Rochester Institute of Technology
Department of Computer Science
Master's Project Proposal

A Study of Probabilistic Cryptography

Kert Richardson
kert_richardson@frontiernet.net

Committee:

Chairman: Stanisław P. Radziszowski,
Rochester Institute of Technology

Reader: Chris Homan,
Rochester Institute of Technology

Observer:

Contents

1 Summary **3**

2 Overview **4**

3 Contents of Project **6**
3.1 Overview of the Theory of Probabilistic Cryptography 6
3.2 Study of Experiments of Paillier Scheme 6

4 Design of Paillier Experiments **8**
4.1 Overview of Design 8
4.2 Detail of Design 8

5 Deliverables **11**

6 Proposed Timeline **12**

7 Draft Table of Contents **13**

1 Summary

The first part of the project will consist of an overview of what has been discovered in the area of probabilistic cryptography and will conclude with my impression of what potential the area holds. The first goal of the project is to do a study of what different types of probabilistic cryptography are out there. For each type I will see how much research has been put into them and how secure they seem to be. The paper will also need to check how efficient the encryption and decryption process is. All of this should show if the different encryption schemes have potential to be used in applications and whether probabilistic encryption as a whole has more potential to be studied. I have decided to study 3 different schemes: Goldwasser-Micali, Blum-Goldwasser, and Paillier's scheme. There are others out there but these seem to be the main ones that are mentioned. The others are based on altering these schemes.

In the second part of my project I will create my own programs that will use one or more of the probabilistic encryption schemes. The plan is to test the timing of several versions of Paillier's scheme versus RSA. It will be interesting to compare the schemes in speed and run some practical tests on them. By creating my own program of the schemes it should allow plenty of opportunities to test every aspect of it. If time permits, other probabilistic schemes could be implemented for more testing.

2 Overview

Goldwasser and Micali were the first people to come up with a workable scheme for probabilistic cryptography. The main benefit of probabilistic cryptography is that you can encrypt the exact same plaintext into different ciphertexts but get the same plaintext back when decrypted. The Goldwasser-Micali probabilistic scheme [2] security is based on the quadratic residuosity problem and is considered very slow. The Blum-Goldwasser scheme [2] is much faster scheme, being comparable to an RSA scheme in speed. Its security is based on integer factorization but has been found to be susceptible to chosen-ciphertext attacks. It will be interesting to study how big of a threat these people think these attacks pose.

There is another scheme called Paillier's probabilistic scheme that has been developed more recently. It is based on the composite residuosity class problem. This scheme seems to have a lot of potential as it seems secure and is computationally comparable to RSA [3]. There is a lot of different research on this type of scheme and there seems to be some practical applications for using it in electronic voting. This will be an interesting topic to look into with all the recent attention given to electronic voting.

The Paillier scheme actually has several variations and other people have changed it to create other versions. The following is the first Paillier scheme described in his paper [4]. It gives you the general idea of how encryption based on composite residuosity works. First you set n = pq with p and q being large primes as you would in RSA. Then you select a random base *g*. This can be found by checking if g works in $gcd(L(g^{\lambda}) \ mod \ n^2),n) = 1$. The notation λ is equivalent to $\lambda(n)$ and is called the Carmichael's function. This can be calculated as $\lambda(n) = lcm(p-1, q-1)$. We also define $L(U) = \frac{U-1}{n}$. The encryption is done by the following:

n = modulus
m = the plaintext where $m < n$
r = a random number where $r < n$
c = ciphertext which is generated as follows $c = g^m * r^n \bmod n^2$.

The process of decryption uses the same mathematics described above and can be done as follows:

c = ciphertext where $c < n^2$
m = plaintext and can be found as follows $m = \frac{L(c^{\lambda} mod n^2)}{L(g^{\lambda} mod n^2)} \bmod n$.

One of the main properties of Paillier's scheme is that it is homomorphic. Homomorphic encryption is just like other semantically secure public-key encryption, except it has has the following property:

$E(A) * E(B) = E(A * B)$

where * can be addition or multiplication or any other binary operator (but not more than one)[8]. This means that after plaintext A and B have been encrypted, we can still perform operations on them. A 3rd party can add or multiply the encrypted values together without knowing the plaintext. The value can later be decrypted by another person to find the result. The 3rd party never needs to know the encrypted values or result to do the operation. This property has a lot of promise and has been applied in some e-voting schemes as well as other applications. The Paillier scheme is additively homomorphic.

3 Contents of Project

The project will consist of two different parts. First, there will be an overview of probabilistic cryptography, explaining what has been done to study it and what research is currently out there. Secondly, some different types of probabilistic cryptography will be used to create my own software that will be used to study the schemes in practical circumstances. The project will be split so half the time is spent on both on these parts.

3.1 Overview of the Theory of Probabilistic Cryptography

The goal in the first part will be to explain as completely as possible what probabilistic cryptography is. The explanation will include what the key differences are between this and other deterministic schemes. One key concept to understand is that probabilistic encryption is semantically secure. This implies that no information about the plaintext can be learned from the ciphertext. This can be of great importance in making sure a scheme is not insecure when sending the same message multiple times. Most schemes, like RSA, will send the same thing over and over. This can be a clue to someone spying on the encrypted message to tell what is being sent.

It will look into all of the underlying theory that allows these schemes to work and shows how robust the theory is. All of the probabilistic schemes seem to have some similarities in encrypting and decrypting. The paper will explain what these are and what the differences are. It will include an explanation on how efficient the algorithms are and about how many operations are necessary for a typical encryption and decryption. It will be related to the speed of more popular private key schemes in use today.

The paper will look into the practical application of the Paillier scheme. Currently it seems to be the only scheme that is being studied for this purpose. The big advantage that Paillier scheme has is that it is homomorphic. This means that you can perform certain mathematical operations on the ciphertext and they will be preserved when it is decrypted. According to Paillier there are several practical areas that area being studied beause of this property, including: voting protocols, threshold cryptosystems, watermarking, and secret sharing schemes [4]. It will be good to look into these and any others uses that can be found. I would also like to see if there are any other applications that would use Paillier's properities well. The applications could be tested in the implementation if something is found that looks interesting.

3.2 Study of Experiments of Paillier Scheme

The second part of the project will consist of a program that will use several probabilistic encryption schemes. The program will use the basic Paillier scheme described in his paper. The program will also implement another version of the Paillier scheme that was developed by Dåmgard and Jurik. This scheme is like Paillier but is done using modulo n^{s+1} for any $s >= 1$ instead of n^2 in Paillier. We will call this scheme Dåmgard-Jurik [3].

Once I have these schemes implemented it will be interested to vary the numbers used

in the schemes to see how they perform for both securty and speed. One thing that I will test is the "base" g from the Paillier scheme. In both the Paillier and Dåmgard-Jurik papers they usually use the number $n + 1$ as the base g. It will be interesting to see if this works best in practicality. There will also be other numbers that can be altered and I may even try my own version of the schemes. The goal will be to find the best values that perform securly.

These seem to be the schemes that are being studied the most currently. There aren't any recent papers on the other schemes, Goldwasser-Micali and Blum-Goldwasser, since they are all from the 1980's. All the papers on Paillier's scheme and other versions of it are within about six years, although some of the ideas seem to have been around longer. This seems to suggest that this scheme has much more promise and can be used in practical applications. The application of using Paillier for electronic voting is another bonus. I haven't found any applications that have used the other schemes or any mention that it might happen.

The plan is to implement an RSA scheme as well to use for comparison. Since RSA is a well established public-key cryptosystem it will be a good standard to compare the less established Paillier. The main thing that will be tested is the speed of the two schemes to see how they compare. It will be interesting to compare the encryption and decryption speeds and see if Paillier can perform at a similar level.

4 Design of Paillier Experiments

The goal of this program is to create software to test the Paillier schemes and see how well they perform. While all the theory and proofs are good to study, we must realize that the implementation must perform or the scheme will never be used. Using this software will give some practical proof of how well they can compete with what is currently being used. The design will be made so I can alter as many variables in the schemes as possible to see which ones outperform the others. The following is the current design of the software I will use to reach these goals.

4.1 Overview of Design

- Create a peer to peer type of cryptographic application.
- Front end application will be simple implemenation of cryptography classes.
- Use RMI to comunicate messages between peers.
- Can send messages as plaintext or encrypted by any scheme implemented.
- Use the Java language along with Java Cryptographic Extension (JCE).
  - Cipher Class
  - CipherSpi Class
- Create Paillier scheme as provider to implement in application.
- Create Dåmgard-Jurik (another version of Paillier) scheme as provider to implement in application
- Use a current RSA scheme already provided with RSA in application.
- Create time tracking methods to compare timings.
- Create simple message cracking functions to see if I can crack simple implemenations.
- Test all schemes implemented for timing and security.

4.2 Detail of Design

The plan is to create a peer to peer type of application that allows a user to set up a private-key encryption scheme and then pass an encrypted message from from one instance to another. The instances of the program can be set up on different machines and could possibly involve more than 2 instances at once. Either peer can send an encrypted message to the other peer. The program will be able to use either of the Paillier schemes or RSA to send the message and the user will not be able to see any real difference. The user interface and message sending will be very simple and most of the work will be done in

the specific methods in the back end. These cryptography methods will be created in a cryptographic service provider, which can be simply called a provider [7]. A provider is the term used in the Java Cryptography Extension (JCE) to describe someone that provides implementations for cryptographic functions.

I am planning on using the Java language to write the program. Java seems to be an obvious choice since most people are using it and the support it gives to specific applications with its API's. This is especially helpful because of the Java Cryptography Extension (JCE) that is provided in the JDK. There are also specific providers that supports the RSA scheme. The classes that are given by the provider will be used to encrypt and decrypt with RSA. These should allow a realistic look at how a well established encryption scheme performs. If this is successful then it will allow more time to be spent on probabilistic schemes. If time permits I would also like to try implementing the Blum-Goldwasser scheme in my application as well.

The concept of a provider is essential in the use of the JCE. The providers have the job of implementing the complex cryptographic algorithms and making them available to other Java developers. The normal mainstream developer looking for cryptographic security will have very little understanding of the how the provider implements the algorithms. They should have a general knowledge of cryptography and understand how to bring a provider into their development and use the implementations. There are many providers around, available as open-source or for purchase. Sun includes a provider in the JCE that implements many basic cryptography functions. Many providers can be installed in Java that contain overlapping algorithms. The JCE has a group of rules based on the setup of the providers that chooses the algorithm. My provider will work in the same way to allow the implementation to be plugged into other applications.

The JCE is actually an extension of the Java Cryptography Architecture (JCA). They both work together to give you a complete set of cryptographic functions. The only reason they were split into separate libraries is because of United States export laws, which prevented the export of certain types of cryptography. These laws have been changed and the split is no longer necessary, but still exist in the current Java version. Each of the different categories of cryptography components are put into different classes, which are referred to as engines. Any of these engines can be invoked in the code to create an instance of the engine. The engines all use the factory pattern to return the instance. This pattern was found to be the most beneficial for allowing a developer independence to choose to the best algorithm from a provider and then later changing it if he finds a better one. The Service Provider Interface (SPI), which all providers must use, specifies that the engines be created using this pattern. To access a certain cryptographic function the developer just has to use the getInstatnce("algorithm") method on the engine where the algorithm is classified. The "algorithm" parameter is the specific name of the algorithm the developer is looking for. The JCE will look for this algorithm in the providers that have been installed and return an instance (if one exists) according to the rules in the JCE. The developer can also specify a certain provider to ensure they receive the specific algorithm they want [6].

The two main classes that will be needed in the JCE are Cipher and CipherSpi

/citeWeiss. The CipherSpi class is an abstract class that allows you to create your own cipher. Implementing this class when you create your own cipher will force you to use all the same procedures that have been predefined. This allows the user to know how to use your implementation of the cipher. The Cipher class is actually an implementation of CipherSpi that gives you access to all the cryptography methods by different providers. Currently Cipher supports implementations of DES, AES, RSA, Blowfish /citeKnud along with some others that come with the JCE. You can get more schemes from other third party vendors. For my purposes I am going to create the methods for my Paillier schemes as a provider and load it into the JCE. This will allow me to access my schemes through the Cipher class and use it for encryption.

A main part of the project will be understanding the JCE and how to use it effectively to implement an encryption scheme. I want to understand how to make a front end application that uses encryption from a provider. This will allow the application to change the back end cryptography by just changing to other classes that implement the same interface. I want to be able to create a provider with the cryptography classes that implement the interface correctly. This will allow me to create cryptography schemes that can effectively be used in an application already set up to use the interface.

The design of the system will involve RMI. The plan is to allow communication between two separate instances of the program by using RMI to send the information across the network. RMI will allow 2 instances of the program to be set up on different machines and perform simple communication.

The actual capabilities of the front-end of the system will be simple but will accomplish the testing needed. The RMI will allow a message object to be sent across the network to another instance of the program. The message can be sent as not encrypted or encrypted by any implemented scheme. The message object will tell what encryption was used if any. The message will be decrypted when it is received by the accepting instance. The message will be able to be viewed either encrypted or decrypted by either the sender or receiver. There will also be time tracking functionality that will allow me to do some of my testing.

It would be good to create some functions that allow me to try to crack the message code. These functions would be based on very simplistic, brute force methods of finding the key. They would mainly be testing the encryption with an artificially small key. A normal size key would nearly impossible to break with my limited computing resources. The message cracking would be done to compare the robustness different schemes implemented.

5 Deliverables

Final Report

1. Summary of Previous Research
2. Mathematical Background
3. Explanation of Algorithms
4. Results of Experiments
   (a) Design of experimental system
   (b) Use of system
   (c) In depth explanation of all tests
5. Summary of All Work
6. Promising Areas for Reasearch

Java Code

1. Provider Classes
2. Front-end Application

Java Executable

6 Proposed Timeline

Task	**Target Date**	**Completed Date**
Proposal Completed	5/28/05	
Proposal Approved	6/1/05	
Background Report Completed	6/15/05	
Design and User Tests Completed	6/29/05	
Probabilistic Program Completed	7/05/05	
Testing of Program Completed	7/12/05	
Final Write Up and Summary Complete	7/19/05	
Project Defense	?	

7 Draft Table of Contents

1. **Introduction - Abstract**
2. **Overview of Probabilistic Cryptography**
3. **Goldwasser-Micali**
    - History
    - Mathematical Background
    - Summary of Scheme
4. **Blum-Goldwasser**
    - History
    - Mathematical Background
    - Summary of Scheme
5. **Paillier's Scheme**
    - History
    - Mathematical Background
    - Summary of Scheme
6. **Design of System**
    - Architecture of System
    - How to Use System
7. **Experiments**
    - Explanation of Experiments
    - Results of Experiments
8. **Summary**
    - Summary of My Work
    - Areas for Future Research

References

[1] Mao Wenbo, On Probabilistic Encryption and Semantic Security, *Modern Cryptography, Theory & Practice*, (2004) 472-497.

[2] A. Menezes, P. van Oorschot, and S.Vanstone, 8.7 Probabilistic public-key encryption, *Handbook of Applied Cryptography*, (1996) 304-319.

[3] Ivan B. Dåmgard and Mads J. Jurik, Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes, *BRICS Report Series*, March 2000, `http://www.brics.dk/RS/00/5/BRICS-RS-00-5.pdf`.

[4] Pascal Paillier, Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, *Advances in Cryptography - EUROCRYPT '99*, Springer Verlag LNCS series (1999) 223-238.

[5] I. Dåmgard and M.J. Jurik, A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System, *PKC 2001, LNCS series*, volume 1992 (2001) 119-136.

[6] Jason Weiss, Java Cryptograghy Extensions, Practical Guide for Programmers *Morgan Kaufmann Publishers*, (2004).

[7] Jonathan Knudsen, Java Cryptography, *O'Reilly Associates, Java Series* (1998).

[8] Prof. Rafail Ostrosky, Lecture 8 of Foundations of Cryptography, http://www.cs.ucla.edu/˜ rafail/TEACHING/WINTER-2005/L8/L8.pdf, (2005).