A Primer on Encryption

Robert M. Townsend (rtownsen@mit.edu)
Elizabeth & James Killian Professor of Economics, MIT

Nicolas XY. Zhang (nxyzhang@mit.edu)
Computer Science and AI Laboratory, MIT

Yinhong Zhao (yinhong.zhao@duke.edu)
Department of Economics, Duke University

July 26, 2022

Abstract

As an essential component of distributed ledgers, encryption enables smart, programmable contracts dealing with private information, limited communication, limited commitment, and confidential data. The idea of encryption has been applied since ancient times, but modern cryptography features the usage of public- and private-key pairs in different settings, including encryption, signature, and hashing. Zero-knowledge proof enables the verifier to verify the knowledge of the prover without revealing the knowledge, so it allows privacy. Three more advanced ideas of encryption - homomorphic encryption, multi-party computation, and ring learning with error - enable encryption to be applied in broader settings.

1 Brief history of encryption

Cryptography is the practice and study of secure communication in the presence of adversary behavior. Limited information and limited communication are intertwined due to the need to keep a secret and yet convey the secret to an allocation mechanism. In mechanism design, incentives are such that agents are induced to tell the truth, but those messages are assumed to be going to a third party, a planner, yet in reality a third party may not be trusted. Thus, in many contexts, messages must be kept secret with secure communication.

1.1 Ancient Times

The idea and practice of encryption have emerged in ancient times with applications similar to contemporary analogues [20]. Mesopotamians sealed clay tokens in envelopes as invoices to accompany the shipment of goods and split English tally sticks to convey debt, allowing it

(a) Cuneiform script in Mesopotamia (1500 B.C.) [21]

(b) Willow tally sticks in England (1299) [13]

Figure 1: Examples of Encryption in Ancient Times

to pass through third parties. These are similar to the modern practices of using distributed ledgers to track shipping and record change in the ownership of debts.

To elaborate, for clay tokens and sealed envelopes, the deal between shipper and receiver is consummated only if the shipment arrives intact and matches the unbroken envelope, assuming the shipper ships the merchandise to the receiver as expected in their overall trusting relationship. By comparing the actual shipment to the envelope, one can see tampering by third parties if either the shipments of goods do not match the tokens in the sealed envelope or the envelope itself is broken.

The idea behind the willow tally sticks is that wood branches are split into matching halves as records of the same underlying transaction. One of the halves, the foil, remains with the issuer/borrower. Uniquely grained wood prevented counterfeiting of the other half, the stock, given to the original lender and passed to third party investors. The stock with some writing serves as an immutable record of debt issuance and circulates among multiple third parties as money. The original debt is extinguished when the stock is presented for redemption by a third party, and the supposed stock matches the foil, i.e., the halves are pieced together and checked.

The ancient systems are hybrids. They utilized trust and mitigated malicious behavior from non-trusted parties. In the case of Mesopotamia, there was trust between shippers and receivers. In the case of Medieval England, it was assumed that the borrower would repay the loan to original investor or verified third party. Encryption also considers the potential existence of malicious parties in between, for instance, someone surreptitiously pilfering goods or a third party counterfeiting promises to pay. It is fascinating that both ancient systems involved dual matching, as the public and private keys in modern cryptography.

Figure 2: Enigma machine in World War II time

1.2 World-War II Cryptography

The importance of encryption and decryption was fully demonstrated in World War II. For example, Nazi Germany employed a cipher device called the Enigma machine (shown in Fig. 2) to encrypt their messages. The electromechanical rotor mechanism of Enigma scrambles the 26 letters of the alphabet in a particular way. For example, a given machine configuration that encoded A to L, B to U, C to S, ..., and Z to J could be represented compactly as

$$LUSHQOXDMZNAIKFREPCYBWVGTJ.$$

Essentially, each encryption scheme is a permutation of the 26 letters, so it is called a mono-alphabetic substitution design. This type of cryptography requires both senders and receivers to know the "private key" or the encryption scheme, creating security flaws.

One straightforward way of attacking the mono-alphabetic substitution design is to exploit the frequency distribution of the English language. Assuming the language being used is English, the letter frequencies on average are known, as shown in Fig. 3. Therefore, the attacker merely needs to compare the frequency of each letter in the ciphertext to the average frequency of each letter in the plaintext to solve the encryption scheme.

As early as 1933, Poland had managed to crack the Enigma machine. During WWII, decryption of the Enigma Cipher allowed the Allies to read important parts of German radio traffic on critical networks and was an invaluable source of military intelligence throughout the war [5]. During the war, Nazi Germany had to change the security system of the machine daily, sometimes even for each message. The insecurity of this symmetric encryption system motivated the development of modern cryptography as an asymmetric public-private key system.

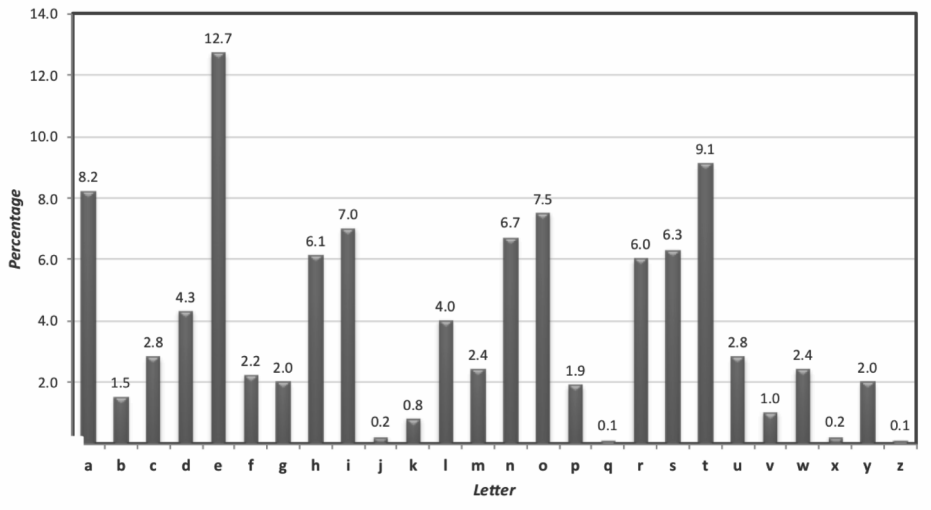

Figure 3: Average letter frequencies in English language

1.3 Modern Encryption

Historically, cryptography was more of an art than a science. Schemes were designed in a heuristic manner and evaluated based on their perceived complexity or cleverness [14]. Modern encryption defines **security** formally and provides rigorous proofs of the security of a proposed encryption scheme under certain conditions. Specifically, a well-articulated encryption scheme should satisfy the following properties to be called "secure".

1. It's impossible for an attacker to find the key.
2. It's impossible for an attacker to recover any information about the plaintext from the ciphertext without knowing the key.
3. It's impossible for an attacker to recover any additional information about the plaintext, regardless of any information the attacker already holds.

Note that the second and third principles are not straightforward. A case in point is the mono-alphabetic substitution scheme. One can infer that the most frequently used letter in the ciphertext is "e" in plaintext though one might not easily distinguish "h" and "r", so the second principle is not satisfied. If one already knows which ciphertext letter maps to "e", it is possible to guess some words with more than one "e" in them, revealing more information about the encryption mapping.

In the rest of the paper, we will introduce a variety of encryption schemes in modern cryptography that satisfy the three desirable properties proposed above. Many of these schemes also carry their desirable properties as a bonus.

2 Introduction to modern cryptography

Cryptography is the practice and study of secure communication in the presence of adversary behavior. Modern cryptography, unlike clay envelops, willow branches, and the Enigma machine, is based on some generally unsolvable mathematical problems.

Example 2.1. Let p, q be two large enough and arbitrarily chosen prime numbers, then pq can be factored into the original parts. We note the following three facts:

1. Inferring both p and q from pq, the factoring problem, involves exploding amount of computational work and thus virtually impossible to solve.
2. Given one of the factors, say p, finding q is easy through division of pq by p.
3. Given both p and q, verification that these are two primes that compose pq is easy through multiplication.

As a note to Fact 1, from a perspective of computation, no efficient, non-quantum prime factorization algorithm is known [6]. Up to now, no algorithms have been published that can factor integers in polynomial time,[1] and the algorithm with the best theoretical asymptotic running time is sub-exponential.[2] It is generally suspected that there does not exist a polynomial-time algorithm, though this has not been proven.

To further illustrate how difficult the problem is, a group of researchers factored a 240-digit number (RSA-240) in 2019, utilizing approximately 900 core-years of computing power [5]. The result is:

$$124620366781718784065835044608106590434820374651678805754818788883289666801188210855036039570272508747509864768438458621054865537970253930571891217684318286362846948405301614416430468066875699415246993185704183030512549594371372159029236099 = 509435952285839914555051023580843714132648382024111473186660296521821206469746700620316443478873837606252372049619334517 \times 24462420883831815056781313902400289665380209257893140145204122133655847709517815525821889773503059066904130204590807144 7.$$

Although it takes extremely arduous work to discover this fact, it is easy to verify even by any regular laptop, which speaks to the fact 3 here.

Example 2.1 motivates four distinct but related principles with different uses: encryption, hashing, and cryptography puzzles. Not all are needed in any given application. Each of these is taken up below.

2.1 Public-key encryption

Public-key encryption, or asymmetric cryptography, is an encryption algorithm that uses a pair of keys. A public key is used to encrypt plaintext into ciphertext, the encrypted message. It can in principle be seen by anyone and reveals nothing, unlike the Enigma ciphertext illustrated above. A private key is used for decryption from ciphertext to plaintext. Algorithms are designed to make it virtually impossible to decipher an encrypted message without the private key. The ciphertext here is analogous to the product of the two primes

[1] An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm. Here, the input size is simply the number of digits in pq, the number to be factorized. Suppose an algorithm can factorize any pq of n digits within a running time of $T(n) \leq f(n)$, where f is a polynomial of fixed positive degree k. Then the algorithm is said to be of polynomial time.

[2] Similarly, an algorithm is said to be of sub-exponential time if for any $\epsilon > 0$, the running time can be upper bounded by a function at the level of 2^{n^ϵ}, i.e., $T(n) \leq M_\epsilon 2^{n^\epsilon}$, M_ϵ a constant related to ϵ.

in Example 2.1, virtually meaningless on its own. Only authorized parties (those who hold private keys) can decrypt the ciphertext into plaintext. Encryption solves the confidentiality problem in the presence of adversary behavior.

2.2 Messages and Signature

The idea of public-key can also be applied to messages and digital signatures. A first participant can generate a private- public- key combination for incoming messages and give the public key to a second participant. The second participant can then encrypt plaintext into ciphertext using the public key and send it securely to the first party, who can use the private key to decipher the message. The second participant does not need to know the private key.

For outgoing messages, the private key generates a digital signature that can be verified by those who know the sender's public key. They can also verify the message's origin and that the message has not been modified since it was signed. Likewise, the sender cannot repudiate the message after signing. More complicated schemes with multiple public keys allow transactions among multiple parties on a shared public platform without revealing their identities or transaction amounts to third parties.

2.3 Hashing

A hash function H is an encoding function that maps data (string) of arbitrary size to a fixed-size value. The output of the hash function is called the hash of the data. A good hash function should satisfy the following properties:

- (Weak) collision resistant: it is computationally infeasible to find $x \neq y$ such that $H(x) = H(y)$.
- Hiding: Given $H(x)$, it is computationally infeasible to find x.
- Puzzle-friendly: for every possible n-bit output y, if k is chosen uniformly at random, then it is computationally infeasible to find x such that $H(k|x) = y$ in time significantly less than 2^n, where $k|x$ means concatenating string x immediately after string k.

We note that a hash function maps strings from an infinite space to a finite space, so there must exist collision by the pigeonhole principle.[3] Weak collision resistance only implies that these collisions are computationally infeasible to find. The puzzle-friendliness guarantees that in general, there is no better solving strategy than random trying of x. One can not select an input x in the hope of having a more significant chance to get some specific output y. For example, even if an attacker already knows that the first part of the plaintext is k, then the computation work of solving the entire plaintext $k|x$ will not be easier than the case when the attacker has no prior information.

Example 2.2. First developed by the U.S. National Institute of Standards and Technology (NIST), SHA-256 is a very widely used hash function nowadays. It converts an input string of arbitrary length to a 64-digit output (so 256 bits). While the technical detail of the algorithm

[3]If n items are placed in m sets, where $n > m$, then at least one set contain more than one item. This is called the pigeonhole principle.

Input	Output
“hello world”	“b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9”
“hello world1”	“916e14036f2d86a479ab16a3f2cffaf73a5419d12576497cc2d837fb423571a5”
“Hello world”	“64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c”
“h”	“aaa9402664f1a41f40ebbc52c9993eb66aeb366602958fdfaa283b71e64db123”

Table 1: Input and output of SHA-256 hash function

is beyond the scope of this paper, we provide some examples of the outputs generated by this hash function given some specific inputs.

As shown in Table 1, one can not spot any patterns in how the output of SHA-256 function relates to the input. Even if we only make a minimal change to the input string, the output is entirely different. However, the output string is fixed given the same input string, so there are no stochastic components in the hash function.

The hash can serve as proof of ownership of some document (e.g., this paper). One can claim to be the document's creator by publicizing the output of the hash function as evidence. While this move does not disclose the original document, it protects the author. Suppose any copyright conflict happens in the future. In that case, the author can provide the document whose hash is the publicized one to prove that the author completed the document by the time the hash was publicized. The hash serves as a certification under this circumstance. More generally, a document can be hashed at the moment with the hash made public. In the future, one can compare the document's hash with the hash previously publicized to validate that the document has not been altered.

2.4 Merkle Tree

In practice, data are stored all over the internet in a decentralized and distributed way. Then, hash functions are used in conjunction with hash tables to store, retrieve, and allow validation of these data records. An accompanying indexing system can inform the user of where an item is stored and how to verify its validity.

One way to organize large masses of data is **Merkle tree** (or a hash tree). It organizes data by successive group hashing, from the outer “leaves” of individual transactions or data to the thin “limbs”, thicker “branches”, and so on back down to a common “trunk”, as shown in Fig. 4. Further, the search backward to the underlying data hash reverses the path. Not all paths need to be explored, which reduces dimensions dramatically.

To be specific, in Fig. 4 suppose data block L3 is updated, then hash 1-0 will change. Subsequently, hash 1 will change because it's the hash of hash 1-0 juxtaposed with hash 1-1. The top hash will also change because it's the hash of hash 0 juxtaposed with hash 1. When someone revisits the Merkle tree later, one will notice that top hash is different from the last visit and then that hash 1 and hash 1-0 have been changed. Therefore, one will quickly notice that data block L3 has been updated without going through all the data blocks, which significantly improves search efficiency when the number of data blocks is large enough. The update can be benevolent or malicious, and one can determine later after checking the update in data block L3.

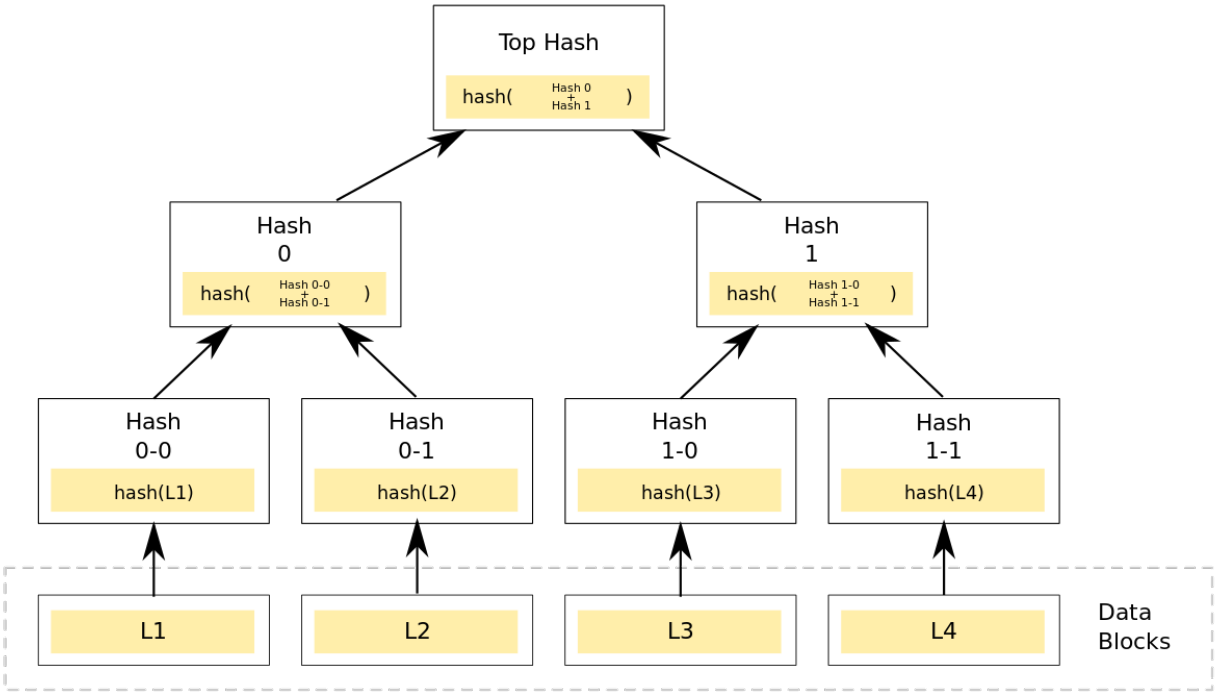

Figure 4: Illustration of Merkle Tree

Example 2.3. A particular application to financial accounts is Inveniam [1], a real estate investment company selling digital assets as a claim on a designated property. The assets are linked with hashes to requisite data and documentation, such as deeds and inspections. These digital assets are traded in a secondary market under the company Tokeny. In sum, digital assets as a contract are exchanged among third parties along with a decentralized data system.

3 ElGamal Cryptography

The realization of encryption, signature, and hashing all depend on a well-designed cryptographic puzzle. Ideally, the computational difficulty of a mathematical puzzle can be controlled manually. For example, the scale of the prime numbers in Example 2.1 can be controlled. However, trial and error can take time and energy, requiring enormous computational power, as we see in Bitcoin. We present here a classic cryptographic puzzle, **ElGamal encryption**, which operates on the algebraic structure of finite rings.

3.1 Mathematical Foundation

Definition 3.1. $a, b \in \mathbb{Z}$ are defined to be congruent modulo n if there exists $k \in \mathbb{Z}$ such that $a = nk + b$.

Example 3.1. $\mathbb{Z}_7$. We first introduce a motivating example of an algebraic structure. Consider the addition and multiplication operation under the modulus of 7. Under this circumstance, $-6, 1, 8, 15...$ are deemed to be "equivalent" because they are congruent to each other ($-6 = 7 * (-1) + 1$, $1 = 7 * 0 + 1$, $8 = 7 * 1 + 1$, $15 = 7 * 2 + 1...$). Thus, 1 essentially represents the set $\{7k + 1, k \in \mathbb{Z}\}$.

We can define the addition and multiplication in this set simply by the addition and multiplication of integers modulo 7. The sum (product) of two elements is simply the element in $\mathbb{Z}_7$ congruent to the sum (product) of two arbitrary elements from each of the two equivalent classes (e.g., $3 + 5 = 1$ because $3 + 5 = 8 \equiv 1$. $2 * 6 = 5$ because $2 * 6 = 12 \equiv 5$.) The algebraic structure of $\mathbb{Z}_7$ can be summarized as the set $\{0, 1, 2, 3, 4, 5, 6\}$ together with the addition and multiplication defined in the modulus of 7.

We further look at the multiplication structure of elements in these congruent sets modulo 7, picking elements in $\mathbb{Z}_7$, the set of integers $\{0, 1, 2, 3, 4, 5, 6\}$. For 1, $1^k = 1$. For 2, we notice that $2^{3k} = 8^k = 1^k = 1$, $2^{3k+1} = 2$, and $2^{3k+2} = 4$ for $k \in \mathbb{Z}$. There does not exist k such that $2^k = 3$. For 3, we notice that $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, and $3^6 = 1$. This means that the set $\{3^k\}$ in fact go through all the non-zero elements in $\mathbb{Z}_7$. In other words, the non-zero part of $\mathbb{Z}_7$ can be written as $\{3^k | k \in \mathbb{Z}\}$. This is an important alternative expression of $\mathbb{Z}_7$. Notice that among all the elements in $\mathbb{Z}_7$, only $\{3^k | k \in \mathbb{Z}\}$ and $\{5^k | k \in \mathbb{Z}\}$ cover all non-zero elements in $\mathbb{Z}_7$, while other numbers only cover a part of it. We call 3 and 5 generators of $\mathbb{Z}_7$ because when they generate the non-trivial part of the set simply by repeatedly multiplying themselves.

With the intuition given above, we formally define rings, finite rings and their generators.

Definition 3.2. A ring is a set R equipped with two binary operations + (addition) and $\cdot$ (multiplication) satisfying the following three sets of axioms:

1. R is an abelian group under addition, meaning that:
   - $(a + b) + c = a + (b + c)$ for all a, b, c in R.
   - $a + b = b + a$ for all a, b in R.
   - There is an element 0 in R such that $a + 0 = a$ for all a in R.
   - For each a in R there exists $-a$ in R such that $a + (-a) = 0$.
2. R is a monoid under multiplication, meaning that:
   - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all a, b, c in R.
   - There is an element 1 in R such that $a \cdot 1 = a$ and $1 \cdot a =$a for all a in R.
3. Multiplication is distributive with respect to addition, meaning that:
   - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ for all a, b, c in R.
   - $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$ for all a, b, c in R.

One example of a ring is a set G consisting of integers $\{0, 1, ...n - 1\}$ with addition and multiplication defined with modular to n, denoted $\mathbb{Z}_n$. Formally,

Definition 3.3. $\mathbb{Z}_n$ is a ring defined on the set of integers $\mathbb{Z}/n\mathbb{Z} = \{0, 1, ...n - 1\}$[4]. We define addition and multiplication with the following rules:

[4]Essentially, each element k in $\mathbb{Z}/n\mathbb{Z}$ represent an equivalence class of $\{x | x = mn+k, m \in \mathbb{Z}$. The addition and multiplication are defined on these equivalence classes.}

- (Addition) $a + b = a + b - nk$ such that $k \in \mathbb{Z}$ and $a + b - nk \in \mathbb{Z}/n\mathbb{Z}$.
- (Multiplication) $ab = ab - nk$ such that $k \in \mathbb{Z}$ and $a + b - nk \in \mathbb{Z}/n\mathbb{Z}$.

It is easy to verify that $\mathbb{Z}_n$ is indeed a ring that satisfies the properties in Definition 3.2.

The number of elements in G is termed the order of the ring G, and it is n. Power of an element is defined similarly as in the integers by multiplying the element with itself repeatedly. This finite ring will be our main object of interest in this section. Recall as mentioned earlier that $3k|k \in Z$ and $5k|k \in Z$ cover all non-zero elements in $\mathbb{Z}_7$, so that 3 and 5 generators of $\mathbb{Z}_7$ because when they generate the non-trivial part of the set simply by repeatedly multiplying themselves.

Definition 3.4. A generator of a p-order ring $\mathbb{Z}_p$ is an element $g \in G$ such that the set

$$S = \{x|x = g^n, n = 0, 1, 2, ...\} = \{1, 2, ...p - 1\} = R \setminus \{0\}.$$

This means that the ring can be generated by multiplying the generator with itself repeatedly. Rings with generators are called cyclic rings.

Example 3.2. Consider powers of a generator g, $S = \{x|x = g^n, n = 0, 1, 2, ...\}$. By definition, $S = \{1, 2, ...p - 1\}$. So, for any $y = 1, 2, ...p - 1$, there exists $x \in S$ such that $g^x = y$.

Given y, is there a quick way to solve for such an x? This problem is called discrete logarithm because heuristically $x = \log_g y$. No efficient algorithm is known to solve for such an x, even though its existence is mathematically obvious. For example, if we want to solve $3^x \equiv 2 \pmod{47}$, then computationally there is no significantly better way than computing $3^1, 3^2, ...$ one by one until one finds that $3^{17} = 129, 140, 163 \equiv 2 \pmod{47}$. When the prime number 47 gets large, the computation effort of powers can be overwhelming.

A well-known algorithm for this problem is Pohlig–Hellman algorithm, which can achieve an algorithmic complexity of $O(\sqrt{p})$. This is not feasible to compute if a sufficiently large prime number p is selected.

With these definitions, we can formally introduce the ElGamal encryption.

3.2 ElGamal Encryption

ElGamal Encrption. ElGamal Encrption is composed of the following key generation, encryption, and decryption process.

1. Key Generation: The first party Alice generate a key pair by
   - Choosing a finite group $\mathbb{Z}_q$ with generator g,
   - Randomly choosing $x \in \{0, 1, 2...q - 1\}$,
   - Computing $h = g^x$.
   - The public key consists of (q, g, h), and the private key is x.
     The private key x is safe if p is sufficiently large because no efficient, non-quantum algorithm is known to solve the discrete logarithm problem, as we introduced in Example 3.2.

2. Encryption and Incoming Message: A second party Bob can send encrypted message $m \in \mathbb{Z}_q$ to Alice by

   - Randomly choosing integer $y \in \{0, 1, 2...q-1\}$,
   - Computing $s = h^y$, $c_1 = g^y$, and $c_2 = m \cdot s$,
   - Sending (c_1, c_2) to Alice.

3. Decryption: Alice can decrypt (c_1, c_2) with her private key x by

   - Computing $s = c_1^x$ because $c_1^x = (g^y)^x = (g^x)^y = h^y = s$
   - Computing s^{-1} in $\mathbb{Z}_q$, which is computationally easy,
   - Computing $m = c_2 s^{-1}$, which is the original message.

3.3 ElGamal Signature

ElGamal signature scheme uses a similar idea, but it's used for outgoing messages, specifically hashed messages.

ElGamal Signature. ElGamal signature scheme is composed of the following key generation, signature, and verification process.

1. Key Generation: The first party Alice generate a key pair by

   - Choosing a cyclic group G of order p with generator g, where p is a prime number,
   - Randomly choosing x from $\{1, 2, ...p-2\}$,
   - Computing $y = g^x$. (G, p, g, y) is the public key, and x is the private key.

2. Signature: A message $m \in G$ is signed by:

   - Choosing $k \in \{2, 3, ...p-2\}$ randomly,
   - Computing $r = g^k$, and $s = (m - xr)k^{-1} \pmod{p-1}$.
   - The signature is (r, s).

3. Verification: When a second party Bob receives a signature, Bob can verify that (r, s) is valid if and only if:

   - $0 < r < p$ and $0 < s < p-1$.
   - $g^m = y^r r^s \pmod{p}$.

This is because if x is known, $xr + sk = m \pmod{p-1}$. Let $m = xr + sk + A(p-1)$ (1), then $g^m = g^{xr+sk} g^{A(p-1)}$ (2). By construction, $g^k = r$ (3) and $g^x = y$ (4), and Fermat's

Little Theorem[5] gives $g^{p-1} = 1 \pmod{p}$ (5). So,

$$\begin{aligned} g^m &= g^{xr+sk} g^{A(p-1)} && \text{By (2)} \\ &= g^{xr+sk} g^{(p-1)A} \\ &= g^{xr} g^{sk} (1)^A && \text{By (5)} \\ &= y^r r^s \pmod{p}. && \text{By (4), (3)} \end{aligned}$$

Therefore, any other parties can verify the signature of *Alice* to confirm the message is indeed from Alice. Any parties other than Alice cannot make a valid signature because they do not own the private key x.

3.4 Example

To further illustrate the procedure of ElGamal cryptography, we give a concrete example here. Say the sender Bob wants to inform the receiver Alice of the day to meet in the upcoming month. So the plaintext space is $\{1, 2, ...31\}$. As we described above, Alice should first generate a pair of public- and private- key, Bob can then encrypt the message with Alice's public key, and Alice can finally decrypt it with her private key. Specifically,

1. Key Generation: Alice chooses a finite group $\mathbb{Z}_{167}$ with $q = 167$ a prime and a generator $g = 4$. Say the private key chosen is $x = 37$, then Alice can compute $h = 4^{37} \equiv 76 \pmod{167}$. Alice can then announce the public key $(q, g, h) = (167, 4, 76)$.

2. Encryption and Incoming Message: Say the date Bob chooses is 16, so the plaintext is $m = 16$. Bob can randomly choose $y = 5 \in \{0, 1, 2...166\}$ and compute that $s = 76^5 \equiv 114$, $c_1 = 4^5 \equiv 22$, and $c_2 = 16 * 114 \equiv 154$. Bob then send $(c_1, c_2) = (22, 154)$ to Alice.

3. Decryption: When Alice receives the message, Alice first compute $s = c_1^x = 22^{37} \equiv 114$, and then $s^{-1} = 63$ in $\mathbb{Z}_{167}$. Alice get the plaintext message by computing $m = c_2 s^{-1} = 154 * 63 \equiv 16 \pmod{167}$. Alice will just go ahead to meet Bob at the correct day.

If Bob wants to send information more complex than a number, say a string, he can always use an encoding system to convert his message to a number. ASCII, the most widely used system, encodes the letter A as 65, B as 66, ... Z as 90. The lowercase letter a is encoded as 97, lowercase b 98,... lowercase z 122. A string can occupy much more digits than a simple number, so the finite group selected must be of a much higher order. In those cases, ElGamal encryption can be slower than what we desire.

4 Zero-knowledge proof (ZKP)

The idea of a zero-knowledge proof is for one party, the prover, to convince another, the verifier, that the prover knows something without revealing anything about that knowledge.

[5]Fermat's Little Theorem is first stated by Pierre de Fermat in 1640. The theorem states that if p is a prime and a is not divisable by p, then $p|a^{p-1} - 1$. In other words, $a^{p-1} \equiv 1 \pmod{p}$.

The concern is two-folded: the prover might deceive the verifier, and the verifier might learn something from the prover and leak or misuse it. For example, a user as prover owns a password that is subsequently hashed. Then in the future, to verify one is the correct user who knows the password, the server as a verifier re-enters the password and compares the hashes. If the hashes match, then the user has proved to the verifier that the user is the correct one. However, this is **not** a zero-knowledge proof because in this case, the verifier learns the underlying secret, the password, which can lead to bad outcomes if the server is hacked and becomes malicious.

4.1 Motivating examples

The following two examples demonstrate the idea of zero-knowledge proof. Suppose two pens seem identical to the outside world, including the verifier, but in fact, have a subtle difference that only the prover knows. The two pens are given to the verifier, with one pen claimed to have unique concealed markings, designated, while the other not. Though they appear identical to the verifier, she keeps track and scrambles them behind her back. When shown to the prover, left hand or right hand, the prover picks the designated one. The prover could have been lucky, so the experiment is repeated. If the prover truly knows the difference, the verifier is asymptotically convinced that the prover has this knowledge. However, the verifier never knows the concealed difference between the two pens, so this proof process is zero-knowledge.

A second example is a network of nodes, some of which are connected with edges. The three-color challenge is to paint all the network nodes so that no two connected nodes share the same color. A prover claims to have a solution and must convince a verifier without revealing anything about the solution. The verifier can pick an edge, and the colors of the two connected nodes are shown, which must be different if the prover has a correct solution. After one experiment, the prover recolored all the nodes, so the outcome of a second experiment is not related to the first. The experiment can be repeated until the verifier is convinced. This idea maps into various real-world applications because the three-color problem is NP-complete. To summarize:

4.2 Procedure

Definition 4.1. A zero-knowledge proof of some statement satisfies the following three properties:

1. (Completeness) If the statement is true, an honest prover will eventually convince an honest verifier.
2. (Soundness) If the statement is false, no cheating prover can convince an honest verifier, except with some minimal probability.
3. (Zero-knowledge) If the statement is true, no verifier learns anything other than the fact that the statement is true.

In practice, the colors of the nodes are concealed by hashes of a signature scheme. The process is described as follows:

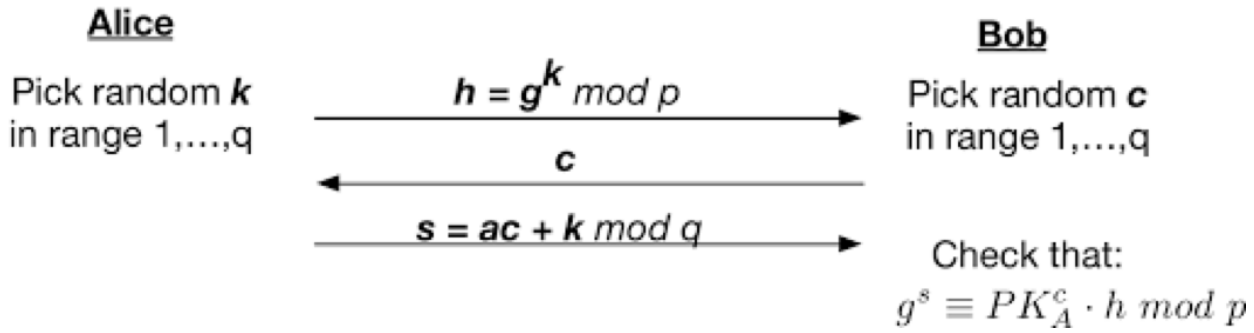

Figure 5: Illustration of Zero-knowledge Proof

1. Key Generation: Let p be a prime number, and g be a generator of a cyclic group of prime-order q. To generate a key pair, Alice would first pick a random integer $a \in \{1, 2, ...q\}$, and then compute the key pair $PK_A = g^a \bmod p$ and $SK_A = a$. Alice keeps the secret key SK_A and publishes her public key PK_A.

2. Later on, Alice needs to prove she has knowledge of her secret key. She can conduct the following interactive protocol with Bob:

   (a) Alice randomly picks $k \in \{1, 2, ...q\}$ and sends $h = g^k \bmod p$ to Bob.

   (b) Bob randomly picks $c \in \{1, 2, ...q\}$ and sends c to Alice.

   (c) Alice compute $s = ac + k \bmod q$ and sends s to Bob.

   (d) Bob checks whether $g^s = PK_A^c$.

This is complete. If Alice performs the protocol honestly, then $g^s \equiv g^{ac+k}$. By construction, $PK_A = g^a$ and $h = g^k$, so

$$g^s \equiv g^{ac+k} \equiv (g^a)^c \cdot g^k \equiv PK_A^c \cdot h \ (\text{mod p})$$

Alice can perform this experiment repeatedly, drawing a distinct k each time until the verifier Bob gets convinced. Further, if Alice was using a different key by mistake or did not know the initial key, Bob could figure this out because $g^s \neq PK_A^c \cdot h$, so the procedure is sound. Lastly, Bob does not acquire any information about the private key a, so the procedure is a zero-knowledge proof.

Such an interactive protocol across the two agents is time-consuming when repeated multiple times. In fact, Alice can effectively run the proof algorithm on her side. She can generate a random c that Bob would have sent her under the algorithm and hashes it. She offers to send messages h and s to Bob, along with the hashing function. Bob can always request to see the c that generated hash, verifying the validity of c, hence Alice has carried out the algorithm.

This idea can be extended to a larger context of a public multi-agent payments DLT platform. For example, Alice, a participant of the platform, is an employee of a company with various affiliations cutting across distinct departments she needs for work. Each department has a public key, and so does the company. Outsiders from other companies, yet still part

of the larger platform, do not know any of these public keys. Alice creates the "public address" of her account by concatenating the hashes of her relevant public keys together and signing it with all the corresponding private keys. The verifier of the transaction, Bob, needs to ensure a valid participant does the transaction without revealing any of Alice's public addresses. Bob picks part of the hash that corresponds to one of Alice's public keys to accomplish this. Alice can then match that part of Bob's hashed public keys with the corresponding private key. In this way, Alice verifies that part of the signature. The process continues successively as Bob randomly picks encrypted public keys. An iterative process is used in this zero-knowledge proof. Here, zero-knowledge proofs are used to allow the privacy of transactions in the system. In more general settings, some private information can be revealed voluntarily to some subset of participants.

5 More advanced ideas in encryption

In this section, we present three more advanced ideas in encryption, homomorphic encryption (HE), multi-party computation (MPC), and ring learning with error (RLWE). HE is more powerful in that it allows operations and actions in the ciphertext space, permitting users to perform computations on its encrypted data without decrypting it. MPC is an extension of encryption for multiple parties. RLWE is a quantum-resistant encryption scheme in a multi-party setting.

5.1 Homomorphic encryption

Homomorphic encryption is a type of encryption that permits users to perform certain operations on the encrypted data without decrypting it. The idea comes from homomorphism in algebra, which usually refers to a structure-preserving map between two algebraic structures of the same type. One primary example is ElGamal Encryption introduced in Section 3.2. It is homomorphic in multiplication, which means that the multiplication of plaintexts can be mapped to the multiplication of ciphertexts with the same encryption map.

Example 5.1. ElGamal Encryption For an ElGamal Encryption Scheme with public key (q, g, h) and private key x, the plaintext space $P = \mathbb{Z}_q$, the ciphertext space $C = \mathbb{Z}_q \times \mathbb{Z}_q$. The full space Ω is also $\mathbb{Z}_q \times \mathbb{Z}_q$, which contains both P and C. The encryption and the decryption maps are given by

$$\begin{aligned} Enc(m) &= (c_1, c_2) = (g^y, mh^y), \\ Decipher(c_1, c_2) &= c_2((c_1)^x)^{-1}. \end{aligned}$$

As shown in Section 3.2,

$$Decipher \circ Enc(m) = c_2((c_1)^x)^{-1} = mh^y((g^y)^x)^{-1} = m(ss^{-1}) = m,$$

which means that $Decipher = Enc^{-1}$. Consider then the multiplicative product of cipher-

texts of two different messages. We have

$$\begin{aligned} Enc(m_1)Enc(m_2) &= (g_1^y, m_1 h^{y_1})(g_2^y, m_2 h^{y_2}) \\ &= (g^{y_1+y_2}, m_1 m_2 h^{y_1+y_2}) \\ &= Enc(m_1 m_2) \end{aligned}$$

if selecting $y = y_1 + y_2$ when encrypting $m_1 m_2$. Alternatively, we can write

$$Decipher(Enc(m_1)Enc(m_2)) = m_1 m_2.$$

Therefore, suppose one receives two ciphertexts of two different plaintexts. Then the multiplicative product of the two ciphertexts is still meaningful in that it represents the ciphertext of the multiplicative product of the plaintexts in a specific way. We formalize this idea in the definition below.

Definition 5.1. Let $Enc : P \to C$ be an encryption scheme that encrypts the plaintexts space $P \subset \Omega$ to ciphertexts space $C \subset \Omega$, where Ω is the full space. Let $f : \Omega \times \Omega \to \Omega$ be an operation defined on Ω (so both P and C). The encryption scheme Enc is said to be homomorphic if $f(Enc(m_1), Enc(m_2)) = Enc(f(m_1, m_2))$.

The homomorphic property means that one does not need the "plaintext" information to utilize a function f in some applications. One can apply the same function f on the ciphertexts and get a result in the encrypted space. This result is the same as if one first decrypts the ciphertext, performs operation f, and then encrypts the outcome. That is

$$f \circ Enc = Enc \circ f.$$

The agents' original data can be kept encrypted and thus private. Equivalently, we can write this equation with the term in the plaintext space on the right-hand side. Let $Decipher = Enc^{-1}$ be the inverse of the encryption map, then

$$Decipher \circ f \circ Enc = f.$$

The "operability" of homomorphic encryption provides flexibility in navigating between the normal and encrypted spaces. The distributive property tells us what rules manage the parenthesis that order the sequence of operations - a distributive scheme allows operations outside the parenthesis to be "distributed" to the elements within. This allows us to perform a series of composed transitive functions g all on top of the encrypted message. Thus, the agent or contract performing operations in the ciphertext space does not need to see the actual content of the message nor the true outcome of f in the normal space.

Notice that with only multiplication, the operability of homomorphic encryption is very limited. Operability over addition is more desirable in many cases. We give a specific example of a more sophisticated homomorphic encryption here. Paillier encryption is an encryption scheme with f as simply the addition. In this encryption scheme, one can recover the sum of two plaintext messages with their ciphertext without knowing either of them.

Paillier Encryption. To achieve homomorphism in terms of both addition and multiplication, Paillier encryption operates on a more complicated plaintext space than ElGamal

encryption. Specifically, it works in the space of $n = p^2q^2$, where p and q are two large enough primes of similar scale. For a more detailed introduction, we refer readers to P.532-539 of [14] or the original paper of Pallier [17].

Definition 5.2. The Euler's function of a positive integer n is defined to be the number of elements in $\mathbb{Z}_n$ that are coprime to n, denoted as $\phi(n)$. Euler's Theorem gives that for any a coprime to n, $a^{\phi(n)} \equiv 1 \pmod{n}$.

1. Key Generation: the first party Alice generate a key pair by
   - Choosing two large enough primes p and q such that $p < q < 2p$. Then $gcd(pq, (p-1)(q-1)) = 1$, where gcd means greatest common divisor.
   - Compute $n = pq$ and $\lambda = lcm(p-1, q-1) = \phi(n)$, where lcm means least common multiple and $\phi(n)$ is the Euler's function of n.
   - Let $\mu = \phi(n)^{-1} \pmod{n}$ and $g = n + 1$.
   - The pair (n, g) is the public key, and (λ, μ) is the private key.
2. Encryption and Incoming Message: Let m be a message to be encrypted with $0 < m < n$.
   - Randomly select $0 < r < n$ and $gcd(r, n) = 1$
   - The ciphertext can be computed by the following encryption map $E : \mathbb{Z}_n \to \mathbb{Z}_{n^2}$, then
   $$c = E(m) = g^m r^n \pmod{n^2}.$$
3. Decryption: Let $c \in \mathbb{Z}^*_{n^2}$, where $\mathbb{Z}^*_{n^2}$ is a subset of $\mathbb{Z}_{n^2}$ co-prime to n^2, be a ciphertext to be decrtypted. The plaintext can be computed by the following decryption map $D : \mathbb{Z}_{n^2} \to \mathbb{Z}_n$
   $$m = D(c) = L(c^\lambda \bmod n^2) \cdot \mu \bmod n,$$
   where $L(x) = \lfloor \frac{x-1}{n} \rfloor$ is the largest integer no bigger than $\frac{x-1}{n}$.

   Theorem: $D(E(m)) = m$ for any $0 < m < n$ and E, D as encryption and decryption maps defined above.

   The proof of this theorem relies on a key mathematical observation that $r^{n\lambda} \equiv 1(\pmod{n^2})$. We prove this as a lemma. We omit the full proof here[6].

   Lemma: $r^{n\lambda} \equiv 1(\pmod{n^2})$ for any $gcd(r, n) = 1$.

   Proof of Lemma: By definition $\lambda = \phi(n)$ and Euler's Theorem, $r^\lambda \equiv 1 \pmod{n}$. Suppose $r^\lambda = k_1 n + 1$, then

$$r^\lambda = (k_1 n + 1)^n = k_1^n n^n + n k_1^{n-1} n^{n-1} + ... + n \cdot n k_1 \cdot 1^{n-1} + 1 \equiv 1 (\bmod\ n^2).$$

[6]The outline of the full proof is as follows: by definition of the encryption map, $c = g^m r^n \pmod{n^2}$, so $c^\lambda = g^{m\lambda} r^{n\lambda} \pmod{n^2}$. By lemma, $r^{n\lambda} \equiv 1 \pmod{n^2}$. So $c^\lambda = g^{m\lambda} \pmod{n^2}$. By definition $g = n + 1$, so $c^\lambda = (n+1)^{m\lambda} \equiv m\lambda n + 1(\pmod{n^2})$. Therefore, $L(c^\lambda \bmod n^2) \cdot \mu = L(m\lambda n + 1) \cdot \mu \pmod{n} = m\lambda\mu \pmod{n} = m \pmod{n}$.

4. Homomorphic Property: let $E : \mathbb{Z}_n \to \mathbb{Z}_{n^2}$ be the encryption map and $D : \mathbb{Z}_{n^2} \to \mathbb{Z}_n$ be the deciphering map, then the Pallier scheme satisfies the addition of Plaintexts:

$$m_1 + m_2 \bmod n = D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2).$$

Proof: By definition, the encryption map $E(m) = g^m r^n \bmod n^2$. So $E(m_1, r_1) \cdot E(m_2, r_2) = g^{m_1+m_2} r_1^n r_2^n$. Then applying the lemma again,

$$\begin{aligned} D(E(m_1, r_1) \cdot E(m_2, r_2)) &= L((g^{m_1+m_2} r_1^n r_2^n)^\lambda \bmod n^2)\mu \bmod n \\ &= (m_1 + m_2)\lambda\mu \bmod n^2 \\ &= m_1 + m_2. \end{aligned}$$

Example 5.2. As introduced above, Pallier encryption is additive homomorphic. We discuss how it can be applied in a setting of a distributed ledger. Suppose A is an agent of the distributed ledger B. A updates the balance in the account to B by sending an encrypted message on the net increase with Pallier encryption. Suppose the plaintext messages A intend to send are $\{m_1, m_2, ...m_n\}$, which are encrypted by the Pallier encryption map E to $\{c_1 = E(m_1), c_2 = E(m_2), ...c_n = E(m_n)\}$. B as the receiver holds the private key and thus has knowledge of the decryption map D, which satisfies $D(E(m_i)) = m_i$. Specifically, in the Pallier encryption,

$$c_i = E(m_i) = g^{m_i} r^n \pmod{n^2}$$
$$m_i = D(c_i) = L(c_i^\lambda \bmod n^2) \cdot \mu \bmod n$$

While B can decrypt every single message A send and store it in its database, B does not need to do so because this increases the risk of disclosing the privacy of A in case B is hacked. Instead, the ledger only needs to know the account balance of A at a given moment. B can first store all the messages A sends in the form of ciphertexts, and whenever B needs to access the account balance of A, B can first sum up all the ciphertexts received and then apply the decryption function D on it:

$$\sum_{i=1}^{n} m_i = D(\sum_{i=1}^{n} c_i).$$

This is guaranteed by the additive homomorphism of the Pallier encryption scheme.

Ideally, we desire a homomorphic encryption scheme with homomorphic properties on not only addition and multiplication but also on all kinds of mapping, so the structure of the whole ring can be mapped from plaintext space to ciphertext space. These encryption schemes are called fully homomorphic encryption (FHE), which are developed in the recent cryptography literature [8, 19, 9].

5.2 Multi-party computation

The primitive that powers MPC is to split a piece of data into multiple encoded parts known as secret shares. Each share reveals nothing about the original data on its own. However,

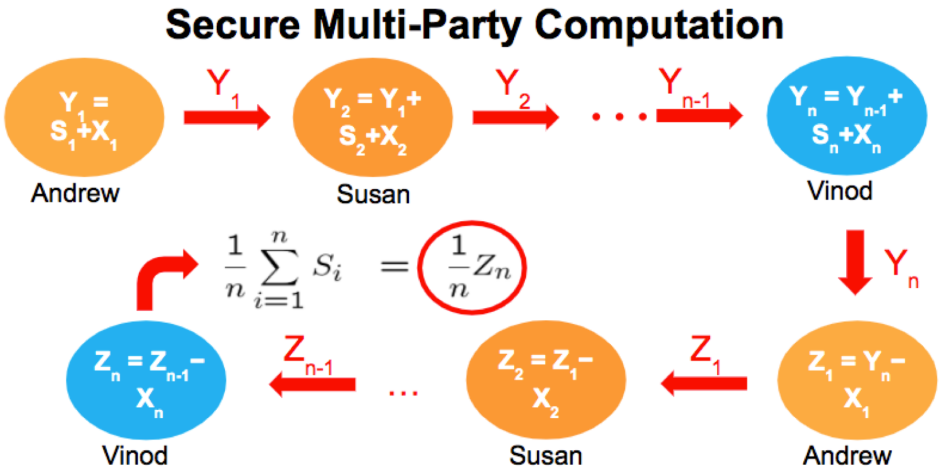

Figure 6: Multiple-party computation for a common sum

if two parties perform the same operation on a set of shares and recombine them, it is the same as the operation performed on the original data. Further, one can design parameters such that at least a critical number of participants have to agree to share.

An intuitive example is a multi-party aggregation. A group of users wants to find the sum of their private information, but they do not want to reveal their information. To realize that, each user compute $T_i = S_i + X_i$, where S_i is their private information and X_i is a random noise. Then, starting from the first user, they circulate by adding their T_i to the Y_i informed by the previous person. After the first round, they get a sum $Y_n = \sum_{i=1}^{n} T_i$. Then, they minus the X_i term from Y_n to clean the noise in the same order. After two rounds of computation, the real sum of their private information $\sum_{i=1}^{n} S_i$ is computed. This procedure is illustrated in Fig. 6. More generally, theoretical results have shown that one can construct protocols for securely computing any desired multi-party functionality assuming the existence of trapdoor permutations and honesty of all participants [10, 11].

5.2.1 Secret Sharing

As a special case, one type of problem that multi-party computation solves is the so-called Shamir secret sharing [18], which has its mathematical foundation in polynomial extrapolation. In this case, the functionality is access to a secret owned by a group of parties.

Consider a ring of polynomials of a specified maximal degree with randomly chosen coefficients for each power term. The essential idea of the algorithm is that it takes k points to uniquely determine a polynomial of (no more than) $k-1$ degree. We describe the procedure of sharing a secret constant with n participants, a minimum of k participants needed to check the secret.

For example, in Fig. 7, we show three degree-2 polynomials passing through two points on the plane. In fact, we can find infinitely many degree-2 polynomials passing through these two points. However, if we are given three points, there exists a unique polynomial with degree no larger than 2 passing the three points. To summarize,

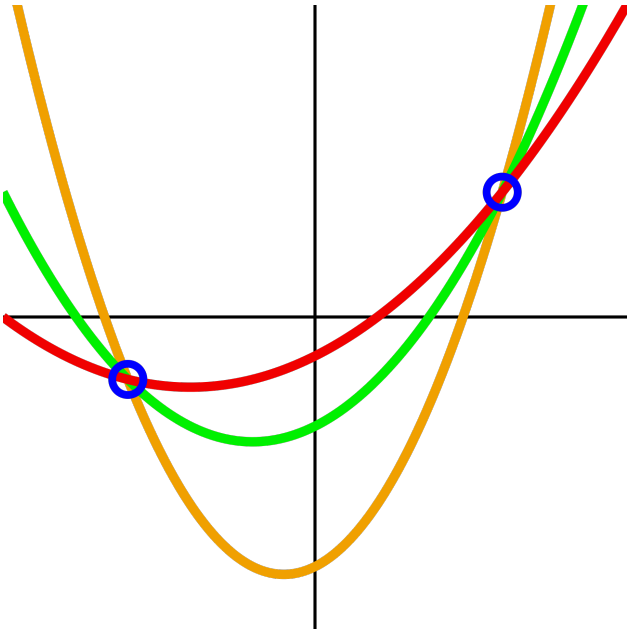

Figure 7: Three degree-2 polynomials passing through two points in the plane

1. Construct a polynomial $f = a_0 + a_1 x + .. + a_k x^k$ of degree k where $0 < k < n < P$ and P is a prime number. $a_0 = S$ is the secret constant.
2. Each of the n participants is given a non-zero integer input x_i and the corresponding integer output from the polynomial $f(x_i)$.
3. Given any $k + 1$ pairs of $(x_i, f(x_i))$, we can find the polynomial including the secret constant term $a_0 = S$. With less than $k+1$ pairs, the secret remains concealed because the polynomial f is not uniquely specified.

5.2.2 Byzantine Agreements

In real-world applications, especially in a permissionless setting, not all parties honestly report their information. Some parties might be malicious, some can be hacked, while others can make mistakes. For example, in the setting of Shamir key sharing, what if there is a malicious party that either deliberately or mistakenly reports a wrong input? If exactly $k+1$ parties are involved when the degree of the polynomial is k, then all parties will receive a wrong constant term as the secret. To make things worse, sometimes it might not be straightforward to distinguish honest parties from malicious parties.

Special designs are needed to obtain a cryptography protocol robust to dishonest parties. A Byzantine fault is defined as a condition of a computer system where components may fail, and there is imperfect information on whether a component has failed [15]. A Byzantine fault tolerant protocol refers to the distributed systems that can normally run even with a certain portion of Byzantine fault participants. Specifically,

Definition 5.3. Given n participants, t of which are dishonest, and assuming only point-to-point communication between participants, a protocol is said to be Byzantine fault tolerant if when a participant A broadcasts a value x,

1. if A is honest, then all honest participants agree on the value x;
2. if A is dishonest, all honest participants agree on a same value y.

A classic result in BFT is a protocol designed by Lamport, Shostak, and Pease in 1982 that can tolerate at most n Byzantine faults for total participants of $3n+1$ [15]. More recent literature focuses on more efficient and broadly applicable BFT protocols and a looser bound of Byzantine faults. An example of using BFT is Bitcoin, a peer-to-peer protocol that solves the BFT problem with a proof-of-work design.

5.3 Ring learning with error

Ring learning with error (RLWE) is a closely related encryption scheme that is secure against quantum computing. Quantum computing is a new type of computation that harnesses the collective properties of quantum states to perform calculations [12]. It is believed that quantum computing can solve specific problems substantially faster, including the integer factorization problem introduced in Section 2. If quantum computing becomes a reality in the future, the encryption systems introduced in the paper might be easily attacked. Therefore, the development of quantum-resistant encryption schemes is a new field in cryptography and cybersecurity.

RLWE is built on the arithmetic of polynomials with coefficients from a finite field. It is essentially the shortest vector problem (SVP) defined as below:

Definition 5.4. A lattice Λ is an infinite subset of points in the $\mathbb{R}^n$ space with the following properties:

1. Coordinatewise addition or subtraction of two points in the lattice produces another lattice point.
2. The lattice points are all separated by some minimum distance.
3. Every point in the space is within some maximum distance of a lattice point [7].

Definition 5.5. The shortest vector problem (SVP) is, on input of an arbitrary lattice $\Lambda \in \mathbb{R}^n$ and a basis N, to determine a vector $x \in \Lambda$ with length $||x|| = \lambda(\Lambda) = \min\{||x|| : x \neq 0, x \in \Lambda\}$ [4].

Example 5.3. Fig. 8 illustrates a lattice structure and an SVP problem on the plane $\mathbb{R}^2$. Let $\Lambda = \{m(4,4)+n(7,5)|m,n \in \mathbb{Z}\} = \{(4m+7n, 4m+5n)|m,n \in \mathbb{Z}\}$. $\{b_1, b_2\} = \{(4,4),(7,5)\}$ forms a basis of the lattice Λ (though there are alternative representations like $\{b_1, b_2\} = \{(1,3),(3,1)\}$). Then the SVP problem on lattice Λ and basis N is to find $x \in \Lambda$ such that $||x|| = \sqrt{x_1^2 + x_2^2}$ is minimized for $x \in \Lambda$.

Notice that $x = (1,3) = 2b_1 - b_2 \in \Lambda$, which gives $||x|| = \sqrt{10}$. We are not sure if $\min\{x \neq 0 | x \in \Lambda\}$. After multiple experiments, we can eventually conclude that x, among some other vectors, indeed minimize $||x||$. Either the search or the proof of SVP problem becomes very difficult when the base vectors are strange (e.g. $\{b_1, b_2\} = \{(10+5\sqrt{3}, \pi + 0.7), (2\sqrt{2}, 2-\sqrt{2})\}$) or the dimensions gets higher. The solution to this problem takes a large number of experiments, and the process cannot be simplified even with quantum computers.

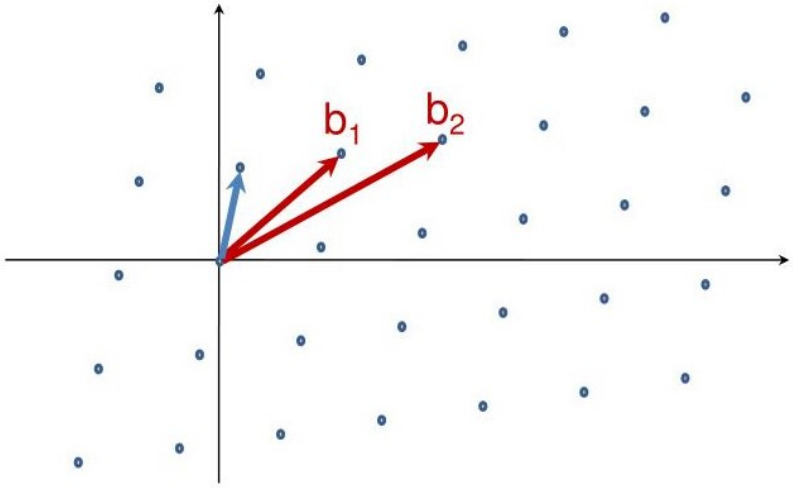

Figure 8: Lattice and SVP Problem. The red arrows are two basis vectors in $\mathbb{R}^2$, while the blue arrow is a vector that represents a potential solution to the SVP problem [7].

Previous studies have shown that the SVP problem is very difficult to solve [2, 3]. This can be attributed to the nature of lattices in number theory. Based on the hardness of the SVP problem, RLWE is defined on a ring of integer polynomials modulo both $f(x)$ and q.

Definition 5.6. $\mathbb{Z}[x]$ is the ring of integer polynomials formed from the set $S = \{\sum_{i=1}^{k} a_i x^i : a_i \in \mathbb{Z}, k > 0\}$ with the polynomial addition and multiplication.

Definition 5.7. For a given prime number q, $\mathbb{Z}_q[x]$ is the ring of integer polynomials formed from the set $S = \{\sum_{i=1}^{k} a_i x^i : a_i \in \mathbb{Z}_{\|}, k > 0\}$ with the polynomial addition and multiplication in $\mathbb{Z}_q$. Notice that by Fermat's Little Theorem, $x^{q-1} = 1$ in any $x \in \mathbb{Z}_q$, so the highest degree of polynomials in $\mathbb{Z}_q$ should be $q - 2$, and the coefficient of each term should be in $\{0, 1, 2, ...q - 1\}$.

Let $f(x) = x^{2^k} + 1 \in \mathbb{Z}[x]$, so $f(x)$ cannot be factored into any two non-degenerate rational polynomials. Let $R = \mathbb{Z}[x]/\langle f(x)\rangle$ be the ring of integer polynomials modulo $f(x)$, then elements of R include polynomials with degree no more than 2^k. Let $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ be the ring of integer polynomials modulo $f(x)$ and a large prime number q, which include polynomials with degree no more than 2^k and integer coefficients less than q [16]. R_q will be the main working space for the RLWE problem.

The RLWE problem can be intuitively described as follows: fix a certain error distribution over R_q that has a high density in "small" elements,[7] which means the elements with relatively small coefficients. Let $s = s(x) \in R_q$ be uniformly random, then the goal of RLWE is to distinguish s between arbitrarily many independent noise equations in the form $(a, b) = (a, as + e)$, where a is uniformly drawn from R_q and e is drawn from the distribution of "small" elements as an error term [16].

[7]Please refer to the original paper for the boundary of smallness [16]. Usually, these are the polynomials with most coefficients lying in $\{-1, 0, 1\}$.

A special case of RLWE. Fix the ring $R = \mathbb{Z}[x]/\langle f(x)\rangle$ for $f(x) = x^{2^k} + 1$.

1. Key Generation: Alice chooses a random element a uniformly from R_q and two random "small" elements $s, e \in R$ from the error distribution. Let $b = as + e$. Alice keep s as the secret key and announces (a, b) as public key.

2. Encryption and Incoming Message: To encrypt a n-bit message $m \in \{0,1\}^n$ where $n < 2^k$, Bob can map m to $z \in R$ using the 0s and 1s as coefficient of the polynomial.

   - Choose three "small" random elements from the error distribution r, e_1, e_2.
   - Compute
   $$u = ar + e_1 \text{ (mod q)},$$
   $$v = br + e_2 + \lfloor q/2 \rfloor z \text{ (mod q)}.$$
   - Send (u, v) to Alice as ciphertext.

3. Decryption: When Alice receives the message, Alice simply need to compute
   $$z' = v - us = \lfloor q/2 \rfloor z + er + e_2 - e_1 s.$$
   If all parameters are specified appropriately (s, r, e, e_1, e_2 are all small enough so as to be negligible), Alice can recover the bits of z from z' by rounding each coefficient to either 0 if it's closer to 0 than $\lfloor q/2 \rfloor$ or 1 if its closer to $\lfloor q/2 \rfloor$ than 0.

Example 5.4. Numerical Example of RLWE. Let $k = 3$ and $q = 17$, then $f(x) = x^8 + 1$.

1. Key Generation: Alice chooses a random element from R_q $a = 7x^5 + 4x^2 + 2$ and two random small elements $s = x^3 - 1$ and $e = x + 1$. Alice then keeps $s = x^2 - 1$ as private key and announces $(a, b) = (7x^5 + 4x^2 + 2, 7x^8 - 3x^5 + 2x^3 - 4x^2 + x - 1) = (7x^5 + 4x^2 + 2, -3x^5 + 2x^3 - 4x^2 + x - 8)$ because the polynomial ring operates under the modulus of $f(x)$ where $x^8 + 1 = 0$.

2. Encryption and Incoming Message: Bob now wants to encrypt a message $m = 00101010$ in the binary system. The corresponding polynomial is $z = x^5 + x^3 + x$. Then Bob randomly draws three small polynomials $r = x^2 - 1$, $e_1 = x^3 - x + 1$, and $e_2 = x - 1$. Then
   $$u = ar + e_1 = 7x^7 - 7x^5 + 4x^4 + x^3 - 2x^2 - x - 1,$$
   $$v = br + e_2 + \lfloor q/2 \rfloor z = -3x^7 + 13x^5 - 4x^4 + 7x^3 - 4x^2 + 8x + 7$$

3. Decryption: Alice can compute that
   $$z' = v - us = \lfloor q/2 \rfloor z + er + e_2 - e_1 s = -x^6 + 8x^5 + x^4 + 10x^3 + x^2 + 8x - 3.$$
   Alice can then round each coefficient to either 0 if it's closer to 0 than 8 (smaller than 4) or 1 if closer to 8 than 0 (at least 4). After the rounding, Alice can recover the original $z = x^5 + x^3 + x$.

   Notice that if a malicious party wants to attack the system without knowing the private key s, one must solve the shortest vector problem on the grid formed by u. The hardness of SVP ensures that one cannot easily discover the true value of s and z without knowing the secret key.

The SVP-based hardness guarantees the security of RLWE even in the existence of quantum computation. If an attacker attempts to solve z without knowing s, RLWE will become an SVP problem on the finite ring of polynomials with some ideal lattice. This is fundamentally different from most encryption schemes mentioned before that root their mathematical foundation in integer factorization, which can be solved using quantum computers. In contrast, studies have shown that it is difficult to solve SVP problems even for quantum computers. Therefore, the hardness of RLWE is guaranteed even in the existence of quantum computers.

Example 5.5. We can apply RLWE encryption in keeping line items of individual private financial accounts secret while allowing aggregation to a shared public account. Suppose there are many participants, including A, B, and C. Each is holding cash as a line item. A sufficiently large number of participants is needed to ensure that aggregate cash holding reveals nothing about anyone's individual holding.

1. Let $b = as + e$ be denote a given polynomial, where a is a known set of polynomial, s is the secret small polynomial, and e is a small additive error. The cipher text is a two-dimensional vector $(a, as + \delta m + e)$.
2. Fix the degree of the polynomial. For each participant i, convert his/her number to coefficients of the polynomial using binary translator, and denote this m_i.
3. A send a translated message m_A to B, and B add B's translated message m_B and sends it to C. So, at the end we have added up all three. For MPC with ciphertext $ct = (a, a \cdot s + \delta \cdot m + e)$, let the public and private secret keys be $(pk_i, sk_i) = (a, a \cdot s_i + e_i)$ respectively, for first and second argument.
4. A sends cipher text ct to B with his message m_A. B adds a layer of encryption on ct_1' to make it ct_1'' by adding the second component, namely that $ct_1'' = (a, a \cdot s_1 + a \cdot s_2 + \delta \cdot m_1 + e_1 + e_2)$ B then pass it along to C who also adds his layer of encryption, and we then have $ct_1^{final} = (a, a \cdot s + \delta \cdot m_A + e)$ Where s is the sum of the s_i and e is the sum of the e_i.

   B does the same with his message m_B, passes it along to A and C who add their encryption, so that B's final encrypted message is $ct_2^{final} = (a, a \cdot s + \delta \cdot m_B + e)$

   The same with C, whose final encrypted message m_3 as : $ct_3^{final} = (a, a \cdot s + \delta \cdot m_1 + e)$ then we can just add ct_1^{final} with ct_2^{final} with ct_3^{final} : $(a, a \cdot s + \delta \cdot (m_A + m_B + m_C) + 3 \cdot e)$ which we can call $(m_A + m_B + m_C) = m$

5.4 Distributed Decryption

In both HE and MPC, data must be encrypted beforehand, usually in different ways. For instance, encryption for HE can be just ElGamal or Paillier Encryption (or more sophisticated versions of them). Encryption for MPC involves Shamir secret sharing to allow for "common public/private keys" to be built (e.g., summing all s_i in the polynomial example returns a common result s), which need not take place in HE schemes. So the different ways to encrypt data are inherent to the type of schemes/computation we want to perform

- HE-specific encryption for HE, MPC-specific encryption for MPC, and HE-MPC-specific encryption for HE-MPC

Overall, with the introduction of HE and MPC, what private information to reveal (or not) even as computation is run on it, and who can decide on it, become a new choice element of mechanism design. It takes thoughts to understand what is needed exactly and what is the best available technology.

References

[1] Inveniam, 07 2022.

[2] M. Ajtai. Generating hard instances of lattice problems (extended abstract). *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, 1996.

[3] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, 1998.

[4] I. Blanco-Chacón. Ring learning with errors: a crossroads between post-quantum cryptography, machine learning and number theory. *Irish Mathematical Society Bulletin*, 0086:17–46, 2020.

[5] Fabrice Boudot and Pierrick Gaudry. [cado-nfs-discuss] 795-bit factoring and discrete logarithms, 12 2019.

[6] J. P. Buhler, H. W. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. *Lecture Notes in Mathematics*, pages 50–94, 1993.

[7] Wikipedia Contributors. Lattice (group), 04 2022.

[8] Craig Gentry. Fully homomorphic encryption using ideal lattices. *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, 2009.

[9] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *Advances in Cryptology – CRYPTO 2013*, pages 75–92, 2013.

[10] Oded Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *STOC*, 1987.

[11] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38:690–728, 07 1991.

[12] Emily Grumbling and Mark Horowitz, editors. *Quantum Computing*. National Academies Press, 03 2019.

[13] Tim Harford. What tally sticks tell us about how money works, 07 2017.

[14] Jonathan Katz. *Introduction To Modern Cryptography.* CRC Press, 2019.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 07 1982.

[16] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60:1–35, 11 2013.

[17] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology — EUROCRYPT '99*, 1592:223–238, 04 1999.

[18] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 11 1979.

[19] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. *Advances in Cryptology – EUROCRYPT 2010*, pages 24–43, 2010.

[20] Kaveh Waddell. The long and winding history of encryption, 01 2016.

[21] Sharon Weadick. Cuneiform tablets: the genesis of documentation, 2012.