# RSA and El-Gamal Cryptosystems

Yanan Xiao, *Student Member, IEEE* Maryam Al Mehrezi

**Abstract**—We present our analysis of RSA and ElGamal cryptosystem with great detail. We show that there are some attacks on RSA. The mathematical foundation of ElGamal cryptosystem, namely discrete logarithm problem is discussed. Basic structure of our implementation codes is also mentioned.

**Keywords**—RSA, El-Gamal, implementation, public key, cryptosystem

## 1 Introduction

CRYPTOGRAPHY is a fascinating subject. The spread of computers and communication systems require private sector to protect the information in digital form, and to provide security services. Further, Cryptography has a great importance in most

technology today, not only in computers but also in credit cards, government and it remains the standard means for securing electronic commerce for many financial institutions around the world.

Cryptography is the science of encoding data, typically using a key, so that people without the key cannot read the data. Many different algorithms are used to encrypt data and they are either symmetric or asymmetric. In symmetric key cryptography, the best known algorithm is the Data Encryption Standard (DES). DES, which was developed at IBM in 1977, was a popular standers algorithm for years, until Triple DES and AES began to replace it. Whereas, symmetric key algorithms require a shared secret to be exchanged between the communicating parties to have a secured communication see Figure 1. The security of the symmetric key algorithm depends on the secrecy of the key. So, the problem with symmetric keys is how to securely get the secret keys to each end of the exchange and keep them secure after that. For this reason, an

• *Yanan Xiao and Maryam Al Mehrezi are with the Department of Electrical Engineering and Computer Science, Masdar Institute of Science and Technology, Masdar City, Abu Dhabi, UAE, 54224.*
*E-mail: {yxiao,malmehrezi}@masdar.ac.ae*

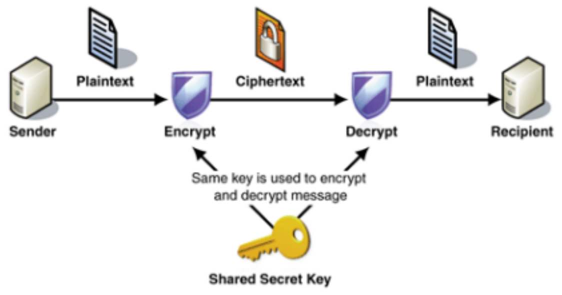

Fig. 1. The process of symmetric encryption

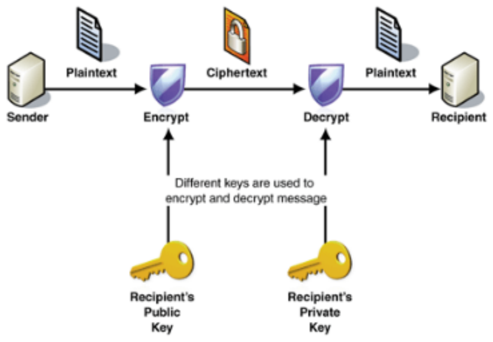

Fig. 2. The process of asymmetric encryption

asymmetric key system is now often used that is known as the public key infrastructure (PKI). Public-key encryption does not require that you openly exchange a secret key with the recipient of an encrypted message. [1]

Public-key cryptography also referred to as asymmetric encryption, uses a pair of cryptographic keys, a public key and a private key see The private key is kept secret and defines the associated decryption transformation, while the public key defines an encryption transformation and can be distributed openly, thereby negating the need to transmit a secret key in advance. The keys are related mathematically, allowing the sender of a message to encrypt his message using the recipient's public key. The message can then only be decrypted using the recipient's private key. Since encryption transformation is public knowledge, public-key encryption alone does not provide data origin authentication or data integrity. Public-key decryption by using private key can provides authentication guarantees in entity authentication and authenticated key establishment protocols. So, public-key encryption provides both privacy and confidentiality.

Public-key encryption schemes are simple, elegant and relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart. However they are typically substantially slower than symmetric-key encryption algorithms. For this reason, public-key encryption is most commonly used in practice for the transport of keys subsequently used for bulk data encryption by symmetric algorithms and other applications including data integrity and authentication, and for encrypting small data items such as credit card numbers and PINs. [2]

Public-key cryptography being discovered in the mid-1970s and it does not have an extensive history as symmetric-key encryption, but the most familiar public-key algorithms are RSA and DiffieHellman key exchange, while the Digital Signature Algorithm is the most widely used digital signature system.

There are several public key algorithms. The first one proposed is the knapsack cryptosystem but it is insecure and an easily understood system. After the knapsack, RSA algorithm is known as the gold standard of public key cryptography that we will focus on it in this report. Then, The DiffieHellman key exchange was published in 1976 by Whitfield Diffie and MartinHellman and it provided a common secret key between the communicating parties over an insecure channel.

Also there are more public key algorithms such as El-Gamal Cryptosystem and Elliptic Curve Cryptosystem (ECC), [3] but in this report we are mainly focused on some asymmetric algorithms which are mostly used. They are RSA and ElGamal cryptosystems.

## 2 RSA Cryptosystem

RSA algorithm is used for public key cryptography and digital signatures. It is the most commonly used public key encryption algorithm. The security of the RSA algorithm is based on the difficulty of factoring large numbers n = pq, where p and q are large prime numbers. The RSA algorithm keys are often of length a power of two, like 512, 1024, or 2048 bits, RSA to be secure, the key length has to be at least 1024-bits. [4]

This section will provide the history of RSA, describes the RSA encryption scheme, its security, and provides some Practical Considerations.

### 2.1 RSA Background

RSA invented by Rivest, Shamir and Adleman in 1977, RSA stands for the first letter in each of its inventors' last names. It was first published in the paper A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, but the National Security Agency (NSA) did not like the idea of the distribution the cryptography source code internationally, and requested to be stopped. However, they continued distribution because the NSA failed to provide a legal basis for their request. [5]

### 2.2 RSA Details

RSA system involves three steps: key generation, encryption and decryption. First step: key generation, the mathematical details of the algorithm used in obtaining the public and private keys are explained as follows:

1) Take two large primes, p and q, and compute their product n = pq, where n is called the modulus.
2) Choose a number, e which is less than n and relatively prime to $(p-1)(q-1)$, that means e and (p-1)(q-1) have no common factors except 1.

3) Find another number d such that $(ed-1)$ is divisible by $(p-1)(q-1)$. Whereas the values e and d are called the public and private exponents, respectively.

The public key is the pair (n, e); the private key is (n, d). The factors p and q may be destroyed or kept with the private key.

It is currently difficult to obtain the private key d from the public key (n, e). However if one could factor n into p and q, then one could obtain the private key d. Thus the security of the RSA system is based on the assumption that factoring is difficult. The discovery of an easy method of factoring would "break" RSA.

Second step: encryption, the ciphertext C is found by the equation $C = M^e \mod n$ where M is the original message and e is the public or encryption exponent.

Third step: decryption, the message M can be found form the ciphertext C by the equation $M = C^d \mod n$ where d is the private or decryption exponent.

The RSA system can be used for public key cryptography (encryption) and digital signatures as we said before, there is slightly different between them.

In RSA, encryption keys are public, while the decryption keys are private, so no one can decrypt an encrypted message except the person who has the private key. Everyone has their own public and private keys. The keys must be made in such a way that the private key may not be easily construed from the public encryption key. [5]

Digital signatures : The recipient may need to verify that the message sent in fact originated from the sender (sign), just did not come from there (authentication). This is done by using the decryption key of the sender, and anyone can use the corresponding public key encryption to verify the signature. Therefore, Signatures cannot be forged and after signed the message no one can deny the site at a later time. [5]

## 2.3 Known Attacks on RSA Cryptosystem

The RSA function is a one-way trapdoor function. It is not known to be easily invertible without the trapdoor, d. General attacks on RSA involve trying to compute d, or some other way to get to M (plaintext). The difficulties and the main security of RSA algorithm is based on prime factorization problem. [6]

There has been much research and many publications on attacks on the RSA Cryptosystem. We can categorize attacks on RSA into five categories as below, and we will present a summary of them.

### 2.3.1 *Factoring large integers*

Given (C, e, N), Marvin can do a brute force search to find M. But factoring N involves a much smaller search space and thus, more efficient to do. Factoring is a well-researched problem. Pomerance [7] has an interesting introduction into what clever methods can be done. For example, how to factor 8051? Naive approach is trying all primes up to square root of 8051, or almost 90. Clever trick is to find a square of primes that subtract to the number, i.e.$90^2 7^2 = 8051$. Thus $(907) * (90 + 7) = 8051$, $83$ and $97$ are the factors. However, this trick is only easy if factors are close to square root of the number. There are only a very small set of numbers where this is true. But much more sophisticated methods are known. Currently the general number field sieve is the best known running time with, reported by Boneh [6], a time on n-bit integers of $\exp^{(c+o(1))n^{1/3}\log^{2/3} n}$ for some $c < 2$.

To be complete, Peter Shor [8] has shown a quantum computer has a polynomial time algorithm for factoring integers. But engineering a quantum computer still has a way to go, so RSA is safe for now.

RSA security depends on the computation to be unreasonable for 1) the factoring of N, and 2) computing the e*th* roots modulo N. The first is fairly well researched and complexity is understood. However, the second is an open question. It has not yet been proven that taking the eth roots modulo N is at least as hard as factoring N. This means RSA security has not been proven to be at least as hard as factoring. But it has withstood time and a large amount of research, which is encouraging that it will stay secure well into the future.

#### 2.3.2 *Elementary attacks*

These attacks show blatant misuse of RSA. There are many such attacks exist, but in this report we will illustrate two of them:

Common Modulus: Fixing N. One blatant misuse of RSA would be using the same modulus N for all users. So instead, have a central authority issue out unique e,d pairs to each user. This sort of works, in the fact that $C1 = M^{e1} \mod N$ can only be decrypted with corresponding pair d1. So a different user with $e_2, d_2$ does not appear able to decrypt. But from the property that knowing (e,d,N) you can factor N, then the game is over. Any user can use their $(e_i, d_i)$ pair to factor $N$. Then, given any other users $e_j$ and the factors of $N, d_j$ can be computed. RSA key pairs should never use the same N twice [6].

Blinding attack:signature forgery can be done with a technique called blinding. If Alice ever blindly signs M, then Marvin might be able to forge Alices signature. If Marvin sends evil plaintext M for Alice to sign, its easy to assume Alice would reject and not sign it. But what if Marvin sends a random, harmless looking M? Depending on the implementation, Alice might be fooled and sign it. Marvin then can take advantageous of this is by generating $M = r^e M \mod N$, for some random r. If Alice provides a signature, S on M, then Marvin can compute Alices signature S for M by $S = S/r \mod N$. [6]

#### 2.3.3 *Low private exponent attacks*

Having a low private exponent d will reduce decryption and signature computing costs. However, too low of a d is insecure. Boneh [6] describes approximations using fractions can allow Marvin to solve for a small d that is $d < \frac{1}{3} \cdot N^{1/4}$. If N is 1024 bits, then d should be at least 256 bits long order to avoid this attack.

#### 2.3.4 *Low public exponent attacks*

Having a low public exponent will reduce encryption and signature validation computing costs. However, too low of an e is also insecure. Todays standard e is set at $2^{16} + 1$. This is a large enough value to avoid attacks and needs only 17 mod multiplications for $M^e \mod N$ using repeated squares. But if a very small e is used instead, it can be subject to attacks such as Hastads Broadcast Attack. If the same M is encrypted with many users (e,N) keys and broadcasted out, Marvin can collect each and compute M. If all users have the same e, then Marvin needs to collect at least e messages. Restating Boneh [6], take example if $e = 3$,

$$C1 = M^3 \mod N1 \tag{1}$$

$$C2 = M^3 \mod N2 \tag{2}$$

$$C3 = M^3 \mod N3 \tag{3}$$

$$M < N1, N2, N3 \;\; thus \;\; M^3 < N1N2N3 \tag{4}$$

Using Chinese Remainder Theorem (CRT) $C1C2C3 = M^3 \mod N1N2N3$, thus taking cube root of C1C2C3 gives M. Stronger attacks are also known on a small e. If you pad M in the above scenario to make it unique for each message, then the broadcast attack fails. But Hastad shows if the padding scheme is a public, fixed polynomial function it doesnt defend from the attack. Franklin and Reiter found an attack on two related messages encrypted with same modulus in time quadratic to e. And Coppersmith took it farther to show an attack on same messages that used a short, random pad (1/9th the size of M). So using a small e is not wise. To defend against all above low public exponent attacks, large e such as the standard $2^{16} + 1$ should be used. A good randomized pad also helps make random Ms to remove relationship amongst messages.

#### 2.3.5 *Implementation attacks*

Research [9] has shown that it is possible to recover the private keys of RSA without directly breaking RSA. This type of attack is known as "timing attack" an attacker notes runtime encryption algorithm and informed and thus justify the extracted parameter involved in secret operations. While it is generally agreed that the RSA is secure from direct attack, not so well known and often overlooked weakness of RSA to timing attacks.

In 1996, Kocher was the first one who discuss about timing attacks. Timing attacks are a form of "side channel attack" where an attacker gains information from the implementation of a coding rather than from any inherent weakness in the mathematical properties of the system. The operation in RSA that involves the private key is thus the modular exponentiation $M = C^d \mod N$, where N is the RSA modulus, C is the text to decrypt or sign, and d is the private key. The attacker wants to find d. For a timing attack, the attacker must have a target system compute Cd mod N for several carefully selected values of C. By precisely measuring the amount of time required and analyzing the timing variations, the attacker can recover the private key d one bit at a time until know the entire exponent. [9]

For example the repeated squaring algorithm does a round of computation for each bit of d. If the bit is 1, then an additional multiplication mod N is performed. Thus, analyzing the timing/power to determine the operations can give away d. To prevent, fix the algorithm to provide the same timing and power for each bit of d regardless of 0 or 1. [6]

If using the Chinese Remainder Theorem (CRT) to speed up computation, a random fault could expose secrete key d on a signature. With CRT to compute $M^d \mod N$, you instead work with $M^d \mod q$ and $M^d \mod p$ to reduce the computation time by working with smaller modulus. However, if a random fault happens in only one of the equations, then Marvin can figure out one of the factors of N. If a fault happened in $M^d \mod q$, then for the final produced C, Marvin would observe $C^e$ does not equal M mod N. Now $C^e = M \mod q$ is also not true, but $C^e = M \mod p$ is. Thus gcd(N, $C^eM$) exposes a factor of N To prevent, use random padding so Marvin doesnt know M. Or, double check the signature to observe faults before releasing. [6]

Timing attacks illustrate that attackers do not necessarily play by the presumed rules and that they tend to attack the weakest link in a system. The best way to protect against such attacks is to think like an attacker and find these links before attacker does.

## 3 El-Gamal Cryptosystem

As stated in the Section 1, after the introduction of public key cryptosystems concept by Diffie and Hellman in [10], a lot of trials and errors have been made to find feasible cryptosystems. The security of RSA system discussed above has much to do with large integers factorization. The knapsack public key encryption scheme relies on the complexity of subset sum problem, which is NP-complete [11]. The first example of *provably secure* public key encryption scheme, i.e. the Rabin scheme, is based on the problem of finding square roots of a modulo a prime. In a more generic manner, the Rabin encryption scheme is derived from the problem of finding $d^{th}$ roots in a finite field, which is intensively discussed in [12]. In this section, we discuss another cryptosystem that is still being widely used, i.e. ElGamal cryptosystem.

It is well recognized that the ElGamal cryptosystem could be regarded as Diffie-Hellman key agreement [13] in key transfer mode. Thus, the security of ElGamal cryptosystem has much to do with the intractability of discrete logarithm problem as well as the Diffie-Hellman problem. We analyze them one by one thereafter. We follow the definition style in [2].

### 3.1 Diffie-Hellman Problem

The Diffie-Hellman key exchange agreement and its derivatives, alongside with ElGamal public key encryption scheme are formed on the basis of Diffie-Hellman problem.

*Definition 1:* For a group $G$, if there exists an element $\alpha \in G$ such that for each $b \in G$ there is an integer $i$ satisfying $b = \alpha^i$, we then call $G$ a cyclic group and element $\alpha$ a generator[1] of $G$.

*Definition 2:* The Diffie-Hellman problem (DHP): find $\alpha^{ab} \mod p$, provided that a prime $p$, a generator $\alpha$ of $Z_p^*$, and elements $\alpha^a \mod p$ and $\alpha^b \mod p$

*Definition 3:* The *generalized Diffie-Hellman problem (GDHP)*: find $\alpha^{ab}$, provided that a finite cyclic group $G$, a generator $\alpha$ of $G$, and group elements $\alpha^a$ and $\alpha^b$ are known.

The link between Diffie-Hellman problem and discrete logarithm problem (DLP) is established as follows. Under the assumption that it is easy to solve discrete logarithm problem in $Z_p^*$, one is able to compute $a$ from $\alpha$, $p$, $\alpha^a \mod p$ by way of solving a discrete logarithm equation.

1. There could be more than one generator in a group, or none at all.

And then he can compute $(\alpha^b)^a = \alpha^{ab} \mod p$ with the knowledge of $\alpha^b \mod p$ at the same time.

The most recent findings still show that it remains unknown whether generalized discrete logarithm problem (GDLP) and GDHP are computationally equivalent. Nevertheless, we summarize some recent progress with regard to this open problem below. The Euler phi function is marked as $\phi$. $B$-smooth is defined under the fact that all prime factors of an integer are $\leq B$. (B is a given positive integer.)

1) Assume that $p$ is a prime and the factorization of $p-1$ is known. Under the circumstance that $\phi(p-1)$ is $B$-smooth, in which $B = O((\ln p)^c)$ for some constant $c$, the DHP and DLP in $Z_p^*$ are computationally equivalent. Proof of this statement can be found in [14].
2) A more general case is that when $G$ is an order $n$ finite cyclic group where the factorization of $n$ is known. In this case we can also conclude the GDHP and GDLP in $G$ are computationally equivalent.
3) In this situation, group $G$ is assumed to have the same property as above. When either $p-1$ or $p+1$ for $p$ as a prime divisor of n is $B$-smooth ($B$ has the same property as above, too), we then conclude that the GDHP and GDLP in $G$ are computationally equivalent. Proof of this statement, and some stronger ones can be found in [15].

Diffie-Hellman key exchange scheme is based on the Diffie-Hellman problem discussed above. It was proposed by Whitfield Diffie and Martin Hellman in *New Directions in Cryptography* [10]. What's more important in their invited paper is that they carefully examined two kinds of then contemporary development in cryptography, and shed light on methods of utilizing theories of communication and computation as tools to solve future cryptography problems. The Diffie-Hellman key exchange scheme has been widely used in Secure Shell (SSH), Transport Layer Security (TLS), and Internet Protocol Security (IPSec) since its proposal. Moreover, it is a key exchange protocol and *not* used for encryption. On the other hand, the ElGamal cryptosystem that is discussed below can be employed in both encryption and digital signature.

Diffie-Hellman key exchange scheme serves as a fundamental technique which provides unauthenticated key exchange. Herein we analyze the basic Diffie-Hellman protocol and briefly introduce some related ones which provide various authentication assurances.

As the first practical key exchange scheme proposed to tackle key distribution problem, which becomes with an urgent issue after the discovery of public key cryptosystem, Diffie-Hellman key exchange scheme (also called *exponential key exchange*) makes it possible that two parties of communication can establish a common secrete, which is strongly against eavesdropping theoretically, by way of a simple exponential calculation through an open channel. And this seemingly unrealistic approach is done without the prerequisite of, say Alice and Bob having met each other before or shared critical component of the scheme in advance. However, the primitive Diffie-Hellman technique has its defects. Even though this sort of protection performs well when eavesdroppers (passive adversaries) are cutting in the communication, it lacks key procedures to protect users from active adversaries who possess the ability to intercept, modify or inject messages. As reflected in Fig. 3, in real world communication scenarios, neither party (of one specific communication) can 100% tell that the source identity of the incoming message is the one it makes the request for. The identity of the other party may be the one that "happened" know the resulting key, i.e. entity authentication or key authentication.

In order to deal with key authentication issue brought up by basic Diffie-Hellman key exchange scheme, a simple method is to set $\alpha^x$ and $\alpha^y \mod p$ constant public keys for the corresponding parties. In this way, these public keys can be distributed through signed certificates, and the problem of long-term shared key is "seemingly" fixed. When such certificates are available *a prior*, key exchange process evolves into a zero-pass key exchange scheme, which no cryptographic message is required to finish the exchange. Nevertheless, an obvious

SUMMARY: $A$ and $B$ communicates with each other through open channel. In this simplified case, they are sending the other one message.
RESULT: A common secrete known to both $A$ and $B$ is established.

1. *One-time Setup.* Select and publish an appropriate prime $p$ and generator $\alpha$ of $Z_p^*$.
2. *Protocol messages.*

$$A \rightarrow B : \alpha^x \mod p \tag{5}$$

$$A \leftarrow B : \alpha^y \mod p \tag{6}$$

3. *Protocol actions.* Each time a shared key is required, the following actions would be carried out.
   1) $A$ selects some random secrete $x$, $1 \leq x \leq p-2$. Then sends $B$ message (5).
   2) $B$ selects some random secrete $y$, $1 \leq y \leq p-2$. Then sends $A$ message (6).
   3) $B$ computes $K = (\alpha^x)^y \mod p$ as the shared key after receiving $\alpha^x$.
   4) $A$ computes $K = (\alpha^y)^x \mod p$ as the shared key after receiving $\alpha^y$.

Fig. 3. **Protocol** Diffie-Hellman key exchange scheme (basic version)

drawback is that as *a prior*, once being hacked, all messages encrypted with this key are under attack. One solution for this drawback is proposed as MTI/A0 key exchange protocol in [16]. The MTI/A0 variant of Diffie-Hellman key exchange scheme patches this mentioned drawback by producing time-variant session keys and use them in mutual authentication in both directions of key exchange. This approach is done implicitly so it performs well against passive attacks. Another roundabout solution that takes in key update technique is used in ElGamal key exchange scheme, which we discuss in later subsections.

Generally speaking, the Diffie-Hellman protocol as well as other similar ones based on it could be carried out in any group where both the discrete logarithm problem is difficult to solve, and exponential computation is easy. In practice, the following groups are more commonly used.

- Multiplicative group $Z_p^*$ of $Z_p$.
- Analogous multiplicative group $F_{2^m}$.
- The group of points defined over a finite field by an elliptic curve.

### 3.2 Discrete Logarithm Problem

The security issues associated with the intractability of discrete logarithm problem have much to do with many cryptographic techniques derived from this fascinating problem. Some well-known and widely used schemes include Diffie-Hellman key exchange protocol, ElGamal encryption, and the ElGamal signature scheme and its derivatives. This subsection deals mainly with the mainstream knowledge with regard to algorithms developed to solve discrete logarithm problem.

The general mathematical setting implied to describe algorithms in this subsection is a (multiplicatively written) finite cyclic group $G$ of order $n$. And there is at least a generator (see definition in 1) named $\alpha$. Following a popular approach, it would be convenient to think of $G$ as the multiplicatively group $Z_p^*$ of order $p-1$, and what the group does is merely multiplication modulo $p$.

*Definition 4:* Suppose that $G$ is a finite cyclic group of order $n$, $\alpha$ a generator of $G$, and $\beta \in G$. We denote $\log_\alpha \beta$ as the discrete logarithm of $\beta$ to the base $\alpha$, and it is the unique integer $x$, $0 \leq x \leq n-1$, such that $\beta = \alpha^x$.
In cryptography, there are some groups (of numbers) that are more interesting for researchers than others. Cases of examples are multiplicative group $F_p^*$ of the field $F_p$. More specifically, multiplicative group $Z_p^*$ of the integers modulo a prime $p$, multiplicative group $F_{2^m}^*$ of the finite field with characteristic two. Another group that becomes popular after the invention of elliptic curve cryptosystem is the group of units $Z_n^*$ with $n$ being a composite integer. Specific examples are the group of

points which are defined within a finite field on an elliptic curve, as well as the jacobian of a hyperelliptic curve that is also defined over a finite field.

*Definition 5:* The *discrete logarithm problem* (DLP) is defined as follows: Compute the integer $x$, $0 \leq x \leq p-2$ that satisfies $\alpha^x \equiv \beta(\mod p)$, where $p$ is a prime, $\alpha$ a generator of $Z_p^*$, and $\beta \in Z_p^*$.

*Definition 6:* The *generalized discrete logarithm problem* (GDLP) is defined as follows: Compute the integer $x$, $0 \leq x \leq n-1$ that satisfies $\alpha^x = \beta$, where $\alpha$ is a generator in a finite cyclic group $G$ of order $n$, and $\beta \in G$.

Due to the fact that elliptic curve cryptography is yet another grand topic, we do not discuss them intensively in this paper. It is generally considered to be associated with discrete logarithm problem through a method called *composite moduli* [2].

When analyzing the generalized problem, it is not hard to derive that the difficulty of GDLP is independent of generator. Assume that there are two generator $\alpha$ and $\gamma$ for a cyclic group $G$ of order $n$, and $\beta$ is an element in $G$. From the definition of group generator we can thus have the following equations: $x = \log_\alpha \beta$, $y = \log_\gamma \beta$, and $z = \log_\alpha \gamma$. Combining those equations, we have $\alpha^x = \beta = \gamma^y = (\alpha^x)^y$. Consequently $x = zy \mod n$, and

$$\log_\gamma \beta = (\log_\alpha \beta)(\log_\alpha \gamma)^{-1} \mod n \tag{7}$$

From equation (7) we can see once an algorithm is developed to compute logarithms to the base $\alpha$, can easily be utilized to compute any other base $\gamma$ serving as a generator of group $G$.

Before stepping into known algorithms for DLP, we talk about some more general cases here. A even more generalized formulation of GDLP can be defined as follows: provided that such an integer exists that satisfies $\alpha^x = \beta$, find it in a finite group where $\alpha, \beta \in G$. Generally speaking, this problem may be harder to solve than GDLP. Nevertheless, under the circumstance where $G$ is a cyclic group, i.e. $G$ being the multiplicative group of a finite field and the order of $\alpha$ already known, it would be much easier to recognize the existence of integer $x$ satisfying $\alpha^x = \beta$. For this requirement, the only condition that has to be met is $\beta^n = 1$, where $n$ is the order of element $\alpha$.

In chapter 3 of "Handbook of Applied Cryptography" [2], the authors point out that "Solving the DLP in a cyclic group $G$ of order $n$ is in essence computing an isomorphism between $G$ and $Z_n$". Isomorphism for cyclic groups refers to the fact that basic structure of the two groups is the same while representation of their elements differs. It is pointed out that some algorithm which is pretty efficient with one group may not be adopted to handle problems with another without any modifications. A self-obvious example is each cyclic group of order $n$ is isomorphic to the additive cyclic group $Z_n$. The latter is merely a group composed of integers $0, 1, 2, \ldots, n-1$ with addition modulo $n$ as the group operation.

We classify known effective algorithms for DLP as follows:

1) algorithms working in arbitrary groups, e.g. exhaustive search, the baby-step-giant-step algorithm, Pollard's rho algorithm;
2) algorithms working in arbitrary groups with on-purpose design to tackle groups consisting of small prime factors, e.g. Pholig-Hellman algorithm; and
3) algorithms working only effectively in certain groups, e.g. the index-calculus algorithm.

### *3.2.1 Exhaustive search*

As with all other cryptosystems, exhaustive search works well, at least theoretically. For GDLP (see definition in 6), all we have to do is to compute $\alpha^0$, $\alpha^1$, $\alpha^2$, ... until $\beta$ is obtained. This is a generic method since under this circumstance, all is does is $O(n)$ multiplications, in which $n$ serves as the order of $\alpha$. So it will turn inefficient when $n$ is substantially large. And in modern cryptography, this is extremely common.

### *3.2.2 Baby-step Giant-step algorithm*

Suppose $m = \lceil \sqrt{n} \, \rceil$, in which $n$ serves as the order of $\alpha$. The baby-step giant-step algorithm patches the exhaustive search method with a large memory consumption. This intuitive approach is based on observation discussed above.

Assume $\beta = \alpha^x$, therefore we could rewrite $x$ with $x = im + j$, in which $0 \leq i, j \leq m$. Henceforth, we obtain an equation $\alpha^x = \alpha^{im}\alpha^j$. Combining these two equations, we have $\beta((\alpha^{-m})^i) = \alpha^j$. From this equation we could infer key idea lying behind this baby-step giant-step algorithm.

Judging from the algorithm in Figure 4, we need $O(\sqrt{n})$ size of storage for group elements. Constructing a table like this would require $O(\sqrt{n})$ multiplications and $O(\sqrt{n}\log n)$ for sorting it. A $O(\sqrt{n})$ times of multiplication as well as look-up is needed for step 4 in Figure 4 even after table construction.

Running time. Let's assume that $\log_{10} n$ times comparison is much less time-consuming than group multiplications, we then make this statement that the computational complexity for baby-step-giant-step algorithm is $O(\sqrt{n})$.

### 3.2.3 Pollard's rho algorithm

Pollard's rho algorithm patches the memory consumption defect of baby-step-giant-step algorithm. It is a randomized algorithm and works with the same computational complexity as algorithm in Figure 4. Due to the superiority in memory usage, it is much more preferable for real-world problems. As usual and for simplicity, we assume $G$ is a cyclic group with prime $n$ as order.

A series of group elements in $G$ is defined as follows, in which $x_{i+1} = 0$:

$$x_{i+1} = f(x_i) \overset{def}{=} \begin{cases} \beta \cdot x_i, & if\ x_i \in S_1, \\ x_i^2, & if\ x_i \in S_2, \\ \alpha \cdot x_i, & if\ x_i \in S_3 \end{cases} \tag{8}$$

for $i \geq 0$. The sub-sequences defined in turn, i.e. $a_0$, $a_1$, $a_2$, …and $b_0$, $b_1$, $b_2$ which satisfying $x_i = \alpha^{a_i}\beta^{b_i}$. Furthermore, $a_0 = 0$, $b_0 = 0$.

$$a_{i+1} = \begin{cases} a_i, & if\ x_i \in S_1, \\ 2a_i \mod n, & if\ x_i \in S_2, \\ (a_i + 1) \mod n, & if\ x_i \in S_3 \end{cases} \tag{9}$$

and

$$b_{i+1} = \begin{cases} (b_i + 1) \mod n, & if\ x_i \in S_1, \\ 2b_i \mod n, & if\ x_i \in S_2, \\ b_i, & if\ x_i \in S_3 \end{cases} \tag{10}$$

According to Floyd's cycle-finding algorithm [17], we can find two group elements $x_i$ and $x_{2i}$ satisfying $x_i = x_{2i}$. Substituting $x$ with $a$ and $\beta$ we obtain $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$. Imposing base $\alpha$ logarithm on both sides of this equation, we get

$$(b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i) \mod n \tag{11}$$

With Equation 11 at hand, it would be easy to work out $\log_\alpha \beta$.

When algorithm in Figure 5 returns error, initialization process should start over by randomly selecting $a_0$, $b_0$ from scale $[1, n-1]$. At the same time, it's not hard to tell from the algorithm that a negligible storage is required, when comparing with baby-step-giant-step algorithm.

### 3.2.4 Pholig-Hellman algorithm

Pholig-Hellman algorithm makes the use of factorization. It factorize $n$ which is the order of group $G$. Assume $n$ can be prime factorized as $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$. Suppose what we want to get is $x = \log_a \beta$, this leads to determine $x_i = x \mod p_i^{e_i}$ for $1 \leq i \leq r$. There are two common practices here, i.e. Gauss's algorithm and Chinese remainder theorem, to solve simultaneous congruences. Pholig-Hellman algorithm takes the former. Using Gauss's algorithm, we have $x_i = l_0 + l_1 p_i + \cdots + l_{e_i - 1} p_i^{e_i - 1}$ in which $l_0$, $l_1$, …, $l_{e_i-1}$ serve as coefficients of $x_i$'s $p_i$-ary representation.

According to [2], computational complexity for this algorithm under the condition that the factorization of $n$ is known, is

$$O(\sum_{i=1}^{r} e_i(\log_{10} n + \sqrt{p_i})) \tag{12}$$

From the computational complexity (Equation 12) we can see that only if each $p_i$ (prime divisor) is relatively small then this approach to generalized discrete logarithm problem is efficient.

```
Require: A cyclic group $G$ of order $n$ with a generator $\alpha$. Element $\beta \in G$.
Ensure: solved discrete logarithm $x = \log_\alpha \beta$.
    1) Initiate $m \leftarrow \lceil \sqrt{n}\ \rceil$.
    2) Set up a look-up table with entries $(j, \alpha^j)$, in which $0 \le j \le m$. For performance
       consideration, rank this table by second component. (We could also employ hashing
       techniques here. Write programs to hash the second component and then store them in a
       hashing table. After that, it just takes constant time to place an entry and search it in the
       table.)
    3) Calculate $\alpha^{-m}$ then evaluate $\gamma \leftarrow \beta$
    4) For $i$ from $0$ to $m-1$ do:
        a) Validate if $\gamma$ equals to the second component of one entry in the table.
        b) If $\gamma = \alpha^j$ then return $(x = im + j)$
        c) Evaluate $\gamma \leftarrow \gamma \cdot \alpha^{-m}$
```

Fig. 4. **Algorithm** Baby-step-giant-step algorithm for solving discrete logarithms

```
Require: A cyclic group $G$ of order $n$ with a generator $\alpha$. Element $\beta \in G$.
Ensure: solved discrete logarithm $x = \log_\alpha \beta$.
    1) Initialization. $x_0 = 1$, $a_0 = 0$,$b_0 = 0$.
    2) For $i = 1, 2, \ldots$ do
        a) Calculate $x_i$, $a_i$, $b_i$ and $x_{2i}$, $a_{2i}$, $b_{2i}$ by imputing $x_{i-1}$, $a_{i-1}$, $b_{i-1}$ and $x_{2i-2}$, $a_{2i-2}$, $b_{2i-2}$ to
           Equation 8,Equation 9 and Equation 10.
        b) If $_i = x_{2i}$, do
            i) Evaluate $\gamma \leftarrow (b_i - b_{2i}) \mod n$.
            ii) If $\gamma = 0$, return error; else evaluate $x = \gamma^{-1}(a_{2i} - a_i) \mod n$.
            iii) Return $(x)$.
```

Fig. 5. **Algorithm** Pollard's rho algorithm for computing discrete logarithms

```
Require: A cyclic group $G$ of order $n$ with a generator $\alpha$. Element $\beta \in G$.
Ensure: solved discrete logarithm $x = \log_\alpha \beta$.
    1) Work out the prime factorization of $n$:$n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, in which $e_i \ge 1$.
    2) For $i$ from $1$ to $\gamma$ do:
       (Work out $x_i = l_0 + l_1 p_i + \cdots + l_{e_i-1} p_i^{e_i-1}$, in which $x_i = x \mod p_i^{e_i}$)
        a) Initialization. $q \leftarrow p_i$, and $e \leftarrow e_i$.
        b) Evaluate $\gamma \leftarrow 1$ and $l_{-1} \leftarrow 0$.
        c) Calculate $\bar{\alpha} \leftarrow \alpha^{n/q}$.
        d) (Calculate the $l_j$) For $j$ from $0$ to $e-1$ do:
            i) $\gamma \leftarrow \gamma \alpha^{l_{j-1} q^{j-1}}$ and $\bar{\beta} \leftarrow (\beta \gamma^{-1})^{n/q^{j+1}}$.
            ii) $l_j \leftarrow \log_{\bar{\alpha}} \bar{\beta}$
        e) Set $x_i \leftarrow l_0 + l_1 p_i + \cdots + l_{e_i-1} p_i^{e_i-1}$
    3) Calculate integer $x$ satisfying $0 \le x \le n-1$ and for $1 \le i \le r$ $x \equiv x_i \mod p_i^{e_i}$.
```

Fig. 6. **Algorithm** Pholig-Hellman algorithm for computing discrete logarithms

### *3.2.5 Index-calculus algorithm*

The last but never the least algorithm to "combat" against discrete logarithm problems that we discuss here is index-calculus algorithm. By fay when it is applied to its specific-purpose group, its computational complexity would often reduce to subexponential. From this perspective, it is regarded as the most powerful method nowadays. We describe a general setting here, namely the working group is a cyclic one called group $G$.

What makes the index-calculus algorithm unique is that it “selects” a relatively small subset out of vast elements in $G$. The subset is named *factor base* due to the fact that from using elements in *factor base* a significant amount of elements in group $G$ could be expressed, with high efficiency. Therefore in practice such a database is precomputed most of the time and reused when it comes that a particular group element is required.

### 3.3 Basic El-Gamal Encryption

After reviewing related parts of ElGamal cryptosystem in 1, and subsection 3.2, we discuss the encryption and digital signature schemes in the following subsections.

Therein we describe ElGamal public-key encryption algorithm as follows:

A tiny proof that decryption works.

$$\gamma^{-a} \cdot \delta \equiv \alpha^{-ak} m a^{ak} \equiv m \mod p \tag{16}$$

### 3.4 Comparison ElGamal Cryptosystem with RSA

There are many differences between RSA and El-Gamal, as following: El-Gamal relies on discrete logarithm problem while RSA relies on prime factorization problem. RSA encryption (with the public key) is faster than the corresponding operation with El-Gamal. RSA decryption (with a private key) is a bit slower than El-Gamal decryption. Historically, RSA was patented in the US, while El-Gamal was not. which is why OpenPGP specified El-Gamal usage, and GnuPG tends to favour it over RSA (the RSA patent expired a decade ago, so this is no longer an issue). RSA is described by a clear standard, which is free (as in ”free beer”), and says precisely where each byte should go. That’s very good for implementers: it reduces the possible mishaps. On the other hand, There is no established standard at all for El-Gamal, except the bits about it in OpenPGP, which od not deal with El-Gamal ”in general”.

## 4 Implementation

We implement prototypes of both RSA and ElGamal cryptosystems. The steps should be self-evident in source codes. In this manner and to save valuable space, we do not paste our codes here.

## 5 Conclusion

We carry out a comprehensive literature review of both the representative of contemporary public key cryptosystem, i.e. RSA, and another popular cryptosystem based on the intractability of discrete logarithm problem, i.e. ElGamal. Due to the lack of time, we can hardly discuss them thoroughly, but we think that key ideas relating to these two cryptosystems are covered. In the future paper we will analyze them with greater details. Note, due to the characteristic of IEEE template, there is still one figure that can not be typeset.

## 6 Member Contributions

Each team member’s contribution is described as follows:

- Yanan Xiao. He discusses the ElGamal cryptosystem and carries out most of the implementation task. He also typesets this paper in IEEE Computer Society template.
- Maryam Al Mehrezi. She finishes the introduction part as well as the discussion of RSA cryptosystem. She also tries her best to help with the prototype implementation.

## Acknowledgments

The authors would like to thank Dr. Zeyar for his amazing lectures throughout the semester, as well as assigning us a challenging but rewarding project like this. In addition, we would show our gratitude to Masdar Institute for creating the world-class research environment.

**Require:** A cyclic group $G$ of order $n$ with a generator $\alpha$. Element $\beta \in G$.
**Ensure:** solved discrete logarithm $x = \log_\alpha \beta$.

1) (*Factor base selection*) The criterion for choosing a subset as *factor base* is to effectively express a "significant proportion" of each element in group $G$ as a multiplication of elements from the subset. The subset is marked as $S = p_1, p_2, \ldots, p_t$.
2) (Constructing linear relations using elements from $S$)
   a) Randomly select an integer $k$ satisfying $0 \leq k \leq n-1$. Calculate $\alpha^k$.
   b) Factorize $\alpha^k$ by elements in $S$:

$$\alpha^k = \prod_{i=1}^{t} p_i^{c_i},\ c_i \geq 0 \tag{13}$$

   If successful, take base $\alpha$ logarithm of both sides of Equation 13, thus obtaining a linear equation

$$k \equiv \sum_{i=1}^{t} c_i \log_\alpha p_i \mod n \tag{14}$$

   c) Repeat the above two steps till $t+c$ relations taking the form 14 are obtained. ($c$ serves as a small positive integer, e.g. $c = 10$ thus providing a system of equations composed of $t+c$ relations having unique solution with high probability).
3) (Calculate logarithms of $S$'s elements) Solving the linear system constructed above ($t+c$ equations with $t$ unknowns) of the form Equation 14 to get a series of values, namely $\log_\alpha p_i$, in which $1 \leq i \leq t$.
4) (Calculate $y$)
   a) Randomly select an integer $k$ satisfying $0 \leq k \leq n-1$. Calculate $\beta \cdot \alpha^k$.
   b) Express $\beta \cdot \alpha^k$ using elements from *factor base* $S$ if possible:

$$\beta \cdot \alpha^k = \prod_{i=1}^{t} p_i^{d_i},\ d_i \geq 0 \tag{15}$$

   If failed, re-select a random integer $k$, and start this step. Else, imputing logarithm $\alpha$ on both sides of Equation 15 and we will get $\log_\alpha \beta = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \mod n$. The rest is to evaluate $y = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \mod n$ and return $(y)$.

Fig. 7. **Algorithm** Index-calculus algorithm for discrete logarithms in cyclic groups

**Ensure:** A public key and its corresponding private key is created for every entity.
Steps to generate key pairs are described as follows:

1) Generate a prime $p$ that is large enough and cannot be predicted, i.e. it should be generated randomly. Find a generator $\alpha$ of the multiplicative group $Z_p^*$ of integers modulo $p$.
2) Randomly select an integer $a$ satisfying $1 \leq a \leq p-2$. Then calculate $\alpha^a \mod p$.
3) The public key is returned as $(p, \alpha, \alpha^a)$; The private key is returned as $a$.

Fig. 8. **Algorithm** Key generation for ElGamal public-key encryption

## References

[1] J. E. Sarlabous. (2000) Introduction to cryptography. [Online]. Available: http://cel.archives-ouvertes.fr/cel-00374740

[2] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2010.

[3] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011.

[4] M. Evgeny. (2009) The rsa algorithm. [Online]. Available: http://www.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf

[5] W. Dan. (2007) The rsa algorithm. [On-

line]. Available: http://imps.mcmaster.ca/courses/SE-4C03-07/wiki/wrighd/rsa_alg.html

[6] D. Boneh, R. Rivest, A. Shamir, L. Adleman *et al.*, “Twenty years of attacks on the rsa cryptosystem,” *Notices of the AMS*, vol. 46, no. 2, pp. 203–213, 1999.

[7] C. Pomerance, “A tale of two sieves,” *Biscuits of Number Theory*, vol. 85, 2008.

[8] P. W. Shor, “Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM J.Sci.Statist.Comput.*, vol. 26, p. 1484, 1997.

[9] W. H. Wong, “Timing attacks on rsa: revealing your secrets through the fourth dimension,” *Crossroads*, vol. 11, no. 3, pp. 5–5, 2005.

[10] W. Diffie and M. Hellman, “New directions in cryptography,” *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[11] A. M. Odlyzko, “The rise and fall of knapsack cryptosystem,” *Cryptology and Computational Number Theory*, vol. 42, pp. 75–88, 1990.

[12] E. Bach and J. Shallit, *Algorithmic Number Theory*. Cambridge, Massachusetts: MIT Press, 1996, vol. 1: Efficient Algorithms.

[13] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *Information Theory, IEEE Transactions on*, vol. 31, no. 4, pp. 469–472, 1985.

[14] B. Boer, “Diffie-hellman is as strong as discrete log for certain primes,” in *Advances in Cryptology CRYPTO 88*, ser. Lecture Notes in Computer Science, S. Goldwasser, Ed. Springer New York, 1990, vol. 403, pp. 530–539. [Online]. Available: http://dx.doi.org/10.1007/0-387-34799-2_38

[15] U. Maurer, “Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms,” in *Advances in Cryptology CRYPTO 94*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed. Springer Berlin Heidelberg, 1994, vol. 839, pp. 271–281. [Online]. Available: http://dx.doi.org/10.1007/3-540-48658-5_26

[16] T. MATSUMOTO, Y. TAKASHIMA, and H. IMAI, “On seeking smart public key distribution systems,” *The Transactions of the IECE of Japan*, vol. E69, no. 2, pp. 99–106, Feb. 1986.

[17] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

**Yanan Xiao** is a first year master student at Masdar Institute. For his undergraduate, he spent three years in information security related area, mainly computer networks. Right now he is with Dr. Chi-Kin Chau to earn his MSc degree. His research interests are wireless networks, embedded systems and all kinds of algorithms. When he does not have much research workload, he usually takes some tea while reading books.

**Maryam Al Mehrezi** is a first year Computing and Information Science student at Masdar Institute. She got a bachelor degree in Computer Science (United Arab Emirates University). Watching movies, riding bicycle and all things that are related to computer science are her interests.

**Ensure:** $B$ uses $A$'s public key to encrypt a message $m$. Then $A$ uses decrypts using his private key.

1) *Encryption*. The steps for $B$ to take are as follows:
    a) Require or obtain $A$'s authentic public key $(p, \alpha, \alpha^a)$.
    b) Express the plaintext message as an integer in the scale $0, 1, \ldots, p-1$.
    c) Randomly select an integer $k$ satisfying $1 \leq k \leq p-2$.
    d) Calculate $\gamma = \alpha^k \mod p$ and $\delta = m \cdot (\alpha^a)^k \mod p$.
    e) Transmit the ciphertext $c = (\gamma, \delta)$ to $A$.
2) *Decryption*. The decrypt steps for $A$ are described as follows:
    a) Calculate $\gamma^{p-1-a} \mod p$ using $A$'s private key. (note: due to the characteristic of modulus, $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$).
    b) Calculate plaintext $m$ by evaluating $(\gamma^{-a} \cdot \delta \mod p)$.

Fig. 9. **Algorithm** ElGamal public-key encryption