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

PUBLIC key cryptosystem. The rest

2 Public Key Cryptosystem

2.1 More Details

Some problems with this template...I mean, the subsubsection part.

3 RSA Cryptosystem

This is just another testing case.

4 El-Gamal Cryptosystem

As stated in the Section 1, after the introduction of public key cryptosystems concept by Diffie and Hellman in [1], 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 [2]. 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 [3]. 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 [4] 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 [5].

4.1 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 α a generator[1] of G.

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

Definition 3: The *generalized Diffie-Hellman problem (GDHP)*: find α^{ab}, provided that a finite cyclic group G, a generator α of G, and group elements α^a and α^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

• *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

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

is able to compute a from α, p, $\alpha^a \mod p$ by way of solving a discrete logarithm equation. 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 ϕ. 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 [6].
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 [7].

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* [1]. 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. 1, 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 α^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,

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 α of Z_p^*.
2. *Protocol messages.*

$$A \rightarrow B : \alpha^x \mod p \tag{1}$$

$$A \leftarrow B : \alpha^y \mod p \tag{2}$$

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 (1).
2) B selects some random secrete y, $1 \leq y \leq p-2$. Then sends A message (2).
3) B computes $K = (\alpha^x)^y \mod p$ as the shared key after receiving α^x.
4) A computes $K = (\alpha^y)^x \mod p$ as the shared key after receiving α^y.

Fig. 1. **Protocol** Diffie-Hellman key exchange scheme (basic version)

which no cryptographic message is required to finish the exchange. Nevertheless, an obvious 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 [8]. 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.

4.2 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 α. 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, α a generator of G, and $\beta \in G$. We denote $\log_\alpha \beta$ as the discrete logarithm of β to the base α, 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, α 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 α 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* [5].

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 α and γ for a cyclic group G of order n, and β 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

$$\log_\gamma \beta = (\log_\alpha \beta)(\log_\alpha \gamma)^{-1} \mod n \tag{3}$$

From equation (3) we can see once an algorithm is developed to compute logarithms to the base α, can easily be utilized to compute any other base γ 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 α 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 α.

In chapter 3 of "Handbook of Applied Cryptography" [5], 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.

4.2.1 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 α^0, α^1, α^2, ... until β 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 α. So it will turn inefficient when n is substantially large. And in modern cryptography, this is extremely common.

4.2.2 Baby-step Giant-step algorithm

Suppose $m = \lceil \sqrt{n}\ \rceil$, in which n serves as the order of α. 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 2, 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 2 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 $O(\sqrt{n})$.

4.2.3 *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 2. 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 $x_{i+1} = 0$:

$$x_{i+1} = f(x_i) \overset{def}{=} \begin{cases} \beta \cdot x_i, & if\ x_i \in S_1, \\ x_i^2, & if\ x_i \in S_2, \\ \alpha \cdot x_i, & if\ x_i \in S_3 \end{cases} \quad (4)$$

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, $a_0 = 0$, $b_0 = 0$.

$$a_{i+1} = \begin{cases} a_i, & if\ x_i \in S_1, \\ 2a_i \mod n, & if\ x_i \in S_2, \\ (a_i + 1) \mod n, & if\ x_i \in S_3 \end{cases} \quad (5)$$

and

$$b_{i+1} = \begin{cases} (b_i + 1) \mod n, & if\ x_i \in S_1, \\ 2b_i \mod n, & if\ x_i \in S_2, \\ b_i, & if\ x_i \in S_3 \end{cases} \quad (6)$$

According to Floyd's cycle-finding algorithm [9], we can find two group elements x_i and x_{2i} satisfying $x_i = x_{2i}$. Substituting x with a and β we obtain $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$. Imposing base α logarithm on both sides of this equation, we get

$$(b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i) \mod n \quad (7)$$

With Equation 7 at hand, it would be easy to work out $\log_\alpha \beta$.

When algorithm in Fig. Figure 3 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.

4.2.4 *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 [5], computational complexity for this algorithm under the condition that the factorization of n is known, is

$$O(\sum_{i=1}^{r} e_i(\log_{10} n + \sqrt{p_i})) \quad (8)$$

From the computational complexity (Equation 8) 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 $0 \leq j \leq m$. 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. 2. **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 4,Equation 5 and Equation 6.
     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. 3. **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 $e_i \geq 1$.
  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 $0 \leq x \leq n-1$ and for $1 \leq i \leq r$ $x \equiv x_i \mod p_i^{e_i}$.
```

Fig. 4. **Algorithm** Pholig-Hellman algorithm for computing discrete logarithms

4.2.5 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.

4.3 Basic El-Gamal Encryption

After reviewing related parts of ElGamal cryptosystem in 1,and subsection 4.2, we discuss the encryption and digital signature schemes in the following subsections.

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

4.4 Generalized El-Gamal Encryption

I planned to clarify everything, thus there is a lot to write. Right now I would make them more concise and compact.

4.5 El-Gamal in Digital Signature

5 Implementation

Implementation process will be discussed here. Let the hunt begin [4].

5.1 RSA

5.2 El-Gamal

6 Conclusion

Conclusion and Contributions.

Appendix A
Life is never easy

Appendix one text goes here.

Appendix B
Some related math stuff will be displayed here

Appendix two text goes here.

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.

References

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

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

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

[4] 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.

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

[6] 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

[7] 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

[8] 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.

[9] 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.

Require: A cyclic group G of order n with a generator α. 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 α^k.
   b) Factorize α^k by elements in S:
   $$\alpha^k = \prod_{i=1}^{t} p_i^{c_i},\ c_i \geq 0 \tag{9}$$
   If successful, take base α logarithm of both sides of Equation 9, thus obtaining a linear equation
   $$k \equiv \sum_{i=1}^{t} c_i \log_\alpha p_i \mod n \tag{10}$$
   c) Repeat the above two steps till $t+c$ relations taking the form 10 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 10 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:
   $$\beta \cdot \alpha^k = \prod_{i=1}^{t} p_i^{d_i},\ d_i \geq 0 \tag{11}$$
   If failed, re-select a random integer k, and start this step. Else, imputing logarithm α on both sides of Equation 11 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. 5. **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 α 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, α, α^a); The private key is returned as a.

Fig. 6. **Algorithm** Key generation for ElGamal public-key encryption

Maryam Al Mehrezi is a first year master student at Masdar Institute.