Development of Cryptography since Shannon*

Funda Özdemir† & Çetin Kaya Koç‡

Abstract

This paper presents the development of cryptography since Shannon's seminal paper "Communication Theory of Secrecy Systems" in 1949.

1 Introduction

Shannon's work [80] was a turning point, and marked the closure of classical cryptography and the beginning of modern cryptography. Indeed, starting from 1949, cryptography theory and applications have gone through significant progress, certainly much faster than the previous several centuries.

Humans' interest in cryptography is as old as the invention of writing. While we have good information and insights about cryptographic methods in the past 2 millennia, we surmise that older algorithms like their more recent successors were all letter- or word-based "codes" in which one substitutes each letter or word with the corresponding code-letter or code-word found in the codebook, according to a selection algorithm. Sender and receiver must share the codebook to "encode" or "decode" the messages. On the other hand, based on the frequencies of the code-letters, cryptanalysts attempted to make sense of the encoded (encrypted) message without having access to the codebook. The competition between the cryptographers and cryptanalysts have been real and fierce, especially when applications involved state or military data. We do get into history of classical cryptography due to the lack of space in this paper, and recommend a new and well-written book for interested readers, "History of Cryptography and Cryptanalysis" [33].

The interplay between cryptographic theory and applications opened up new areas of applications and also motivated the practitioners of the theory to develop new methods and algorithms. There is much to write about cryptography, but given the space, we will limit our focus to secret-key cryptography, public-key cryptography, post-quantum cryptography, and homomorphic encryption, which are also section headers in this paper.

*This work was supported by TÜBİTAK Projects 2232-118C332 and 1001-121F348.

†İstinye University

‡Iğdır University, NUAA, UCSB

The development of **secret-key cryptography** started soon after Shannon's insights how one builds complex, usable and efficient secret-key cryptographic algorithms. Horst Feistel's LUCIFER [38, 37, 54] at IBM, followed up by US NIST Data Encryption Standard [64], and a plethora of academic, commercial, cyberpunk algorithms and standards, to finally AES [65] which is another US standart. These algorithms were all built upon Shannon's ideas. The driving factor comes from banking application, that is for our need to relay confidential financial information. This is an ongoing work, and the academic, industrial and government bodies will continue to develop newer secret-key cryptographic algorithms.

A second revolution in cryptography happened somewhere between 1976 and 1978, interestingly right around time when the secret-key cryptographic algorithm DES was standardized by the US. While trying to address the problem of how to share secret keys between two or more parties, researchers at Stanford and MIT invented **public-key cryptography**. The Diffie-Hellman key exchange algorithm [32] and the RSA public-key cryptographic algorithm [76] have indeed changed cryptography as significantly as Shannon's contribution. In the ensuing years, practical solutions to key exchange between parties, digital signatures, and public-key encryption methods allowed us to build trust architectures into Internet-connected servers, desktop and and mobile computers. The public-key cryptography provided techniques, mechanisms, and tools for private and authenticated communication, and for performing secure and authenticated transactions over the Internet as well as other open networks. This infrastructure was needed to carry over the legal and contractual certainty from our paper-based offices to our virtual offices existing in the cyberspace. The timing of the invention of public-key cryptography was near perfect!

The first two decades of the 21st Century presented two challenges for cryptographers. The first and formidable challenge was that quantum computers were becoming feasible. Experimental quantum computers developed or sponsored by two major companies (IBM and Microsoft) and several research institutes in major research universities are available for researchers to test their quantum algorithms [88]. It has already been established by Peter Shor [81, 83] that a quantum computer (if available) with several thousands quantum bits (qubits) can be programmed to break public-key almost all of the public-key cryptographic algorithms. Therefore academic and governmental efforts started to design public-key cryptographic algorithms that would be resistant to quantum computing attacks, which gave birth to **post-quantum cryptography**. In April 2015, the US NIST held a "Workshop on Cybersecurity in a Post-Quantum World" to discuss cryptographic algorithms for public-key based key agreement and digital signatures that are not susceptible to cryptanalysis by quantum algorithms. In this direction, NIST recently launched the so-called "Post- Quantum Cryptography Standardization" process, a multiyear effort aimed at selecting the next-generation of quantum-resistant public-key cryptographic algorithms for standardization.

Another formidable challenge has been the desire to compute with the encrypted text without decrypting, which is termed as **homomorphic encryption**. The potential applications of homomorphic encryption were recognized and appreciated almost about the same time as the first public-key cryptographic algorithm RSA was invented, which is multiplicatively homomorphic. The ensuing 30 years have brought on several additively or multiplicatively homomorphic encryption functions with increasing algorithmic complexity. In 2009, Craig Gentry and several other authors later on proposed fully (both additively and multiplicatively) homomorphic encryption algorithms and addressed issues related to their formulation, arithmetic, and security. We now have a variety of fully homomorphic encryption algorithms that can be applied to various private computation problems in healthcare, finance and national security.

We start with Shannon's ideas in Section 2, and show how Feistel used them to create his seminal cryptographic algorithm LUCIFER. In this paper, we focus only on the DES and AES, the US standardized algorithms since the first was the chosen algorithm for applications ranging from banking to Internet for 2 decades, while latter has been in use as its replacement for more than 2 decades, going into the 3rd.

The Section 3 covers public-key cryptography which tackles key management, public-key encryption and digital signatures, providing authentication and nonrepudiation properties for the exchanged data and communicating parties. We will cover the basic ideas and algorithms of public-key cryptography briefly, and move into post-quantum cryptography in the Section 4. The advent of quantum computing is bound to change public-key cryptography, and the changes have already started. We will give an overview of post-quantum cryptography in this section.

Finally the Section 5 covers homomorphic encryption which will bring a kind of luxury to data science such that we can keep *everything encrypted* and still accomplish the necessary computations for maintaining the data as well as inferring from it. This will indeed be a revolution, and will bring nearly-absolute security for our precious data.

2 Secret-Key Cryptography

Claude Shannon wrote his "secrecy" paper in 1945, however it was declassified and published only in 1949 [80]. Shannon suggested that cryptanalysis using statistical methods might be defeated by the mixing or iteration of non-commutative operations. Shannon refers to these operations as confusion (or substitution) and diffusion (transposition). His ideas were used in the design of the top 3 encryption algorithms in the following 5 decades. The LUCIFER [38, 37] and DES S-boxes and P-boxes [63, 64] are Feistel's interpretation of Shannon's confusion and diffusion. Similarly, AES or Rinjdael also uses many rounds to mix confusion and diffusion [65, 29].

2.1 LUCIFER

Horst Feistel changed cryptology. Historically (pre-Shannon days), encryption algorithms have been dictated by the hardware available. The pinwheels of the 1934 Hagelin machine, the rotors of the 1918 German Enigma machine, the telephone dial switches used in the 1937 Japanese Purple machine and nonlinear versions of the linear feedback shift register were subsequently based on the 1947 transistor breakthrough [54]. Horst knew about some of these, but he realized that hardware was a limitation; a program could directly implement encryption and so he started in the reverse direction. When asked by about the idea behind his algoritm LUCIFER, Horst said "The Shannon secrecy paper [10] reveals all" [54]. He understood the power of Shannon's idea, followed the master's advice, leading to LUCIFER and DES.

LUCIFER was the very first encryption algorithm designed for software. In fact, LUCIFER is the name for the software implementation of the block cipher described in the 1971 patent designated by Feistel. Coded in the APL language, LUCIFER originally was stored in the APL directory (folder) with the intended name DEMONSTRATION. Early versions of APL limited the character length of a file name, and a colleague suggested the name DEMON, modified by Horst to LUCIFER. Shannon's secrecy paper alone may have provided the real inspiration for a person of Horst's creative genius. Still, Horst Feistel went in his own cryptographic direction, providing a fresh point of view. A modified LUCIFER became the Data Encryption Standard (DES), affirmed in 1976 as a Federal Information Processing Standard (FIPS 46-1). AES became the new FIPS, replacing DES in 2000 as a standard. The Triple DES-variant (3DES) continues to be used for authenticated transactions in banking [52, 53].

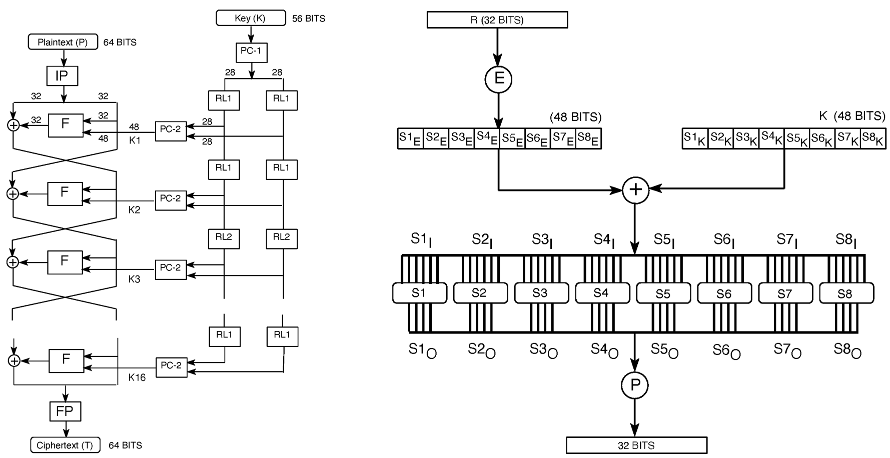
2.2 DES

The Data Encryption Standard is a US standard that provided confidentiality for financial transactions from 1970s till to the end of 1990s. It was developed by IBM, based on ideas of Horst Feistel, and submitted to the National Bureau of Standards (the precursor of the National Institute of Standards and Technology) following an invitation to propose a candidate for the protection of sensitive, unclassified electronic data. After consulting with the National Security Agency (NSA), the NBS eventually selected a slightly modified version, which was published as an official Federal Information Processing Standard (FIPS) for the United States in 1977, with the number FIPS 46. It quickly became an international standard, and enjoyed widespread deployment.

However, there was also some controversy about the DES for several years. The design philosophy of certain elements (S-boxes) was never explained (classified), and its key length was unnaturally short (56 bits) while it could have been 64 bits. The NSA involvement was found suspicious by some researchers, especially on its key length [48]. There were also conspiracy theories about the DES having a "backdoor" for easy decryption (which was never proven to-day). Academic community approached the DES with caution, however, in the end, it significantly contributed to the development of modern cryptography for our communication and computing systems

The fundamental building block in DES is a substitution followed by a permutation on the text, based on the key. This is called a round function. DES has 16 rounds.

Figure 1: The 16 rounds of DES and the round function.



A cryptographic algorithm should be a good pseudorandom generator in order to foil key clustering attacks. DES was designed so that all distributions are as uniform as possible. For example, changing 1 bit of the plaintext or the key causesthe ciphertext to change in approximately 32 of its 64 bits in a seemingly unpredictable and random manner.

However, Biham and Shamir [8] observed that with a fixed key, the differential behavior of DES does not exhibit pseudorandomness. If we fix the XOR of two plaintexts P and P^* at P' then T' (which is equal to $T \oplus T^*$) is not uniformly distributed. In contrast, the XOR of two uniformly distributed random numbers would itself be uniformly distributed. The attack (called differential cryptanalysis) based the nonrandom behavior of the DES still could not break DES, primarily due to the fact that 16 rounds made the tracing of the differences of the plaintexts and ciphertexts very difficult. Biham and Shamir showed that DES reduced to 6 rounds can be broken by a chosen plaintext attack in less 0.3 seconds on a PC using 240 ciphertexts; the known plaintext version requires 2^{36} ciphertexts. On the other hand, DES reduced to 8 rounds can be broken by a chosen plaintext attack in less than 2 minutes on a PC by analyzing about 2^{14} ciphertexts; the known plaintext attack needs about 2^{38} ciphertexts. Yet, the full DES (16 rounds) can only be broken by analyzing 2^{36} ciphertexts from a larger pool of 2^{47} chosen plaintexts using 2^{37} time. The differential cryptanalysis confirmed the importance of the number of rounds and the method by which the S-boxes are constructed.

On the other hand, the variations on DES turn out to be easier to cryptanalyze than the original DES. Most importantly, certain changes in the structure of DES hase catastrophic results, as shown in Table 1.

Table 1: Effectiveness of differential cryptanalysis.

Modified Operation	Chosen Plaintexts
Full DES (No change)	2^{47}
Random P permutation	2^{47}
Identity P permutation	2^{19}
Order of S-boxes	2^{38}
Change XOR by Addition	2^{31}
Random S-boxes	2^{21}
Random Permutation	$2^{44} \sim 2^{48}$
One Entry S-box	2^{33}
Uniform S-boxes	2^{26}
Eliminate Expansion E	2^{26}
Order of E and subkey XOR	2^{44}

Feistel ciphers take an important part in secret key cryptography from both theoretical and practical point of view. After DES, new schemes have been published, like GOST in Russia, IDEA, and RC-6 in the United States. From a practical point of view, Feistel ciphers had their days of glory with the DES algorithm and its variants (3DES with two or three keys, XDES, etc.) that were the most widely used secret key algorithms around the world between 1977 and 2000. After 2000, the AES algorithm, which is not a direct Feistel cipher, but still based on the Shannon's ideas (confusion and diffusion) which were very effectively utilized by Feistel, has become the standard for secret key encryption.

2.3 AES

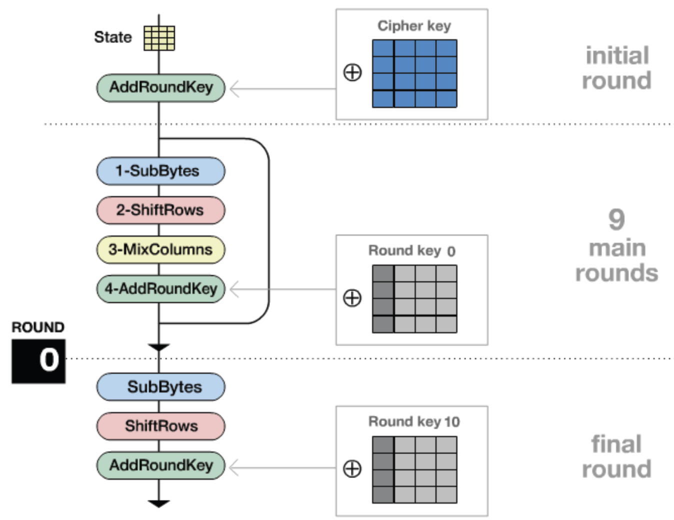
AES is a key-alternating block cipher, with plaintext encrypted in blocks of 128 bits. The key sizes in AES can be 128, 192, or 256 bits. It is an iterated block cipher because a fixed encryption process, usually called a round, is applied a number of times to the block of bits. Finally, we mean by key-alternating that the cipher key is XORed to the state (the running version of the block of input bits) alternately with the application of the round transformation. The original Rijndael design allows for any choice of block length and key size between 128 and 256 in multiples of 32 bits. In this sense, Rijndael is a superset of AES; the two are not identical, but the difference is only in these configurations initially put into Rijndael but not used in AES [29].

The state matrix of AES is formed from the input data as a 4×4, 4×6, and 4×8 matrices, for 128, 192 or 256 bits, respectively. Given the 128-bit data $(A_0 A_1 A_2 \cdots A_{14} A_{15})$ such that each of A_i is 8 bits (1 byte), the 4×4 state matrix is formed as

$$\begin{bmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{bmatrix}$$

The 8-bit (1-byte) binary data is usually represented in hexadecimal, such as $(\texttt{a3}) = (1010\ 0011)$. While the 8-bit input data block is a binary number in its most generic form, the Rijndael/AES treats each one of the bytes in the state matrix as elements of the Galois field $\mathrm{GF}(2^8)$. The irreducible polynomial of the field $\mathrm{GF}(2^8)$ is $p(x) = x^8 + x^4 + x^3 + x + 1$. A field element $a(x) \in \mathrm{GF}(2^8)$ is represented using a polynomial of degree at most 7 with coefficients $a_i \in \mathrm{GF}(2)$ such that $\sum_{i=0}^{7} a_i x^i = a_7x^7 + x_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. For example, $(\texttt{a3}) = (1010\ 0011) = x^7 + x^5 + x + 1$. AES has 4 sub-rounds, named as AddRoundKey, SubBytes, ShiftRows, MixColumn. Except the ShiftRows operation, all of them involve finite field addition, inversion, and linear and non-linear operations in the field $\mathrm{GF}(2^8)$.

Figure 2: The 10 rounds of AES and the round function.



Here we describe only the MixColumn operation which multiplies a fixed 4×4 matrix with every 4×1 column vector of the 4×4 state matrix. The MixColumn matrix M in hex and polynomial representation is

$$\begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix}$$

Given a 4×1 column vector u of the state matrix, such that each vector entry is an element of the finite field $\mathrm{GF}(2^8)$, we perform a matrix-vector multiplication operation

Mu using field multiplications and additions to compute the new column vector of the state matrix. We give an example of the MixColumn operation example below:

$$\begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} \texttt{d4} \\ \texttt{bf} \\ \texttt{5d} \\ \texttt{30} \end{bmatrix} = \begin{bmatrix} \texttt{04} \\ \texttt{66} \\ \texttt{81} \\ \texttt{e5} \end{bmatrix}$$

We show the computation of the first entry of the resulting vector, in other words, the computation of

$$[\texttt{02 03 01 01}] \begin{bmatrix} \texttt{d4} \\ \texttt{bf} \\ \texttt{5d} \\ \texttt{30} \end{bmatrix} = (\texttt{04})$$

By representing the column vector $[\texttt{d4 bf 5d 30}]^T$ as a polynomial vector, we can write this MixColumn operation in polynomial representation as

$$[x \quad x+1 \quad 1 \quad 1] \begin{bmatrix} x^7+x^6+x^4+x^2 \\ x^7+x^5+x^4+x^3+x^2+x+1 \\ x^6+x^4+x^3+x^2+1 \\ x^5+x^4 \end{bmatrix}$$

To compute the inner product, we perform polynomial multiplications, additions, and reductions modulo $p(x)$ whenever necessary:

$$\begin{gathered} x \cdot (x^7+x^6+x^4+x^2) + \\ (x+1) \cdot (x^7+x^5+x^4+x^3+x^2+x+1) + \\ 1 \cdot (x^6+x^4+x^3+x^2+1) + \\ 1 \cdot (x^5+x^4) \end{gathered}$$

The first product $x \cdot (x^7+x^6+x^4+x^2)$ needs to be computed and reduced if necessary. Here, we need reduction modulo the irreducible polynomial $p(x)$ since the resulting polynomial would be of degree 8

$$\begin{aligned} x \cdot (x^7+x^6+x^4+x^2) &= x^8+x^7+x^5+x^3 \\ &= (x^4+x^3+x+1)+x^7+x^5+x^3 \\ &= x^7+x^5+x^4+x+1 \\ &= (1011\ 0011) = (\texttt{b3}) \end{aligned}$$

After the polynomial multiplication, we reduced the highest degree term (which is x^8) by substituting it with x^4+x^3+x+1, which is the lower half of the irreducible polynomial $p(x) = x^8+x^4+x^3+x+1$.

The second product, after the multiplication gives

$$(x+1) \cdot (x^7+x^5+x^4+x^3+x^2+x+1) = x^8+x^7+x^6+1$$

We also need to reduce it modulo $p(x)$ since its degree is larger than 8. By substituting x^8 with $x^4 + x^3 + x + 1$, we obtain

$$(x^4 + x^3 + x + 1) + x^7 + x^6 + 1 \;=\; x^7 + x^6 + x^4 + x^3 + x$$

which is equal to (1101 1010) = (`da`). However, we do not need reductions for the third and fourth products:

$$\begin{aligned} 1 \cdot (x^6 + x^4 + x^3 + x^2 + 1) &= x^6 + x^4 + x^3 + x^2 + 1 &= (\texttt{5d}) \\ 1 \cdot (x^5 + x^4) &= x^5 + x^4 &= (\texttt{30}) \end{aligned}$$

Finally, adding all 4 resulting polynomials, we obtain the top entry as

(`02`) · (`d4`) = (`b3`)	$x^7 + x^5 + x^4 + x + 1$
(`03`) · (`bf`) = (`da`)	$x^7 + x^6 + x^4 + x^3 + x$
(`01`) · (`5d`) = (`5d`)	$x^6 + x^4 + x^3 + x^2 + 1$
(`01`) · (`30`) = (`30`)	$x^5 + x^4$
(`04`)	x^2

The remaining 3 entries are obtained by repeating the above operations by multiplying the second, third, and fourth rows of the fixed MixColumn matrix with the same state column.

3 Public-Key Cryptography

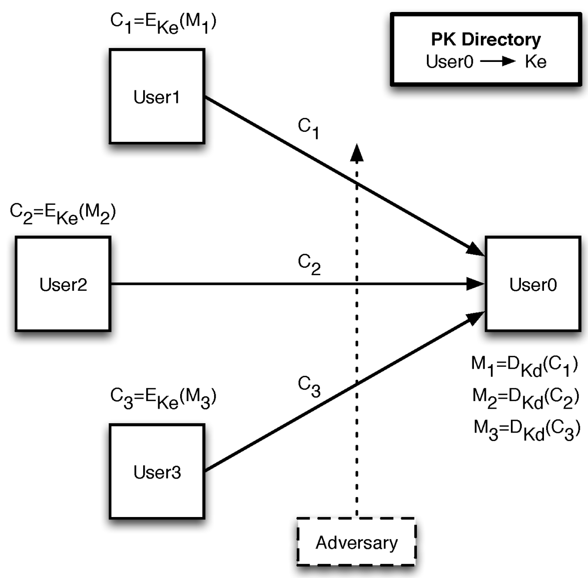
In public-key cryptography, the encryption $E_{K_e}(M)$ and the decryption $D_{K_d}(C)$ functions are inverses of one another, and use different keys

$$C = E_{K_e}(M) \quad \text{and} \quad M = D_{K_d}(C) \ .$$

These processes are asymmetric, and the keys are not equal, i.e., $K_e \neq K_d$. The naming conventions are

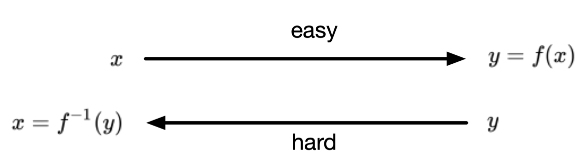
- K_e is the public key, which is expected to be known by anyone;
- K_d is the private key, known only to the user;
- K_e may be easily deduced from K_d;
- However, K_d is not easily deduced from K_e.

Figure 3: The general concept of public-key encryption.



The User publishes his own public key K_e and anyone can obtain it and can encrypt a message M, and send the resulting ciphertext to the User $C = E_{K_e}(M)$. The private key K_d is known only to the User and only the User can decrypt the ciphertext to get the message $M = D_{K_d}(C)$. The adversary may be able to block the ciphertext, but it cannot decrypt. A public-key cryptographic algorithm is based on a function $y = f(x)$ such that given x, computing y is easy; while, given y, computing x is hard:

Figure 4: The general concept of a one-way function.



Such functions are called **one-way functions**. In order to decide which function is hard according to this criteria, we can resort to the theory of complexity. However, a one-way function is difficult for anyone to invert, including the receiver of the encrypted text. Instead, we need a function that is easy to invert for the legitimate receiver of the encrypted message, but hard for everyone else. Such functions are called **one-way trapdoor functions**.

In order to build a public-key encryption algorithm, we need a one-way trapdoor function. As this fact is understood around 1975-1976, researchers at Stanford and MIT were looking for such special functions which are either based on the known one-way functions or some other "unknown" constructions. Since then the following one-way functions have been identified, allowing us to build public-key encryption algorithms with the help of trapdoor mechanisms:

- Discrete Logarithm:
  Given p, g, and x, computing y in $y = g^x \pmod p$ is EASY
  Given p, g, y, computing x in $y = g^x \pmod p$ is HARD

- Factoring:
  Given p and q, computing n in $n = p \cdot q$ is EASY
  Given n, computing p or q in $n = p \cdot q$ is HARD

- Discrete Square Root:
  Given x and y, computing y in $y = x^2 \pmod n$ is EASY
  Given y and n, computing x in $y = x^2 \pmod n$ is HARD

- Discrete eth Root:
  Given x, n and e, computing y in $y = x^e \pmod n$ is EASY
  Given y, n and e, computing x in $y = x^e \pmod n$ is HARD

3.1 The Diffie-Hellman Key Exchange Method

The first one-way function in this list gave birth to the **Diffie-Hellman Key Exchange Method**, whose trapdoor mechanism is based on the commutativity of exponentiation $(g^a)^b = (g^b)^a$. It was invented by Martin Hellman and Whitfield Diffie who published their paper "New Directions in Cryptography" in 1976 [32], introducing a radically new method of distributing cryptographic keys, and thus solving one of the fundamental problems of cryptography.

- A and B agree on a prime p and a primitive element g of $\mathcal{Z}_p^*$. This is accomplished in public: p and g are known to the adversary

- A selects $a \in \mathcal{Z}_p^*$, computes $r = g^a \pmod p$, and sends r to B

- B selects $b \in \mathcal{Z}_p^*$, computes $s = g^b \pmod p$, and sends s to A

- A (having received s) computes $K = s^a \pmod p$

- B (having received r) computes $K = r^b \pmod p$

- These two quantities are equal since

$$K = r^a = (g^a)^b = g^{ab} \pmod p \ ,$$
$$K = s^b = (g^b)^a = g^{ab} \pmod p \ .$$

At the end of these computation and communication steps, the parties A and B have the key value K, which is known to them but computing K is hard by anyone who sees and records all communicated values. The difficulty of computing the key K depends on the Discrete Logarithm Problem, whose general definition is given as: The computation of $x \in \mathcal{Z}_p^*$ in $y = g^x \pmod{p}$, given p, g, and y.

For example, given $p = 23$ and $g = 5$, how can we find x such that $7 = 5^x \pmod{23}$? The answer in this case easy $x = 19$ since we can find it by trying all possible values in $\mathcal{Z}_{23}^* = \{1, 2, \ldots, 22\}$. However, the difficulty of computing the discrete logarithm for a larger p will significantly higher; consider $p = 158(2^{800} + 25) + 1 =$

1053546280395016975304616582933958731948871814925913489342608
7342587178835751858673003862877377055779373829258737624519904
5043066135085968269741025626827114728303489756321430023716636
9174066615907176472549470083113107138189921280884003892629359

and $g = 17$, and the computation of $x \in \mathcal{Z}_p^*$ such that $2 = 17^x \pmod{p}$. Such x exists since 17 is a primitive root of p, however, the number of trials to find it will require insurmountable time and energy.

If the Discrete Logarithm Problem is difficult in a group (such as $\mathcal{Z}_p^*$), we can use it to implement not only the Diffie-Hellman key exchange method, but also several other public-key cryptographic algorithms, such as the ElGamal public-key encryption method and the Digital Signature Algorithm. As we described, we can resort to the exhaustive search of the unknown value by trying all possible values of $x \in \mathcal{Z}_p^*$ iteratively.

```
z = g
for i = 2 to p − 1
    z = g · z (mod p)
    if y = z
        return x = i
```

This algorithm requires $p - 2$ multiplications. However, it is an exponential algorithm in terms of the input size, which is the number of bits in the prime p. Since, the multiplications of two k-bit operands are of order $O(k^2)$, the search complexity is exponential in k, as $O(pk^2) = O(2^k k^2)$. There are better algorithms, such as Shanks Algorithm, Pollard Rho Algorithm, Pohlig-Hellman Algorithm, and the Index Calculus Algorithm; the first three algorithms are still of exponential complexity. The analysis is of the Index Calculus Algorithm is more complicated, and is estimated to be

$$O\left(e^{c \cdot (\log p)^{1/3} (\log \log p)^{2/3}}\right) .$$

This time complexity is sub-exponential since it is faster than exponential (in $\log p$) but slower than polynomial. Therefore, the Discrete Logarithm Problem remains to be a hard problem on a digital computer, making the Diffie-Hellman key exchange method a strong public-key cryptographic algorithm. Currently much of wireless communication and internet security depends on it.

3.2 The RSA Algorithm

The second important algorithm in the search for one-way trapdoor functions came from the 3 MIT professors, Ronald Rivest, Adi Shamir, and Leonard Adleman in the Summer and Fall of 1976. Their paper was published in 1978 [76], and MIT patented the method 1983 (which ended in 2000). The Rivest-Shamir-Adleman Algorithm or briefly as the **RSA Algorithm** constructs public and private keys for the User as follows:

- The User generates 2 large, about same size random primes: p and q
- The modulus n is the product of these two primes: $n = p \cdot q$
- Euler's totient function of n is given by $\phi(n) = (p-1) \cdot (q-1)$
- The User selects e as $1 < e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$ and computes $d = e^{-1} \pmod{\phi(n)}$ using the extended Euclidean algorithm.
- The **public key**: The modulus n and the public exponent e.
- The **private key**: The private exponent d, the primes p and q, and $\phi(n) = (p-1) \cdot (q-1)$

Once the keys are available, the encryption and decryption operations are performed by computing

$$\begin{aligned} C &= M^e \pmod{n} , \\ M &= C^d \pmod{n} , \end{aligned}$$

where M, C are the plaintext and ciphertext such that $0 \leq M, C < n$.

The security of the RSA Algorithm depends on the discrete eth root problem, i.e., given y, n and e, computing x in $y = x^e \pmod{n}$ is known to be a hard problem. One we can attempt to break the RSA algorithm in several ways:

- Compute eth Root of $M^e \pmod{n}$ and obtain M
- Factor $n = pq$, compute $d = e^{-1} \bmod (p-1)(q-1)$
- Obtain $\phi(n)$ by some method, and compute $d = e^{-1} \bmod \phi(n)$

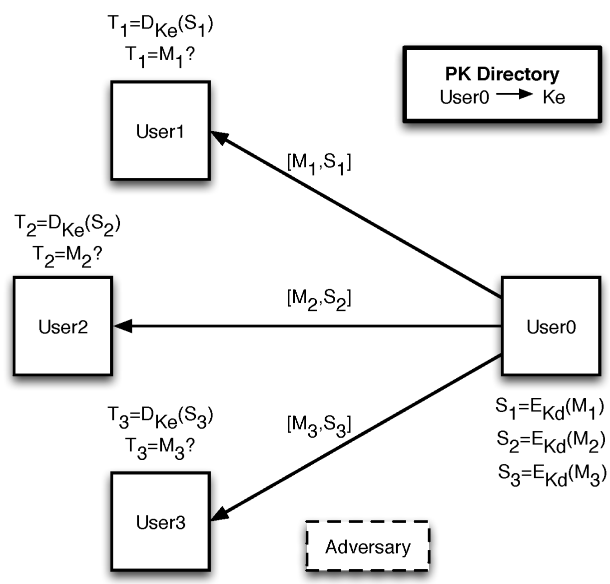
There is no known algorithm for computing discrete eth root mod n directly, and it is obvious factoring n indeed breaks the RSA encryption algorithm. However, "Breaking RSA" does not mean that we can factor n. There is no general proof for such a claim.

3.3 Digital Signatures

A digital signature or digital signature algorithm is a mathematical method for demonstrating the authenticity of a digital message or document. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication) such that he cannot deny sending it (non-repudiation) and that the message was not altered in transit (integrity). Digital signatures are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering. A public-key encryption algorithm is also a digital signature algorithm, the most notable example being the RSA algorithm.

Diffie and Hellman first described the concept of a digital signature scheme, and they conjectured that such methods exist. The RSA algorithm can be used as a public-key encryption method and as a digital signature algorithm

Figure 5: The general concept of digital signatures.



However, the plain RSA signatures have certain security problems. Other digital signature algorithms have been developed after the RSA: Lamport signatures, Merkle signatures, and Rabin signatures. Several more digital signature algorithms followed up, and are in use today: ElGamal, the Digital Signature Algorithm (DSA), the elliptic curve DSA (ECDSA).

The steps of the (plain) RSA signatures follows as:

- User A has an RSA public key (n, e) and private key (n, d)
- User A creates a message $M < n$, and **encrypts the message using the private key** to obtain the signature S as

$$S = M^d \pmod{n}$$

and sends the message (plaintext) and the signature $[M, S]$ to User B

- User B receives $[M, S]$, obtains User A's public key from the directory, and **decrypts the signature using the public key**:

$$T = S^e \pmod{n}$$

If $T = M$, then the User B decides that the signature S on the message M was created by User A

The plain RSA signatures have several problems to be used directly as a signature scheme in practice. First of all, the message length is limited to the modulus length, and longer messages cannot be directly signed. A biggest concern is that legitimate signatures can be used to create **forged** signatures. Consider that $[M, S]$ is a legitimate pair of message and signature, created by the owner of the public and private key pair such that $S = M^d \pmod{n}$ and $M = S^e \pmod{n}$. The pair $[M^2 \bmod n,\ S^2 \bmod n]$ also verifies:

$$(S^2)^e = (S^e)^2 = M^2 \pmod{n}$$

It appears that $[M^2 \bmod n,\ S^2 \bmod n]$ is a legitimate signature.

The solution of these problems with plain RSA signatures are avoided by employing a hash function $H(\cdot)$. Instead of encrypting M with the private key, we encrypt $H(M)$: the hash of M

$$h = H(M) \quad \rightarrow \quad S = h^d \pmod{n} \quad \rightarrow \quad [M, S]$$

The receiving party verifies the message and signature pair $[M, S]$ using

$$h = H(M) \quad \rightarrow \quad T = S^e \pmod{n} \quad \rightarrow \quad T \stackrel{?}{=} h$$

The cryptographic hash function $H(\cdot)$ is a publicly available function, and does not involve a secret key.

The Diffie-Hellman and RSA algorithms opened up new avenues for cryptography, particularly in internet security and wireless communication. The next 4 decades from 1980s to now have seen their proliferation and implementations. New methods and standards have been developed by NIST, as well as banking, communication, and internet communities. Public-key cryptography has become an household term, including the software packages and communication utilities, such as SSL and https.

4 Post-Quantum Cryptography

A quantum computer is based on the principles of quantum physics to perform computations. Classical computers use bits which is either 0 or 1 whereas quantum computers use quantum bits (called qubits) which can be either in a 1 or 0 quantum state or in a superposition of these states. A quantum computer is useful only if a quantum algorithm which solves a particular problem exists. It is important to make the distinction between Quantum Cryptography and Post-Quantum Cryptography (PQC). Quantum cryptography refers to quantum mechanical solutions to achieve communication secrecy or quantum key distribution. On the other hand, post-quantum cryptography aims to design and deploy algorithms that are secure against classical and quantum attacks. The security proofs of current widely used public-key cryptosystems (namely, RSA, Diffie-Hellman and ECC) are based on the hardness of integer factorization, discrete logarithm and elliptic curve discrete logarithm problems. Solving these problems using classical computation technology, even with hardware accelerators, takes hundreds of years. However, in 1994 Peter Shor [82] proposed an algorithm which solves these problems in polynomial time with a large-scale (a few thousands qubits) quantum computer. Although the key sizes for RSA and ECC used today are resistant against currently available small-scale quantum computers, the transition from public-key cryptography to post-quantum cryptography is needed in the near future, before any large-scale computers are built. Compared with public-key cryptography, symmetric cryptography is less affected by quantum attacks like Grover's algorithm which halves the security level.

In 2016, National Institute of Standards and Technology (NIST) released a report that announced a standardization plan for PQC and called for new quantum-resistant cryptographic algorithms for key encapsulation mechanisms (KEM), public-key encryption (PKE) and digital signatures. The evaluation criteria used throughout the NIST PQC standardization process are: 1) security, 2) cost and performance, and 3) algorithm and implementation properties.

Table 2: NIST 3rd round finalists
(Cb: Code-based; Lb: Lattice-based; Mv: Multivariate).

Scheme	**Type**	**Security Problem**
Classic McEliece	KEM, Cb	Decoding Goppa codes
CRYSTALS-KYBER	KEM, Lb	Module-LWE
NTRU	KEM, Lb	NTRU problem
SABER	KEM, Lb	Module-LWE
CRYSTALS-DILITHIUM	Sign, Lb	Module-LWE and Module-SIS
FALCON	Sign, Lb	Ring-SIS over NTRU lattices
Rainbow	Sign, Mv	Unbalanced Oil-Vinegar (UOV)

Among 82 received submissions by the November 2017 deadline, 69 of them were accepted into the first round of the standardization process in December 2017 as they met the submission requirements and minimum acceptability criteria. In January 2019, NIST moved 26 algorithms on to the second round of the process by consulting public feedback and internal reviews of candidates. 17 of them were KEMs/PKEs and

9 were digital signatures. Four of the 26 candidates were mergers of the first round algorithms. In July 2020, NIST announced the 15 candidates moved on to the third round of the standardization process. Of the 15 advancing candidates, seven have been selected as finalists and eight as alternate candidates. The alternate candidates are considered as potential candidates for future standardization, most likely after another round of evaluation.

Table 3: NIST third round alternate (Cb: Code-based; Lb: Lattice-based; Ib: Isogeny-based; Mv: Multivariate; Sym: Symmetric; Hb: Hash-based).

Scheme	Type	Security Problem
BIKE	KEM, Cb	Decoding QC-MDPC codes
HQC	KEM, Cb	Decisional QCSD with parity
FrodoKEM	KEM, Lb	LWE
NTRU Prime	KEM, Lb	NTRU
SIKE	KEM, Ib	Isogenies of elliptic curves
GeMSS	Sign, Mv	Hidden Field Equation (HFE)
Picnic	Sign, Sym	ZKP
SPHINCS+	Sign, Hb	security of the hash functions

There are five competing families of PQC algorithms: Code-based encryption, Isogeny-based encryption, Lattice-based encryption and signatures, Multivariate signatures, and Hash-based signatures.

4.1 Code-based Cryptography

Code-based cryptography uses error-correcting codes to build public-key encryption algorithms. The first code-based cryptosystem was proposed by Robert J. McEliece in 1978 [61]. Athough it is as old as RSA and has much stronger security history than RSA, due to large key sizes (large matrices as its public and private keys) it was not deployed in practical applications so far. However, today it is a strong candidate for PQC as it is resistant to attacks using Shor's algorithm.

The security of McEliece cryptosystem is based on the hardness of efficient decoding of a selected linear code. A decoding algorithm corrects errors which might have occurred during the transmission of a message over a communication channel. The classical decoding problem is to find the closest codeword $\mathbf{c} \in C$ to a given $\mathbf{y} \in \mathbb{F}_q^n$ assuming that there is a unique closest codeword. Berlekamp, McEliece and van Tilborg [5] showed that the general decoding problem for binary linear codes (over $\mathbb{F}_2$) is NP-complete. The original McEliece cryptosystem uses secretly generated random binary Goppa codes [46] which can be efficiently decoded with the algebraic decoding algorithm of Patterson [72]. Before presenting the algorithms of McEliece, we give a brief description of linear codes.

Let $\mathbb{F}_q$ be the finite field with q elements, where q is a prime power. A q-ary *linear code* of length n and dimension k is a k-dimensional vector subspace of $\mathbb{F}_q^n$. The elements of the code are called *codewords*. The *minimum distance* of the code is minimum *weight* of its nonzero codewords, where the weight of a codeword is the number of its nonzero coordinates. A linear code of length n, dimension k and minimum distance d is referred to as an $[n, k, d]$ code. A code of minimum distance $d \geq 2t + 1$ can correct up to t errors, i.e., C is a $\lfloor (d-1)/2 \rfloor$-error correcting code. A vector with more errors will likely get decoded incorrectly.

Since a linear code is a vector space, it admits a basis. Any codeword can be expressed as the linear combination of these basis vectors. A *generator matrix* G of an $[n, k, d]$ code C is a $k \times n$ matrix whose rows form a basis for C. Namely, $C = \{\mathbf{x}G : \mathbf{x} \in \mathbb{F}_q^k\}$. A *parity-check matrix* of C is an $(n-k) \times n$ matrix H such that $\{\mathbf{c} \in \mathbb{F}_q^n : H\mathbf{c}^T = 0\}$ where $\mathbf{c}^T$ is the transpose of c. If G has the form $[I_k|A]$, where I_k is the $k \times k$ identity matrix, then G is said to be in *systematic form*. The matrix $H = [A^T|I_{n-k}]$ is then a parity-check matrix for C. There are many generator matrices for a linear code, but there is a unique one in systematic form if it exists.

The algorithms of the original McEliece cryptosystem is as follows:

KeyGen: A t-error correcting binary $[n, k, d]$ linear code C with a generator matrix G' is picked. Further, a $k \times k$ random binary invertible matrix S and an $n \times n$ random binary permutation matrix P are chosen. Multiplying a vector by a permutation matrix, which has exactly one 1 in each row and each column and 0s elsewhere, permutes the entries of the vector. The public key is the pair $(G = SG'P, t)$ and the secret key is the triple (G', S, P) with an efficient decoding algorithm for C.

Enc: To encrypt a plaintext $\mathbf{m} \in \mathbb{F}_2^k$, a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight t is chosen and the ciphertext is computed as

$$\mathbf{c} = \mathbf{m}G + \mathbf{e}.$$

Dec: To decrypt a ciphertext $\mathbf{c} \in \mathbb{F}_2^n$ the legitimate receiver, who knows the matrices S, G', P and an efficient decoding algorithm for C, computes first

$$\mathbf{c}P^{-1} = \mathbf{m}GP^{-1} + \mathbf{e}P^{-1} = \mathbf{m}SG'PP^{-1} + \mathbf{e}P^{-1} = \mathbf{m}SG' + \mathbf{e}P^{-1}.$$

Note that the weight of $\mathbf{e}P^{-1}$ is t and since C is a t-error correcting code, the codeword $\mathbf{m}SG'$ is obtained. Using the decoding algorithm for C, the legitimate receiver recovers $\mathbf{m}S$ and then covers $\mathbf{m}$ by multipliying the inverse of S.

McEliece's original parameter sizes $n = 1024, k = 524, t = 50$ were designed for 2^{64} security which was broken in 2008 [6] with more than 2^{60} CPU cycles. Further, the new parameters were designed to minimize public-key size while achieving 80-bit, 128-bit, or 256-bit security against known attacks [6]. For a detailed security analysis of these codes, we refer the reader to [56].

4.2 Isogeny-based Cryptography

Isogeny-based cryptography uses maps between elliptic curves, called isogenies, to build public-key cryptography. The first such cryptosystem was discovered by Couveignes in 1997, but became better known in 2006 [28]. This system further developed by Rostovtsev and Stolbunov in [77] and Stolbunov in [87]. All these proposed systems are based on the difficulty of computing isogenies between ordinary elliptic curves. This hardness assumption is totally different from the hardness of the elliptic curve discrete logarithm problem for security. Therefore, Shor's quantum algorithm [82] cannot break these systems. However, Couveignes–Rostovtsev–Stolbunov (CRS) cryptosystem based on ordinary elliptic curves can be broken with a subexponential quantum attack [23]. In 2011, Jao and De Feo [51] used isogenies between supersingular elliptic curves rather than ordinary ones to construct a novel key-exchange protocol, called *Supersingular Isogeny Diffie-Hellman (SIDH)*. The extended version of SIDH was later released by Jao, De Feo and Plût with [39]. SIDH addressed both the performance and security drawbacks of CRS system. Thenceforth SIDH has attracted almost all research focus of isogeny-based cyrptography. SIDH resists against the attack proposed in [23] which exploits the commutativity of the endomorphism ring of an ordinary elliptic curve, since SIDH is constructed using the isogenies between supersingular elliptic curves whose endomorphism ring is non-commutative. Currently known fastest classical and quantum attacks against SIDH are both exponential. The SIDH algorithm also provides perfect forward secrecy which improves the long-term security of encrypted communications. Further, compromise of a key does not affect the security of past communication.

SIDH was used to build the key encapsulation mechanism SIKE (Supersingular Isogeny Key Encapsulation) [50] based on pseudo-random walks in supersingular isogeny graphs, that was submitted to the NIST standardization process on post-quantum cryptography and selected as a third round alternate candidate. One of the main advantages to SIKE is that it has the smallest public key sizes of all the encryption and KEM schemes, as well as very small ciphertext sizes. Among all the post-quantum cryptosystems, isogeny-based systems are the most recent and their security against quantum attacks needs to be further studied.

In 2018, Castryck et al. [16] presented CSIDH (Commutative Supersingular Isogeny Diffie-Hellman) which directly adopts the CRS cryptosystem based on ordinary elliptic curves to supersingular case. CSIDH is vulnerable to the attack proposed in [23]. On the performance side, CSIDH is much faster than CRS while it is slower than SIDH. CSIDH has not been submitted to NIST's standardization process since it was designed after the submission deadline date.

Table 4: Instantiations of Diffie-Hellman.

	DH	**ECDH**	**SIDH**
elements	integers g modulo prime	points P in curve group	curves $\mathcal{E}$ in isogeny class
secrets	exponents x	scalars k	isogenies ϕ
computations	$g, x \mapsto g^x$	$k, P \mapsto [k]P$	$\phi, \mathcal{E} \mapsto \phi(\mathcal{E})$
hard problem	given g, g^x find x	given P, $[k]P$ find k	given $\mathcal{E}$, $\phi(\mathcal{E})$ find ϕ

In this section, original SIDH key-exchange protocol will be explained. But first, we briefly introduce supersingular elliptic curves over finite fields and isogenies. For more details on elliptic curves and their use in cryptography, we refer the interested readers to [84, 40].

4.2.1 Supersingular Elliptic Curves and Isogenies

Let $\mathbb{F}_q$ be a finite field of q elements where q is a prime power, namely $q = p^n$ for $n > 0$ and $p > 3$. An elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_q$, denoted as $\mathcal{E}/\mathbb{F}_q$, is given by an equation in short Weierstrass form

$$\mathcal{E} : y^2 = x^3 + ax + b, \quad a, b \in \mathbb{F}_q \text{ and } 4a^3 + 27b^2 \neq 0.$$

The set of points on $\mathcal{E}$ over $\mathbb{F}_q$ are the set of pairs $(x, y) \in \mathbb{F}_q^2$ satisfying the curve equation

$$\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}_{\mathcal{E}}\},$$

where $\mathcal{O}_{\mathcal{E}} = (\infty, \infty)$ is the *point at infinity* , which is also considered to be a solution to the Weierstrass equation. The set of points on an elliptic curve $\mathcal{E}$ is an abelian group with the identity element $\mathcal{O}_{\mathcal{E}}$ under the "chord and tangent rule". The number of points on $\mathcal{E}/\mathbb{F}_q$ is $\#\mathcal{E}(\mathbb{F}_q) = q+1-t$ for an integer t lying in the interval $[-2\sqrt{q}, 2\sqrt{q}]$. An elliptic curve is called **supersingular** if $t \equiv 0 \bmod p$, or equivalently $\#\mathcal{E}(\mathbb{F}_q) = 1 \bmod q$, and is called **ordinary** otherwise.

For $k \in \mathbb{N}$ and $P \in \mathcal{E}(\mathbb{F}_q)$, we define $[k]P = P + P + \cdots + P$ (n times). The order of P is k if $[k]P = \mathcal{O}_{\mathcal{E}}$. Since $\mathcal{E}(\mathbb{F}_q)$ is a finite group, the order of any point $P \in \mathcal{E}(\mathbb{F}_q)$ is finite and divides the group order $\#\mathcal{E}(\mathbb{F}_q)$.

Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two elliptic curves over $\mathbb{F}_q$. An **isogeny** $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ is a non-constant rational function which is a group homomorphism (i.e., compatible with the group operations) satisfying $\phi(\mathcal{O}_{\mathcal{E}_1}) = \mathcal{O}_{\mathcal{E}_2}$. Two elliptic curves are **isogenous** if there is an isogeny between them. **Endomorphisms** are a special class of isogenies where the domain and co-domain are the same curve. The endomorphism ring of $\mathcal{E}$ is the set of isogenies from $\mathcal{E}$ to itself, along with the zero map $0 : \mathcal{E} \to \mathcal{E}$ given by $0(P) = \mathcal{O}_{\mathcal{E}}$ for all points P on $\mathcal{E}$. In a set notation, $\text{End}(E) = \{\phi : \mathcal{E} \to \mathcal{E}\} \cup \{0\}$.

Isomorphisms also forms special class of isogenies where the kernel is trivial. If there is a pair of isogenies $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ and $\psi : \mathcal{E}_2 \to \mathcal{E}_1$ such that both $\phi \circ \psi$ and $\psi \circ \phi$ are the identity, then ϕ and ψ are isomorphisms and so $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic curves. Elliptic curves up to isomorphism forms the isomorphism classes. The typical representative for isomorphism classes is the j-invariant which is

$$j(\mathcal{E}) = 1728 \frac{4a^3}{4a^3 + 27b^2}$$

for elliptic curves in short Weierstrass form. The SIDH algorithm establishes the secret key by computing the j-invariant of two isomorphic supersingular elliptic curves generated by the two communicating parties that happens to be isogenous to an initial supersingular curve.

A theorem of Tate states that $\mathcal{E}_1$ and $\mathcal{E}_2$ are isogenous if and only if they have the same number of points over $\mathbb{F}_q$ (indeed over any finite extension of $\mathbb{F}_q$), i.e., $\#\mathcal{E}_1(\mathbb{F}_q) = \#\mathcal{E}_2(\mathbb{F}_q)$ [90]. The set of curves that are isogenous to an elliptic curve $\mathcal{E}$ is called the **isogeny class** of $\mathcal{E}$. Note that if $\mathcal{E}$ is supersingular then all curves in its isogeny class are supersingular; similarly, isogeny class of an ordinary curve consists of ordinary curves. It is well known that every supersingular curve is isomorphic to one defined over $\mathbb{F}_{p^2}$. From now on, we consider the supersingular curves only.

Cryptography is interested in separable isogenies, which does not factor through Frobenius map $(x, y) \mapsto (x^q, y^q)$ over $\mathbb{F}_q$. The **degree** of a separable isogeny is the number of points in its kernel. An isogeny is defined by its kernel in the sense that for every finite subgroup G of $\mathcal{E}_1$, there is a unique $\mathcal{E}_2$ (up to isomorphism) and a separable isogeny $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ such that Ker $\phi = G$. Instead of $\mathcal{E}_2$, we sometimes write $\mathcal{E}_1/G$. Given a finite subgroup G of $\mathcal{E}_1$, an isogeny $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ with kernel G can be constructed by Vélu's formulas [92]. Notice that the number of distinct isogenies of degree ℓ, called as ℓ-isogenies, is equal to the number of distinct subgroups of $\mathcal{E}_1$ of order ℓ.

As an example, for each $m \in \mathbb{Z}$ such that $p \nmid m$ and an elliptic curve $\mathcal{E}$ over $\mathbb{F}_q$, consider the following separable isogeny

$$\begin{aligned} [m] : \quad \mathcal{E} &\to \mathcal{E} \\ P &\mapsto [m]P. \end{aligned}$$

The kernel of this isogeny is the m-torsion subgroup of $\mathcal{E}$, denoted by $\mathcal{E}[m]$, which is the set of points on $\mathcal{E}$ of order m,

$$\mathcal{E}[m] = \{P \in \mathcal{E} : [m]P = \mathcal{O}_{\mathcal{E}}\} = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}.$$

The degree of the above isogeny is equal to $\#\mathcal{E}[m] = m^2$.

4.2.2 Supersingular Isogeny Diffie-Hellman (SIDH)

First, the domain (public) parameters are fixed. Let p be a prime of the form ${\ell_A}^{e_A} {\ell_B}^{e_B} f \pm 1$, where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers and f is a small cofactor such that p is a prime. A supersingular elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_q = \mathbb{F}_{p^2}$ is constructed (this can be done via an efficient algorithm due to Broker [15]) such that it has cardinality $({\ell_A}^{e_A}{\ell_B}^{e_B}f)^2$. Elliptic curve points $P_A, Q_A \in \mathcal{E}[\ell_A^{e_A}]$ are chosen such that the group $\langle P_A, Q_A\rangle$ generated by P_A and Q_A is the entire group $\mathcal{E}[\ell_A^{e_A}]$, i.e., $\{P_A, Q_A\}$ form a basis for $\mathcal{E}[\ell_A^{e_A}]$. In a similar way, elliptic curve points $P_B, Q_B \in \mathcal{E}[\ell_B^{e_B}]$ are chosen such that the group $\langle P_B, Q_B\rangle$ generated by P_B and Q_B is the entire group $\mathcal{E}[\ell_B^{e_B}]$.

The SIDH key exchange protocol between two parties A and B works as follows:

- A picks two random integers $0 \le m_A, n_A < \ell_A^{e_A}$ such that $\ell_A \nmid m_A, n_A$ and computes $[m_A]P_A + [n_A]Q_A$. Similarly, B picks two random integers $0 \le m_B, n_B < \ell_B^{e_B}$ such that $\ell_B \nmid m_B, n_B$ and computes $[m_B]P_B + [n_B]Q_B$.

- A creates a secret isogeny $\phi_A : \mathcal{E} \to \mathcal{E}_A$ with kernel generated by the point $[m_A]P_A + [n_A]Q_A$ by using Vélu's formulas. Then $\mathcal{E}_A = \phi_A(\mathcal{E}) = \mathcal{E}/K_A$ where $K_A = \langle [m_A]P_A + [n_A]Q_A\rangle$ is the kernel. Similarly, B creates a secret isogeny $\phi_B : \mathcal{E} \to \mathcal{E}_B$ for which the kernel is $K_B = \langle [m_B]P_B + [n_B]Q_B\rangle$ and $\mathcal{E}_B = \phi_B(\mathcal{E}) = \mathcal{E}/K_B$.

- In the exchange step of the protocol, A and B publishes the messages $(\mathcal{E}_A, \phi_A(P_B), \phi_A(Q_B))$ and $(\mathcal{E}_B, \phi_B(P_A), \phi_B(Q_A))$, respectively.

- Upon receipt of B's message, A computes an isogeny $\phi'_A : \mathcal{E}_B \to \mathcal{E}_{AB}$ with kernel $\langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)\rangle = \langle \phi_B([m_A]P_A + [n_A]Q_A\rangle = \phi_B(K_A)$. Here, $\mathcal{E}_{AB} = \mathcal{E}_B/\phi_B(K_A)$. Similarly, having received A's message, B computes an isogeny $\phi'_B : \mathcal{E}_A \to \mathcal{E}_{BA}$ with kernel $\langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B)\rangle = \phi_A(K_B)$. Here, $\mathcal{E}_{BA} = \mathcal{E}_A/\phi_A(K_B)$.

- The elliptic curves $\mathcal{E}_{AB}$ and $\mathcal{E}_{BA}$ computed by A and B are isomorphic as they are both isomorphic to $\mathcal{E}/\langle K_A, K_B\rangle$, so they have the same j-invariant. This common j-invariant is the shared secret key.

Given the curves $\mathcal{E}_A$, $\mathcal{E}_B$ and the points $\phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A)$ as described in the above protocol, finding the j-invariant of $\mathcal{E}/\langle K_A, K_B\rangle$ is called as *supersingular computational Diffie-Hellman (SSCDH) problem*. The security of SIDH is based on this problem. SSCDH is more special (due to auxiliary information) than the main problem in this area known as *supersingular isogeny problem* which is described as follows: Given a finite field K and two supersingular elliptic curves $\mathcal{E}_1$, $\mathcal{E}_2$ defined over K such that $|\mathcal{E}_1| = |\mathcal{E}_2|$, compute an isogeny $\phi : \mathcal{E}_1 \to \mathcal{E}_2$. The best known classical algorithm for this problem is due to Delfs and Galbraith [31] and requires $\tilde{O}(p^{1/2})$ bit operations and the best known quantum algorithm is due to Biasse et al [7] and requires $\tilde{O}(p^{1/4})$ bit operations. However, SSCDH problem can be regarded as an instance of the *claw problem* for which the best known classical and quantum attacks requires $O(p^{1/4})$ and $O(p^{1/6})$ bit operations, respectively [95, 89].

Note that the number of possible secret isogenies that A can create is equal to the number of possible distinct kernels, which is $\ell_A^{e_A-1}(\ell_A+1)$ and the number of possible isogeny choices for B is $\ell_B^{e_B-1}(\ell_B+1)$.

Further note that the time needed to compute an isogeny grows linearly with the degree of the isogeny. Representing a large degree isogeny as a composition of small prime degree isogenies makes isogeny crypto feasible. For example, SIDH decomposes a degree $\ell_A^{e_A}$ isogeny into a sequence of e_A isogenies of degree ℓ_A instead of computing the isogeny in a single step using Vélu's formulas. while the computation cost of the latter one is $O(\ell_A^{e_A})$, the cost to compute the former one is proportional to ℓ_A. To reduce the cost of the computation of sequences of isogenies and speed the computation up, Jao and De Feo proposed a new method. For further details, we refer the readers to [51].

The selection of the primes, the selection of the curve equation and the elliptic curve point representation (affine vs projective) together yield efficient implementations of the SIDH algorithm. For the fast arithmetic computation inside the SIDH protocol, it is more convenient to use the primes of the form $p = 2^{e_A}3^{e_B} \pm 1$. For an initial curve $\mathcal{E}/\mathbb{F}_{p^2} : y^2 = x^3 + x$ where $p = 2^{e_A}3^{e_B} \pm 1$, the 751-bit prime $p = 2^{372}3^{239} - 1$ provides 125-bit post-quantum security level matching security of AES-192 and the 964-bit prime $p = 2^{486}3^{301} - 1$ provides 161-bit post-quantum security level matching AES-256.

5 Homomorphic Encryption

Cloud computing offers many services to users, including storage of and computation with large amounts of data. To take the advantage of the cloud computing, users must share their data with the service provider. These data might be sensitive (for example, financial data or patients' medical reords). A simple solution to ensure data privacy is to encrypt the data that is sent to the cloud. However, a user cannot compute on the data in the cloud, and to perform computations, the data must be downloaded and decrypted, or the secret key must be shared with the service provider. The former process nullifies the major advantage of using cloud services while the later process sacrifices privacy. This is where *homomorphic encryption* (HE) comes into play. While the conventional encryption schemes cannot perform operations on the encrypted data without decrypting it first, HE allows the cloud servers to compute on encrypted data without decrypting it in advance. This concept was first introduced in 1978, shortly after the invention of RSA cryptosystem [76], by Rivest, Adleman and Dertouzous [75], in their work entitled "On data banks and privacy homomorphism".

A homomorphic (public-key) encryption scheme $\mathcal{E}$ consists of four efficient algorithms: $\mathbf{KeyGen}_{\mathcal{E}}$, $\mathbf{Enc}_{\mathcal{E}}$, $\mathbf{Dec}_{\mathcal{E}}$ and $\mathbf{Eval}_{\mathcal{E}}$, where the first three algorithms are the usual 3-tuples of any conventional public-key encrption scheme whereas the fourth one is an HE-specific algorithm which is associated to a set of permitted functions $\mathcal{F}_{\mathcal{E}}$. These algorithms are efficient in the sense that their computational complexity must be polynomial in security parameter λ that specifies the bit-length of the keys. $\mathbf{KeyGen}_{\mathcal{E}}$ takes a security parameter λ as input, and outputs a pair of keys (pk, sk), where pk denotes the public key and sk denotes the secret key. $\mathbf{Enc}_{\mathcal{E}}$ takes the public key pk, a plaintext m from the underlying plaintext space $\mathcal{M}$ and some randomness as inputs, and outputs a ciphertext $c \in \mathcal{C}$ where $\mathcal{C}$ is the ciphertext space. $\mathbf{Dec}_{\mathcal{E}}$ takes the secret key sk and a ciphertext c as inputs, and outputs a plaintext m. *Correct decryption* is required to be able to call $\mathcal{E}$ an encryption scheme, i.e. the equality

$$\mathbf{Dec}_{\mathcal{E}}(sk, \mathbf{Enc}_{\mathcal{E}}(pk, m)) = m$$

should be satisfied. $\mathbf{Eval}_{\mathcal{E}}$ takes the public key pk, any ciphertexts $c_1, \ldots, c_t \in \mathcal{C}$ with $\mathbf{Enc}_{\mathcal{E}}(pk, m_i) = c_i$ and any permitted function f in $\mathcal{F}_{\mathcal{E}}$ as inputs. It outputs an evaluated ciphertext that encrypts $f(m_1, \ldots, m_t)$. *Correct evaluation* is satisfied if the following holds:

$$\mathbf{Dec}_{\mathcal{E}}(sk, \mathbf{Eval}_{\mathcal{E}}(pk, f, c_1, \ldots, c_t)) = f(m_1, \ldots, m_t),$$

i.e. the evaluated ciphertext decrypts to the computation of the plaintexts through $f \in \mathcal{F}_{\mathcal{E}}$. If f is not in $\mathcal{F}_{\mathcal{E}}$, with an overwhelming probability, $\mathbf{Eval}_{\mathcal{E}}$ algorithm will not produce a meaningful output.

If $\mathcal{E}$ has the properties of both correct decryption and correct evaluation for the functions in $\mathcal{F}_{\mathcal{E}}$, then it is called an **$\mathcal{F}_{\mathcal{E}}$-homomorphic scheme**. However, mere correctness does not rule out trivial schemes where the evaluation algorithm just output $(f, c_1 \ldots, c_t)$ without processing the ciphertexts at all, and the decryption function decrypts the ciphertexts $c_1, \ldots, c_t$ and then apply f to the resulting plaintexts. Further important attribute of an homomorphic encryption scheme, which is referred as *compactness* (or *compact ciphertext requirement*), excludes this trivial case. Compactness property requires the ciphertext size and decryption time to be completely independent of the homomorphically evaluated function f but only dependent on the security parameter λ. For example, decryption of an evaluated ciphertext takes the same amount of computation as decryption of a fresh ciphertext $c = \mathbf{Enc}_{\mathcal{E}}(pk, m)$. More formally, $\mathcal{E}$ is *compact* if there exists a polynomial g such that, for every value of the security parameter λ, $\mathbf{Dec}_{\mathcal{E}}$ can be expressed as a circuit of size at most $g(\lambda)$. Note that an $\mathcal{F}_{\mathcal{E}}$-homomorphic scheme is not necessarily compact.

An arithmetic function f can also be represented as a circuit which breaks the computation of f down into AND, OR and NOT gates. Addition, subtraction and multiplication operations (in fact, these operations *modulo* 2) are enough to evaluate these gates. For $x, y \in \{0, 1\}$, we have $\text{AND}(x, y) = xy$, $\text{NOT}(x) = 1 - x$ and $\text{OR}(x, y) = 1 - (1 - x)(1 - y)$.

Homomorphic encryption schemes are categorized into three classes according to the set of permitted functions. If an encryption scheme permits only one type of operation (either addition or multiplication) with an unlimited number of times, then it is called a **partially homomorphic encryption (PHE)** scheme; if it allows one type of operation with a limited number of times while allowing another infinitely many times, it is called a **somewhat homomorphic encryption (SWHE)** scheme. In PHE and SWHE schemes, there is no compactness requirement, i.e., the ciphertexts can get quite larger with each homomorphic operation. If an encryption scheme can handle all functions (i.e., allows both addition and multiplication infinitely many times) and fulfill the compactness requirement then it is called as **fully homomorphic encryption (FHE)** scheme.

PHE schemes are deployed in some particular real-life applications like electronic voting [3] and Private Information Retrieval (PIR) [55] whose algorithms support only addition operation. Although, SWHE schemes support both addition and multiplication, the maximum number of operations performed homomorphically is limited since each operation contributes "noise" to the ciphertext and after a threshold decryption fails. However, explosion in demand for cloud computing platforms accelerated the construction of FHE schemes which enables arbitrary computation on encrypted data.

5.1 Partially Homomorphic Encryption

There are several PHE schemes [76, 45, 35, 2, 62, 69, 71, 30], supporting either addition or multiplication operation, in the literature . In this section, we focus on two PHE schemes.

5.1.1 Goldwasser-Micali Algorithm

The Goldwasser–Micali (GM) algorithm [45], developed by Shafi Goldwasser and Silvio Micali in 1982, has the distinction of being the first probabilistic public-key encryption scheme, where each plaintext has several corresponding ciphertexts. The security of the GM algorithm is based on the Quadratic Residuosity Problem modulo $n = p \times q$.

An integer $a \in \mathbb{Z}_n^*$ is called a *quadratic residue* modulo n if there exists an integer $x \in \mathbb{Z}_n^*$ such that $a \equiv x^2 \pmod{n}$. If there is no solution to this congruence, then a is called a *quadratic non-residue* modulo n.

There is a special number-theoretic tool associated with quadratic residues, the *Jacobi symbol*, denoted by $\left(\frac{a}{n}\right)$, which is defined for all $a \geq 0$ and all odd positive integers n. For the sake of a total understanding of the GM algorithm, we refer the reader to Section 2.1.10 (Quadratic Residues) in [70]. If $n > 3$ is an odd composite integer, the problem of determining whether a nonnegative integer a with Jacobi symbol 1 is a quadratic residue modulo n is called the *Quadratic Residuosity Problem*.
KeyGen: Two random large primes p and q are chosen, and $n = p \times q$ is computed. Then a quadratic non-residue $x \in \mathbb{Z}_n^*$ with Jacobi symbol $\left(\frac{x}{n}\right) = 1$ is chosen. This choice is accomplished by finding $x \in \mathbb{Z}_n^*$ such that $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$. By choosing

p and q as Blum integers, i.e. $p \equiv 3 \pmod 4$ and $q \equiv 3 \pmod 4$, the integer $n-1$ is guaranteed to be a quadratic non-residue with $\left(\dfrac{n-1}{n}\right) = \left(\dfrac{-1}{n}\right) = 1$. The public key pair is (n, x) and the private key pair is (p, q).

Enc: After converting the message into a plaintext which is a string of bits $(m_1, m_2, \ldots, m_k)$, the sender picks uniformly at random $y_i \in \mathbb{Z}_n^*$ for each bit m_i and encrypts each bit by computing

$$c_i = E(m_i) \equiv y_i^2 \times x^{m_i} \pmod n.$$

via the encryption function

$$\begin{aligned} E : (\{0,1\}, \oplus) &\rightarrow (\mathbb{Z}_n^*, \times) \\ m &\mapsto y^2 \times x^m, \end{aligned}$$

where $\oplus$ denotes addition modulo 2, $\times$ denotes modular multiplication and $\mathbb{Z}_n^*$ denotes the set of positive integers that are less than n and relatively prime to n. The ciphertext generated is $(c_1, c_2, \ldots, c_k)$, such that $c_i \in \mathbb{Z}_n$ for $i = 1, 2, \ldots, k$.

Dec: To decrypt the message and get the plaintext back, the legitimate receiver, who knows the private key pair (p, q) and can decide the quadratic residuosity of c_i modulo p and modulo q, determines whether c_i is a quadratic residue modulo n for $i = 1, \ldots, k$. If c_i is a quadratic residue modulo both p and q, then c_i is a quadratic residue modulo n, which necessarily yields $m_i = 0$. Otherwise, c_i is a quadratic non-residue modulo n which implies $m_i = 1$.

Eval: Let y_1 and y_2 be randomly selected integers in $\mathbb{Z}_n^*$. For bits m_1 and m_2,

$$\begin{aligned} E(m_1) \times E(m_2) &\equiv ({y_1}^2 \times x^{m_!}) \times ({y_2}^2 \times x^{m_2}) \pmod n \\ &\equiv (y_1 \times y_2)^2 \times x^{m_1 \oplus m_2} \pmod n \\ &\equiv E(m_1 \oplus m_2), \end{aligned}$$

which yields

$$D(E(m_1) \times E(m_2)) = m_1 \oplus m_2.$$

The randomness in the encryption of $m_1 \oplus m_2$ is $y_1 \times y_2$, which is neither uniformly distributed in $\mathbb{Z}_n^*$ nor independent of the randomness in $E(m_1)$ and $E(m_2)$. However this can be addressed by the re-randomization property of GM algorithm. Let $r \in \mathbb{Z}_n^*$ be a random number. Then,

$$r^2 \times E(m) \equiv r^2 \times y^2 \times x^m \pmod n \equiv (r \times y)^2 \times x^m \pmod n,$$

which is a valid encryption of m with the randomness $r \times y \in \mathbb{Z}_n^*$. Hence $D((r^2 \times E(m)) = m$.

5.1.2 ElGamal Algorithm

The ElGamal cryptosystem [35], which is a public-key encryption scheme proposed by Taher ElGamal in 1985, improves the Diffie-Hellman key exchange method [32] into an encryption algorithm. There are two number theoretic versions of this algorithm; one is multiplicatively homomorphic and the other is additively homomorphic. Additively homomorphic version is not practical in use since it forces the legitimate receiver to solve a discrete logarithm problem, which is intractable, to decrpyt a ciphertext. Therefore, we focus our attention on the multiplicatively homomorphic version of ElGamal. Its security is based on the hardness of both the Computational Diffie-Hellman Problem and the Decisional Diffie-Hellman Problem in the underlying group G_q.

KeyGen: Two random large primes p and q satisfying $q \mid (p-1)$ are chosen. Next, a cyclic subgroup G_q of $\mathbb{Z}_p^*$ of order q with generator g is chosen. This choice is accomplished by selecting some $y \in \mathbb{Z}_p^*$ and computing $g \equiv y^{(p-1)/q} \pmod p$. Finally a random $x \in \mathbb{Z}_q$ is selected an $h = g^x \pmod p$ is computed. The public key quadruple is (p, q, g, h) and the private key is x.

Enc: The plaintext is $m \in G_q$. The sender generates a random number $r \in \mathbb{Z}_q$ and computes the ciphertext pair

$$E(m) = (c_1, c_2) = (g^r \pmod p, m \times h^r \pmod p))$$

via the encryption function

$$\begin{aligned} E : (G_q, \times) &\rightarrow (G_q \times G_q, \times) \\ m &\mapsto (g^r, m \times h^r), \end{aligned}$$

For encryption of each message, a new r is chosen to be a uniformly random integer in order to ensure security.

Dec: The legitimate receiver who holds the private key x can decrypt the ciphertext (c_1, c_2), without knowing the value of r, by computing u_1 and u_2 as

$$\begin{aligned} u_1 &= (g^r)^x = (g^x)^r \equiv h^r \pmod p \\ u_2 &= {u_1}^{-1} \times c_2 \equiv h^{-r} \times (m \times h^r) \equiv m \pmod p), \end{aligned}$$

where ${u_1}^{-1}$ is the multiplicative inverse of u_1 in the group G_q. This inverse can be computed using the Extended Euclidean Algorithm in number theory.

Eval: Let m_1 and m_2 be two plaintexts with accompanying random numbers r and r', respectively. Then the pairwise products of the ciphertext pairs are

$$\begin{aligned} E(m_1) \times E(m_2) &= (c_1 \times c_1', c_2 \times c_2') \\ &= (g^r \times g^{r'} \pmod p, (m_1 \times h^r) \times (m_2 \times h^{r'}) \pmod p)) \\ &= (g^{r+r'} \pmod p, m_1 \times m_2 \times h^{r+r'} \pmod p)) \\ &= E(m_1 \times m_2) \end{aligned}$$

where the randomness in the encryption of $m_1 \times m_2$ is $r+r'$, which is neither uniformly distributed in $\mathbb{Z}_q$ nor independent of the randomness in $E(m_1)$ and $E(m_2)$. However this can be addressed by the re-randomization property of the multiplicative ElGamal algorithm.

Let $E(m) = c = (c_1, c_2) \equiv (g^r, m \times h^r) \pmod{p}$ for random $r \in \mathbb{Z}_q$, and $r' \in \mathbb{Z}_q$ be another chosen random number. Then

$$\begin{aligned}(c_1 \times g^{r'}, c_2 \times h^{r'}) &\equiv (g^r \times g^{r'}, m \times h^r \times h^{r'}) \pmod{p} \\ &\equiv (g^{r+r'}, m \times h^{r+r'}) \pmod{p},\end{aligned}$$

which is a re-randomized ciphertext of the original message m where $r + r' \in \mathbb{Z}_q$. $D(c) = D(c \times (g^{r'}, h^{r'}))$.

5.2 Somewhat Homomorphic Encryption

Until 2005, all proposed encryption schemes had partial (either additive or multiplicative) homomorphic property. In 2005, Boneh, Goh and Nissim constructed BGN cryptosystem based on bilinear pairings on elliptic curves that can support arbitrarily many additions and a single multiplication by keeping the ciphertext size constant. While BGN scheme meets the compactness requirement, allowing only one multiplication makes it somewhat homomorphic. After the first plausible FHE published in 2009 [41], some SWHE versions of FHE schemes were also proposed due to the performance issues associated with FHE schemes.

5.3 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is a special type of encryption which is both additively and multiplicatively homomorphic. Since addition and multiplication form a complete set of operations, an FHE scheme allows any polynomial-time computation on encrypted data. In 1978, Rivest, Adleman and Dertouzos [75] first proposed theoretic possibility of a scheme supporting arbitrarily complex computation in their paper titled “On Data Banks and Privacy Homomorphisms”. However, for more than 30 years, this theoretic possibility could not be put into practice and so it has been regarded as a “holy grail” of cryptography. Craig Gentry proposed the first plausible way of obtaining an FHE scheme based on ideal lattices in his seminal Stanford PhD thesis [41].

Gentry’s proposed scheme is not only an FHE scheme but also a blueprint to obtain an FHE scheme from an SWHE scheme. Although this scheme was considered as a major breakthrough, it was not efficient and hard to implement. Since the release of this blueprint, significant progress has been made in the direction of finding efficient and simpler FHE schemes [85, 91, 86, 13, 12]. His construction has three components: an SWHE scheme that can support a limited number of operations (a few multiplications and arbitrarily many additions), *squashing* method which converts the SWHE scheme into a bootstrappable one and finally a method of *bootstrapping* which turns the (bootstrappable) SWHE scheme into an FHE scheme.

Encryption functions of all existing SWHE schemes works by adding a small amount of noise to the plaintext. Homomorphic evaluations on ciphertexts increase this noise and once it exceeds a certain threshold, the decryption fails. *Bootstrapping* refreshes a ciphertext by running the decryption function on it homomorphically. An SWHE scheme $\mathcal{E}$ is called *bootstrappable* if it can evaluate its own decryption function, plus one addition or multiplication gate modulo 2. When these augmented circuits are in the permitted set of functions (or circuits) $\mathcal{F}_{\mathcal{E}}$, one can construct a fully homomorphic encryption scheme from $\mathcal{E}$. A bootstrappable scheme refreshes the evaluated ciphertext for more homomorphic computations by reducing the noise in the ciphertext via the following $\mathbf{Recrypt}_{\mathcal{E}}$ algorithm.

$\mathbf{Recrypt}_{\mathcal{E}}(pk_2, D_{\mathcal{E}}, \overline{sk_1}, c_1)$

- Generate $\overline{c_1}$ via $\mathbf{Encrypt}_{\mathcal{E}}(pk_2, c_{1j})$ over the bits of c_1
- Output $c = \mathbf{Eval}_{\mathcal{E}}(pk_2, D_{\mathcal{E}}, \overline{sk_1}, \overline{c_1})$

First, it is supposed that two different public and secret key pairs are generated, (pk_1, sk_1) and (pk_2, sk_2). Let c_1 be the encryption of the message bit m with pk_1 and let $\overline{sk_1}$ be a vector of ciphertexts encrypted with pk_2 over the bits of sk_1. The public key pk_2, the decryption circuit $D_{\mathcal{E}}$, $\overline{sk_1}$ and c_1 are taken as inputs by the $\mathbf{Recrypt}_{\mathcal{E}}$ function. First, $\overline{c_1}$ is generated as a bitwise encryption of c_1 with the key pk_2 using the encryption function. It is easy to recognize that $\overline{c_1}$ is doubly-encrypted. Since the SWHE scheme $\mathcal{E}$ can evaluate its own decryption function homomorphically, the noisy inner ciphertext is decrypted homomorphically with $\overline{sk_1}$. After the evaluation, a new encryption of m but under pk_2 is obtained. While the noise is decreased by eliminating the noise from the inner ciphertext, additional noise is added during the homomorphic evaluation of the decryption function. As long as the new noise added is less than the old noise removed, there is a progress. Further homomorphic operations can be done repeatedly on the obtained "fresh" ciphertext until reaching again a threshold point.

Gentry's bootstrapping technique can be applied only if the decryption function is simple enough. Otherwise, first *squashing* method should be applied in order to reduce the complexity of the decryption function so that it is in the set of permitted functions. In brief, squashing converts an SWHE scheme into a bootstrappable one.

The development of FHE since the release of Gentry's work [41] can be roughly divided into four generations according to the techniques used in constructing the FHE schemes.

5.3.1 First Generation FHE

This starts with Gentry's original scheme using ideal lattices [41]. The security of the underlying SWHE scheme is based on the hardness of an average-case decision problem over ideal lattices, namely a variant of the "bounded distance decoding problem (BDDP)" on ideal lattices. The semantic security of the achieved FHE scheme is based on an additional assumption called "sparse subset sum assumption". Subsequently, Gentry [42] showed a worst-case to average-case reduction for BDDP over ideal lattices. In the same year of this security reduction, van Dijk et al. [91] presented the second FHE scheme based on the Gentry's idea, but ideal lattice computations were replaced by simple integer aritmetic operations. The security of this fully homomorphic DGHV scheme is based on the "approximate gcd (AGCD) problem" and "sparse subset sum problem (SSSP)". Then, Smart and Vercauteren [85] introduced a third variant of Gentry's scheme which uses both relatively small key and ciphertext size. Afterwards, a series of articles [68, 44, 79] presented optimized the key generation algorithms in order to implement Gentry's FHE scheme efficiently.

These first generation schemes have several bottlenecks in terms of applicability in real life. Firstly, they have limited homomorphic capacity due to very rapid noise growth. Squashing the decryption circuit to make the underlying SWHE schemes bootstrappable comes at the expense of additional and fairly strong security assumption namely the sparse subset sum assumption. Moreover, the schemes that follow Gentry's blueprint have inherent efficiency limitations. The efficiency of an FHE scheme is measured by the ciphertext and key size, the time it takes to encrypt and decrypt, and more importantly per-gate computation overhead which is defined as the ratio between the time it takes to compute a circuit homomorphically on encrypted inputs to the time it takes to compute it on clear inputs. The first generation FHE schemes that follow Gentry's blueprint have a quite poor performance so that their per-gate computation overhead is $p(\lambda)$, a large polynomial in the security parameter.

In 2011, Gentry and Halevi [43] constructed a new approach which is one of the first major deviations from Gentry's blueprint. Their construction still relies on ideal lattices and on bootstrapping but eliminates the need for squashing and thereby does not rely on the hardness of the SSSP. However, there is no noteworthy improvement on the efficiency aside from the optimization that reduces the ciphertext length.

5.3.2 Second Generation FHE

The second generation began in 2011 with the work of Brakerski and Vaikuntanathan [14]. They introduced *re-linearization technique* to control ciphertext dimension in homomorphic multiplications. Further, they showed how to construct a bootstrappable scheme without using squashing, instead using a new method to simplify the decryption algorithm named *dimension-modulus reduction* which does not require sparse subset sum assumption for security. The security of BV scheme is based solely on the hardness of much more standard "learning with error (LWE)" problem introduced by Regev [74] as a generalization of "learning parity with noise" problem. Compared with the previous schemes using squashing method, BV scheme [14] (as well as GH scheme [43]) has no noteworthy efficiency improvement because of costly bootstrapping operation. The real cost of bootstrapping for FHE schemes that follow Gentry's blueprint is much worse than quadratic (see [12] for a detailed analysis). Brakerski, Gentry and Vaikuntanathan leveraged the techniques in [14] and constructed a *leveled*-FHE scheme [12]. Leveled-FHE is a relaxation of FHE, in which the parameters depend (poynomially) on the depth of the circuits that the scheme is capable of evaluating. The depth referred here is the multiplicative depth which is the maximal number of sequential multiplications that can be performed on ciphertexts. The re-linearization and dimension-modulus reduction techniques in [14] were enhanced as the *key switching* and *modulus switching* techniques in BGV scheme. Modulus switching is a powerful noise management technique that control the noise without bootstrapping and it is computationally cheaper than bootstrapping. This technique sacrifices modulus size without jeopardizing the correctness of decryption. In other words, a ciphertext modulo q is replaced with a ciphertext modulo a smaller modulus p which decrypts to the same plaintext. Although BGV scheme does not requires bootstrapping, they used it as an optimization to reduce the per-gate computation overhead. The security of BGV scheme is based on RLWE (ring learning with error) problem [60] with quasi-polynomial approximation factors whereas all the previous schemes relies on the hardness of problems with sub-exponential approximation factors. BGV scheme can also be instantiated with LWE rather than RLWE, albeit with worse performance. After BGV scheme, Brakerski [11], introduced a new scale-invariant FHE without modulus switching. In this scheme, the same modulus is used throughout the homomorphic evaluation process. Compared with previous LWE-based FHE schemes, in [11] the ciphertext noise grows only linearly with the homomorphic operations rather than exponentially. Then, Fan and Vercauteren [36] optimized the Brakerski's scheme by changing the based assumption from LWE problem to RLWE problem. Another improvements of Brakerski's scheme was reducing the computational overhead of key switching, faster execution of homomorphic operations and efficiency improvement [93]. Later Zhang et al. modified and improved Brakerski's scheme [96].

It is also worth noting that in 2012 a NTRU-based multikey FHE scheme was proposed by Lopez-Alt, Tromer and Vaikuntanathan (LTV) [59] for its promising efficiency and standardization properties. However, to allow homomorphic operations and prove security, a non-standard assumption is required in LTV scheme. In the following year, Bos, Lauter, Loftus, and Naehrig [9] showed how to remove this non-standard assumption via Brakerski's scale invariant technique [11].

In second generation FHE schemes, noise growth is slower during homomorphic evaluations compared with first generation FHE schemes. Moreover, although second generation follows Gentry's blueprint in the sense that they first construct a SWHE scheme and then transform it into a FHE scheme using bootstrapping, they can even be operated in the leveled-FHE mode without bootstrapping and this makes them more efficient. However, the complex process of key-switching (or re-linearization) still introduces a huge computational cost which is a main bottleneck for practicality.

5.3.3 Third Generation FHE

In 2013, Gentry, Sahai and Waters proposed a new LWE-based FHE scheme, known as GSW, which uses *approximate eigenvector* method instead of the expensive re-linearization (or key switching) technique. Since the ciphertexts of GSW scheme are matrices that are added and multiplied homomorphically in a natural way, the cipher-text dimension is kept constant. GSW scheme is simpler and asymptotically faster than the previous LWE-based FHE schemes. In the following years, two efficient ring variants of the GSW cryptosystem known as FHEW [34] and TFHE [24] were introduced by Ducas and Micciancio and by Chillotti et al, respectively.

5.3.4 Fourth Generation FHE

All three generation FHE schemes mentioned above support the exact arithmetic operations over some discrete spaces like rings or finite fields. However, majority of real-world applications require computations in a continuous space such as $\mathbb{R}$ or $\mathbb{C}$. To address this issue, Cheon et al. proposed CKKS algorithm [21] which provides a natural setting for performing operations on approximate numbers. The CKKS algorithm is particularly suitable for implementing prediction and machine learning methods. The name of the algorithm originally went by the name HEAAN, but later the authors changed it to CKKS in order to distinguish it from the homomorphic encryption library HEAAN which implements CKKS. After the release of the CKKS scheme, a full residue number system (RNS) variant was introduced in [20].

Bootstrapping to extend the original leveled encryption scheme CKKS to a fully homomorphic encryption was first proposed by Cheon et al [19]. Subsequently, several newer and better algorithms have been presented for bootstrapping CKKS and its full-RNS variants [17, 47, 10, 57].

5.4 Implementation Issues

Several open-source FHE libraries exist today. Below we list the most popular ones with the authors (developers) created them, the schemes they support and the languages they are implemented in.

SEAL : Authored by Microsoft; includes BFV, CKKS and written in C++

PALISADE : Authored by a consortium of DARPA-funded defense contractors; includes BGV, BFV, CKKS, TFHE, FHEW and written in C++

HELib : Authored by Halevi and Shoup; includes BGV, CKKS and written in C++

HEAAN : Authored by Cheon, Kim, Kim, and Song; includes CKKS and written in C++

FHEW : Authored by Ducas and Micciancio; includes FHEW and written in C++

TFHE : Authored by Chillotti et al; includes TFHE and written in C++

FV-NFLlib : Authored by CryptoExperts; includes BFV and written in C++

FV-NFLlib : Authored by EPFL-LDS; includes BFV, CKKS and written in Go

In order to use homomorphic encryption in medical and financial sectors and also for national security, it will have to be standardized via an agreement on security levels for parameter sets by multiple standardization bodies and government agencies. HomomorphicEncryption.org group, co-founded in 2017 by Microsoft, IBM, and others, is an open consortium of people from industry, government and academia for this standardization process [1].

With the rapid development of FHE schemes and libraries, and frameworks, it is important that the cryptography community has a standard for how to safely set the security parameters. In order for homomorphic encryption to be adopted in medical, health, and financial sectors, An important part of the standardization process is the agreement on security levels for varying parameter sets. HomomorphicEncryption.org has undertaken the task of this standardization.

In the remainder of this section, we describe the ideas behind Gentry's lattice based original construction forming the first generation with the conceptually simpler DGHV scheme [91]. Then, BGV scheme will be presented to describe the ideas behind the second generation. Finally, fourth generation CKKS scheme will be explained.

5.5 The DGHV Scheme

In [91], van Dijk et al. described a remarkably simple SWHE scheme using only modular arithmetic and used Gentry's techniques to convert it into a fully homomorphic scheme. The construction is based on the hardness of the Approximate Greatest Common Divisor (AGCD) problem formulated by Howgrave-Graham [49]. It is easy to compute the greatest common divisor of a given set of integers by Euclidean Algorithm. However, given polynomially many near-multiples $x_i = s_i + p \cdot q_i$ of a number p, where s_i is much smaller than $p \cdot q_i$, it is hard to compute p. In fact, AGCD assumption states that when the multiples are "noisy", it is not possible to compute p efficiently. AGCD problem can be reduced to the security of the scheme of van Dijk et al.

A secret-key SWHE scheme will be described first. Then a public-key version will be obtained by invoking the result of Ron Rothblum [78] that shows how to transform any secret-key homomorphic encryption scheme into a public-key one.

DGHV construction uses a number of parameters (all polynomial in the security parameter λ) adapted from AGCD problem and they are set under some constraints. As a convenient parameter setting, set $N = \lambda, P = \lambda^2$ and $Q = \lambda^5$.

KeyGen$_\mathcal{E}$: A random P-bit odd integer p (not necessarily prime) is generated.

Enc$_\mathcal{E}$: To encrypt a bit $m \in \{0, 1\}$, a random N-bit number μ is chosen such that $\mu = m \bmod 2$ and a random Q-bit number q is chosen. Write $\mu = m + 2r$ for $r \ll p$. The output is a fresh ciphertext $c = E(m) = \mu + pq = m + 2r + pq$ with a small "noise" μ which masks the actual message.

Dec$_\mathcal{E}$: The ciphertext is decrypted as $m = D(c) = (c \bmod p) \bmod 2$. Decryption works properly as long as the noise $c \bmod p$ is in the range $(-p/2, p/2)$ such that p divides $c - c'$. This condition put a limit on the number of homomorphic operations performed on the ciphertexts. As the noise of the system grows over $p/2$, the decryption no longer returns the correct result.

Eval$_\mathcal{E}$: Consider the ciphertexts $c_1 = m_1 + 2r_1 + pq_1$ and $c_2 = m_2 + 2r_2 + pq_2$, where c_i's noise is $m_i + 2r_i$. Then homomorphic addition computes

$$E(m_1) + E(m_2) = (m_1 + m_2) + 2(r_1 + r_2) + p(q_1 + q_2)$$

which is a valid ciphertext of $m_1 + m_2$ as long as the noises are small enough so that $|(m_1 + m_2) + 2(r_1 + r_2)| < p/2$. It is possible to perform various number of homomorphic additions before noise goes beyond $p/2$.

Homomorphic multiplication computes

$$E(m_1)E(m_2) = m_1m_2 + 2(2r_1r_2 + r_1m_2 + r_2m_1) + pq'$$

for some integer q'. This is a valid ciphertext of m_1m_2 and can be decryted as long as the noises are small enough so that $| = m_1m_2 + 2(2r_1r_2 + r_1m_2 + r_2m_1)| < p/2$. It is clear that multiplication increases the noise faster than addition.

After performing too many multiplications and additions, the noise can go beyond $p/2$ and the decryption function of the scheme $\mathcal{E}$ no longer outputs the correct plaintext. Hence, this somewhat homomorphic encryption scheme is not fully homomorphic. But still $\mathcal{E}$ is homomorphic enough. It can handle an elementary symmetric polynomial in t variables of degree (roughly) $d < P/(N \cdot \log t)$ as long as $2^{Nd} \cdot \binom{t}{d} < p/2$.

The scheme described so far was the secret-key version of the homomorphic encryption. A public-key version is presented in [91]. The secret key of the scheme is p as before. The public key is a list of encryptions of zero under the secret-key version: $\{x_i = 2r_i + pq_i\}_{i=0}^{k}$ where r_i and q_i are chosen as before. Here the x_i are sampled so that x_0 is the largest, x_0 is odd and $x_0 \bmod p$ is even. To encrypt a bit m, a random subset $S \subset \{1, 2, \ldots, k\}$ and a random integer in a certain range are chosen. The encryption is

$$c = m + 2r + 2\sum_{i \in S} x_i \bmod x_0$$

The ciphertext is decrypted as $(c \bmod p) \bmod 2$ as long as c has a small noise (which is possible only if the encyptions of zero in the public key have small noises).

Now it is time to ask this question: Is the somewhat homomorphic scheme $\mathcal{E}$ described above "bootstrappable" ? Only if $\mathcal{E}$ is capable of evaluating its own decryption circuit (plus some), the answer is "yes".

For the bootsrapping analysis, consider the decryption function

$$m = (c \bmod p) \bmod 2.$$

Since $c \bmod p = c - p \cdot \lfloor c/p \rceil$, where $\lfloor \cdot \rceil$ rounds to the neraest integer, and also p is odd, decryption function can be written more simply as

$$c - \lfloor c/p \rceil \bmod 2 = (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2).$$

This is just the XOR of the least significant bits of c and $\lfloor c/p \rceil$.

Computing the least significant bit and XOR is immediate. However, computing $\lfloor c/p \rceil$ is complicated. Because, each long numbers c and $1/p$ need to be expressed with at least $P \approx \log p$ bits of precision to guarantee that $\lfloor c/p \rceil$ is computed correctly. As two P-bit numbers are multiplied, a bit of the result may be a high-degree polynomial of the input bits. This degree is also roughly P. Since $\mathcal{E}$ can handle an elementary symmetric polynomial in t variables of approximate degree $d < P/(N \cdot \log t)$, it is not possible for $\mathcal{E}$ to handle even a single monomial of degree P, where the noise of output ciphertext is upper-bounded by $(2^N)^P \approx p^N \gg p/2$. It turns out that $\mathcal{E}$ cannot handle its decryption function, which means it is not bootstrappable.

However, it is possible to transform the scheme, by using Gentry's ingenious *squashing* technique, into a bootstrappable one with the same homomorphic capacity but a decryption function that is simple enough. This transformation is accomplished by augmenting the public key with a "hint" about the secret key. The hint is a large set of rational numbers that has a secret sparse subset which sums to the original secret key. The "post-processed" ciphertext via this hint, which contains a sum of a small set of nonzero terms instead of the multiplication of long numbers c and $1/p$, is decrypted more efficiently than the original ciphertext. In order to guarantee that the hint in the public key does not reveal any adversary information about the secret key, an additional security assumption is required, namely "sparse subset sum" assumption. This assumption is based on the difficulty of sparse subset sum problem (SSSP) used by Gentry [41] and studied previously in the context of server-aided cryptography [66]. For more details on this, we refer the reader to [91].

DGHV scheme is conceptually very simple but less efficient than the lattice-based scheme. Several optimizations and new variants over integers was introduced to address the efficiency problem [26, 27, 94, 18, 25, 73, 67, 22, 4].

5.6 The BGV scheme

We will describe here the RLWE instantiation of the BGV scheme [12] which has a considerably better performance compared to the LWE instantiation. Let λ and μ be security parameters. In the setup procedure, a 4-tuple of parameters $params = (q, d, N, \chi)$ is chosen, where $q = q(\lambda)$ is a μ-bit odd modulus, $d = d(\lambda)$ is a power of 2, $N > \lceil 3 \log q \rceil$ and χ is a discrete Gaussian distribution over $\mathbb{Z}$. The underlying ring of this scheme is the ring of polynomials of degree less than d with integer coefficients denoted as $R = \mathbb{Z}[x]/(x^d + 1)$. R_q is used to denote the quotient ring $R/qR = \mathbb{Z}_q[x]/(x^d + 1)$ where the coefficients of polynomials are integers modulo q.

Vectors will be written in bold lowercase letters.

SecretKeyGen: The secret key $\mathbf{s}$ is generated by drawing $\mathbf{s}' \leftarrow \chi$ and setting $\mathbf{s} = (1, \mathbf{s}') \in {R_q}^2$.

PublicKeyGen: The public key is obtained by generating a column matrix $\mathbf{A}' \leftarrow \mathbb{R}_q^{N\times 1}$ uniformly and an error vector $\mathbf{e} \leftarrow \chi^N$, and then setting $\mathbf{b} \leftarrow \mathbf{A}'\mathbf{s}' + 2\mathbf{e}$. The public key $\mathbf{A}$ is an $N \times 2$ matrix over R_q whose first column is $\mathbf{b}$ and the second column is **-**$\mathbf{A}'$.

Enc: A message $m \in R_2$ is encrypted by setting $\mathbf{m} = (m, 0) \in {R_q}^2$, generating $\mathbf{r} \leftarrow {R_2}^N$ uniformly and computing the ciphertext $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^T\mathbf{r} \in {R_q}^2$.

Dec: A ciphertext $\mathbf{c}$ is decrypted as $m \leftarrow [[\langle \mathbf{c}, \mathbf{s}\rangle]_q]_2$ which is the reduction of the dot product of $\mathbf{c}$ and $\mathbf{s}$ first modulo q (into the interval $(-q/2, q/2)$) and then modulo 2.

In order to construct a leveled homomorphic encryption scheme from the encryption scheme defined above, some operations must be defined, namely *BitDecomp*, *Powersof2*, *SwitchKeyGen*, *SwitchKey* and *Scale*.

BitDecomp$(\mathbf{x} \in R_q^n)$ operation decomposes $\mathbf{x}$ into its bit representation

$$(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{\lfloor \log q \rfloor}) \in {R_2}^{n\cdot\lceil \log q \rceil},$$

where $\mathbf{x} = \sum_{j=0}^{\lfloor \log q \rfloor} 2^j \cdot \mathbf{u}_j$ with all $\mathbf{u}_j \in R_2^n$.

Powersof2$(\mathbf{x} \in R_q^n)$ operation outputs the vector

$$(\mathbf{x}, 2\cdot \mathbf{x}, \ldots, 2^{\lfloor \log q \rfloor} \cdot \mathbf{x}) \in R_q^{n\cdot\lceil \log q \rceil}.$$

For vectors $\mathbf{c}$ and $\mathbf{s}$ of equal length, it is easy to observe that

$$\langle \mathit{BitDecomp}(\mathbf{c}, q), \mathit{Powersof2}(\mathbf{s}, q)\rangle = \langle \mathbf{c}, \mathbf{s}\rangle \pmod q.$$

Key switching method consists of two procedures described below.

SwitchKeyGen$(\mathbf{s}_1 \in R_q^{n_1}, \mathbf{s}_2 \in R_q^{n_2})$ operation starts by generating a public key $\mathbf{A}$ from the secret key $\mathbf{s}_2$ for $N = n_1 \cdot \lceil \log q \rceil$ as described above. Then it ouputs a (public key) matrix $\mathbf{B}$ bay adding *Powersof2*$(\mathbf{s}_1) \in R_q^N$ to the first column of the matrix A.

SwitchKey$(\mathbf{B}, \mathbf{c}_1)$ takes the ciphertext $\mathbf{c}_1$ encrypted under the secret key $\mathbf{s}_1$ and the output $\mathbf{B}$ of *SwitchKeyGen*, then outputs a new ciphertext $\mathbf{c}_2$ that encrypts the same message under $\mathbf{s}_2$, namely

$$c_2 = \mathit{BitDecomp}(\mathbf{c}_1)^T \cdot \mathbf{B} \in R_q^{n_2},$$

where n_2 is the dimension of $\mathbf{s}_2$.

Finally, for the sake of completeness, the *Scale* operation must be defined.

Scale$(\mathbf{x}, q, p, r)$ outputs $\mathbf{x}'$ defined as the R-vector closest to $(p/q) \cdot \mathbf{x}$ that satisfies $\mathbf{x}' = \mathbf{x} \pmod r$, where $q > p$.

Let $\mathbf{c}$ be a valid encryption of m under the secret key $\mathbf{s}$ modulo q (i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$) and let $\mathbf{s}$ be a short vector. Further let $\mathbf{c}'$ be a simple scaling of $\mathbf{c}$, that is the R-vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. It turns out that $\mathbf{c}'$ is a valid encryption of m under $\mathbf{s}$ modulo $p < q$ using the usual decryption equation (i.e., $m = [[\langle \mathbf{c}', \mathbf{s} \rangle]_p]_2$). In a nutshell, it is possible to change the inner modulus in the decryption equation to a smaller number while preserving the correctness of decrption under the same secret key. An evaluator, who does not know the secret key but only knows a bound on its length, can transform a ciphertext $\mathbf{c}$ satisfying $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$ into a ciphertext $\mathbf{c}'$ satisfying $m = [[\langle \mathbf{c}', \mathbf{s} \rangle]_p]_2$ (see Lemma 5 in [12]). Most interestingly, if $\mathbf{s}$ has coefficients that are small in relation to q and p is sufficiently smaller than q, then the magnitude of the noise in the ciphertext essentially decreases (Corollary 1 in [12])

$$|[\langle \mathbf{c}', \mathbf{s} \rangle]_p| < |[\langle \mathbf{c}, \mathbf{s} \rangle]_q|.$$

Given the scheme and operations described above, it is now possible to define a leveled FHE scheme which can be transfromed into a FHE scheme by using Gentry's bootstrapping technique.

Let L be a parameter indicating the number of levels of arithmetic circuit that the FHE scheme is capable of evaluating. Further let $\mu = \mu(\lambda, L)$, where λ is the security parameter. The setup procedure defined previously must be called from L(input level of circuit) to 0 (output level) in order to obtain a ladder of parameters. Namely, $\mathit{params}_j = (q_j, d, N_j, \chi)$ where $q_l > q_{L-1} > \cdots > q_1 > q_0$ has size $(j+1)\mu$ bits and $N_j > \lceil 3 \log q_j \rceil$ for $j = 0, 1, \ldots, L$. The parameter sets params_j is used to generate the secret key $\mathbf{s}_j$, by executing the SecretKeyGen procedure, and the public key $\mathbf{A}_j$, by executing the PublicKeyGen procedure described earlier for ecah level $j = L, L-1, \ldots, 1, 0$. Then by tensoring $\mathbf{s}_j$ with itself, set $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$ whose coefficients are each of the product of two coefficients of $\mathbf{s}_j$ in R_{q_j}. Afterwards, set $\mathbf{s}''_j = \mathit{BitDecomp}(\mathbf{s}'_j)$ and perform $\mathbf{B}_j = \mathit{SwitchKeyGen}(\mathbf{s}''_j, s_{j-1})$. Encryption is done by carrying out the encryption operation defined before using the public keys A_j and decryption is done by executing the decryption operation defined before using the secret key $\mathbf{s}_j$. The ciphertexts in depth j of the circuit are assumed to be encrypted under $\mathbf{s}_j$ using the modulus q_j. Homomorphic addition and multiplication operations are executed on the ciphertexts, and after performing each operation, a function named *Refresh* is called. *Refresh* calls the *Scale* function to switch the moduli and then invokes the *SwitchKey* function to switch the key under which the resulting ciphertext is encrypted. Indeed, since addition increases the noise much more slowly than multiplication, it is not necessarily required to refresh after additions.

5.7 The CKKS scheme

Cheon et al. [21] proposed CKKS scheme in 2017 for efficient approximate computation on encrypted data. The CKKS algorithm works in the ring of polynomials with integer coefficients modulo the mth cyclotomic polynomial $\Phi_m(x)$ that is $R = \mathbb{Z}[x]/(\Phi_m(x))$. The degree of $\Phi_m(x)$ is $n = \phi(m)$, where ϕ is the Euler's totient function. In the ring $R_q = \mathbb{Z}_q[x]/(\Phi_m(x))$, the elements are polynomials whose degree is up to $n-1$ with coefficients in the range $(-q/2, q/2]$. If $\zeta = e^{\frac{2\pi i}{m}}$ is a primitive mth root of unity, then the mth cyclotomic polynomial is

$$\Phi_m(x) = \prod_{\substack{1 \leq j \leq m \\ \gcd(j,m)=1}} (x - \zeta^j).$$

In CKKS, $m \geq 2$ is taken as a power of 2. Then $\Phi_m(x) = x^{m/2}+1 = x^n+1$. Before encryption and after decryption of CKKS scheme, encoding and decoding functions are called, respectively. Consider the canonical embedding map

$$\begin{array}{rcl} \sigma : R & \rightarrow & \mathbb{C}^n \\ a(x) & \mapsto & (a(\zeta^j))_{j \in \mathbb{Z}_m^*}, \end{array}$$

where the second half of the complex values in the image vector $\sigma(a)$ are the symmetric complex conjugates of the first half. So we can project the image vectors onto their first half via the natural projction $\pi : \mathbb{C}^n \rightarrow \mathbb{C}^{n/2}$. Then the decoding function transforms an arbitrary polynomial $a(x) \in R$ into a complex vector $\mathbf{z}$ such that $\mathbf{z} = \pi \circ \sigma(a) \in \mathbb{C}^{n/2}$. The encoding function is defined as the inverse of this decoding function. Specifically, it encodes an input vector $\mathbf{z} \in \mathbb{C}^{n/2}$ into a polynomial $a(x) = \sigma^{-1} \circ \pi^{-1}(\mathbf{z})$.

The L-infinity norm of $\sigma(a)$ for $a \in R$ is denoted by $||a||_\infty = ||\sigma(a)||_\infty$, which is equal to the largest of the absolute value of the complex components of the vector $\sigma(a)$. Following notations in [21], we define three distributions as follows. Given a real $\gamma > 0$, $\mathcal{DG}(\gamma^2)$ denotes a distribution over $\mathbb{Z}^n$ which samples it components independently from the discrete Gaussian distribution of variance γ^2. For a positive integer h, $\mathcal{HWT}(h)$ denotes uniform distribution over the set of vectors in $\{0, +1, -1\}^n$ whose Hamming weight is exactly h. For a real $o \leq \rho \leq 1$, the distribution $\mathcal{ZO}(\rho)$ draws each vector from $\{0, +1, -1\}^n$ with probability $\rho/2$ for each of $+1$ and -1, and probability of being zero is $1 - \rho$.

The aim is to construct a leveled HE scheme for approximate arithmetic. Let the integer L be the depth of the arithmetic circuit to be evaluated homomorphically and

$p > 0$ be a base. The ciphertext modulus is $q_k = p^k$ for each level $k = 1, \ldots, L$. Parameters for level k come from $\mathbb{Z}_{q^k}[x]/(x^n + 1)$ for each $k = 1, \ldots, L$. The input level of the arithmetic circuit uses the modulus $q_L = p^L$, and the next level uses $q_{L-1} = p^{L-1}$ and so on. The output level uses the modulus $q_1 = p$.

SecretKeyGen: For $O(2^\lambda)$ security, we choose the parameters of the scheme as a power of two $m = 2n$, a real value γ, an integer h, an integer P, and the base p.

Then we sample $\mathbf{s} \leftarrow \mathcal{HWT}(h)$, $\mathbf{a} \leftarrow R_{q_L}$, $\mathbf{a}' \leftarrow R_{P \cdot q_L}$, $\mathbf{e} \leftarrow \mathcal{DG}(\gamma^2)$ and $\mathbf{e}' \leftarrow \mathcal{DG}(\gamma^2)$ to generate the following secret key **sk**, the public key **pk** and the evaluation key **evk**, respectively.

$$\begin{aligned} \mathbf{sk} &= (1, \mathbf{s}) \\ \mathbf{pk} &= (\mathbf{b}, \mathbf{a}) \in R_{q_L}^2 \text{ where } \mathbf{b} = -\mathbf{a} \cdot \mathbf{s} + \mathbf{e} \pmod{q_L} \\ \mathbf{evk} &= (\mathbf{b}', \mathbf{a}') \in R_{Pq_L}^2 \text{ where } \mathbf{b}' = -\mathbf{a} \cdot \mathbf{s} + \mathbf{e} + P\mathbf{s}^2 \pmod{P \cdot q_L}. \end{aligned}$$

Note that vectors above also represents polynomials whose coefficients are the components of the corresponding vector. So the vectors are multiplied as polynomials in the corresponding polynomial ring and then written back as a vector.

Enc: After encoding an input message $\mathbf{z} \in \mathbb{C}^{n/2}$ into the plaintext $\mathbf{m} \in R$ using the procedure described previously, and sampling $\mathbf{v} \leftarrow \mathcal{ZO}(0.5)$ and $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \mathcal{DG}(\gamma^2)$, we compute the ciphertextvia the encryption function $E_{\mathbf{pk}}$ as

$$\mathbf{c} = E_{\mathbf{pk}}(\mathbf{m}) = \mathbf{v} \cdot \mathbf{pk} + (\mathbf{m} + \mathbf{e}_0, \mathbf{e}_1) \pmod{q_L}.$$

Dec: The plaintext polynomial $\mathbf{m}$ is computed from a ciphertext $\mathbf{c}$ in level k via the decryption function $D_{\mathbf{sk}}$ as

$$\mathbf{m} = D_{\mathbf{sk}}(\mathbf{c}) = \langle \mathbf{c}, \mathbf{sk} \rangle \pmod{q_k}.$$

The CKKS algorithm introduces an error so that the decrypted value is not exactly the same as the input value, indeed we have

$$D_{\mathbf{sk}}(E_{\mathbf{pk}}(\mathbf{m})) \approx \mathbf{m}.$$

During the evaluation of the arithmetic circuit, the CKKS algorithm performs homomorphic addition, homomorphic multiplication, and rescale operations.

The homomorphic addition of two ciphertexts $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ and $\mathbf{c}' = (\mathbf{c}'_0, \mathbf{c}'_1)$ in the same circuit level k is performed using

$$\begin{aligned} \mathbf{c}_{add} &= \mathbf{c} + \mathbf{c}' \pmod{q_k} \\ &= (\mathbf{c}_0, \mathbf{c}_1) + (\mathbf{c}'_0, \mathbf{c}'_1) \pmod{q_k} \\ (\mathbf{d}_0, \mathbf{d}_1) &= (\mathbf{c}_0 + \mathbf{c}'_0, \mathbf{c}_1 + \mathbf{c}'_1) \pmod{q_k}. \end{aligned}$$

Here the input values $\mathbf{c}_0, \mathbf{c}'_0, \mathbf{c}_1, \mathbf{c}'_1$and the output values $\mathbf{d}_0, \mathbf{d}_1$ are the elements of the ring R_{q_k} and the arithmetic is performed in this polynomial ring.

The homomorphic multiplication of two ciphertexts $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ and $\mathbf{c}' = (\mathbf{c}'_0, \mathbf{c}'_1)$ in the same circuit level k is performed using

$$\begin{aligned} \mathbf{c}_{mult} &= \mathbf{c} \odot \mathbf{c}' \pmod{q_k} \\ &= (\mathbf{d}_0, \mathbf{d}_1) + \lfloor P^{-1} \cdot \mathbf{d}_2 \cdot \mathbf{evk} \rceil \pmod{q_k} \end{aligned}$$

where $(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) = (\mathbf{c}_0 \cdot \mathbf{c}'_0, \mathbf{c}_0 \cdot \mathbf{c}'_1 + \mathbf{c}'_0 \cdot \mathbf{c}_1, \mathbf{c}_1 \cdot \mathbf{c}'_1) \pmod{q_k}$ and $\lfloor \cdot \rceil$ stands for rounding to the nearest integer. The output components of $\mathbf{c}_{mult}$ are also the elements of the ring R_{q_k} and the arithmetic is performed in this ring.

Rescale operation $Rescale_{k \to k'}(\mathbf{c})$ transfroms the ciphertext $\mathbf{c}$ from level k to level k' by computing

$$\begin{aligned} \mathbf{c}' &= \lfloor p^{k'-k} \cdot \mathbf{c} \rceil \pmod{q_{k'}} \\ (\mathbf{c}'_0, \mathbf{c}'_1) &= \lfloor p^{k'-k} \cdot (\mathbf{c}_0, \mathbf{c}_1) \rceil \pmod{q_{k'}} \\ &= (\lfloor p^{k'-k} \cdot \mathbf{c}_0 \rceil, \lfloor p^{k'-k} \cdot \mathbf{c}_1 \rceil) \pmod{q_{k'}} \end{aligned}$$

Generally, $k' = k - 1$, and therefore, the resclae transforms $\mathbf{c}$ from k to $k - 1$ (one level closer to the output level)

$$\begin{aligned} \mathbf{c}' &= \lfloor p^{-1} \cdot \mathbf{c} \rceil \pmod{q_{k-1}} \\ (\mathbf{c}'_0, \mathbf{c}'_1) &= \lfloor p^{-1} \cdot (\mathbf{c}_0, \mathbf{c}_1) \rceil \pmod{q_{k-1}} \\ &= (\lfloor \mathbf{c}_0 / p \rceil, \lfloor \mathbf{c}_1 / p \rceil) \pmod{q_{k-1}} \end{aligned}$$

6 Conclusions

This paper presented an extensive summary of the evolution of cryptography since Shannon's seminal paper "Communication Theory of Secrecy Systems" [80]. The first milestone point is the development of secret-key cryptographic methods LUCIFER, DES, and AES [38, 63, 65], that started in 1958 and continue to-day. The second milestone was the invention of public-key cryptography, starting with Diffie-Hellman key exchange [32] and Rivest-Shamir-Adleman [76] between 1976-1978. Followed up public-key cryptography, a variety of post-quantum cryptographic (PQC) algorithms [58] have been developed, that are expected to make us safe with the advent of quantum computers. Then, we have partially homomorphic encryption (HE) methods [70] that have been flourishing since the day public-key cryptography was invented, and finally fully-homomorphic encryption methods which are based on the ideas of Craig Gentry [41]. The PQC and HE methods are the two directions cryptographic research and development will move on in the next 2 decades.

Our interest in cryptography is as old as the invention of writing, and it is doubtful this fascination will vane. There will be many information security challenges ahead, and we will attempt to understand and bring solutions for them using cryptographic ideas and tools.

References

[1] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard, 2018.

[2] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.

[3] J. C. Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1988.

[4] D. Benarroch, Z. Brakerski, and T. Lepoint. Fhe over the integers: Decomposed and batched in the post-quantum regime. In S. Fehr, editor, *Public-Key Cryptography – PKC 2017*, pages 271–301. Springer, LNCS N. 10175, 2017.

[5] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[6] D. J. Bernstein, T. Lange, and C. Peters. Attacking and defending the McEliece cryptosystem. In J. Buchmann and J. Ding, editors, *Post-Quantum Cryptography-PQCrypto 2008*, pages 31–46. Springer, LNCS Nr. 5299, 2008.

[7] J. Biasse, D. Jao, and A. Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In W. Meier and D. Mukhopadhyay, editors, *INDOCRYPT 2014*, pages 428–442. Springer, LNCS Nr. 8885, 2014.

[8] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer Verlag, 1993.

[9] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *IMACC 2013*, pages 45–64. Springer, LNCS Nr. 8308, 2013.

[10] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In A. Canteaut and F.-X. Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617. Springer, LNCS Nr. 12696, 2021.

[11] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, pages 868–886. Springer, LNCS Nr. 7417, 2012.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) Fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3), 2014.

[13] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard)-LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106. IEEE, 2011.

[14] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard)-LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106. IEEE, 2011.

[15] R. Broker. Constructing supersingular elliptic curves. *Journal of Combinatorics and Number Theory 1*, 1(3):269 –273, 2009.

[16] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. Csidh: An efficient post-quantum commutative group action. In T. Peyrin and S. Galbratith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427. Springer, LNCS Nr. 11274, 2018.

[17] H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT*, pages 34–54. Springer, LNCS Nr. 11477, 2019.

[18] J. H. Cheon, J. S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 315–335. Springer, LNCS Nr. 7881, 2013.

[19] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT*, pages 360–384. Springer, LNCS Nr. 10820, 2018.

[20] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full RNS variant of approximate homomorphic encryption. In C. Cid and M. J. Jacobson, editors, *Selected Areas in Cryptography – SAC 2018*, pages 347–368. Springer, LNCS Nr. 11349, 2018.

[21] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *ASIACRYPT*, pages 409–437. Springer, LNCS Nr. 10624, 2017.

[22] J. H. Cheon and D. Stehlé. Fully homomophic encryption over the integers revisited. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 513–536. Springer, LNCS Nr. 9056, 2015.

[23] A. M. Childs, D. Jao, and V. Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

[24] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33. Springer, LNCS Nr. 10031, 2016.

[25] J. S. Coron, T. Lepoint, and M. Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In H. Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, pages 311–328. Springer, LNCS Nr. 8383, 2014.

[26] J. S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 487–504. Springer, 2011.

[27] J. S. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 446–464. Springer, LNCS Nr. 7237, 2012.

[28] J. Couveignes. Hard homogeneous spaces. IACR Cryptology ePrint Archive, Report 2006/291 `https://eprint.iacr.org/2006/291`, 2006.

[29] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer Verlag, 2002.

[30] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In K. Kim, editor, *Public Key Cryptography*, pages 119–136. Springer, 2001.

[31] C. Delfs and S. D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78(2):425 –440, 2016.

[32] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, November 1976.

[33] J. F. Dooley. *History of Cryptography and Cryptanalysis*. Springer, 2018.

[34] L. Ducas and D. Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 617–640. Springer, LNCS Nr. 9056, 2005.

[35] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Crypto – Advances in Cryptology*, pages 10–18. Springer, 1985.

[36] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. IACR ePrint Archive, 144, `http://eprint.iacr.org/2012/144`, 2012.

[37] H. Feistel, W. A. Notz, and J. L. Smith. Some cryptographic technique for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545–1554, 1975.

[38] H. A. Feistel. A survey of problems inauthenticated communication and control. Report, MIT Lincoln Laboratory, 20 May 1958.

[39] L. De Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[40] S. D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[41] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[42] C. Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 116–137. Springer, LNCS Nr. 6223, 2010.

[43] C. Gentry and S. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 107–109, 2011.

[44] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 129–148. Springer, LNCS Nr. 6632, 2011.

[45] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[46] V. D. Goppa. A new class of linear error-correcting codes. *Probl. Peredachi Inf.*, 6(3):24–30, 1970.

[47] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. In S. Jarecki, editor, *Topics in Cryptology - CT-RSA 2020*, pages 364–390. Springer, LNCS Nr. 12006, 2020.

[48] M. E. Hellman. DES will be totally insecure within ten years. *IEEE Spectrum*, 16(7):32–40, 1979.

[49] N. Howgrave-Graham. Approximate integer common divisors. In Joseph H. Silverman, editor, *Cryptography and Lattices - CaLC 2001*, pages 51–66. Springer, LNCS Nr. 2146, 2001.

[50] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, G. Pereira, K. Karabina, , and A. Hutchinson. Sike. National Institute of Standards and Technology Report 2020 `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`, 2020.

[51] D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In BY. Yang, editor, *PQCrypto 2011*, pages 19–34. Springer, LNCS Nr. 7071, 2011.

[52] A. G. Konheim. *Cryptography, A Primer*. John Wiley & Sons, 1981.

[53] A. G. Konheim. Automated teller machines: their history and authentication protocols. *Journal of Cryptographic Engineering*, 6(1):1–29, April 2016.

[54] A. G. Konheim. Horst Feistel: the inventor of LUCIFER, the cryptographic algorithm that changed cryptology. *Journal of Cryptographic Engineering*, 9(1):85–100, April 2019.

[55] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.

[56] T. Lange. Sd8 (post-quantum cryptography) - part 4: Code-based cryptography. ISO/IEC JTC 1/SC 27/WG 2 Cryptography and Security Mechanisms Convenorship: JISC (Japan)- Standing Document `https://www.din.de/resource/blob/721042/4f1941ac1de9685115cf53bc1a14ac61/sc27wg2-sd8-data.zip`, 2020.

[57] Y. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No. Near-optimal polynomial for modulus reduction using L2-norm for approximate homomorphic encryption. *IEEE Access*, 8:144321–144330, 2020.

[58] Z. Liu, P. Longa, and Ç. K. Koç. Guest editors' introduction to the special issue on cryptographic engineering in a post-quantum world: State of the art advances. *IEEE Transactions on Computers*, 67(11):1532–1534, 2018.

[59] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC '12: Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[60] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, pages 1–23. Springer, LNCS Nr. 6110, 2010.

[61] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 44:114–116, 1978.

[62] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, page 59–66. ACM, 1998.

[63] National Institute for Standards and Technology. Data Encryption Standard, FIPS 46 1977.

[64] National Institute for Standards and Technology. Data Encryption Standard. `https://nvlpubs.nist.gov/nistpubs/sp958-lide/250-253.pdf`, 1999.

[65] National Institute for Standards and Technology. Advanced Encryption Standard (AES), FIPS 197, November 2001.

[66] P. Q. Nguyen and I. Shparlinski. On the insecurity of a server-aided rsa protocol. In C. Boyd, editor, *ASIACRYPT*, pages 21–35. Springer, LNCS Nr. 2248, 2001.

[67] K. Nuida and K. Kurosawa. (batch) fully homomorphic encryption over integers for non-binary message spaces. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 537–555. Springer, LNCS Nr. 9056, 2015.

[68] N. Ogura, G. Yamamoto, T. Kobayashi, and S. Uchiyama. Fully homomorphic encryption with relatively small key and ciphertext sizes. In I. Echizen, N. Kunihiro, and R. Sasaki, editors, *Advances in Information and Computer Security – IWSEC 2010*, pages 70–83. Springer, LNCS Nr. 6434, 2010.

[69] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 308–318. Springer, 1998.

[70] F. Özdemir, Z. Ö. Özger, and Ç. K. Koç. *Partially Homomorphic Encryption*. Springer, 2021.

[71] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238. Springer, 1999.

[72] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[73] Y. G. Ramaiah and G. V. Kumari. Efficient public key generation for homomorphic encryption over the integers. In V. V. Das and J. Stephen, editors, *Advances in Communication, Network, and Computing- CNC 2012*, pages 262–268. Springer, LNICST Nr. 108, 2012.

[74] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of ACM*, 56(6), 2009.

[75] R. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[76] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[77] A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. IACR Cryptology ePrint Archive, Report 2006/145 `https://eprint.iacr.org/2006/145`, 2006.

[78] R. Rothblum. Homomorphic encryption: From private-key to public-key. In Y. Ishai, editor, *Theory of Cryptography - TCC 2011*, pages 219–234. Springer, 2011.

[79] P. Scholl and N. P. Smart. Improved key generation for gentry's fully homomorphic encryption scheme. In L. Chen, editor, *Cryptography and Coding – IMACC 2011*, pages 10–22. Springer, LNCS Nr. 7089, 2011.

[80] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, October 1949.

[81] P. W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In Leonard M. Adleman and Ming-Deh A. Huang, editors, *Proceedings of Algorithmic Number Theory, LNCS 877*. Springer, 1994.

[82] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[83] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[84] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 2009.

[85] N. P. Smart and F. Vercauteren. Fully homomorphic encrypton with relatively small key and ciphertext sizes. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography - PKC 2010*, pages 420–443. Springer, LNCS Nr. 5056, 2010.

[86] D. Stehlé and R. Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT 2010*, pages 377–394. Springer, LNCS Nr. 6477, 2010.

[87] A. Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215–235, 2010.

[88] M. Swayne. The world's top 12 quantum computing research universities. `https://thequantumdaily.com/2019/11/18/the-worlds-top-12-quantum-computing-research-universities`, November 18, 2019.

[89] S. Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

[90] J. Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2:134–144, 1966.

[91] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encrypton over the integers. In *EUROCRYPT 2010*, pages 24–43. Springer, LNCS Nr. 6110, 2010.

[92] J. Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sc. Paris, Série A.*, 273:238–241, 1971.

[93] T. Wu, H. Wang, and Y.-P. Liu. Optimizations of Brakerski's fully homomorphic encryption scheme. In *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, pages 2000–2005, 2012.

[94] H. M. Yang, Q. Shia, X. Wang, and D. Tang. A new somewhat homomorphic encryption scheme over integers. In *2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring*, pages 61–64, 2012.

[95] S. Zhang. Promised and distributed quantum search. In L. Wang, editor, *Computing and Combinatorics - COCOON 2005*, pages 430–439. Springer, LNCS Nr. 3595, 2005.

[96] X. Zhang, C. Xu, C. Jin, R. Xie, and J. Zhao. Efficient fully homomorphic encryption from RLWE with an extension to a threshold encryption scheme. *Future Generation Computer Systems*, 36:180–186, 2014.