Implementation of Cramer-Shoup Cryptosystem

By

Steven Ulrick

Table of Contents

Abstract

In 1998, Ronald Cramer and Victor Shoup developed a cryptosystem that is provably secure against adaptively chosen ciphertext attacks while still having the benefit of being a practical scheme. This is a significant breakthrough considering that even the popular RSA Cryptosystem and El Gamal cryptosystem are defenseless against adaptively chosen ciphertext attacks.

The security of Cramer-Shoup cryptosystem is based on the assumption that the Diffie-Hellman decision problem is intractable. I present an implementation of the Cramer-Shoup cryptosystem in C++ programming language developed in the Visual Studio Environment. NTL (Number Theory Library) was used to deal with arbitrary precision integer arithmetic.

Chapter 1 Introduction

The quest for secret communication is age old. One of the earliest examples of this was military generals trying to safely exchange messages through enemy territory. Even if the enemy were to intercept the message, it is crucial that the message is undecipherable to the enemy. This need has continued into the computing age, where computer users must transfer sensitive data such as credit card numbers over insecure networks. This need for secret communication is accomplished through the art and science of cryptography.

As time goes by, the demand for safer and more secure ways to communicate continues to grow. In this project, we implement the Cramer-Shoup cryptosystem which is one of the first of its kind; this cryptosystem is provably secure against adaptively chosen ciphertext attacks and is a practical scheme.

1.1 Cryptography

While there is no universal agreement on the definition of *Cryptography*, we shall use the following definition which is adequate for our purposes: *the scientific study of techniques for securing digital information, transactions, and computations* [1]. One of the main objectives of cryptography is the development of cryptosystems that allow for two parties to secretly communicate over an insecure network. There are numerous

cryptosystems that are being used today, all of which fall in one of two categories: *private key cryptosystems* and *public key cryptosystems*.

1.2 Private Key Cryptosystems

A typical private key cryptosystem works as follows, two parties agree on a private key *sk* in some secure fashion. This can be done face-to-face or by other means but the private key must never be compromised, or the cryptosystem will also be compromised. The parties are now able to communicate over an insecure channel by running an encryption algorithm **Enc**. The sending party encrypts their message *m* by running their encryption algorithm **Enc** using the private key *sk* and message *m* as input. The encryption algorithm being ran with inputs *sk* and *m* is denoted by $\mathbf{Enc}_{sk}(m)$. The message is now "scrambled". This scrambled version is called the ciphertext and is transmitted over the insecure channel. The receiving party will then use their decryption algorithm **Dec** using the agreed private key *sk* and ciphertext as input to decrypt or "unscramble" the ciphertext. The decryption of a ciphertext *c* using a key *sk* is denoted by $\mathbf{Dec}_{sk}(c)$.

1.3 Public Key Cryptosystems

One of the major problems with private key cryptosystems is that the private key must be agreed between the two parties previously by some secure means of communication. This is difficult to do so unless they can meet face to face. The public key cryptosystem provides a way around this problem.

In a public key cryptosystem, each party has a private key *sk* and a public key *pk*. An individual's public key is openly available to everybody while the private key stays a secret to the individual. When a message *m* is to be sent, the sender will use the receiver's public key to encrypt a message *m* to produce a ciphertext *c*, $\mathbf{Enc}_{pk}(m) = c$. Now, for the receiver to decrypt the ciphertext *c* back to the original message *m*, the decryption algorithm will be ran using ciphertext *c* and the receiver's private key *sk* as the inputs, $\mathbf{Dec}_{sk}(c) = m$.

Chapter 2 Notions of Security

A cryptosystem can be evaluated in terms of how secure it is. This evaluation of the cryptosystem is done by recognizing what attacks can and cannot compromise the cryptosystem. There are many different types of attacks used to "break" a cryptosystem. A cryptosystem is said to be resistant against a specified attack if that attack is unable to compromise the cryptosystem.

In the next four sections, we describe four different types of attacks. The attacks are ordered from easiest to defend against, to hardest to defend against; they are: Eavesdropping, Chosen Plaintext Attacks, and Chosen Ciphertext Attacks (CCA and CCA2). We describe each attack by a game that is played with an adversary.

The adversary used in these games is restricted to a probabilistic polynomial time (PPT) adversary. In other words, the adversary is assumed to be a probabilistic algorithm that runs in polynomial time, i.e., for some constants a and c, the algorithm must run in time an^c where n is the security parameter.

2.1 Eavesdropping

Eavesdropping is a common type of attack on cryptosystems. The attack involves a party intercepting a ciphertext c over an insecure channel. For the used cryptosystem to be secure against eavesdropping, the intercepting party must not be able to obtain any partial information about the message m

from ciphertext *c*. The following game provides a formal definition of eavesdropping indistinguishability.

A probabilistic polynomial time adversary *A* creates two plaintext messages m_0 and m_1 (of the same length). We randomly choose one of the two messages and encrypt it producing a ciphertext *c*, known as the *challenge ciphertext*. The adversary is now given *c* and asked to decide whether it is the encryption of m_0 or m_1.

Let *P* be the success probability of the adversary *A* making the correct decision. Obviously, the opponent can succeed with the probability of *P* = ½ by just randomly guessing. We are interested to see if the adversary can do better. If *P* is at most negligibly greater than ½, then the encryption scheme is said to have the property of *eavesdropping indistinguishability*.

The phrase "*P* is at most negligibly greater than ½" can be formally expressed as $P \leq {}^{1}/_{2} + \varepsilon(n)$ where $\varepsilon(n)$ is a negligible function of the security parameter n (used in the key generation). A function is said to be a negligible function if it approaches 0 faster than the reciprocal of any positive polynomial.

2.2 Chosen Plaintext Attack (CPA) Indistinguishability

Chosen plaintext attacks occur when an adversary somehow gains access to an encryption oracle of a cryptosystem allowing them to encrypt anything they would like. An adversary could gain access to an encryption oracle a number of ways, one example would be social engineering. Let's say

that Alice encrypts her emails with her private key *sk* before sending them out. Oscar would like to know the encryption of *m* with Alice's private key *sk*. Oscar could simply ask Alice to forward the message *m* to Bob. Once Alice encrypts the message *m* and sends it out to Bob, Oscar can intercept it giving him $ENC_{sk}(m)$. In this situation Oscar has his own encryption oracle in the form of Alice.

For a cryptosystem to have the property of CPA indistinguishability, the adversary must not be able to obtain any information about message *m* even with this access to the encryption oracle and the ciphertext *c*. A more formal definition of CPA indistinguishability is given by the following game.

Very similar to eavesdropping indistinguishability experiment, an adversary *A* creates two plaintext messages m_0 and m_1. Again, we randomly select one of the two messages and encrypt it producing *c*, the challenge ciphertext. *A* now has to decide whether *c* is the encryption of m_0 or m_1.

Unlike in the eavesdropping experiment, adversary *A* is allowed access to the encryption oracle to encrypt anything they wish including the original plaintext messages to help in making this decision.

Let *P* be the success probability of the adversary *A* making the correct decision. If *P* is at most negligibly greater than ½, then the encryption scheme is said to have the property of *CPA indistinguishability*. Obviously, the adversary could simply use the encryption oracle to encrypt both m_0 and m_1, checking to see if either encryption equals the challenge ciphertext *c* to always

have an outcome of $P = 1$. However, this cannot be the case if the cryptosystem is to have the property of CPA indistinguishability.

A cryptosystem is said to be a *deterministic* cryptosystem if the same plaintext is always encrypted to the same ciphertext while using the same key. A cryptosystem with the property of CPA indistinguishability cannot have an encryptions scheme which is deterministic.

Therefore for a cryptosystem to have the property of *CPA indistinguishability*, an encryption scheme must be a randomized one. That is, the encryption of two identical plaintext messages with the same key should have a very high chance of resulting in two different ciphertexts. It is obvious that an encryption scheme having the property of CPA indistinguishability will also have the property of eavesdropping indistinguishability.

2.3 Chosen Ciphertext Attack (CCA) Indistinguishability

Chosen ciphertext attacks occur when an adversary gains access to not only an encryption oracle of a cryptosystem but also a decryption oracle. Similar to the attacks explained earlier, for a cryptosystem to be considered secure against Chosen Ciphertext Attacks, an adversary cannot gain any useful information about a message *m* from its corresponding ciphertext *c*, even with the added benefit of access to the encryption and decryption oracles. There are some stipulations on how the adversary can access the decryption oracle which is explained formally in the following game.

A probabilistic polynomial time adversary *A* creates two plaintext messages m_0 and m_1 (of the same length). The adversary is now allowed access to both the encryption oracle and decryption oracle. The adversary is allowed to perform as many encryptions/decryptions as they wish. However, only when the adversary is finished using the decryption oracle, are we able to randomly select one of the two messages, encrypt it producing the challenge ciphertext *c*, and give it to adversary *A*. Again, once the challenge ciphertext *c* is given to the adversary, the adversary is no longer allowed access to the decryption oracle. *A* now has to decide whether *c* is the encryption of m_0 or m_1.

Let *P* be the success probability of the adversary *A* making the correct decision. If *P* is at most negligibly greater than ½, then the encryption scheme is said to have the property of *CCA indistinguishability*. An encryption scheme having the property of CCA indistinguishability will also have the property of CPA and eavesdropping indistinguishability.

2.4 Adaptively Chosen Ciphertext Attack (CCA2) Indistinguishability

The modeling for an Adaptively Chosen Ciphertext Attack is almost identical to the Chosen Ciphertext Attack. The one key difference is that once the challenge ciphertext *c* is given to the adversary *A*, *A* is allowed to continue using the decryption oracle. However, there is one stipulation that *A* cannot pass *c* through the decryption oracle. Of course this is because decrypting *c* would simply give the adversary the original message *m*.

An encryption scheme having the property of CCA2 indistinguishability will also have the property of CCA, CPA, and eavesdropping indistinguishability.

Chapter 3 Description of Cramer-Shoup Cryptosystem

Every cryptosystem consists of three basic algorithms: key generation, encryption, and decryption. In this chapter we describe these three algorithms of the Cramer-Shoup Cryptosystem and give a proof of the correctness of the Cramer-Shoup Cryptosystem. We start by giving definitions of some mathematical concepts that we will be using.

3.1 Definitions

A *group* is a set G along with a binary operation $\circ$ for which the following condition hold: [1]

- Closure: For all $g, h \in G$, $g \circ h \in G$
- Associativity: For all $g_1, g_2, g_3 \in G$, $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$
- Existence of an Identity: There exists an identity element $e \in G$ such that for all $g \in G$, $e \circ g = e = g \circ e$.
- Existence of Inverses: For all $g \in G$ there exists a corresponding element $h \in G$ such that $g \circ h = e = h \circ g$. Such a h is called the *inverse* of g.

The *order* of a group G simply refers to the number of elements in G. A group G is said to be *cyclic* if every element in the group can be generated from a single element $g \in G$ such that for all elements in G can be written as g^k for some integer k. This element g would be called a *generator* in group G.

Furthermore, if a group is of prime order, then every element excluding the identity is a generator for the group.

3.2 Key Generation

Assume we have a group *G* of prime order *q*, where *q* is large. A hash function *H* must also be used to hash long strings to elements of $\mathbf{Z}_q$. (An in-depth description of hash functions are given in sections 4.2 and 5.5.2)

Random elements $g_1, g_2 \in G$ are chosen. Also chosen are random elements:

$$x_1, x_2, y_1, y_2, z \in \boldsymbol{Z}_q$$

The following group elements are computed:

$$c = {g_1}^{x_1} {g_2}^{x_2}, d = {g_1}^{y_1} {g_2}^{y_2}, h = {g_1}^{z}$$

A hash function H is chosen from a family of universal one-way hash functions. H should be capable of mapping long bit strings to elements of $\mathbf{Z}_q$

The public key is:

$$(g_1 , g_2 , c , d , h, H)$$

The private key is:

$$(x_1, x_2, y_1, y_2, z)$$

3.3 Encryption

We can assume that the plaintext messages are (or can be encoded as) elements of *G*.

Given a message $m \in G$, the encryption algorithms runs as follows.

Choose $r \in \mathbf{Z}_q$ at random. Then compute:

$$u_1 = {g_1}^r, u_2 = {g_2}^r, e = h^r m, \alpha = H(u_1, u_2, e), v = c^r d^{r\alpha}$$

The ciphertext now consists of:

$$(u_1, u_2, e, v)$$

Note that one group element is actually encrypted to a sequence of four group elements. Therefore, when using this cryptosystem, there will be a blow-up in the size of the ciphertext. This is the price we are paying for making the cryptosystem CCA secure.

3.4 Decryption

Given a ciphertext (u_1, u_2, e, v), the decryption algorithm runs as follows.

Compute $\alpha = H(u_1, u_2, e)$ and test to see if the following equation is true:

$${u_1}^{x_1 + y_1\alpha} {u_2}^{x_2 + y_2\alpha} = v$$

If the equation does not hold good then the decryption algorithm outputs "reject" indicating that the ciphertext is not correct. If the equation holds good, it outputs the following message:

$$m = e/({u_1}^z)$$

3.5 Proof of Correctness

We shall prove that if you encrypt a message, and then decrypt the resulting ciphertext, you will get back the original message.

From the encryption algorithm, we know that $u_1 = {g_1}^r$ and $u_2 = {g_2}^r$. Therefore,

$$ {u_1}^{x_1} {u_2}^{x_2} = {g_1}^{r x_1} {g_2}^{r x_2} $$

$$ {u_1}^{x_1} {u_2}^{x_2} = ({g_1}^{x_1} {g_2}^{x_2})^r $$

$$ {u_1}^{x_1} {u_2}^{x_2} = c^r \quad (1) $$

Similarly,

$$ {u_1}^{y_1} {u_2}^{y_2} = {g_1}^{r y_1} {g_2}^{r y_2} $$

$$ {u_1}^{y_1} {u_2}^{y_2} = ({g_1}^{y_1} {g_2}^{y_2})^r $$

$$ {u_1}^{y_1} {u_2}^{y_2} = d^r \quad (2) $$

Now,

$$ {u_1}^{x_1 + y_1 \alpha} {u_2}^{x_2 + y_2 \alpha} = {u_1}^{x_1} {u_1}^{y_1 \alpha} \cdot {u_2}^{x_2} {u_2}^{y_2 \alpha} $$

$$ {u_1}^{x_1 + y_1 \alpha} {u_2}^{x_2 + y_2 \alpha} = {u_1}^{x_1} {u_2}^{x_2} \cdot {u_1}^{y_1 \alpha} {u_2}^{y_2 \alpha} $$

$$ {u_1}^{x_1 + y_1 \alpha} {u_2}^{x_2 + y_2 \alpha} = {u_1}^{x_1} {u_2}^{x_2} \cdot ({u_1}^{y_1} {u_2}^{y_2})^{\alpha} $$

$$ {u_1}^{x_1 + y_1 \alpha} {u_2}^{x_2 + y_2 \alpha} = c^r \cdot d^{r\alpha} = v \quad (3) $$

Equation (3) is the result of performing the substitutions in equation (1) and (2). Therefore, the test performed in the decryption algorithm will succeed.

Now,

$$u_1{}^z = (g_1{}^r)^z$$

$$u_1{}^z = (g_1{}^z)^r$$

$$u_1{}^z = h^r$$

Therefore,

$$e/u_1{}^z = e/h^r = \frac{h^r \cdot m}{h^r} = m \qquad (4)$$

Equation (4) proves that decrypting the ciphertext results in the original message *m*.

Chapter 4 Proof of Security

No public key cryptosystem is unconditionally secure. Typically you can prove a public key cryptosystem is secure assuming some computation problem is difficult to solve. The security of the Cramer-Shoup cryptosystem is based the hardness of the Decisional Diffie-Hellman problem. In this chapter, we present the proof of security for the Cramer-Shoup cryptosystem.

4.1 Decisional Diffie-Hellman Problem

For a cyclic group G of order q consider the two following distributions:

- **R** – consisting of the 4 elements (g_1, g_2, u_1, u_2) where each element is chosen randomly from the group G
- **D** – consisting of the 4 elements (g_1, g_2, u_1, u_2) where g_1, g_2 are chosen randomly from G. Also, $u_1 = {g_1}^r$ and $u_2 = {g_2}^r$ where r is a randomly chosen element from $\mathbf{Z}_q$.

Given a quadruple, determine whether the quadruple came from **R** or **D**. If there is a statistical test that can distinguish the two distributions in polynomial time then the Decisional Diffie-Hellman problem would not be considered hard. However, no such statistical test currently exists.

The Decisional Diffie-Hellman(DDH) problem is to determine whether a given quadruple $(g_1,\ g_2,\ u_1, u_2)$ is coming from which one of the above two distributions **R** or **D**.

Consider an algorithm A which when given a quadruple $(g_1,\ g_2,\ u_1, u_2)$ outputs either 0 indicating the quadruple is coming from **R** or 1 indicating the quadruple is coming from **D**. Solving the DDH problem means developing such an algorithm A that runs in PPT and has the property that

$$\Pr[A(g_1,\ g_2,\ u_1, u_2) = 1 \mid g_1,\ g_2,\ u_1, u_2\ \in \boldsymbol{D}] -$$

$$\Pr[A(g_1,\ g_2,\ u_1, u_2) = 1 \mid g_1,\ g_2,\ u_1, u_2\ \in \boldsymbol{R}]$$

is non-negligible.

The DDH problem is widely conjectured to be hard (i.e., it is widely believed that such an algorithm does not exist)

4.2 Hash Functions

Hash functions are functions that take arbitrary-length strings as input and produce much shorter strings as output. There are three key important properties of hash functions. The three key properties are one-way, strongly collision resistant, and universal one-way (weakly collision resistant). A description of each property is now given.

- One-Way-

  Given a hash function H and an element *y*, it is difficult to find an element *x* such that H(*x*) = *y*.

- Universal One-Way (Weakly Collision Resistant)-

  Given a hash function H and an element x, it is difficult to find another element x'(where $x \neq x$') such that H(x) = H(x').

- Strongly Collision Resistant-

  Given a hash function H, it is difficult to find two elements x_1 and x_2 (where $x_1 \neq x_2$) such that H(x_1) = H(x_2).

If a hash function does not have the property of one-way, then the hash function will not have the property of universal one-way. Also, if a hash function does not have the property of universal one-way, then the hash function does not have the property of strongly collision resistant.

In the Cramer-Shoup cryptosystem we will have a need to use a hash function. We assume that the hash function used is chosen from the family of universal-one-way hash functions.

4.3 Proof of Security

Theorem 1. *The Cramer-Shoup cryptosystem is secure against adaptively chosen ciphertext attack assuming that (1) the hash function H is chosen from a universal one-way family, and (2) the Diffie-Hellman decision problem is hard for the group G*

To prove this theorem we will assume a probabilistic polynomial time adversary *A* is able to break the Cramer-Shoup cryptosystem. Using the help of

such an adversary *A* as a subroutine, we will develop an algorithm that solves the Decisional Diffie-Hellman problem.

Recall that in the DDH problem, we are given a quadruple (g_1, g_2, u_1, u_2) from either a distribution **R** or **D** and we need to figure out whether the quadruple came from **R** or **D**.

In order to make use of the adversary *A*, we first simulate an instance of Cramer-Shoup cryptosystem (actually similar to it, but not the same) from the given quadruple (g_1, g_2, u_1, u_2). The details of the simulator are described below.

4.3.1 Simulation of Key Generation

We start with (g_1, g_2, u_1, u_2) given as input. The following key generation algorithm is run by the simulator using g_1 and g_2:

- Choose at random $x_1, x_2, y_1, y_2, z_1, z_2 \in \mathbf{Z}_q$
- Compute $c = {g_1}^{x_1}{g_2}^{x_2}, d = {g_1}^{y_1}{g_2}^{y_2}, h = {g_1}^{z_1}{g_2}^{z_2}$
- The simulator chooses a hash function H at random.

The public key that the adversary *A* sees now consists of

$$(g_1, g_2, c, d, h, H)$$

The private key which only the simulator can see now consists of

$$(x_1, x_2, y_1, y_2, z_1, z_2)$$

The key generation of the simulator and the actual cryptosystem differ slightly. The simulator has a random private key element z_2 while the original cryptosystem does not. The only element that has changed in the public key is

h. The real Cramer-Shoup cryptosystem defines *h* as $g_1^{\ z}$ while the simulated Cramer-Shoup cryptosystem defines *h* as $g_1^{\ z_1} g_2^{\ z_2}$.

The element z_2 does not affect the adversary *A*'s view of *h*. As z_1 and z_2 are random elements of $\mathbf{Z}_q$, $h = g_1^{\ z_1} g_2^{\ z_2}$ will be a random element of the group *G*. So from the adversary *A*'s view, *h* is a random element of the group *G* regardless of whether it came from the real Cramer-Shoup cryptosystem or the simulated cryptosystem.

4.3.2 Simulation of Encryption and Decryption

To encrypt message $m \in G$ we compute

$$e = u_1^{\ z_1} u_2^{\ z_2} m,\ \alpha = H(u_1, u_2, e),\ v = u_1^{\ x_1 + y_1 \alpha}\ u_2^{\ x_2 + y_2 \alpha}$$

and output

$$(u_1, u_2, e, v)$$

Given an alleged ciphertext (u_1, u_2, e, v), the decryption oracle will first check whether it is properly formed by checking $v = u_1^{\ x_1 + y_1 \alpha}\ u_2^{\ x_2 + y_2 \alpha}$ and if so we compute the message m as

$$m = e/(u_1^{\ z_1} u_2^{\ z_2})$$

4.3.3 Outline of Proof

The simulator provides the public key to the adversary. The adversary then provides messages m_0 and m_1. The simulator selects $b \in \{0, 1\}$, encrypts m_b and then gives it to the adversary.

We will show that, when the input of the simulator comes from **D**, the cryptosystem that we constructed looks like Cramer-Shoup cryptosystem from the view of the adversary and the output of the encryption oracle will be a perfectly legitimate ciphertext. Therefore, if we have an adversary who can break Cramer-Shoup cryptosystem, then the adversary will have non-negligible advantage in guessing the bit *b*. This is the content of Lemma 1.

On the other hand when the input comes from **R**, the output of the encryption oracle will not be formed legitimately as $\log_{g_1} u_1 \neq \log_{g_2} u_2$ now. Nonetheless, we will show that in this case the adversary's view is independent of the hidden bit *b*. Therefore, the adversary cannot do better than simply guessing bit *b*. This is the content of Lemma 2.

We will now answer that the given quadruple comes from **D** if the adversary is able to find *b* correctly and we will answer **R** if the adversary is not able to answer correctly. Our answer will be correct with a high probability assuming the adversary has the ability to break Cramer-Shoup cryptosystem.

Lemma 1. *When the simulator's input comes from* **D**, *the joint distribution of the adversary's view and the hidden bit b is statistically indistinguishable from that in the actual attack.*

Consider the joint distribution of the adversary's view and the bit *b* when the input is from distribution **D**. In that case, we know that $u_1 = g_1^r$ and $u_2 = g_2^r$. The output (ciphertext) of the simulated encryption oracle is

(u_1 , u_2, e, v)

It is clear that the simulator's u_1 and u_2 are identical to that of the real cryptosystem when the input comes from distribution **D**. We will now see how *e* and *v* are equivalent for the simulated cryptosystem and the real cryptosystem thus being indistinguishable to adversary *A*.

$e = h^r \times m_b$ (Real cryptosystem's definition of *e*)

$e = ({g_1}^{z_1} {g_2}^{z_2})^r \times m_b$

$e = ({g_1}^{z_1 r} {g_2}^{z_2 r}) \times m_b$

$e = ({g_1}^r)^{z_1} ({g_2}^r)^{z_2} \times m_b$

$e = {u_1}^{z_1} {u_2}^{z_2} \times m_b$ (Simulator's definition of *e*)

The simulator's definition of v satisfies the equation used in the real cryptosystem to verify if a ciphertext is valid before decrypting.

$v = c^r d^{r\alpha}$ (Real cryptosystem's definition of *v*)

$v = ({g_1}^{x_1} {g_2}^{x_2})^r \times ({g_1}^{y_1} {g_2}^{y_2})^{r\alpha}$

$v = ({g_1}^r)^{x_1} ({g_2}^r)^{x_2} \times (({g_1}^r)^{y_1} ({g_2}^r)^{y_2})^{\alpha}$

$v = {u_1}^{x_1} {u_2}^{x_2} ({u_1}^{y_1} {u_2}^{y_2})^{\alpha}$

$v = {u_1}^{x_1 + y_1 \alpha} \times {u_2}^{x_2 + y_2 \alpha}$ (Simulator's definition of *v*)

Completing the proof requires us to argue that the output of the decryption oracle has the right distribution. We call $({u'}_1, {u'}_2, e', v') \in G^4$ is a *valid ciphertext* if $\log_{g_1} {u'}_1 = \log_{g_2} {u'}_2$.

If the ciphertext is valid, where $u'_1 = {g_1}^{r'}$ and $u'_2 = {g_2}^{r'}$, then we can see that $h^{r'} = (u'_1)^{z_1}(u'_2)^{z_2}$. The decryption oracle will then correctly output $e/h^{r'}$ for m. Lemma 1 immediately follows from the following claim.

Claim. The decryption oracle – in both an actual attack against the cryptosystem and in an attack against the simulator – rejects all invalid ciphertexts, except with negligible probability.

We prove this claim by considering the distribution of the point $\mathbf{P} = (x_1, x_2, y_1, y_2) \in \boldsymbol{Z}_q^4$ conditioned on the adversary's view. For the rest of Chapter 4, let $\log(\cdot)$ denote $\log_{g_1}(\cdot)$ and $w = \log g_2$.

From the adversary's view, **P** is a random point on the plane ρ formed by intersecting the hyperplanes

$$\log c = x_1 + wx_2 \tag{1}$$

and

$$\log d = y_1 + wy_2 \tag{2}$$

Equations (1) and (2) both come from the public key as the following will show:

$c = {g_1}^{x_1}{g_2}^{x_2}$ (Public Key's definition of c)

$c = {g_1}^{x_1}{g_1}^{wx_2}$

$\log c = x_1 + wx_2$ (1)

and

$d = {g_1}^{y_1}{g_2}^{y_2}$ (Public Key's definition of d)

$$d = {g_1}^{y_1} {g_1}^{wy_2}$$

$$\log d = y_1 + wy_2 \quad (2)$$

P is a point that satisfies these two equations. **P** is not constrained any further by the output of the encryption oracle. The hyperplane defined by

$$\log v = rx_1 + wrx_2 + \alpha ry_1 + \alpha rwy_2 \quad (3)$$

actually contains ρ, the equation is a linear combination of the equations (1) and (2). Equation (3) comes from the simulator's definition of v (which we've also already shown to be equivalent to the real cryptosystem's definition of v).

$v = {u_1}^{x_1+y_1\alpha} \times {u_2}^{x_2+y_2\alpha}$ (Simulator's definition of v)

$\log v = \log({g_1}^{r(x_1+y_1\alpha)} \times {g_2}^{r(x_2+y_2\alpha)})$

$\log v = \log({g_1}^{r(x_1+y_1\alpha)}) + \log({g_2}^{r(x_2+y_2\alpha)})$

$\log v = r(x_1 + y_1\alpha) + \log({g_1}^{wr(x_2+y_2\alpha)})$

$\log v = r(x_1 + y_1\alpha) + wr(x_2 + y_2\alpha)$

$\log v = rx_1 + wrx_2 + \alpha ry_1 + \alpha rwy_2$

If the decryption oracle were to receive an invalid ciphertext (u'_1, u'_2, e', v') from adversary *A*, where $\log u'_1 = r'_1$ and $\log u'_2 = wr'_2$, with $r'_1 \neq r'_2$, then the oracle would reject unless **P** satisfies the following equation

$$\log v' = r'_1 x_1 + wr'_2 x_2 + \alpha r'_1 y_1 + \alpha r'_2 wy_2 \quad (4)$$

where $\alpha' = H(u'_1, u'_2, e)$. Since equations (1), (2), and (4) are linearly independent (there are 4 unknowns and only 3 equations), there are exactly q solutions for (x_1, x_2, y_1, y_2).

Initially it looks as if point **P** consisting of (x_1, x_2, y_1, y_2) would have q^4 possibilities; however this is not the case. Equations (1) and (2) (available from the public key) constrain point **P** since fixing x_1 completely determines x_2 and similarly fixing y_1 completely determines y_2. Because of these constraints, there are now q^2 possibilities. Of these q^2 possibilities, only q will be satisfying the equation. Therefore, the probability of selecting a point **P** that satisfies the equation is q/q^2 or $1/q$. Remember that we are working with groups of very large size so this probability is very small.

If the adversary passes an invalid ciphertext, there will be some information that will be revealed that increases their probability of guessing a point **P** that satisfies the equation. As stated, there are 3 equations and 4 variables, which means the system can be solved uniquely in terms of a single variable. Because of this, each time the adversary passes a ciphertext that is rejected, they can eliminate q different possibilities from **P**. The probability of the adversary choosing a point **P** that satisfies the equation on their first try is q/q^2, the probability of choosing correctly on the second try would be $q/(q^2 - q)$. This can be simplified to the following equation, let i be the number of incorrect guesses made by the adversary, the probability of the adversary correctly guessing a point **P** on their ith attempt is $1/[q - (i - 1)]$.

Lemma 2. *When the simulator's input comes from* **R**, *the distribution of the hidden bit b is (essentially) independent from the adversary's view.*

We start by letting $u_1 = {g_1}^{r_1}$ and $u_2 = {g_1}^{wr_2}$. We can assume that $r_1 \neq r_2$, since $r_1 = r_2$ only occurs with negligible probability. The lemma follows immediately from the following two claims.

Claim 1. If the decryption oracle rejects all invalid ciphertexts during the attack, then the distribution of the hidden bit b is independent of the adversary's view.

Consider the point $\boldsymbol{Q} = (z_1, z_2) \in \boldsymbol{Z}_q^2$. At the beginning of the attack, this is a random point on the line

$$\log h = z_1 + wz_2 \tag{5}$$

determined by the public key value of *h* as shown below.

$h = {g_1}^{z_1} {g_2}^{z_2}$ (Simulator's definition of h)

$\log h = z_1 \log g_1 + z_2 \log g_2$

$\log h = z_1 + wz_2$ (5)

Moreover, if the decryption oracle only decrypts valid ciphertexts $({u'}_1, {u'}_2, e', v')$, then the adversary only obtains a multiple of equation (5). In other words, linearly dependent relations $'\log h = r'z_1 + r'wz_2$. Thus, the adversary gains no new knowledge about $\boldsymbol{Q}$.

Consider the output of the simulator encryption output (u_1, u_2, e, v). Let $\epsilon = {u_1}^{z_1}{u_2}^{z_2}$ so that $e = \epsilon \times m_b$. Now consider this new equation

$$\log \epsilon = r_1 z_1 + wr_2 z_2 \tag{6}$$

Equations (5) and (6) are linearly independent. For a fixed ϵ, there is exactly one solution for z_1 and z_2 that satisfies equations (5) and (6). Hence each pair

of (z_1, z_2) gives rise to a new value for ϵ solving equation (6). In other words, ϵ is essentially a random element of the group **G**. It follows that b is independent of the adversary's view. Since ϵ is a random element of the group **G**, m_b is completely lost when multiplied by ϵ. Therefore, our encryption scheme acts like a perfect one-time pad.

Claim 2. The decryption oracle will reject all invalid ciphertexts, except with negligible probability.

Claim 2 removes the assumptions made in Claim 1. To prove this claim, we look at the distribution of **P** = $(x_1, x_2, y_1, y_2) \in \boldsymbol{Z}_q^4$ conditioned on the adversary's view. From the adversary's view, **P** is a random point on the line **L** formed by intersecting hyperplanes (1), (2), and

$$\log v = r_1 x_1 + w r_2 x_2 + \alpha r_1 y_1 + \alpha r_2 w y_2 \qquad (7)$$

with (7) coming from the output of the encryption oracle.

We now assume that the adversary submits an invalid ciphertext $(u'_1, u'_2, e', v') \neq (u_1, u_2, e, v)$, where $\log u'_1 = r'_1$ and $\log u'_2 = w r'_2$, with $r'_1 \neq r'_2$. Also let $\alpha' = H(u'_1, u'_2, e')$. There are three cases to be considered.

Case 1. $(u'_1, u'_2, e') = (u_1, u_2, e)$.

This case implies that the hash values are the same where $H(u'_1, u'_2, e') = H(u_1, u_2, e)$ and $\alpha' = \alpha$. Because the first three elements of the two quadruples are equal, that must mean the last elements v and v' are not equal. However, $v' \neq v$ implies that the decryption oracle will certainly reject.

Case 2. $(u'_1, u'_2, e') \neq (u_1, u_2, e)$ and $\alpha' \neq \alpha$.

The decryption oracle rejects unless point **P** lies on the hyperplane $\boldsymbol{H}$ defined by equation (4). However, the equations (1), (2), (7), and (4) are linearly independent. As seen below, the determinant of the matrix formed by the coefficients of x_1, x_2, y_1 and y_2 in the four equations does not equal 0.

$$det\begin{pmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r_1 & wr_2 & \alpha r_1 & \alpha w r_2 \\ r'_1 & wr'_2 & \alpha' r'_1 & \alpha' w r'_2 \end{pmatrix} = w^2(r_2 - r_1)(r'_2 - r'_1)(\alpha - \alpha') \neq 0.$$

We can see that hyperplane $\boldsymbol{H}$ intersects the line **L** at a single point. Thus, the decryption oracle rejects, except with negligible probability of $1/q$. The probability of $1/q$ derives very similarly from the explanation given in Lemma 1. Fixing x_1 completely determines x_2 and similarly fixing y_1 completely determines y_2. Because of these constraints, there are now q^2 possibilities instead of q^4. Of these q^2 possibilities, only q will be satisfying the equation. Therefore, the probability of selecting a point **P** that satisfies the equation is q/q^2 or $1/q$.

Case 3. $(u'_1, u'_2, e') \neq (u_1, u_2, e)$ and $\alpha' = \alpha$.

In this case $\alpha' = (u'_1, u'_2, e')$ and $\alpha = (u_1, u_2, e)$, we have succeeded in finding another element that hashes to the same value which contradicts the assumption that *H* is coming from a family of universal one-way functions.

Chapter 5 Implementation

This chapter provides the details of my implementation of the Cramer-Shoup Cryptosystem.

5.1 Summary and Environment

The implementation of Cramer-Shoup's cryptosystem was done in C++ using Visual Studios 2010 IDE. The project is made of up three separate programs: the key generation, encryption, and decryption. All three programs used Victor Shoup's Number Theory Library (NTL). The Number Theory Library provides an efficient implementation of large arithmetic. Note that the numbers that we deal with are larger than what can be held in typical integer variables or large integer variables of ordinary programming language. The data type for large integer is denoted by ZZ in NTL.

5.2 Underlying Group

The Cramer-Shoup cryptosystem assumes that you encrypt elements of a cyclic group of prime order, but leaves the group unspecified. In my project we are using the group of quadratic residues in the group $\mathbf{Z}^*_p$ where p is a strong prime, i.e, $q = (p-1)/2$ is also a prime. An element $y \in \mathbf{Z}^*_p$ is a *quadratic residue modulo p* if there exists an $x \in \mathbf{Z}^*_p$ such that $x^2 = y \bmod p$. [1] Exactly $(p-1)/2 = q$ elements in $\mathbf{Z}^*_p$ are quadratic residues. This is easy to see as every quadratic residue in $\mathbf{Z}^*_p$ has exactly two square roots (which are shown

below). Below is an instance of all the quadratic residues in $\mathbf{Z}^*_p$ where p equals 11(a strong prime).

$1^2 = 1$ $6^2 = 3$

$2^2 = 4$ $7^2 = 5$

$3^2 = 9$ $8^2 = 9$

$4^2 = 5$ $9^2 = 4$

$5^2 = 3$ $10^2 = 1$

The quadratic residues of $\mathbf{Z}^*_p$ forms a subgroup which consists of = {1, 3, 4, 5, 9}. It's easy to see that these elements form a legitimate subgroup as multiplying any two elements will result in another element of the subgroup. Every element of the subgroup (except the identity element) is actually a generator for the subgroup, as the subgroup is of prime order.

5.3 Key Generation Program

The Key Generation program generates a private key and a public key to be used by the Encryption and Decryption programs. For the program to run, it needs the following three inputs: name of the file in which the public key is to be written, name of the file in which the private key is to be written, and the security parameter *n*. The user decides the size of *n* by specifying the number of bits. Of course, the larger *n* is, the more secure the cryptosystem will be.

As stated, the public and private keys are written into two separate files. Each element of the key is of type "ZZ". Every element is written to an

individual line of the output files. The next two sections describe each element that is written to the Public Key and Private Key file.

5.3.1 Public Key File

- p – a strong prime. p is a strong prime only if (p - 1) / 2 also equals a prime number. The size of p is dependent on the size of the security parameter n given by the user.
- q – a germain prime. That is q = (p - 1) / 2 where p is the strong prime given above.
- g_1– is the result of the computation ${\alpha_1}^2 \ mod \ p$. Where α_1 is a random value from 2 to p – 2. Because we are squaring α_1, we are guaranteeing that we get a quadratic residue because we are excluding 1 and p-1.
- g_2– is the result of the computation ${\alpha_2}^2 \ mod \ p$. Where α_2 is a random value from 2 to p – 2.
- c – is the result of the computation ${g_1}^{x_1} \ {g_2}^{x_2} \ \text{mod} \ p$ where x_1 and x_2 are random values from 1 to q.
- d – is the result of the computation ${g_1}^{y_1} {g_2}^{y_2} \ mod \ p$ where y_1 and y_2 are random values from 1 to q.
- h – is the result of the computation ${g_1}^z \ \text{mod} \ p$ where z is a random value from 1 to q.

The following four components are used for the hash function and are written as four separate elements of the public key file.

- h_1, h_2, h_3, h_4 – to calculate each h, choose a new random number r from 1 to p-1,

  $h = r^2 \bmod p$

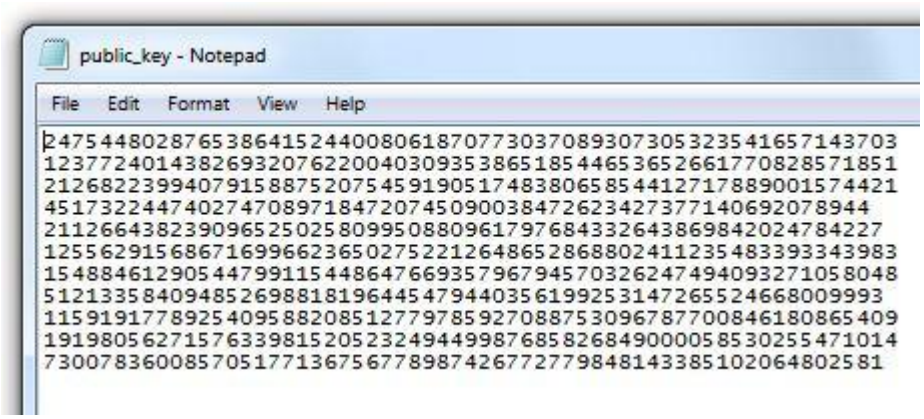

Figure 5.1 public key output file snapshot

5.3.2 Private Key File

The private key output file contains all the lines in the public key output file plus five extra lines (or elements). These five elements are x_1, x_2, y_1, y_2, z. These elements are used to produce some of the elements in the public key file as stated above. Each of these elements is a random integer from 1 to q.

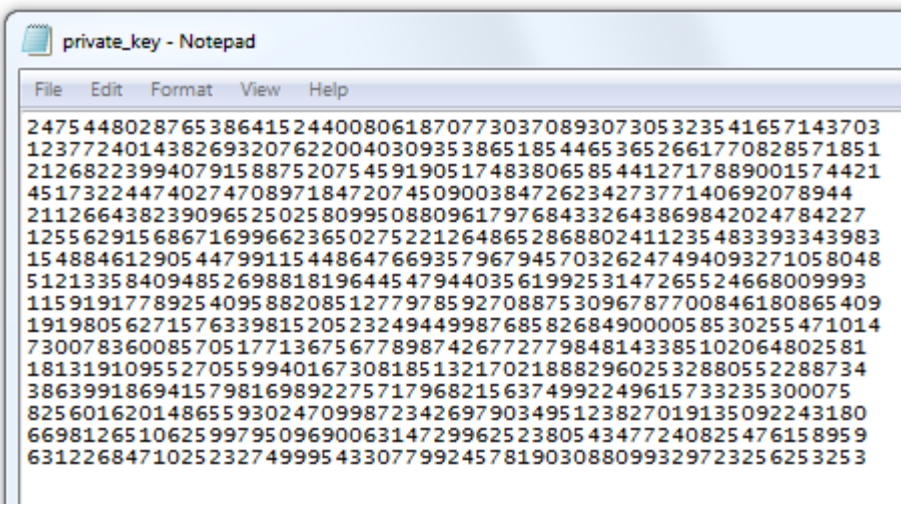

Figure 5.2 private key output file snapshot

5.4 Encryption Program

Once the private key and public key are generated from the key generation program, a file is ready to be encoded and encrypted. The encryption program needs three inputs to run: the public key, name of the file to be encrypted, and name of where the encrypted file is to be written.

The encryption program encodes a single element at a time and then encrypts it. The alternative would be to encode the entire file into elements and then to encrypt each element.

5.4.1 Encoding Scheme Implementation

For a file to be encrypted, the file must first be encoded into elements of group **G** in the subgroup of order q in $\mathbf{Z}^*_p$. To encode the file into elements, the program does the following steps:

- Let l be the number of bits needed to represent q in binary.

- Read l - 1 bits from the file.
- Convert the sequence of bits to its decimal form, call this integer b.
- Add 1 to b.
- Compute b^2 modulo p to get an element of the group **G**.

5.4.2 Hash Function Description

The Cramer-Shoup cryptosystem requires a universal-one way hash function H to compute $\alpha = H(u_1, u_2, e)$ in both the encryption and decryption algorithms.

Our hash function begins by converting the group elements u_1 , u_2 , e of G to random numbers a_1, a_2, a_3, a_4 ranging from 1 to q-1. This is explained in detail in the next section.

Recall that there were four elements h_1, h_2, h_3, h_4 of group **G** in the public key file that will be used for hashing purposes. The hash function is computed with the equation $\alpha = \sqrt{h_1^{a_1} h_2^{a_2} h_3^{a_3} h_4^{a_4}} \bmod p$ where α is the smallest non-negative square root. It follows that for H to be a universal-one way hash function, it must be difficult to find an input (u'_1, u'_2, e') that maps to a'_1, a'_2, a'_3, a'_4 where

$$\alpha = \sqrt{h_1^{a_1} h_2^{a_2} h_3^{a_3} h_4^{a_4}} \bmod p \; = \sqrt{h_1^{a'_1} h_2^{a'_2} h_3^{a'_3} h_4^{a'_4}} \bmod p$$

and

$(u'_1, u'_2, e') \neq (u_1, u_2, e)$.

To find such a (u'_1, u'_2, e') would require solving the discrete logarithm problem which is widely conjectured to be hard (i.e., it is widely

believed that an algorithm that solves discrete logarithm problem in probabilistic polynomial time does not exist).

The discrete logarithm problem is now described. For a cyclic group **G** of order q, given values for elements $g, y \in \mathbf{G}$, find x such that $g^x = y \bmod q$. It is very easy to see that finding values for a'_1, a'_2, a'_3, a'_4 to equal α would require solving the discrete logarithm problem. Thus, H has the property of being a universal-one way hash function.

5.4.3 Hash Function Implementation

In this section we give the implementation details of the hash function H that we will be using in the encryption and decryption programs. In particular, we will describe exactly how the programs compute α from u_1, u_2 and e (**Note:** p, h_1, h_2, h_3, h_4 are imported from the public key file to calculate α while $u_1, u_2,$ and e comes from the encryption):

- Create variables a_1, a_2, a_3, a_4
- Let i be the number of bits needed to represent p in binary.
- Check the binary lengths of $u_1, u_2,$ and e to see if they are equal to i.
- If they are not equal, append leading 0's to $u_1, u_2,$ and e until their binary lengths are equal to i.
- Remove the last bit of u_1 and append this bit to the front of u_2.
- Set a_1 equal to u_1 (u_1 has its last bit removed).
- Remove the last two bits of u_2 and append the bits to the front of e.
- Set a_2 equal to u_2 (u_2 has a bit from u_1 appended to the front and its last two bits removed).

- Remove the last three bits from e and add them to the front of a_4.
- Set a_3 equal to e (e has two bits from u_2 appended to the front and its last three bits removed).
- Append trailing 0's to a_4 until its binary length is equal to i.
- $\alpha = \sqrt{h_1^{a_1} h_2^{a_2} h_3^{a_3} h_4^{a_4}} \bmod p$

5.4.4 Encryption

The program does the following steps to encrypt an encoded element E (**Note:** p, q, g_1, g_2, c, d, h are imported from the public key file):

- Create variables $u_1, u_2, e, r, \alpha, v$.
- r = a random number between 0 and q.
- $u_1 = g_1^r \bmod p$
- $u_2 = g_2^r \bmod p$
- $e = h^r \bmod p$
- $e = e \times E \bmod p$
- Calculate α using u_1, u_2, e. (See **5.4.3 Hash Function Implementation**)
- $v = c^r \times d^{r\alpha} \bmod p$
- The encryption of element E is complete and the ciphertext now consists of u_1, u_2, e, v.
- Each ciphertext element is written to the ciphertext file on four lines. The four components u_1, u_2, e, v are written on a separate line.

5.5 Decryption Program

After the encryption program has been run, the next and final step is to run the decryption program. This program first decrypts the entire encrypted file and then decodes it. The decryption program needs three inputs to run: the name of the file to decrypt, private key, and the name of the file which to write the decrypted version.

5.5.1 Decryption

An encrypted element in the ciphertext consists of four components u_1, u_2, e, v each stored on a separate line of the ciphertext file. The program does the following steps to decrypt a single encrypted element (**Note:** x_1, x_2, y_1, y_2, z are imported from the private key):

- Read in u_1, u_2, e, v from the ciphertext file.
- Calculate α using u_1, u_2, e. (See **5.4.3 Hash Function Implementation**)
- $v' = u_1^{x1 + y1\alpha} \times u_2^{x2 + y2\alpha} \bmod p$
- Check the integrity of the ciphertext by testing if $v' = v$.
- If the condition does not hold true, then output that the ciphertext is not well formed and exit the program. Otherwise, continue with decryption.
- Perform e / u_1^z to get the original decrypted(but encoded) element E
- Write element E to the specified destination file.

5.5.2 Decoding Scheme

Once the file has been decrypted, the last step is to decode the elements back into its binary form. The decrypted (but encoded) file has a single element per line. The following steps are done to decode a single element E:

- Perform $\sqrt{E} \bmod p$ (where p is the strong prime from the public key file), call this value E'.
- Subtract 1 from E'.
- Convert E' from a decimal number to a binary number.
- Let l be the number of bits needed to represent q in binary.
- If the length of E' is less than $l - 1$, add leading 0's to E' until it is of length $l - 1$.
- E' is now ready to be written to the decrypted/decoded file.

Chapter 6 Conclusion

The demand for more secure cryptosystems continues to grow. However, the cryptosystem must still be efficient and practical. In my project, I implemented the Cramer-Shoup cryptosystem, which is currently one of the most secure cryptosystems while still having the benefit of being a practical scheme. This cryptosystem is provably secure against adaptively chosen ciphertext attacks (CCA2 secure). Even the popular RSA and El Gamal cryptosystems are not secure against adaptively chosen ciphertext attacks.

My project consisted of implementing all three parts of the cryptosystem: key generation, encryption, and decryption. Each part was separated into their own program. Victor Shoup's Number Theory Library (NTL) was used throughout the programs for efficient large integer arithmetic.

Apart from merely implementing the Cramer-Shoup cryptosystem, we studied in detail its proof of security which is quite sophisticated.

Although the Cramer-Shoup cryptosystem is CCA2 secure and is also efficient to implement, it has the disadvantage of the ciphertext becoming 4 times the size of the plain text. We leave this as an open problem to development a cryptosystem that is CCA2 secure but at the same time providing less expansion as the Cramer-Shoup cryptosystem, if possible.

References

[1] Jonathan Katz, Yehuda Lindell, “Introduction to Modern Cryptography,” Chapman & Hall/CRC, 2008

[2] Douglas Stinson, “Cryptography Theory and Practice,” Chapman & Hall/CRC, 2006

[3] Ronald Cramer, Victor Shoup, “A Practical Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack,” 1998