the success probabilities $\Pr[\mathsf{Succ1}]$ of an adversary for $30 \leq \mu \leq 64$. The y-axis is represented is logarithmic scale i.e. -45 means 10^{-45}.

Thus, if Succ2 denotes the probability of success in the second case then, we have

$$\Pr[\mathsf{Succ2}] = \Pr[X \geq n - \mu + 1]. \tag{4.3}$$

Using eq. (4.1) and the values stated above we have $\Pr[\mathsf{Succ1}] = 1.045104711 \times 10^{-21}$. By varying the values of μ we compute the success probabilities which are shown in Fig. 4.1. We see that the success probabilities decreases with the increase of μ. But if μ is very big then the reconciliation protocol will have lesser flexibility and would have lesser tolerance to transmission errors or adversarial corruption of the pages.

For estimating $\Pr[\mathsf{Succ2}]$ we tabulate the values of probabilities for various values of k ranging from 35 to 64 in Table 4.1. The values are also plotted in Figures 4.2 and 4.3. Figures 4.2 and 4.3 clearly show that the probability decreases with increase of k. Also, the probability for $k = 35$ is 7.21×10^{-10}.

A thing to note is that the success probabilities here are pretty bigger than the usual success probabilities that we encounter in typical cryptographic scenarios. For example, consider a block-cipher with a key length of 128 bits, the block-cipher is considered secure if no adversary can guess the key with probability more than 2^{-128}. This probability is far lesser than the probabilities of the events Succ1 and Succ2. But, a thing to be noted is that the scenario described here is quite different from a typical cryptographic scenario. As, in case of the example of the block-cipher, the adversary can go

Table 4.1: Hypergeometric distribution probabilities

k	Probability	k	Probability
35	$7.21753484047e-10$	36	$1.4503968582e-10$
37	$2.73781506864e-11$	38	$4.85180920525e-12$
39	$8.06659727036e-13$	40	$1.25720700707e-13$
41	$1.83497037238e-14$	42	$2.50530702627e-15$
43	$3.19541308517e-16$	44	$3.80157817897e-17$
45	$4.21124176911e-18$	46	$4.33504068502e-19$
47	$4.1372667416e-20$	48	$3.65111375824e-21$
49	$2.97037912873e-22$	50	$2.22000633385e-23$
51	$1.51806931357e-24$	52	$9.45299853007e-26$
53	$5.33054233034e-27$	54	$2.70411525713e-28$
55	$1.22426611811e-29$	56	$4.89893950498e-31$
57	$1.71177788566e-32$	58	$5.14287654257e-34$
59	$1.30182442e-35$	60	$2.70008027851e-37$
61	$4.40626661827e-39$	62	$5.30545584755e-41$
63	$4.19076148158e-43$	64	$1.62911666151e-45$

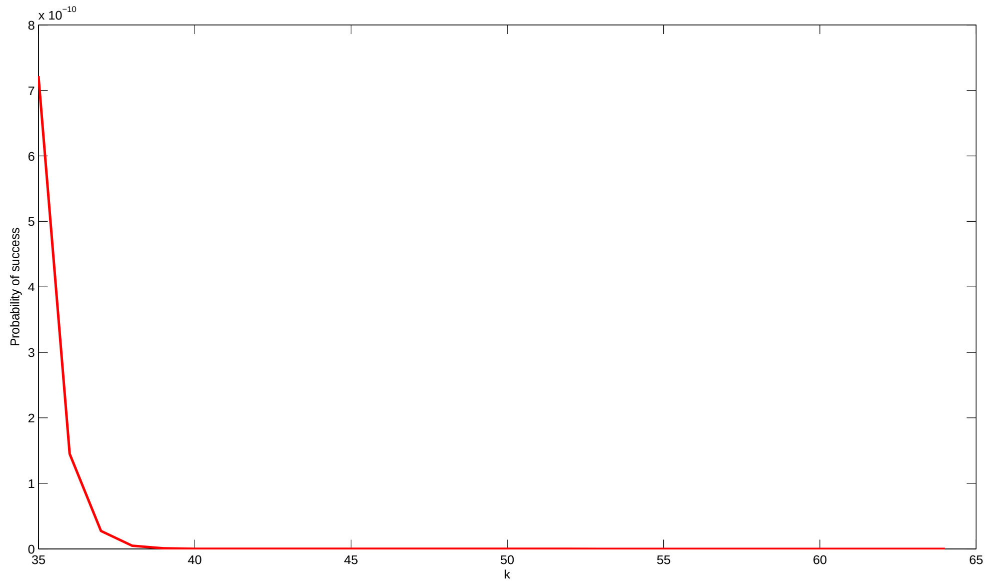


Figure 4.2: Graph for the success probabilities $\Pr[\text{Succ2}]$ of an adversary when $N = 2^{16}, n = 64$ and $m = \lceil \frac{N}{5} \rceil$

on trying every key and each time he checks a key his probability of success increases in the successive steps. Also the interpretation of security based on success probability depends on the computational power of the adversary, like if the adversary can check

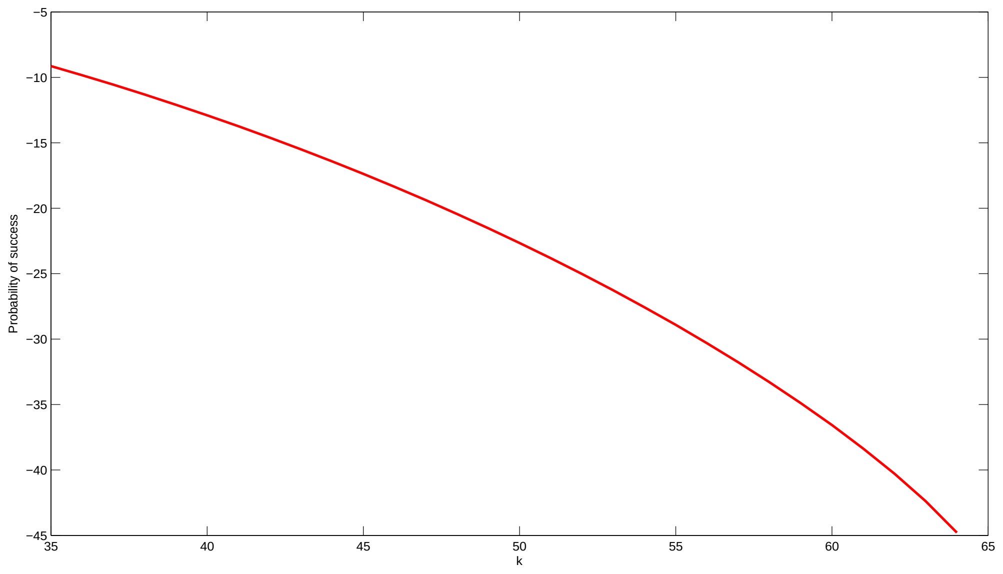


Figure 4.3: The same graph of Fig. 4.2 using a logarithmic scale

2^{128} keys in a reasonable amount of time then there would be no security. But the scenario for our scheme is quite different. Here the adversary has no way that he can try his/her method multiple times. As each session starts with a different key, hence for each session the probability of success remains unchanged. Given this interpretation, the probability of the events Succ1 and Succ2 are low enough to provide reasonable security.

The success probabilities in case of the second scenario, i.e., $\Pr[\mathsf{Succ2}]$ can be further reduced if the number of pages downloaded in each session (the parameter n) is increased. We show the probabilities for $67 \leq k \leq 96$ in Fig. 4.4. But an increase in n would lead to an increase in the initial key length and would also increase the communication overhead of the page reconciliation protocol.

4.3.4 Security Concerning Poly and MAC

The functions *Poly()* and $MAC()$ are two other important components of the system used in the page reconciliation protocol. The security of these functions are addressed in this section.

The function $Poly_t^\lambda()$ is used to create a footprint of the pages. Note that a ℓ byte page is converted into λ bits using the function $Poly_t^\lambda()$. The output of $Poly_t^\lambda()$, is masked with a random string and then sent to the communicating party. Thus, the footprints of

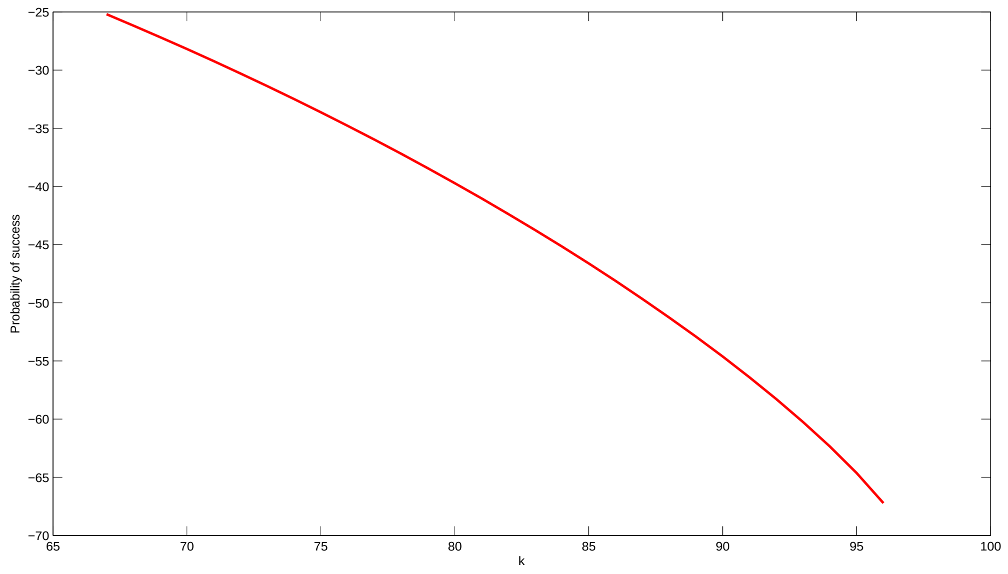


Figure 4.4: Graph for the success probabilities $\Pr[\mathsf{Succ2}]$ of an adversary when $N = 2^{16}, n = 96$ and $m = \lceil \frac{N}{5} \rceil$ using a logarithmic scale

the pages as transmitted are random strings and the adversary has no way to guess the original page from the footprint that gets transmitted. But, the adversary can use some weaknesses in the function $Poly_t^\lambda()$ to make the protocol fail. As described in Chapter 3, a page X is partitioned into blocks of λ bits and written as $X = x_1||x_2||\ldots||x_s$, where each x_i is λ bits long. Given a λ bit long key t, $Poly_t^\lambda(X)$ is computed as

$$Poly_t^\lambda(X) = x_1 t \oplus x_2 t^2 \oplus \cdots \oplus x_s t^s,$$

where the multiplication and the addition (denoted by $\oplus$) are in the field $GF(2^\lambda)$. If the adversary has access to a PSN from where the users are downloading, (s)he can potentially serve two different pages $X' = x_1'||x_2'||\ldots||x_s'$ and $X'' = x_1''||x_2''||\ldots||x_s''$ to the two users such that $Poly_t^\lambda(X') = Poly_t^\lambda(X'')$. If the adversary can do this then the users would think that they have the same page but in reality they do not. Let us call this event as Succ3, and thus the probability of success for an adversary in doing this would be denoted by $\Pr[\mathsf{Succ3}]$. Recall that t is a part of the key hence can be assumed to be a λ bit random string unknown to the adversary. With this assumption we have

$$\begin{aligned}\Pr[\mathsf{Succ3}] &= \Pr[Poly_t^\lambda(X') = Poly_t^\lambda(X'')] & (4.4)\\ &= \Pr[Poly_t^\lambda(X') \oplus Poly_t^\lambda(X'') = 0] & (4.5)\\ &= \Pr[(x_1' \oplus x_1'')t \oplus \cdots \oplus (x_1' \oplus x_1'')t^s = 0]. & (4.6)\end{aligned}$$

Now, $(x_1' \oplus x_1'')t \oplus \cdots \oplus (x_1' \oplus x_1'')t^s$ is a non-zero polynomial of degree s, as X' and X'' are different. Thus, this polynomial can have at most s roots in $GF(2^\lambda)$. And, t is a random element in $GF(2^\lambda)$, thus the probability that this equation in eq. 4.6 is satisfied is same as the probability that t is a root of the polynomial. Thus we have

$$\Pr[\mathsf{Succ3}] = \frac{s}{2^\lambda}. \tag{4.7}$$

According to our choice of parameters $\lambda = 32$ bits and each page is 4096 bytes long, hence $s = (4096 \times 8)/32 = 1024$. Thus, using eq. 4.7 we have

$$\Pr[\mathsf{Succ3}] = \frac{1024}{2^{32}} = 2^{-22}. \tag{4.8}$$

Again, as discussed in the previous section this success probability should not be equated with the success probabilities of typical crypto-systems, as the adversary cannot repeat his attempts to break the scheme. The probability can be reduced by increasing λ. In particular if we increase λ to 64 bits then the probability of success reduces to 2^{-55}. But, the increase in the size of λ would mean that larger footprints would be used, and this would increase the length of the initial key and the communication overhead of the page reconciliation protocol.

Next we discuss the security of the function $MAC()$. This function is a message authentication code employed by the users to ensure the integrity of the messages sent during the page reconciliation protocol. The $MAC()$ for a message Y is computed as follows

$$MAC_{\delta,\zeta}(Y) = Poly_\delta^\gamma(Y) \oplus \zeta.$$

Note, here the keys δ and ζ are γ bits long and thus the polynomial $Poly_\delta^\gamma()$ is computed over the field $GF(2^\gamma)$. ζ is a random string obtained from the key, and ζ is not reused, i.e., in the reconciliation protocol Alice and Bob uses different values of ζ.

The goal of the adversary here would be to tamper the messages to a different message which would have the same MAC. Let this event be called Succ4. The structure of the MAC used is very similar to that of any polynomial based MACs [17], which are generally computed as $Poly_\delta^\gamma(Y) \oplus E_K(N)$, where $E_K()$ is a block-cipher and N a non-repeating quantity usually called a nonce. It is known that probability of forging of such MACs is at most of the order of $s^2q^2/2^\gamma + \nu$, where q is the number of message MAC pairs previously seen by the adversary and ν is the probability with which a random string can be distinguished from the outputs of the block cipher $E_K()$. In our construction $E_K()$ is replaced by a truly random string, thus $\nu = 0$ and the adversary gets to see at most one message MAC pair. Hence the probability of success in our case becomes

$$\Pr[\mathsf{Succ3}] \leq \frac{s^2}{2^\gamma}.$$

With our choice of parameters $\gamma = 128$ bits, we obtain $s = 4096 \times 8/128 = 256$ we have

$$\Pr[\mathsf{Succ3}] \leq \frac{2^{16}}{2^{128}} = 2^{-112},$$

which is pretty small for all practical purposes.

Chapter 5

Our Random Number Generator

If you aren't going all the way, why go at all?

Joe Namath

A random number generator is a very important component of the model since the one-time pad encryption scheme (which is part of the system) requires random bits to assure *perfect secrecy*. We recall that a one-time pad encryption scheme uses as much key material as the size of the plaintext. One-time pad provides information theoretic security only if the key bits are perfectly random. In our scheme the randomness is provided by the page server nodes. Thus, each PSN must be equipped with a mechanism which can generate random bits. Also, it is required that the random bits provided by the PSNs are "truly random". Though it is debatable whether true random bits can really be produced by any device, but it is well accepted that to generate true randomness one has to depend on some natural physical phenomenon. Thus, in our attempt to generate "true randomness" we designed a physical random number generator which is supposed to be embedded in every PSN. There are many previous designs available for physical random number generators suited for different environments [18, 19, 20]. For a proof of concept we designed a simple but efficient generator. Our basic design is based on the work of Aaron Logue [21]. In this chapter we will describe some basic issues in generating randomness and then we shall describe the random number generator which we designed.

5.1 Background and types of generators

A random number generator (RNG) is required in a wide range of areas, like statistics, cryptography, etc. The required quality of randomness is dictated by the type of application. Since most cryptographic protocols and applications depend on the generation of random numbers, the security of such protocols generally depend on the quality of randomness. Thus, cryptographic applications in general demand very high quality of randomness. Examples of random material required in cryptography are:

- Secret keys.
- Nonces and initialization vectors.
- Seeds required by pseudo-random generators.
- Challenge data and parameters used in challenge-response type security protocols, among others.

For cryptographic purposes, a RNG can be a computational or physical device that gives as output sequences which are expected to be unpredictable to an adversary. This component is important in cryptographic protocols since their security essentially rely on the unpredictability of the random material. In this sense, even if an adversary has knowledge about the design and previously generated random numbers, (s)he would not be able to predict the future outputs from the RNG.

There can be different types of generators based on the techniques to generate randomness. In one hand, we have Pseudo Random Number Generators (PRNG) which are deterministic algorithms whose output is a sequence of bits that approximates the properties of truly random bits. They completely depend on an initialization process for generating the sequence of bits. This initialization is performed by using a short random value called *the seed*. These generators are called deterministic since if they are initialized with the same seed, then they will always produce the same sequence of bits. Generally a PRNG cannot generate unlimited amount of random bits from a single seed. After generating say t many bits if a PRNG repeats the sequence, we say that the PRNG has a period of t. Most PRNGs are periodic (i.e. they have a period). On the other hand, we have True Random Number Generators (TRNG) which extract statistically independent and unbiased bits from a physical process. Hence, the TRNGs are non-deterministic generators and also have no period. They always produce a sequence which never repeats. The major difference between PRNGs and TRNGs is that an internal state is kept in the former whereas nothing is kept in the TRNGs, producing bits independent of the previously generated. PRNGs are computational algorithms which are easier to describe and also generally more efficient than TRNGs.

For our work, PRNGs are unsuitable since an encryption scheme in the limited access model considers an adversary without limitations in the computational capabilities. Thus, security provided by PRNGs (which depends on the assumption that it should be computationally infeasible to compute the next output even if the algorithm used and all previous outputs generated from the PRNG are known) does not hold anymore under this scenario. Then, from now onwards we will only discuss TRNGs which are of interest for our protocol.

5.2 Classification of true random number generators

We start this section by defining a very important term called *the entropy*, as a measure of the uncertainty associated with a random variable. This is written formally as follows:

DEFINITION 2. *The entropy H of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ is:*

$$H(X) = -\sum_{i=1}^{n} \Pr[x_i] \lg(\Pr[x_i])$$

This definition was introduced by Claude E. Shannon [22] in 1948. Entropy is used to measure uncertainty of a specific event X. An event X is more uncertain if X has high entropy. Thus, natural sources of randomness are generally called as entropy sources. We now focus on the discussion regarding the true random number generators. These generators can be divided in:

- **Software based generators**. The entropy source in which these generators relies on, is extracted from random events that take place in a computer system. Examples of such events can be the user movement of the mouse, keystrokes, network traffic, etc. These events are captured and used by software procedures and converted into numerical values which serves as random numbers. They are also called non-physical TRNGs.
- **Hardware based generators**. The randomness is extracted from physical phenomena like electronic noise, radioactive decay, etc. These sources of entropy are better in quality than the software based ones.

The option of Hardware based generators (also called as Physical TRNGs) are more reliable, since they are more robust and secure and with higher entropy than the software based generators. The following methods are generally employed with physical TRNGs for generating random numbers:

- using analog components
- using exclusively digital components
- using a combination of analog and digital components

We now discuss the possible sources of randomness usually considered when constructing physical true random number generators.

5.2.1 Source of randomness in physical TRNGs

When constructing physical TRNGs, the entropy source is always extracted from a physical source. The decision regarding which source better fits the necessities are based on the performance, security and efficiency required for the specific case, the costs for constructing as well as the type of components wished to be present in the random generator.

We now describe the different source of randomness that can be exploited for constructing a physical generator [20]:

- **Quantum mechanical properties**. The nuclear decay from a radioactive substance or the quantum mechanical properties of a photon are examples of entropy sources that can be employed for a physical TRNG.
- **Electronic noises**. Noise is an undesirable characteristic of all electronic circuits. Undesired random fluctuation in electrical signals within a circuit constitutes noise. This noise can vary significantly with time and thus can act as a rich source of randomness. There can be various types of noise present in a circuit. We mention some of them as follows:
  - Thermal noise. This kind of noise is generated by the thermal agitation of electrons in a conductor and can be observed in resistors. This noise is approximately white and it is also known as Johnson Noise.
  - Shot noise. This type of electronic noise occurs when a voltage differential causes electrons and holes cross a barrier. Examples of components producing shot noise are: a diode, a transistor or a vacuum tube.
  - Avalanche noise. This noise is produced by an avalanche breakdown of an avalanche diode when a specified reverse bias voltage is applied. This can be observed on zener diodes where the phenomenon is called zener breakdown.
  - Flicker noise. It is also known as $1/f$ noise or *pink* noise, is a signal with a frequency spectrum that falls off steadily into the higher frequencies. It occurs in almost all electronic devices, and results from a variety of effects. It is always related to the direct current flow.
- **Metastability**. In electronic systems metastability refers to the ability of the system to persist in an unstable equilibrium state for an un-bounded time. This phenomenon can be frequently encountered in flip-flops. A flip-flop has two well-defined stable states designated as 0 and 1, but in certain conditions it can hover between its stable states for more than one clock cycle. This can result in an un-predictable behavior of the system. This unpredictability can act as a suitable source of randomness.

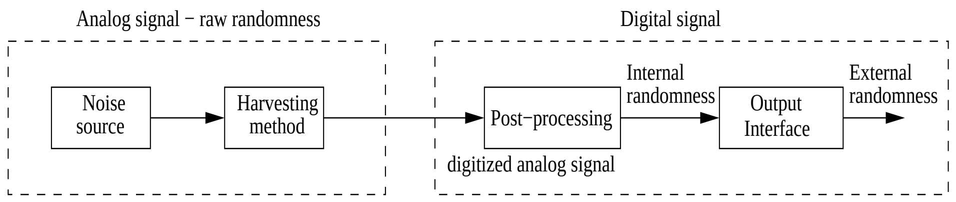


Figure 5.1: TRNG generic architecture

- **Jitter**. Roughly speaking, jitter can be defined as short-term variations or displacement of some aspects of the pulses in a digital signal. It can be seen as a variation in characteristics of the pulse such as amplitude, frequency or phase of successive cycles.

After briefly describing these source of randomness, it is important to note that if a TRNG is required to be designed with pure digital components, then using the first two options is not suitable. Moreover, if a generator design makes use of any electronic noise as described above, then commonly it is necessary to use an amplifier in order to bring the output of the physical source into a macroscopic realm; also, a sampler is needed for converting the analog output into a digital signal.

5.2.2 Typical architecture of a physical TRNG

We now discuss regarding the general architecture needed for a TRNG design. Without loss of generality, a typical TRNG design is comprised of three basic components:

- the source of randomness
- the sampler
- the post processing phase

This generic architecture is depicted in Fig. 5.1. As pointed out before, a good TRNG design is based on the non-deterministic characteristic of a physical phenomena. This component is the most important part of the generator, since it is crucial for the quality of the random numbers that would be extracted and used later in applications and protocols. The process starts with a noisy source generating an analog signal that is immediately fed into the sampler (the second component of the TRNG) in order to convert the analog signal into a digital one (also called the digitized analog signal or *das*). Thus, a harvesting method is employed for sampling the entropy source with the requirement of not disturbing the physical source in order to collect the maximum entropy. The harvesting method will depend on the noisy source of randomness selected. The *das* numbers so far are called *raw random* numbers.

There is the possibility that some noisy sources can exhibit biases in their outputs. This bias has to be eliminated by a post processing on the *das* numbers. Post processing increases the randomness provided by a TRNG at the price of reducing the throughput of the generator. As in general a post processing technique would discard some bits or compress a bit string to correct the bias. The decision regarding the post processing technique depends on the entropy provided by the raw source of randomness and on the efficiency of the post processing algorithm used. After the post processing phase, it is expected that the probability distribution of the random bits will be closer to the uniform distribution, which may not be the case with the *das* number bits. Thus, the post processing mechanism improves the quality of the randomness by masking the imperfections in the entropy source, providing also robustness in the design. Examples of post processing methods are:

- The Von Neumann corrector
- The XOR corrector
- Extractor functions
- Hash functions (like SHA-1 algorithm)
- Resilient functions

It is important to point out that post processing might not be needed for every single design for a TRNG. It is usually employed to strengthen the generator if the randomness source selected exhibit a bias.

5.3 Our physical TRNG design

5.3.1 Description

Our design is based on the use of electronic devices as noise generators. We use certain electronic components, and the output of these components is converted to a digital signal which is measured by a micro-controller. Then we perform a post-processing to unbias the bits and deliver the randomness. The generator has a serial output compliant with RS232 standard at 115200 bps. It is also possible to use other designs which uses different kinds of interfaces, like, USB, Ethernet etc. in order to achieve higher throughputs. In Fig. 5.2 we show the schematic circuit of the true random number generator.

The circuit consists of five basic parts. The first part is the noise generator, which is composed of two transistors ($T1$ and $T2$) with their bases interconnected creating an

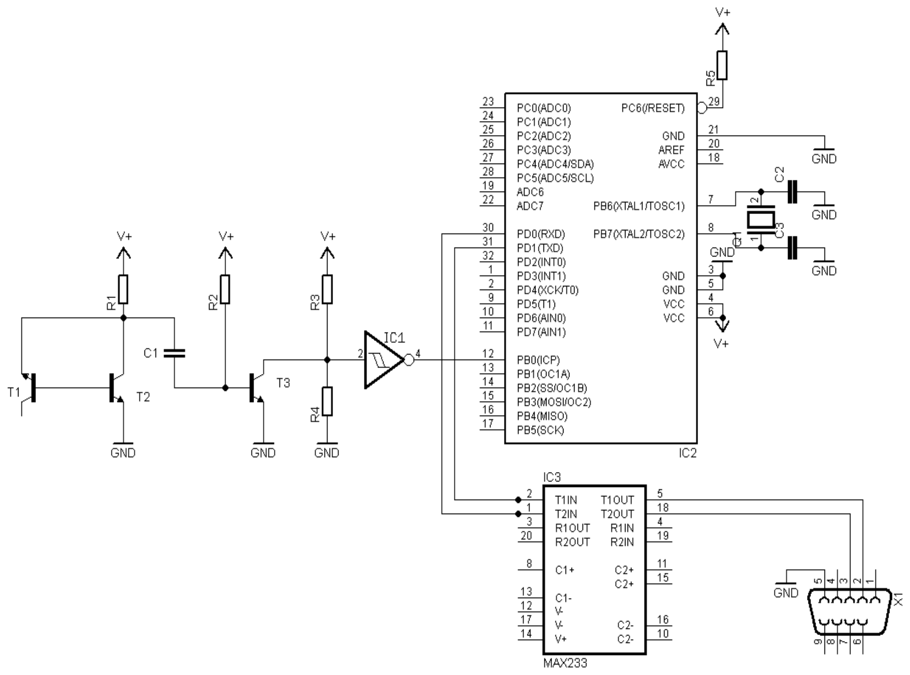


Figure 5.2: Schematic circuit of the TRNG

avalanche noise in the reverse-biased transistor. The second part is the amplifier, it is composed of one transistor ($T3$) that amplifies the noise generated by the components in the previous part. The third part, the digitizer, is composed of a Schmitt trigger gate ($IC1$). This produce a square wave with an unpredictable frequency and pulse width. The fourth part, the processing unit, is composed of a micro-controller ($IC2$) with an embedded C program to measure the pulse width and then post-process the bits to generate the output. The fifth part, the interface, is composed of an integrated circuit ($IC3$) to translate voltage levels between TTL and RS232 standard.

The micro-controller unit (MCU) used in our design, is an ATMEGA8 from Atmel. These MCUs has RISC architecture and can execute 10^6 instructions per second at 1 MHz. We chose C language to write the firmware that measures the pulse width and process the bits. The basic idea for measuring the pulse width is to use an internal module called Timer1 with *Input Capture* capability. Timer1 is implemented internally as a counter, so when the module starts, the counter increments its value at each clock cycle. Then when a falling/rising edge is detected the MCU produces an interrupt and the value of Timer1 is recovered. Its value is a variable of type *int* (16 bits), but for this work we only use the Least Significant Bit because it has the higher probability to change in any measurement.

The next step in the program is to perform a procedure to unbias the bits and store them in groups of 8 before being sent to the computer. There are different methods for bias corrections, i.e., the Von Neumann corrector, the XOR corrector, hash functions

Sampled bits	**Output bit**
$0, 0$	None
$0, 1$	0
$1, 0$	1
$1, 1$	None

Figure 5.3: The Von Neumann corrector

Table 5.1: NIST test suite results

seq. length	bias corrector	NIST test suite results
$100,000$	Von Neumann	Passed all tests but *RandomExcursions*
$1,000,000$	Von Neumann	Passed all tests but *CumulativeSums*
$100,000$	XOR	Failed 5 tests
$1,000,000$	XOR	Failed 4 tests
$100,000$	None	Passed all tests but *LinearComplexity*
$1,000,000$	None	Failed 3 tests

(i.e. SHA-1), etc. The Von Neumann corrector [23] is employed in our design because of its simplicity and the quality of the randomness obtained in our design as explained next. The Von Neumman method is very simple as can be seen in Fig. 5.3; per every 2 bits sampled, the program makes a comparison, discarding them if they are equal. If not, we store the first bit and accumulate bits until we have 1 byte to send it over the serial interface. We measured the throughput of our genetaror with the Von Neumann corrector and is giving 4096 bytes every 0.03 secs (note that the size of each random page we require for our scheme is 4096 bytes). We remark that using the serial interface can slow data transmission between MCU and the computer, but fortunately we can replace it with USB or Ethernet interfaces to achieve higher throughputs.

5.3.2 Testing the randomness of the generator

The quality of the generated randomness and its entropy is crucial for the system. We analyzed the randomness associated with the sequence of bits output by our generator. There exist many statistical tests that can be applied to verify the quality of randomness. In our case, we decided to make use of the battery of test (composed of 15 tests) from NIST [24]. For performing our tests, we collected files with at least 20 MB of random binary sequences that were provided by our physical TRNG connected to the computer by the USB-Serial interface. Different files were created from the physical TRNG with the Von Neumann corrector, the XOR corrector and without any correction method (raw randomness). Later, these (different) files were feed to the NIST's battery of tests. We run all the test using the suggested parameters by NIST. It is also required to define how many bit streams (sequences of zeros and ones) of a desired length will be parsed by the tests using every input file. We selected 100 as the number of bit streams. For the bit-sequence length, we chose $100,000$ and $1,000,000$ as the values. The results in terms of bit-sequence length, post-processing method and test results are presented in Table 5.1.

With the Von Neumann method, the generator presents a nice performance in terms of its randomness quality in contrast with the use of the XOR method. This is at the cost of reducing its throughput (got reduced to $\frac{1}{4}$). Also, we remark that without any post-processing (raw randomness) the generator has a good performance. Regarding the entropy of the random data provided by the generator, we used a software utility to calculate the entropy for input blocks of 4096 bytes. Recall the entropy formula $-\sum_{i=1}^{n} p_i \times \lg p_i$. In short, the test calculates the frequency of every possible value in a byte $(0 - 255)$ and then proceeds to apply the entropy formula. We had as a result for the entropy value 7.948536 per byte. These results thus convinced us that the TRNG provides adequate randomness.

Chapter 6

The Implementation of the Scheme

The way to be safe is never to be secure.

Benjamin Franklin

We designed and developed a software application that allows users to securely send and receive e-mails messages by using a prototype implementation of a crypto-system in the limited access model. All the theoretical aspects and contributions were analyzed and tested through our implementation. The prototype covers the whole system and it is meant as a proof of concept of the possibility of a future practical and real implementation. As stated before, we know of a single work in this direction as reported in [2]. We take that implementation as a starting point and fill many gaps and develop a working prototype of the system. Our implementation gave us the opportunity to analyze the model in terms of the implementational constraints, and also allowed us to test the suitability of the chosen parameters in the functioning of the system as described in Chapter 3.

In this chapter we describe the details of our implementation addressing the different features required and the aspects to be concerned with in a real environment. Among the different entities and algorithms which forms the model, we have implemented the page reconciliation protocol which plays an important role in the security of the scheme and our design of the protocol also helped to decrease the size of the initial key. Our implementation was completely written in python[25] with the help of some other software tools intended to provide a more realistic scenario, but users need only to have in their computers the python interpreter in order to use the application. The solution is divided into two basic modules: the *psn system* and the *user client system*, which are explained in this chapter. Finally we performed some tests of the system and argued about the efficiency and security of our implementation.

6.1 Brief description of the system

In order to simulate a real scenario, our implementation covers both the software for the PSN side and the user client side. In contrast to the previous attempt as reported in [2], we built a physical random number generator (being used by every PSN in the model) in order to provide true random bits to users and not pseudo-randomness which is not suitable anymore in this context.

As a summary, our implementation from the PSN point of view has the following characteristics:

- The servers are always listening through an specific port in which they can communicate with the users.
- Every time a new page needs to be constructed, the required randomness is extracted from the TRNG connected to the server by an USB-serial interface.
- Pages are created only after the first request. And gets stored till the second request and are destroyed there-after to preserve security.
- An eviction policy has been defined for those pages that has been served only once and no new request has arrived to the server in a reasonable amount of time.
- For this implementation we did not had access to multiple computers. We had two physical computers where we have set up multiple servers using a virtualization software tool called VirtualBox [26]. These virtual servers all have an independent IP address and can independently listen and reply to user request.

The client software has been designed to allow users to easily and securely send e-mail messages (of any length) to their contacts at any moment. Users can add all the contacts they want, and the application has been built to internally manage all the features of the model without requiring the participation of the client user. We now list the basic features that we considered for our design of the application:

- Users can manage the contacts they desire to have registered in the client application.
- There is an interface designed for composing new e-mail messages to the registered contacts.
- Users have an inbox meant to receive all the incoming e-mail messages from their registered contacts. Also they can reply or forward received messages.

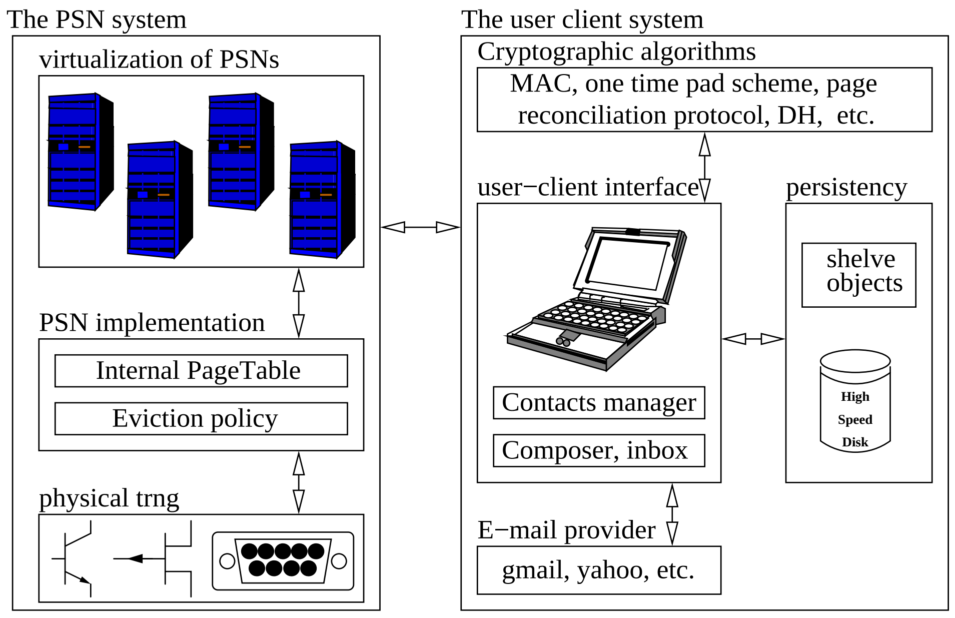


Figure 6.1: Architecture of the prototype

- All data needed by the application is stored in the local computer, so persistency is assured to all the users.

- A user does not need to stick to a single computer; if they happen to run the application in another computer, they only need to copy the application folder and everything will work without inconveniences.

- The application makes use of the user email account in order to send all the system messages (i.e. encrypted messages, protocol messages, etc.).

- We have implemented the cryptographic algorithms and protocols needed by the system to securely transmit the e-mail messages. We remark that we implemented the page reconciliation protocol as was described in Chapter 3, where a user (say Alice) starts the protocol by computing the foot prints of every downloaded page and sends them encrypted with one-time pad along with the corresponding MAC tag. Then, the other user (Bob) receives the enciphered foot prints and compared them with his own computed foot prints, so finally he construct a list of the common pages between them.

In Figure 6.1 we present the architecture of the prototype implementation, depicting all the elements involved in our development. Now, we describe in detail the PSN system and the user-client system.

6.2 The PSN system

As pointed out before, this part of the development has been done using python [25]. For our tests, an instance of the physical TRNG explained in Chapter 5 is connected to the PSN by the USB-Serial port. Moreover, the python program reads the port every time a new random page is required. Our implementation allows users to communicate with a PSN through a TCP socket on a selected port. In our case, the number of this port is always 17078 but it could be some other agreed number above 1024.

For running this python application, a server that will be acting as a PSN will have no special requirements. In fact it only needs enough memory to hold the maximum amount of pages allowed for every PSN which is 2^{16} as pointed out in the choice of parameters section in Chapter 4. Moreover, this server does not need graphic interface, so a lightweight installation of any distribution of Gnu/Linux (we recommend Slackware for instance) will do along with the python interpreter.

So, as a starting point the application checks the connection with the USB-Serial port to be prepared with the construction of random pages. Also, it creates a TCP socket object and binds it to the corresponding *IP address* and port, and at this point it is ready to receive requests from the users.

Each time a user wants to request a page from a selected PSN, (s)he sends the page identifier (16 bits long) through the TCP socket. The random bytes are then sent back to the user through the open TCP socket. It is important to specify that for every request (which may come from many different users), a new connection must be done with the PSN. We now describe the procedure followed by the server every time an user requests a page of random data to any PSN:

- Since the server is always listening for connections, when a new one arrives then in background it starts to work.
- First, it checks internally in the PageTable stored in its memory for a page associated with the incoming page identifier.
- If found, then it proceeds to internally delete the page and its identifier and returns back the randomness.
- If not found, then issues the execution of the *genNewRandPage* method for generating a new page out of the randomness provided by the TRNG connected to the USB-Serial interface.
- Finally the random page is returned to the user through the established connection. In our development, this procedure is performed in packets of 32 bytes each until the total 4096 bytes are delivered.

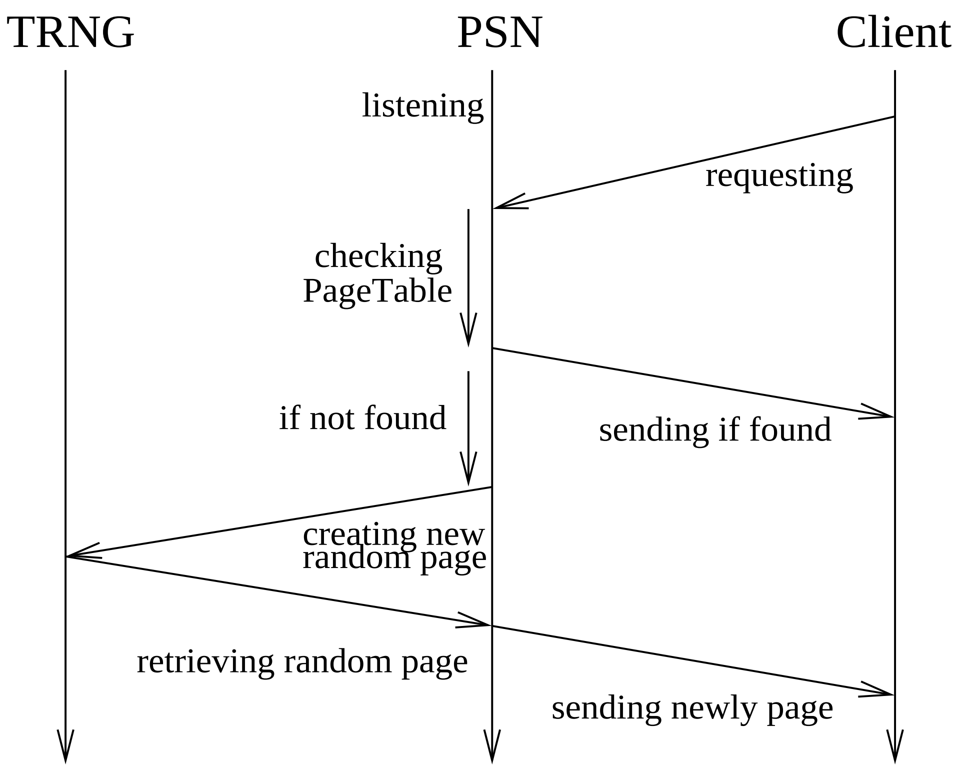


Figure 6.2: The time line during a page request to any PSN

This procedure has been depicted in Figure 6.2. Also, an eviction policy regarding stale pages has been implemented. In this regard, if a page which has been served-only-once has not been again requested in 24 hours, then automatically the page is destroyed along with its identifier from the PSN internal memory. This policy is needed to minimize the threat of denial of service attacks. In terms of implementation, two simple classes have been written in python code (this code can be seen in Appendix 1) and they are explained below:

- The class *PSNEngine*. This class is in charge of the internal PageTable management, keeping track of every page served-only-once and its timestamp for the eviction policy. It has special methods for performing the connection to the TRNG and reading the port, the generation and grouping of the bytes in a single page, the daemon for the eviction duty and the management of the internal PageTable.

- The class *PSNListener*. This class is meant to create the socket connections for the communication with the clients, being always ready listening for incoming requests. This class works along with the previously explained class for serving the randomness. Then, this class after receiving the random data from the class *PSNEngine* delivers directly the page to the corresponding user.

Finally, we also measured the average time elapsed in a local network when a user request and receives a new page generated by a PSN. After performing 100 tests, our average measure is 35 ms.

6.3 The user client system

The user client system has also been written in python with pyqt4 [27] for the user interface. The system lets parties communicate securely with an easy-to-use interface. All a user needs is to have an email account (like gmail) and can register all the parties that (s)he wants to communicate with. Later, (s)he can send e-mail messages securely with the contacts. In this sense, it is important to point out that only registered contacts will possibly communicate with the user.

In this section we now describe the functionality of our implementation regarding the software to be run in the client side of the encryption scheme. Later we will discuss the implementational details of the application.

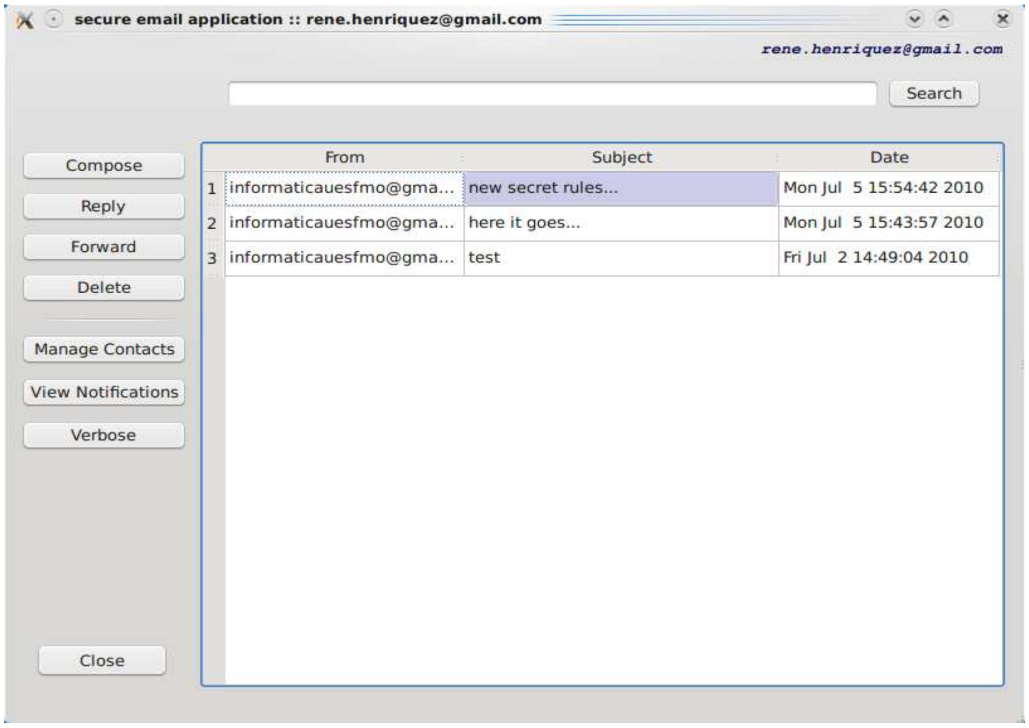
6.3.1 The functionality of the application

Every time a user desires to communicate securely through our encryption scheme in the limited access model, it is only necessary to start a simple and friendly python program from a text-based console or by the graphic environment. There are only two requirements in the user side to run the application, which we state as follows:

- *the python interpreter*. Since all the instructions has been codified using the python programming language.
- *the qt bindings*. Since the graphical user interface of our application has been developed using this application framework.

Once these requirements are completed and the user has started the application, a login form will appear asking for a valid e-mail account along with the corresponding password. As it might be seen, this is necessary because the application will be using this communication medium in order to send and receive all the system messages associated with the encryption scheme. Otherwise, the application will not allow any user to connect with it. Different users can login with different accounts in the same computer. The application has capability to manage the different files for every possible user.

After logging in, the principal window of the application will show up as seen in Figure 6.3. This main window is composed of the following elements:



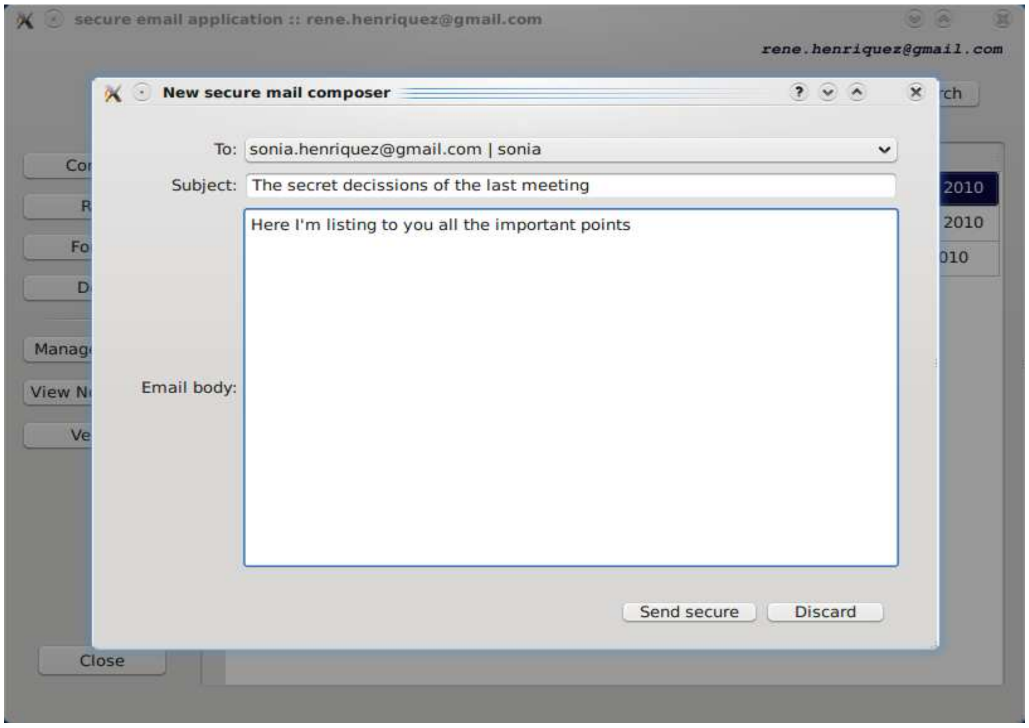
	From	Subject	Date
1	informaticauesfmo@gma...	new secret rules...	Mon Jul 5 15:54:42 2010
2	informaticauesfmo@gma...	here it goes...	Mon Jul 5 15:43:57 2010
3	informaticauesfmo@gma...	test	Fri Jul 2 14:49:04 2010

Figure 6.3: The main window of the client application

- *The inbox pane*. This element will show chronologically (starting from the newest one) all the incoming messages from all the registered contacts of a user. Each row represents a single message and it shows information about the user who sent the message, the subject of the incoming e-mail and the received date.

- *The search bar*. Users can search for messages stored by the client application by simply entering some text in the field to the left of the search button. This text will try to be matched by the application in order to find possible messages related to the search string written by the user. If there are multiple messages returned by the search mechanism, then a grid with the possible messages will be presented to the user for selecting one of them.

- *The buttons pane*. This is a set of options for actions over incoming messages (reply, forward and delete) as well as to create new ones (compose). Moreover, there is the option for the management of the contacts and two special windows, one for viewing notifications of unregistered contacts who wants to get in touch with the user, and the other for viewing the verbose of the application session.


secure email application :: rene.henriquez@gmail.com

rene.henriquez@gmail.com

New secure mail composer

To: sonia.henriquez@gmail.com | sonia

Subject: The secret decissions of the last meeting

Email body: Here I'm listing to you all the important points

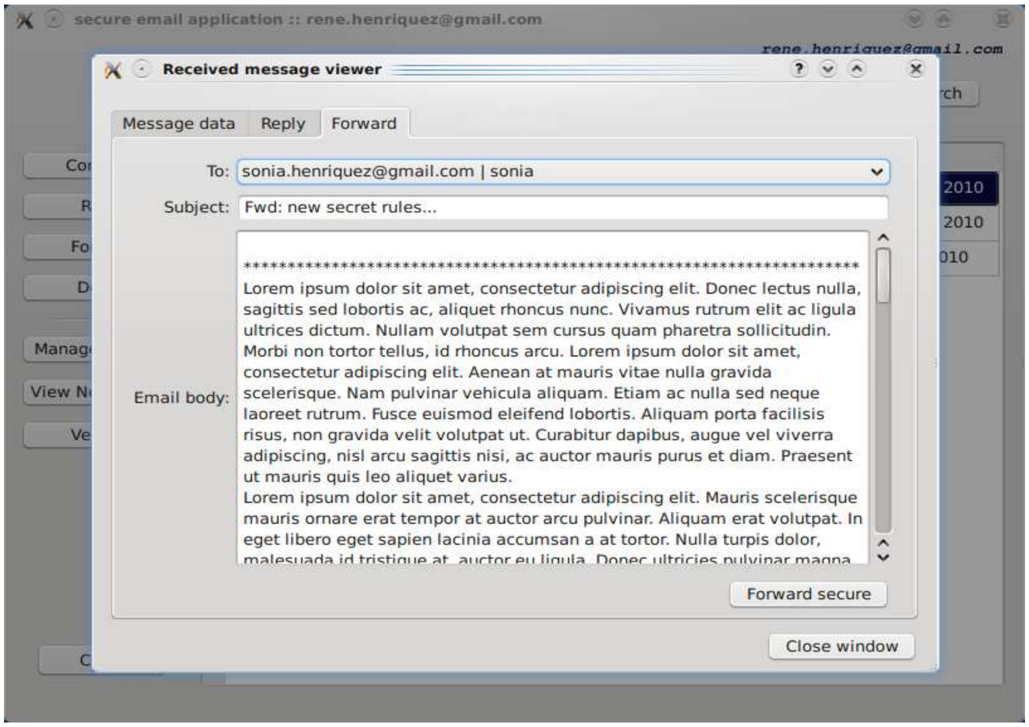
Figure 6.4: The mail composer window of the client application

Composing a new message

Every time a user needs to create and send a new message securely to one of the contacts, first it is necessary to click in the *Compose* button part of *the buttons pane*. Right away, a simple window as the one depicted in Figure 6.4 will appear for allowing the user to compose the new message. As seen in Figure 6.4, it is only necessary to fill up the fields of the subject and body of the new e-mail. After these fields are completed, the user needs to click the *Send secure* button at the bottom of the window. Starting from that point, the application will start to perform all the tasks involved in the model to communicate securely with the selected contact.

Visualization and actions over the messages

The client side application is in charge of safely performing all the tasks required to receive every message from any of the previously registered contacts. In this regard, again the user needs not to worry about anything related to the cryptographic functionalities since everything is done by the application. Thus, all the decrypted messages that appear in the inbox pane can then be viewed by the user by just double clicking the desired message. Now it is possible to see the complete e-mail body along with the subject and the received date. Moreover, the user can also reply that e-mail right away



To: sonia.henriquez@gmail.com | sonia

Subject: Fwd: new secret rules...

Email body:

```
************************************************************************
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec lectus nulla,
sagittis sed lobortis ac, aliquet rhoncus nunc. Vivamus rutrum elit ac ligula
ultrices dictum. Nullam volutpat sem cursus quam pharetra sollicitudin.
Morbi non tortor tellus, id rhoncus arcu. Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Aenean at mauris vitae nulla gravida
scelerisque. Nam pulvinar vehicula aliquam. Etiam ac nulla sed neque
laoreet rutrum. Fusce euismod eleifend lobortis. Aliquam porta facilisis
risus, non gravida velit volutpat ut. Curabitur dapibus, augue vel viverra
adipiscing, nisl arcu sagittis nisi, ac auctor mauris purus et diam. Praesent
ut mauris quis leo aliquet varius.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris scelerisque
mauris ornare erat tempor at auctor arcu pulvinar. Aliquam erat volutpat. In
eget libero eget sapien lacinia accumsan a at tortor. Nulla turpis dolor,
malesuada id tristique at, auctor eu ligula. Donec ultricies pulvinar magna
```

Figure 6.5: The received message viewer window of the client application

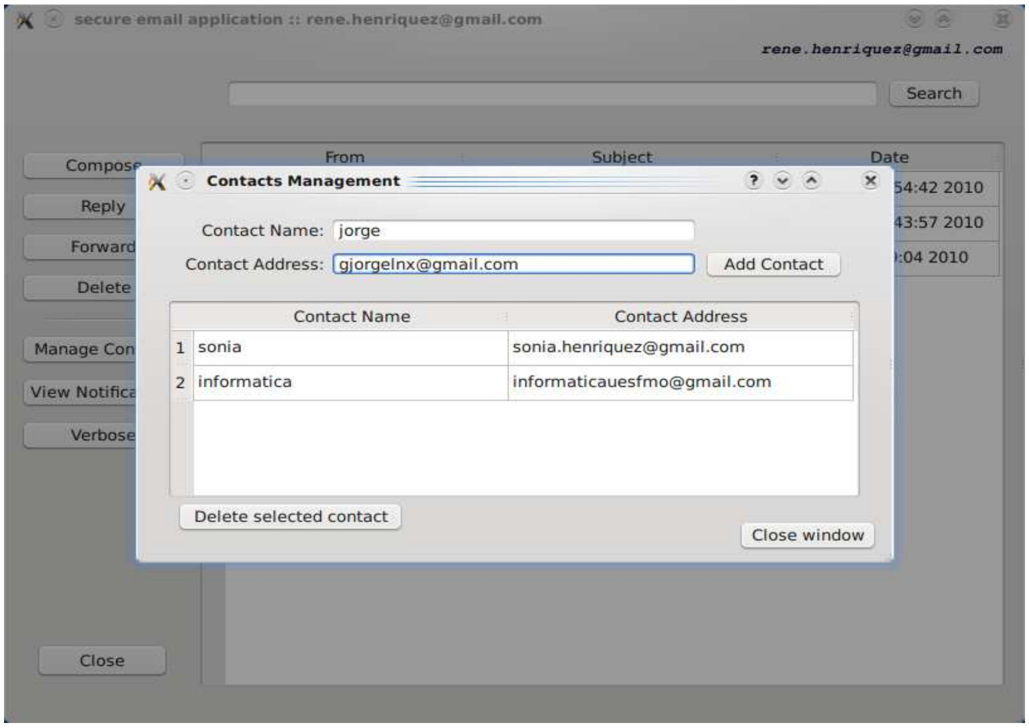
or forward it securely to another of the registered contacts as seen in Figure 6.5. Also, any message can be deleted forever by first selecting it from the inbox pane and then clicking the *Delete* button.

Management of contacts

Since all messages have to be exchanged only between the contacts registered in the application, then a way for their management is needed. For this purpose, our implementation has a specific interface in which it is only necessary to add the contact name and email address in order to register successfully. The application takes care of not letting the user register the same account twice so the integrity in the management of the data is assured. Also, it is possible to modify the data of any already registered contact as well as delete it. This window can be seen in Figure 6.6.

Other functionalities

It is possible that a non-registered contact tries to communicate and sends a message to the connected user. In this sense, the application will detect that the incoming message belongs to a person not present in the contact list and will notify of this to the connected



Contact Name: jorge

Contact Address: gjorgelnx@gmail.com

	Contact Name	Contact Address
1	sonia	sonia.henriquez@gmail.com
2	informatica	informaticauesfmo@gmail.com

Figure 6.6: The contacts management window of the client application

user. Then he can decides to accept and register this new contact or just ignore the notification. Also, there exist an option for viewing the verbose of the application having the opportunity to monitor or check all the activities that the application is performing. This is important since we have pointed out that users do not need to get involved regarding the execution of cryptographic protocols, but still (s)he can check out the status of the application.

6.3.2 Structure of the Software Implementation

The solution has been structured into different python classes, where each class represent a window in the client application. This modularization of the the client application has brought benefits for simple management of the whole system. Moreover, for maintaining storage and persistence of the data involved we use the so-called python *shelve* objects[1], which provides the required storage features without depending on complex database management systems. Then, these persistence key-value-like objects increase the portability of the client software since no more dependencies than the python interpreter are needed. Thus, it is a lightweight software. If it is necessary to port the application along with the stored data to a different computer, it can be

[1]http://docs.python.org/library/shelve.html

easily realized by copying the application directory to the new location.

EmailClientGui class

The core of the implementation resides in this class which is responsible of managing the main application window. This class performs a variety of different tasks like being aware of incoming requests, performing decryptions, manipulating and discarding the already used keys among many other duties. We describe now the most important methods of the class:

- *callbackSetText*. Every time a user has logged in successfully to the application, this method will be executed. It is in charge of storing the received objects from the logging phase (like the user connection to the email account), the creation of the cryptographic tool objects, the initialization of the shelve objects and threads needed to check for new messages coming from the contacts.
- *searchMail*. This method performs a search through the received and stored e-mails by using a text phrase entry by the user. If it encounters many options, a window with all of them will appear for the user to select the right one.
- *deleteAction*. This procedure will be triggered every time a user decides to delete a received message forever.
- *checkAccount*. This method checks the inbox of the user email account, for new system messages identified by special e-mail headers recognized by the application.
- *mailDaemon*. This method schedules the execution of the checkAccount method.
- *initDiffieHellman*. If two contacts wish to communicate and they do not agree yet on an initial key, then this method is performed for an initial key exchange request to the other party. This method makes use of the crypto tools codified for our implementation. Also, this key exchange message is sent with an special header called *X-Lam-Message* with code 100.
- *replyDiffieHellman*. This method is executed after receiving a message with code 100 and performs the last part of the Diffie-Hellman protocol for the initial key-exchange. This message travels with the code 200 in the header *X-Lam-Message*. When this message has been received in the client application of the other party, the method *finalizeDiffieHellman* is executed so now users will be ready to start the limited access model protocols.
- *initSecureEngine*. This method is performed before starting the connection to the PSNs for requesting the randomness. It checks the length of the message to be exchanged in order to know how many random bytes will be needed for encrypting

with one-time pad scheme. After this, then it start the connection with the randomly selected PSNs from the common list of servers hold among the users of the system.

- *requestPsnBytes*. It is executed by receiving messages with codes 300 and 400 in the *X-Lam-Message* header. This method is used to make the connections to the PSNs for requesting the pages of random data that will be used later to form the one-time pad. These requests are performed by threads saving some time for downloading the randomness. Also, here it is computed the tag of every page by using the MAC function which is part of the application crypto tools. Per every tag, it only takes the last 32 bits. After this process, one of the users starts the page reconciliation protocol by sending a message with code 500 in the *X-Lam-Message* header. This message with all the page tags travels encrypted with one-time pad using part of the initial shared key.

- *connect2Psn*. This is the procedure used by the threads mentioned in the method above for connecting and retrieving the randomness from the servers. Per every server connection, it tries twice to request the pages of random data. After these two trials, if the connection could not be done then the NULL value is stored.

- *reconcilePages*. This method receives the first reconciliation message for the downloaded pages of randomness. Basically, after decrypting the received message it compares every foot print computed by the other party with the corresponding page foot print computed locally. A list of the common indices is sent to the contact in a new message with code 600 in the *X-Lam-Message* header.

- *checkReconciledMessage*. It is executed for processing the second message in the page reconciliation protocol, based on the received message it selects the common page indices and form the final page. If there are enough bytes in the one-time pad, it proceeds to encrypt. Otherwise, it starts a new round of connections with the PSNs.

- *otpEncryption*. This method is used for encrypting the plaintext that a user wish to send with the one-time pad encryption scheme. It computes the bit-wise XOR of the plaintext with the constructed pad. Then, this new message with the encrypted string is sent with code 700 in the *X-Lam-Message* header.

- *hyperEncryptedResponse*. This procedure receives the encrypted message and proceeds to decrypt it with the shared long key. The newly received message is shown in the inbox pane. Moreover, the bytes already used are discarded by this method. A message of confirmation is sent with code 800 in the *X-Lam-Message* header.

- *finalizeHyperProtocol*. This method finalizes the encryption scheme in the limited access model. Here, with the confirmation message the client application knows that the other party received and decrypted the message, so now it can just discard the used keys.

- *createOtpPages*. Method for creating new OTP final pages. This is done by using the reconciled pages in groups of 30 pages. In this way, the initial key is updated and the encryption/decryption bytes are increased.

StartApp class

This class implements the login window of the system where the users connect with the client application. The class has a main method called *signIn* which verifies the credentials of the e-mail account given by the user who is trying to log in. If the connection to the email account of the user could be done, then the main application window is loaded ready to use.

ComposerManagement class

This class provides the window for letting the users compose new messages to be sent to a selected contact. In this window, it is only necessary to fill out the subject and body fields and then click the corresponding send button. The first task that this class does is to check if the users already share an initial secret key to start the protocols of the limited access model. Otherwise, they issue the Diffie-Hellman protocol for an initial key-exchange which will be performed only once.

ContactsManagement class

This class implements the contact management window. It has the following methods:

- *fillContactsTable*. Method that fills the contacts grid with all the registered data. This data can be viewed, modified and deleted by the user.

- *addContact*. Performs the registration of a new contact, saving the username and email account for future communications.

- *modifiedDataHandler*. It is triggered every time a user alter the data corresponding to a selected user in the grid. It checks if an email account has not been repeated for maintaining the integrity of the contacts information.

- *delContact*. This method performs the delete action over a selected contact.

ViewerManagement class

This class has been designed for allowing users to have a detailed view of any received message in their inbox pane. In this regard, they can also perform actions like replying the received e-mail or forward it to a new registered contact. The first task that these class does is to check if the users already share an initial secret key to start the protocols of the limited access model. Otherwise, they issue the Diffie-Hellman protocol for an initial key-exchange which will be performed only once.

ImapServerConn class

Since there is a strong necessity of connecting periodically to email accounts for retrieving and sending e-mails so we have implemented this class for providing all the necessary methods that facilitate the tasks of the application. The class has the following methods:

- *imapConnect*. It tries to connect to the mail server of the user email account with the corresponding credentials.
- *getImapStatus*. It returns the status of the connection performed by the method above.
- *getNewMessages*. This method is used by the application to check and retrieve new messages (but only those who belong to the system by checking the presence of the *X-Lam-Message* header) from the user email account.
- *getEmailFields*. This method helps for extracting the required fields for every message retrieved from the email account. Specifically, it extracts the *from, subject* and *body* fields.
- *deleteEmail*. It performs the delete action over a selected e-mail. This is necessary since every message processed by the scheme is not useful anymore.

SmtpServerConn class

This is a tiny but useful class employed for sending all the messages that take place during the execution of the different methods in the whole system. Basically, it sets the headers *from, subject, body* and also the *X-Lam-Message* new header. Then, it performs all the steps needed to communicate with the SMTP server and finally sends the message.

GaloisFieldArithmetic class

This class has been implemented for performing arithmetic operations in binary fields, which are needed for cryptographic functions like the MAC algorithms required by the protocols in the limited access model. This class has the following useful methods:

- *binaryAddition*. The basic binary addition is performed by this method, taking as input parameters the two operands as binary strings and giving as output the resulting addition.
- *int2bin*. This method convert an integer value into its binary representation.
- *karatsubaOfmanMultiplier*. Since the information theoretically secure MAC algorithm employed in our development calls for polynomial multiplications with coefficients in the binary field, then we have implemented the renowned Karatsuba-Ofman multiplier. It takes two operands each of 128 bits long and gives as a result the corresponding multiplication. Also for the computation of foot prints performed by the Poly function, it takes operands of 32 bits long each and gives the result of the multiplication.
- *polynomialReduction*. Since the multiplication will output an element outside the desired extension field ($GF[2^{128}]$) then a reduction must be done. In our implementation, this is performed by using the irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$ for computing $Poly_\delta^\gamma()$.

CryptographicTools class

This class implements the secure MAC and the Diffie-Hellman protocol. The secure MAC makes use of the arithmetic provided by the *GaloisFieldArithmetic* class explained above. In the case of the key-exchange protocol, it performs exponentiations in the prime field arithmetic by using the add and multiply method.

6.4 Testing the system

The user client application was run in many laptops and desktop in which many users provided an e-mail account, and the prototypical implementation was tested by sending and receiving e-mail messages. The goal was to use and test the different application features described in the previous section. The application was run in the local area network (LAN) of our Computer Science Department.

We observed and measured the time required by the whole system to deliver the e-mail messages sent among the users. This is an interesting aspect to analyze since in the performance of the model there are many entities involved (i.e. the PSNs, the run of protocols and algorithms, etc.) as we shown before. This complex structure of the protocol can have a dramatic impact on the amount of time needed to encrypt and decrypt an e-mail message with the scheme. For example, for messages of $12,000$ bytes it took on average between 4 and 5 minutes to get delivered by the other communicating party.

We are aware that these timings are not efficient and of course they will increase as the size of the e-mail messages increase. But the performance of the system can be improved significantly with the following ideas:

- Changing the programming language for the implementation. Instead of using the interpreted language *python* we can use the more efficient C language.
- It is possible to improve the Poly and MAC functions which are widely use in the page reconciliation protocol, by not only implementing better algorithms for the finite fields operations but also by codifying in assembly code.
- Perform the parallelization of some operations (like the request of pages to the PSNs in every round, where n pages are requested to n different servers).

The last idea has been implemented in our case, by using python threads. Still, we know there are other parallelization options that work more efficiently than threads. We also tested the security of the system in the scenario where a fraction of PSNs were compromised, delivering dangerous data to the users posing a threat to the system. These PSNs were virtualized in another machine, acting as available PSNs to the users. The servers were receiving request, and they served different pages to the users so they would not be able to reconciliate properly, and so decreasing the number of commonly shared pages. We had tests in which 25 PSNs were running and 5 of them were compromised, following the limited access model assumption. In this regard, we could practically notice how the system still worked out well even when some of the servers were injecting compromised data in the network.

Chapter 7

Conclusions

The one unchangeable certainty is that nothing is certain or unchangeable.

John F. Kennedy

In this chapter we point out the achieved results in our work and the conclusions regarding the analysis of our implementation. Moreover, we propose some directions for future work.

7.1 Conclusions

This work presented the analysis regarding a practical implementation of a crypto-system that can provide *information theoretic security* in the *limited access model*. We were interested in exploring how to overcome the natural drawbacks of information theoretically secure schemes, by analyzing and modifying the components involved in the limited access model toward a realistic implementation.

The importance of the project lies in the possibility of a practical *provably unbreakable* encryption scheme in which we are neither making any assumptions regarding the computational capabilities of an adversary nor the computational complexity of any mathematical problem as modern schemes used nowadays do.

The prominent contribution of this work is the precise description of every entity and algorithm associated with an encryption scheme in the limited access model, such that they are amenable for a prototypical implementation of a practical crypto-system with the appeal of *everlasting secrecy* characteristic.

Even though there exist a previous attempt of a practical implementation [2], that work presents many gaps with many interesting parts left open for possible improvements or redefinements. Thus, we took the implementation in [2] as a starting point and filled many of these gaps developing a working prototype of the system.

Furthermore, we showed modifications in the page reconciliation protocol which is an important component of the system and proposed reasonable values for the various parameters involved in the model, improving in the security of the system. This also resulted in a dramatic decrease of the initial key length.

We also argued regarding the security of the scheme, pointing out possible vulnerabilities and pitfalls that can be exploited by an adversary and then presented countermeasures of how our system can protect itself against those vulnerabilities, justifying the various choice of parameters made previously.

Additionally, we described the design of a simple but efficient random number generator which plays an important role in the system, since this generator is supposed to be embedded in every Page Server Node for providing the required randomness to the users. Finally, we presented the prototypical implementation of our encryption scheme in the limited access model.

Our work can be seen as a step forward towards a practical system in the limited access model. Though we claim improvements over [1] and [2] but we still do not claim that this scheme can be widely deployed and replace existing schemes. Computational security as provided by modern cryptographic algorithms are in most practical cases sufficient and there are various kinds of crypto-systems which till date are considered secure. The model and the encryption scheme described in this thesis is no doubt much more complicated than popular existing schemes and has characteristics which are very different from existing schemes. Though the security of the scheme does not depend on computational assumptions but it does depend on a strong assumption on the existence of the PSN network and the inability of the adversary to control the whole network. Though this seems to be a feasible assumption given the current state of technology but this assumption is far less studied than the assumption that "factorization is hard". So, to conclude we think we have made a significant advance in analyzing a non-conventional crypto-system and have shown evidence of the practicality of the limited access model and the encryption scheme under the model.

7.2 Future work

We note down below some directions of future works in the area of the thesis:

1. A deep cryptographic analysis of the cryptosystem is required to find out vulnerabilities present in the system. We have pointed out certain vulnerabilities but there may exist more which we failed to detect.

2. A unified security definition of the protocol is missing. If such a definition can be given then one can try to work out a security proof for the whole protocol. Such a

proof would increase our confidence over the scheme and additionally may help to refine the parameters involved.

3. The implementation that we did is purely prototypical. We chose python as our target language to reduce the over-head of development thus sacrificing efficiency. We believe that a much more efficient implementation of the protocol is possible which we would like to attempt in future.

Appendix A

The PSN Implementation

In n this appendix we show the code we have written for our Page Server Node implementation. The PSN program has been written in the python programming language.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

# ---------------------------------------------------------------
# psn.py
#
# Implementation for a Page Server Node using a
# physical true random number generator, via serial port.
#
# Rene E. Henriquez G.
# 29/03/2010
# ---------------------------------------------------------------

import socket
import struct
import threading
import datetime
import sched
import time

class PSNEngine:
     """

     """
```

```
    dbPages = {}
    randSource = None
    maxDbSize = 1000
    timeFrame = {}

    def __init__(self):
        """

        """
        self.randSource = open('/dev/ttyUSB0', 'r')
        tep = threading.Thread(target=self.evictPagesDaemon)
        tep.start()

    def genNewRandPage(self):
        """

        """
        newPage = ''
        randomness = ''
        randomness = self.randSource.read(4096)
        for rbyte in randomness:
            newPage += chr(struct.unpack('B', rbyte)[0])

        return newPage

    def evictStalePages(self):
        """

        """
        tempDict = dict(map(lambda item: (item[1], item[0]),
          self.timeFrame.items()))
        timeList = sorted(tempDict.keys())
        currentTime = datetime.datetime.now()
        for arrivedTime in timeList:
            if (currentTime - arrivedTime).days >= 1:
                keyPage = tempDict[arrivedTime]
                del self.dbPages[keyPage]
                del self.timeFrame[keyPage]
            else:
                break

    def evictPagesDaemon(self):
        """
        """
        scheduler = sched.scheduler(time.time, time.sleep)
        time.time()
        scheduler.enter(90000, 1, self.evictStalePages, ())
        while 1:
            scheduler.run()
            time.sleep(10)
            scheduler.enter(90000, 1, self.evictStalePages, ())

     def serveRandPage(self, keyPage):
        """

        """
        reqPage = self.dbPages.pop(keyPage, None)
        if reqPage is None:
            self.dbPages[keyPage] = reqPage = self.genNewRandPage()
            self.timeFrame[keyPage] = datetime.datetime.now()
            if len(self.dbPages) > self.maxDbSize:
                self.dbPages.popitem()
        else:
            del self.timeFrame[keyPage]
            print "served and destroyed"

        return reqPage

class PSNListener:
    """

    """

    psnPort = 17078
    psnIp = ''

    def listenOnPort(self):
        """

        """
        print "Initializing psn resources..."
        psnDb = PSNEngine()

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((self.psnIp, self.psnPort))
        s.listen(5)

        print "Psn ready! Listening through port 17078..."

        while 1:
            (conn, addr) = s.accept()

            try:
                cKey = conn.recv(4)
                rndData = psnDb.serveRandPage(cKey)
                for i in range(128):
                    conn.sendall(rndData[i*32:(i*32)+32])
                    conf = conn.recv(2)
            except:
                conn.close()
                s.close()
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                s.bind((self.psnIp, self.psnPort))
                s.listen(5)
                continue

if __name__ == '__main__':
        server = PSNListener()
        server.listenOnPort()
```

Bibliography

[1] Michael O. Rabin. Provably unbreakable hyper-encryption in the limited access model. *IEEE Information Theory Workshop on*, pages 34–37, Oct. 2005.

[2] Jason K. Juang. Practical Implementation and Analysis of Hyper-Encryption. Master of engineering in electrical engineering and computer science, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2009.

[3] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[4] Ueli M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *J. Cryptology*, 5(1):53–66, 1992.

[5] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, India*, pages 175–179. IEEE Computer Society Press, 1984.

[6] Yonatan Aumann, Yan Zong Ding, and Michael O. Rabin. Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, 48(6):1668–1680, 2002.

[7] Erkay Savas and Berk Sunar. A practical and secure communication protocol in the bounded storage model. In Pascal Lorenz and Petre Dini, editors, *ICN (2)*, volume 3421 of *Lecture Notes in Computer Science*, pages 707–717. Springer, 2005.

[8] Michael O. Rabin. Hyper encryption and everlasting secrets. In Rossella Petreschi, Giuseppe Persiano, and Riccardo Silvestri, editors, *CIAC*, volume 2653 of *Lecture Notes in Computer Science*, pages 7–10. Springer, 2003.

[9] Danny Harnik and Moni Naor. On everlasting security in the h*ybrid* bounded storage model. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2006.

[10] Chi-Jen Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *J. Cryptology*, 17(1):27–42, 2004.

[11] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[12] http://en.wikipedia.org/wiki/One-time_pad.

[13] Yan Zong Ding and Michael O. Rabin. Hyper-encryption and everlasting security. In Helmut Alt and Afonso Ferreira, editors, *STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2002.

[14] Stefan Dziembowski and Ueli M. Maurer. Optimal randomizer efficiency in the bounded-storage model. *J. Cryptology*, 17(1):5–26, 2004.

[15] Stefan Dziembowski and Ueli M. Maurer. On generating the initial key in the bounded-storage model. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2004.

[16] Seymour Lipschutz and Marc Lipson. *Schaum's outline of Probability; 2nd Edition*. Schaum's outline. McGraw-Hill, New York, NY, 2000.

[17] Daniel J. Bernstein. The poly1305-aes message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[18] Wolfgang Killmann and Werner Schindler. A design for a physical rng with robust entropy estimators. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2008.

[19] Berk Sunar, William J. Martin, and Douglas R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Trans. Computers*, 56(1):109–119, 2007.

[20] Shashi Prashanth Karanam. Tiny True Random Number Generator. Master of science, George Mason University, Department of Electrical and Computer Engineering, 2006.

[21] http://www.cryogenius.com/hardware/rng/.

[22] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423,623–656, July, October 1948.

[23] Bruce Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C, on page 425.* John Wiley & Sons, Inc., New York, NY, USA, 1995.

[24] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudo-random number generators for cryptographic application. NIST Special Publication 800-22, May 2001.

[25] http://www.python.org/.

[26] http://www.virtualbox.org/.

[27] http://www.riverbankcomputing.co.uk/news.