 doi:10.20944/preprints202207.0389.v1

Comparison of Modern Cryptography Methods

Amandip Dutta[1]

[1]Independent Researcher

Correspondence: amandip.dutta@gmail.com

Abstract

In this paper, different cryptography algorithms were reviewed, and their relative merits were discussed. Symmetric and asymmetric cryptography algorithms were compared for time and space efficiencies. These experiments were run to measure the execution time and the memory footprints of various algorithms to understand the implications of their real-life applications. In the experiment performed, it was observed that elliptic curve cryptography was the most efficient in terms of encryption/decryption time, as well as memory usage.

1 Introduction

With the emergence of computers in the 20th and 21st centuries, there was a need to address security. Software engineers and computer programmers have designed software systems to help and address security. Specifically, engineers have focused on trying to conceal their data as it is transmitted through the public internet, thus making it hard for unauthorized users to read and understand the data. This is formally known as cryptography, and it has come to be the foundation of computer security. The objective of cryptography is to enable specific people to communicate over a channel where others cannot understand what is being sent in it [27]. The need for private online communication in applications involving electronic transactions and wireless communications led to cryptography becoming more widespread [26]. There are two main parts to cryptography: encryption and decryption. In encryption, a normal message (plaintext) is transformed into a ciphered message (ciphertext). In decryption, the ciphertext gets changed back to the plaintext [2]. The two main types of algorithms in cryptography are symmetric and asymmetric. Symmetric algorithms use the same private key to both encrypt and decrypt messages. Asymmetric algorithms require different keys (cannot be derived from each other) to encrypt and decrypt [26]. The figure below explains how symmetric algorithms use the same secret key, while asymmetric algorithms require a public key for encryption and a private key for decryption:

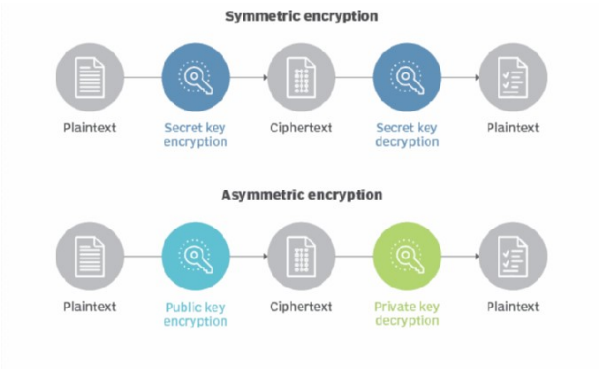

Figure 1: A Comparison of Symmetric and Asymmetric Algorithms [7]

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

Due to the different variations of cryptography algorithms, many different algorithms have been created in the last 50 years. Most of the ones created have been asymmetric, and more recently symmetric methods, but some algorithms aren't symmetric or asymmetric. For example, SHA-256, a cryptography hash function that was created by the NSA, can only encrypt plaintexts, and not decrypt ciphertexts [22].

In this paper, we looked at RSA (Rivest, Shamir, Adelman), ECC (Elliptic Curve Cryptography), ElGamal, and Blowfish, all of which are asymmetric methods except for Blowfish (symmetric). Specifically, we looked at the advantages and flaws of each method and compared them based on execution time and memory efficiency. We then conducted an experiment in which we wrote code to compare the encryption times, decryption times and memory usage of the different algorithms.

2 Related Work

Many researchers have compared cryptography methods before. Especially since algorithms have to be used with inputs and values that are thousands or potentially millions of characters long, algorithms and techniques need to be compared to determine how efficient they are at certain tasks. For example, in [6], Siahaan, Elviwani, and Oktaviana compared RSA and ElGamal in terms of encryption and decryption speeds. They analyzed and concluded that RSA is faster than ElGamal as the generated RSA ciphertext has fewer characters than the ElGamal ciphertext. However, they noted that ElGamal will be more challenging to solve as it requires more complicated calculations. In addition, Yadav and Mahto of the National Institute of Technology Jamshedpur compared RSA and ECC in terms of memory consumption. They analyzed that ECC would perform better than RSA on memory-constrained devices, as ECC requires less memory to run [19]. Lastly, researchers Ahmed, Ali, Maqsood, and Shah compared the Advanced Encryption Standard, the Data Encryption Standard, RSA, and ElGamal for encryption/decryption times, key generation, and file size. Their results show that AES (Advanced Encryption Standard) surpassed all other methods in terms of time, file size efficiency, and key generation [2]. As more cryptographic algorithms are developed, more research and analysis need to be done to understand the impact of the algorithms.

3 Chronology

3.1 History

Cryptography was first cited in Circa 600 BC when Spartans used a device called a "scytale" to send encoded messages during battle. Despite this, the most significant and earliest discovery of the scytale was in 1917, when Edward Hebern invented the electromechanical machine. The electromechanical machine contained the mechanical and automatic components of a typewriter while having a rotor that connects through a scrambler [15]. Only a year later, a German engineer named Arthur Scherbius invented a device similar to the electro-mechanical machine, which used three rotors, which take in the inputted letters and bounces off a reflector. The reflector then pushes the letters back through the three rotors in the other direction [16]. These letters that were encoded were formally known as the enigma code.

Alan Turing, a British mathematician, and logician would later crack the enigma code. During WW2, Turing took a job where he would have to decipher military codes used by Germany and their allies. He eventually created a machine known as the "Bombe", which helped reduce the work of the "code-breakers" he had designed before. As a result, this would later be shared with the French and British by Poland, in WW2. Few people like Alan Turing were able to decode the key, which would change daily [21]. This effort along with a few others eventually led the Allies to win WW2. Shortly after the end of WW2, Claude E. Shannon of Bell Labs published an article called "A mathematical theory of cryptography" which not only shaped modern cryptography but was the first mathematical interpretation of codes and keys.

Later on in the 1970s, IBM formed a cryptography team that created block ciphers to protect IBM's

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

customers. The US will have adopted it three years later, and it would come to be known as the Data Encryption Standard (DES). In 1976, Martin Hellman and Whitfield Diffie created a paper known as the "Diffie-Hellman Key Exchange" [15]. This was the first time where no pre-arranged keys were used. Instead, they used one private key and one public key to carry out their algorithm. In the early 2000s, asymmetric key cryptography replaced DES. More recently in 2005, Elliptic Curve Cryptography was created and allowed for smaller key sizes, as well as fast decryption times. As a result, it became more difficult to break than RSA, and the Diffie-Hellman key exchange as brute-forcing ECC was a lot more difficult as it uses only the prime number points on a cubic function [29]. The rise of so many different cryptography methods has led me to research and compare the different methods in terms of time, space, and efficiency.

3.2 Future

Although cryptography is relatively new, the idea of quantum computers is very probable in the near future of cryptography. Quantum computing was introduced in 1982 by Richard Feynman and has been researched to be the destructor of modern asymmetric cryptography. Due to their sheer power in brute force operations such as factorization of large primes, they pose a risk to asymmetric methods that rely on discrete logarithm [17]. As the demand for cryptographic computation grows, the computing requirements will also rise. For example, increasing key sizes (128 to 256 bits) can make symmetric algorithms less vulnerable to quantum attacks [12]. This increase in bits will likely trend towards quantum bits (qubits). Due to their susceptibility to errors, qubits suffer from bit-flips (switching zeroes and ones) and are easily affected by heat and noise in their operating environment [17]. Due to the strengths and challenges of quantum computing, cryptoanalysts are unsure as to whether or not this could be a probable future of cryptography.

4 Algorithms

4.1 RSA

4.1.1 Implementation Details

The RSA Algorithm was created by three MIT colleagues, whose names were Rivest, Shamir, and Adleman. Their approach to creating this was to take readable data and scramble it in a way such that only a person with a key made from prime numbers can decrypt it. To implement RSA, assume we have a public key (a,b), and assume an original plaintext message to be p. Assuming c is the ciphertext, we can encrypt the plaintext into ciphertext by modular exponentiation: $c = p^a \bmod b$ [3]. To decrypt, we reverse the process of encryption. A receiver can use his private key (d,n) to get the original plaintext. The same modular exponentiation is also required to retrieve the original plaintext as well. The sender must compute the modular exponentiation of c with respect to modulus n ($p = c^d \bmod e$) [3].

Because RSA is a well-known public-key system, that is increasingly being used in the world of cryptography, fast implementations of RSA are greatly needed. One implementation that is being researched is efficient modular multiplication. As explained above, RSA requires modular multiplication and performs the modulus and exponentiation in a single line ($c = p^a \bmod b$). However, if we apply Newton's method, where an expression $(xy) \bmod m$ can be reciprocally approximated [23], the modular multiplication can be shown through the following:

$$\begin{gathered} \text{let } a = xy \\ \text{which then } q = \tfrac{s}{m} \\ \text{and } r = s - qm \text{ [23]} \end{gathered}$$

Combining this method of modular exponentiation along with others such as Montgomery's method and residue number systems can greatly increase the speed of RSA implementation.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

4.1.2 Advantages

RSA has numerous benefits as it is one of the most commonly-used methods. For example, it has larger key sizes compared to more modern methods (ECC) for better and stronger security of data [19]. Due to its larger key size, its encryption speed doesn't significantly change from larger keys as shown in the graph below:

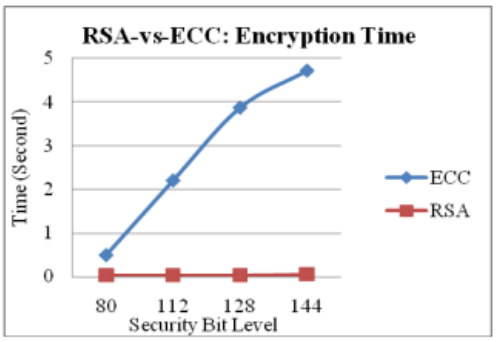

Figure 2: A Comparison of RSA and ECC Encryption Times [19]

Additionally, constructing larger keys is relatively easy as one can increase the modulus or exponents of the public key. Therefore, one can use a padding scheme that makes the message larger by adding random elements and thus, makes it harder to compute and hack using math or brute force operations. If an attacker would decrypt encryption that is padded, they would most likely get decryption far different than the actual decryption.

4.1.3 Flaws

Despite its strengths, RSA has flaws as well and may be replaced by other asymmetric methods such as ECC. For example, its decryption is relatively slow compared to ECC (reference chart). Due to RSA being so widely popular, several attacks have occurred on it. For example, a plaintext attack is when the attacker knows some of the blocks of the original message. With this, the attacker could try to encrypt the blocks he knows, and try to convert them into ciphertexts. As a result, it makes it easier for the attacker to discover the original message. To combat this, many companies use padding bits to confuse the attacker (characters that fill up unused portions of data) [24]. For example, if one would need to pad a 9-bit message such as 110110000 into a 16-bit message, one would first append a "1" to the end of the message, followed by zeros. This would result in 1101100001000000. Another type of padding that is used is Trailing Bit Complement Padding (TBC Padding) [24]. If the end of the message ends in a "0", 1s are appended, and vice-versa.

	Original	Padded
0-Bit Ending	1100	110011111
1-Bit Ending	1101	110100000

Table 1: Trailing Bit Complement Padding

4.2 ECC

4.2.1 Implementation Details

ECC generates an elliptic curve based on the level of encryption (the larger the curve, the more secure the encryption). An elliptic curve E over a finite field F is the set of all points lying on the curve: $y^2 = x^3 + ax + b$ [25]. It requires finite fields, therefore the only way to solve encryption is by trying random integers, which makes brute-forcing extremely difficult. It includes two keys, one public, and one private key. To generate a key, choose a suitable curve in which the global parameters fall in, and select the base point $P = (x_1, y_1)$, from which we can choose private keys as long as they are in the range from one to n (cannot be infinite) [25]. To generate a public key, we can use the formula $P_k = Pr_k * P$, where P_k is the public key, Pr_k is the private key, and P is a global parameter [25]. P must be on the curve as well.

4.2.2 Advantages

Although this method is relatively new, cryptoanalysts have been able to compare its efficiencies to other cryptography methods. ECC contains smaller security bits than RSA, thus, point-multiplication is better than RSA private key operation. It has also been shown to provide the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

same level of security as RSA, yet using shorter keys [19]. Additionally, it can be adapted to lots of different cryptographic schemes and protocols like the Elliptic Curve Digital Signature Algorithm. As a result, it could be used for high-level security decryption as its use of points on a curve makes it harder to brute force [28]. Victor S. Miller proposed an encryption scheme that used ECC and was faster than the Diffie-Hellman key exchange protocol by around 20 percent. In addition, Neal Koblitz, Alfred Menezes, and Scott Vanstone were able to implement the discrete logarithm problem in ECC, which provided smaller block spikes, faster speeds and higher security [33].

4.2.3 Flaws

Despite crypto analysts claiming that ECC will be the future of cryptography, there are many flaws in ECC. For example, the encryption is relatively slow compared to RSA. It is also prone to twist-security attacks (leakage of the victim's private key). In future quantum computers, ECC will be easier to break than RSA cryptosystems because ECC will have lower qubits (quantum equivalents for bits). As a result, incorrect implementations can lead to ECC privacy key leaks, which can cause branch or cache-timing errors [19]. In addition, longer ECC keys can be broken into, as timing attacks are very apparent. Specifically, timing attacks can leak execution time, which can lead to even more timing attacks happening in the future. [8] Other side-channel attacks can leak power consumption and electromagnetic emanation (exposure) [19].

Power attacks are also similar to timing attacks except that voltage peaks are analyzed by the hacker. Another type of hack that can occur is fault attacks, also known as twist-security hacks. During this attack, the attacker shares a selected public key that does not lie on the ECC curve [28]. Once the victim creates a shared key with his private key, and the attacker's public key, the attackers can then extract the victim's secret key.

One we may encounter is known as a "Grover attack" where an attacker creates a superposition over all possible inputs and continually destroys invalid states, which results in finding inputs that satisfy a given function. Due to the introduction of quantum computers, it can also result in brute force attacks, but that will have to overcome very difficult physical and hardware limitations [28].

4.3 ElGamal

4.3.1 Implementation Details

In ElGamal, a public-key cryptosystem, the encryption key is published, but the decryption key is kept private. The mathematical relationship between encryption and decryption cannot be exploited easily because there are many different inputs one can decrypt from a single output. The mathematical relationship between encryption and decryption relies on the discrete logarithm problem [1].

To encrypt with ElGamal, assume one has a prime number n and the public key (p,m,k). Assuming one has a random integer a, and the intended message to send is b. The ciphertext pair would be as shown down below:

To decrypt with ElGamal, first, calculate the following parameter ${c_1}^{p-1-z}$ where the private key is z. To recover the secret message S, multiply the calculated parameter by c_2 by using the following equation: $S = ({c_1}^{p-1-z})c_2 \bmod p$ [1].

4.3.2 Advantages

Since ElGamal uses the discrete logarithm problem, the encryption process is faster than decryption than RSA. In addition, the strength of the algorithm relies on the calculation of discrete logarithms [14]. Also, encrypting the same plaintext multiple times will result in different ciphertexts. In regards to encryption, the encryption process requires two modular exponentiations ($\alpha^k \bmod p, (\alpha^a)^k \bmod p$), which can be sped up by selecting a random exponent k [20]. Similar to the Digital Signature Algorithm, ElGamal can also support the electronically signing of messages, in addition to encryption and decryption.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

4.3.3 Flaws

The three main types of attacks that occur on ElGamal encryption and decryption are chosen-plaintext attacks, non-adaptive chosen-ciphertext attacks, and adaptive chosen-ciphertext attacks. A chosen-plaintext attack is when an attacker chooses random plaintexts and encrypts them to obtain the ciphertexts, their goal is to use the ciphertexts to gain more information regarding the original message [14]. A non-adaptive chosen-ciphertext attack is when the attacker can gather information by gathering decryptions of chosen ciphertexts, like the opposite of the chosen-plaintext attack. An adaptive version of the previous attack is when an attacker sends specific ciphertexts to be decrypted in a certain order [10].

4.4 Blowfish

4.4.1 Implementation Details

Blowfish is a key-using, symmetric cryptography block cipher designed by Bruce Schreiner. It has a 64-bit block size and a key length from 32 to 448 bits. Also, it consists of a key-expansion part and a data-encryption part [30]. The key expansion converts keys into different subkeys. It is then encrypted using modulo 232 and XORed (eXclusive OR) to create a 32-bit output as shown below:

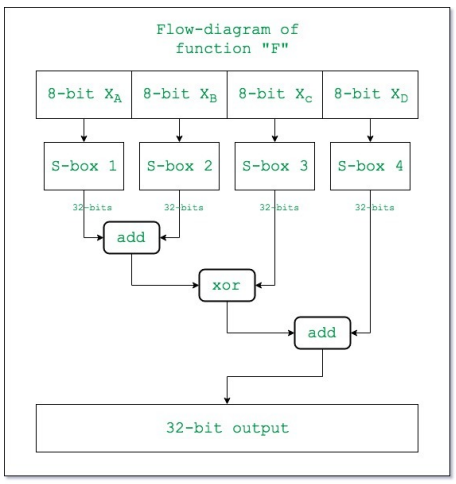

Figure 4: Key Expansion in Blowfish [5]

In addition to Blowfish being the first symmetric method ever introduced, its implementation in hardware is also unique. In addition to computing on a central processing unit (CPU), cryptoanalysts have been able to implement it on graphical processing units (GPUs). They observed that even if input file size increases, the encryption and decryption time can be reduced by using GPUs [18].

4.4.2 Advantages

Blowfish is used in password management, file/disk encryption, backup tools, email encryption, operating systems, as well as secure shells. In addition, applications such as AEdit, Foopchat, and Freedom, use the Blowfish algorithm for encrypted emails and chats, as well as privacy for web browsing [32]. Due to Blowfish's fast decryption speeds, it is commonly used in password-hashing. Because of this, Blowfish is commonly used in the cryptography space as it has one of the fastest decryption speeds among most cryptography algorithms. In addition to its use in applications, Blowfish can be used for bulk encryption of data files, multimedia voice encryption, and hard disk backup [4].

4.4.3 Flaws

Some key disadvantages are that it is one of the fastest block ciphers, but very slow when changing keys. The decryption process is far slower than other algorithms in terms of time [30]. Each new key requires the equivalent of encrypting 4KB of text which prevents its use in certain applications. Each pair of users needs a new unique key, so as the number of users increases, key management becomes more complicated. In addition, it has not been tested as much as other symmetric cryptography methods such as DES [32]. The algorithm can't provide authentication as two people have the same key (since Blowfish is a symmetric algorithm). Because of Blowfish's slow decryption times and its vulnerability to plaintext attacks, crypto analysts are recommending switching to Blowfish's successor, TwoFish.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

5 Experiment

For this experiment, we gathered code from various sources that demonstrated both the encryption and decryption algorithms of RSA, ECC, ElGamal, and Blowfish. We implemented encryption and decryption timers to measure the encryption and decryption times. In addition, we implemented a memory dump function that collects the memory used from encryption and decryption. The links to the source code used can be found here: https://github.com/amandipd/Comparison-of-Modern-Cryptography-Methods-Source-Code-[9, 11, 13, 31]. We collected the data in five trials and averaged them to get a generalized time/memory size for the data. Due to system limitations and multiple processes running during data collection, there may be a small margin of error in the data.

RSA

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Encryption Time (ms)	44.16	57.03	58.32	41.00	63.45	52.79
Decryption Time (ms)	0.494	0.878	0.548	0.477	0.987	0.677
Memory Used (KB)	1392	922	1952	1960	1172	1479.6

ECC

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Encryption Time (ms)	0.855	0.577	0.907	0.682	0.757	0.756
Decryption Time (ms)	0.770	0.397	0.750	0.399	0.649	0.593
Memory Used (KB)	583	945	428	758	1043	751.4

ElGamal

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Encryption Time (ms)	1.782	2.548	2.135	2.446	3.108	2.404
Decryption Time (ms)	2.180	2.319	1.899	2.218	2.950	2.313
Memory Used (KB)	1176	1637	715	1176	715	1084

Blowfish

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Encryption Time (ms)	51.54	52.51	57.37	57.53	52.66	54.32
Decryption Time (ms)	0.201	0.190	0.248	0.375	0.189	0.241
Memory Used (KB)	1386	1637	1637	1386	1637	1536.6

These results were collected on a AMD Ryzen 9 4900HS with 8 cores, 16 threads, and 16GB of RAM.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

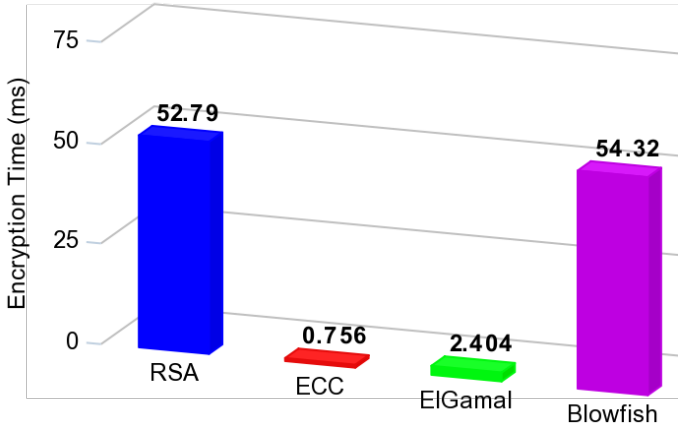

Figure 5: A Comparison of Encryption Times

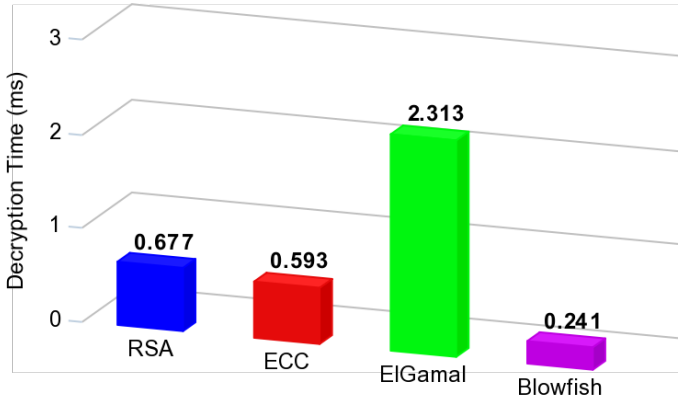

Figure 6: A Comparison of Decryption Times

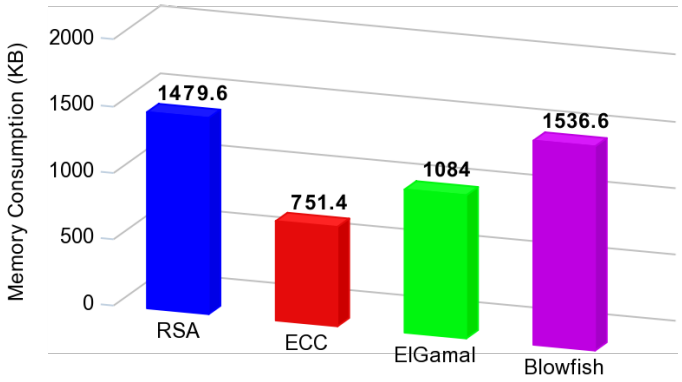

Figure 7: A Comparison of Memory Consumption

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 July 2022 doi:10.20944/preprints202207.0389.v1

6 Comparison of Cryptography Techniques

6.1 Space

Firstly, we believe that the memory usage collected from all algorithms are high as the initial source codes were not optimized for efficient memory use. With this in mind, ECC used the least memory compared to any of the other cryptography methods. Despite requiring the least memory, ElGamal's memory usage was arbitrarily similar. ElGamal used the second least amount of memory to encrypt and decrypt data. Due to its use of a random exponent to speed up the two modular exponentiations required, it yields small memory usage. However, after ECC and ElGamal, Blowfish and RSA also had similar memory consumption. However, Blowfish had the largest amount of memory consumption overall. This would explain the growth in use of asymmetric cryptography, as it requires less space to perform encryption/decryption. In server-level programs, tens of thousands of kilobytes can be wasted when using a symmetric method versus an asymmetric method.

6.2 Time

In terms of encryption speed, ECC can encrypt plaintext the fastest among any other methods, and by a large margin. ECC's encryption speed was roughly half of ElGamal and more than 50 times faster than RSA and Blowfish. Due to its low-bit keys, ECC was able to encrypt faster than any other method. In terms of decryption, the results were slightly different. Blowfish had the fastest decryption time and came second, and only around 0.3 milliseconds after Blowfish. Although this number may seem extremely small, in brute-force operations, this could save large amounts of time. ElGamal took close to one millisecond, and Blowfish took nearly two milliseconds. As a result, we can conclude that ECC and RSA had the fastest decryption times among all of the methods compared.

6.3 Efficiency

Overall, the most efficient method in terms of time and space was ECC. Although it didn't have the least memory consumption, it was the quickest in encryption and decryption. Although RSA did come close to ECC in terms of decryption speeds because of how close the times were, and the potential bloatware in the code, we tried to optimize where we collected our time. For example, we placed the timers before any other code we wrote in order to minimize the time that wasn't spent encrypting/decrypting.

```
System.out.println("c^r mod p = " + crmodp);
System.out.println("d = " + d);
System.out.println("Alice decodes: " + ad + "\n");

long end2 = System.nanoTime();
// stopping the decryption timer before memory collection code

long afterUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
long actualMemUsed = afterUsedMem-beforeUsedMem;
// calculating total memory usage

actualMemUsed = actualMemUsed / 1000;
// converting memory to Kilobytes

System.out.println("Elapsed Encryption Time in Nanoseconds: "+ (end1-start1));
System.out.println("Elapsed Decryption Time in Nanoseconds: "+ (end2-start2));
System.out.println("Memory Used in KB: " + actualMemUsed);
```

Figure 8: Memory Collection Code in ElGamal

 doi:10.20944/preprints202207.0389.v1

In addition, ElGamal was significantly more efficient than RSA and Blowfish. It used a small amount of memory and had quick encryption speeds, however, it had extremely slow decryption speeds, which explains why it is used in privacy software, as encryption would be more important than decryption. Although RSA and Blowfish were the least efficient compared to ECC and ElGamal, there are still big reasons as to why people still use these methods. RSA is one of the oldest and most secure cryptography methods to existing. In addition, Blowfish had the fastest decryption speed among all the methods.

7 Conclusion

All in all, cryptography methods have been used in computer systems to optimize security, efficiency, time, and space. The four cryptography methods we looked at were Elliptic Curve Cryptography, Rivest Shamir Adleman, ElGamal, and Blowfish. We analyzed how older methods such as RSA and Blowfish may have larger memory consumption, slower encryption times, and slower decryption times, however, they own offer their benefits. For example, RSA is commonly used in everyday applications since it laid the foundations for asymmetric cryptography and was traditionally used in Transport Layer Security. Blowfish is unique as it uses a single key to both encrypt and decrypt data. Also, it can accommodate any key length from 32 to 448 bits and can expand keys as well. Newer cryptography methods offer unique approaches to encrypt and decrypt data, while still being efficient and reliable. For example, ECC generates elliptic curves and picks prime number keys from the curve. Not only is this unique, but it makes it harder for hacks to occur as hackers would have to guess prime number possibilities. Lastly, ElGamal is unique as many different inputs can be decrypted from an output, and it requires two modular exponentiations.

In conclusion, certain cryptography methods have different encryption and decryption speeds, as well as memory usage, and are utilized accordingly based on what is needed in a program. For example, ECC is used in digital signatures, as well random-number generators, and tasks such as Lenstra elliptic curve factorization. Because of its unique strengths, it is used in security and password generation as it requires a small amount of memory and has fast encryption speeds. Due to RSA's age and reliability, it is commonly used in web browsers, VPNs, email, chat, and other communication channels as well. Especially since it has slow encryption and decryption speeds, it is not as commonly used in the security of data or password management. Because ElGamal has relatively fast encryption speeds, it is commonly used in hybrid cryptosystems, where messages are encrypted with ElGamal but are then decrypted with another cryptography method. Due to ElGamal being older and being a symmetric method, it is not as commonly used. However, it is used in some password management tools, as well as backup software.

8 Future Work

Looking forward, this experiment can be modified to better compare modern cryptography methods. For instance, longer plain texts may affect the results. Also, the time capturing method does have a margin of error, which could me made smaller by restricting the computer to only necessary applications. The addition of these ideas will further the research in comparing cryptography methods.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

blowfish, cryptography, elgamal, elliptic-curve, encryption, decryption, Rivest-Shamir-Adleman

References

[1] Wisam Najm Al-Din Abeda, Isam Salah Hameeda, Ali Thaeer Hammid, Omar A. Imran, and Sura F. Yousifa. Implementation of el-gamal algorithm for speech signals encryption and decryption. pages 1–10. ICCIDS, 2020.

[2] Muhammad Ahmed, Muhammad Mumtaz Ali, Faiqa Maqsood, and Munam Ali Shah. Cryptography: A comparative analysis for modern techniques. volume 8, pages 442–448, 2017.

[3] Rekib Uddin Ahmed, Prabir Saha, Mridupawan Sonowal, and Sheba Diamond Thabah. Fast and area efficient implementation of rsa algorithm. pages 525–526. Elsevier B.V., 2019.

[4] V. Josephraj B. Shamina Ross. Performance enhancement of blowfish encryption using rk-blowfish technique. volume 12, pages 9236–9244. Research India Publications, 2017.

[5] Abhay Bhat. Blowfish algorithm with examples. 2021.

[6] Andysah Putera Utama Siahaan Boni Oktaviana, Elviwani. Comparative analysis of rsa and elgamal cryptographic public-key algorithms. pages 1–10.

[7] Kate Brush. Asymmetric cryptography (public key cryptography). September 2021.

[8] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. volume 3, pages 241–265, 2013.

[9] Christian d'Heureuse. How to use the blowfish algorithm to encrypt 64-bit blocks with a constant key.

[10] J. Wu D.R. Stinson. On the security of the elgamal encryption scheme and damgard's variant. pages 1–15, 2014.

[11] Washington Edu. Security of elgamal.

[12] Wajdi Fegali. The future of cryptography in hardware processors. EE Times, June 2021.

[13] Geeks For Geeks. Implementation of diffie-hellman algorithm. May 2021.

[14] Jaspreet Kaur Grewal. Elgamal:public-key cryptosystem. pages 1–12, 2015.

[15] Thales Group. A brief history of encryption. 2021.

[16] Alex Hern. How did the enigma machine work: On the day the imitation game hits cinemas, a look at how allied codebreakers untangled the enigma. The Guardian, 2014.

[17] Audun Jøsang, Vasileios Mavroeidis, and Mateusz D. Zych Kamer Vishi. The impact of quantum computing on present cryptography. volume 9, pages 1–4, 2018.

[18] Mohammad Qatawneh Mahmoud Rajallah Aasassfeh. Performance evaluation of blowfish algorithm on supercomputer iman1. volume 9, pages 219–223, 2018.

[19] Dindayal Mahto and Dilip Kumar Yadav. Rsa and ecc: A comparative analysis. volume 12, pages 9053–9061. Research India Publications, 2017.

[20] Amer R. Zerek Mohamed A. Abuinjam, Amer Daeri. Elgamal public-key encryption. pages 1–3, 2014.

[21] Imperial War Museums. How alan turing cracked the enigma code. 2021.

[22] N-able. Sha-256 algorithm overview. 2019.

[23] David Pearson. A parallel implementation of rsa. pages 1–9, 1996.

[24] J. Pieprzyk S. Bakhtiar, R. Safavi-Nain. Cryptographic hash functions: A survey. page 7.

[25] J. Renita Johnson Shantha Arumugam, N. Edna Elizabeth. Analysis and implementation of ecc algorithm in lighweight device. page 6. International Conference on Communication and Signal Processing, 2019.

[26] Sarah Simpson. Cryptography defined/brief history. 1997.

[27] Douglas Robert Stinson. Cryptography theory and practice. page 1. Taylor and Francis Group, 2006.

[28] Veronika Stolbikova. Can elliptic curve cryptography be trusted? a brief analysis of the security of a popular cryptosystem. volume 3, page 5, 2016.

[29] Nick Sullivan. A (relatively easy to understand) primer on elliptic curve cryptography. 2013.

[30] Tibco. Managed file transfer platform server for z/os: Data encryption algorithms.

[31] Quick Programming Tips. Java asymmetric encryption decryption example with rsa.

[32] UKEssays. Blowfish algorithm advantages and disadvantages. November 2018.

[33] S. Vasundhara. The advantages of elliptic curve cryptography for security. volume 13, pages 4995–5011. Research India Publications, 2017.