African Journal of Academic Publishing in Science and Technology (AJAPST)

Volume 1, Issue 2, 2025
Page No: 1-15
Website: https://easrjournals.com/index.php/AJAPST/index

Securing Mobile Messaging on Android

Mabrouka Algherini *
Information Technology Department, Higher Institute of Sciences and Technology, Soukna-Aljofra, Libya

*Corresponding author: kokyrbab1@gmail.com

Received: February 02, 2024	Accepted: March 27, 2025	Published: April 09, 2025

Abstract:
As the sensitive use of SMS grows every day, researchers have been devising ways to secure it. However, some of the algorithms proposed by researchers or solutions that have flooded the market have not fully aided SMS security, making it difficult to choose appropriate solutions that align with users' needs, which is key to protecting sensitive information. Finding the right solution to secure SMS is still a problem today in sectors like banking, E-commerce and even individual usage. The idea of using encryption is not new, but employing the right solution to secure SMS is yet to be achieved by all. In this thesis, we analyzed and evaluated AES, Camellia and RC6 algorithms for securing SMS on Android platform and over the network. A comparative study was carried out on these algorithms using performance metrics: Encryption, decryption, and key generation time. The result shows that Camellia with key sizes 128, 192 and 256 bits was the fastest to encrypt a text while RC6 with key sizes 128, 192 and 256 bits was the fastest to decrypt a text. Camellia and RC6 with key sizes 128, 192 and 256 bits generate keys at almost the same speed. AES of key sizes 128, 192 and 256 bits was slower in generating keys, encrypting and decrypting a message when compared to Camellia and RC6. The findings from this research support the idea of using Camellia and RC6 more often for SMS encryption.

Keywords: SMS, algorithms, sensitive information, encryption and decryption.

Introduction

Mobile devices have emerged as a necessary technology for communicating. The demand for mobile phones is amazing that in 2010, 100.9 million mobile phones were shipped out worldwide and mobile phones is selling more than personal computers today. There are about nine popular mobile operating systems (OS) installed on mobile phones. In future, mobile phones may replace personal computers when it comes to e-mailing, instant messaging, web browsing and SMS [1]. There is a large number of mobile phone users that prefer to use short message service (SMS) as a form of communication more than mail-based service, voice call, Email and even web-based communication having possess the three important element of communication which are good response rate, fastness and inexpensive nature. SMS users tend to receive instant response when they send a message to a recipient. They experience fast message delivery which also cheaper than the voice calls for mobile communication [2].

Literature review

The first SMS message was first sent to a recipient in the United Kingdom in 1992 and ever since then the SMS has grown to become a common communication tool for the masses. The SMS mobility, good response rate, fastness and inexpensive nature make it the best bearer for mobile applications [3]. For many businesses and government agencies, SMS play a huge role in their communication strategies. Moreover, individuals also use the SMS to exchange information every day.

SMS comes with benefit such as allowing mobile phone users to swiftly send and receive various types of important data or information like bank account details, usernames and passwords, social security numbers of credit cards. These types of information or data are only meant for intended users and therefore it is important that they are kept secured [3].

Some risks crops up when using SMS. SMS can be intercepted when transmitted over the network. That is, a hacker using several techniques can tap a message that is sent over a network that does not guarantee protection of SMS. If this happens, confidential information can be exposed to unauthorized persons. Another risk is that a person can mistakenly send SMS to an unintended recipient and this will allow the wrong person to read the sender's message [4].

In our lives today, mobile phone devices have gained so much ground as becoming one of the most important communication tools. Most people around the world depend on these devices to communicate with loved ones, business partners and so on. An executive summary by GSMA titled The Mobile Economy states that as at March 2016 there are above 4.7 billion unique mobile phone subscribers on the GSM network.

GSM network mobile operators offer a wide range of services, out of those services offered the SMS stands tall as regards communication. Most people prefer SMS when communicating compared to other services available. This is because the SMS is cost-effective, fast and high response rate.

Problem formulation

Despite all the tangible and intangible benefits of SMS, SMS comes with security issues. The information that goes with the SMS can be exposed to unwanted viewers or unauthorized persons and this is a big challenge to user's privacy. This may affect our rights as extremely important data exchange like credit card social security numbers and logins to bank accounts are being seen by unauthorized persons. There are fraudulent activities that can happen when an unauthorized person has access to customer's online bank login details or credit card information. Before the invention of SMS, this would not have been a problem as this extremely confidential information would be kept safe in files with lock and keys but now can be visible in SMS (Jibril et al, 2014). There are several encryption algorithms that have been designed to solve the security issues of SMS but finding the best encryption algorithm should be key to protection of sensitive information. Using the right algorithm is an issue and a problem.

The main aim of this article is providing a study on Securing for Mobile Messaging utilizing an Android gives by a comprehensive insight on cryptography and encryption algorithms used in this research. The remain sections are classified as follows: Section 2 presenting the methodology by defining cryptography and encryption citing some examples. In section 3, the obtained results and its summary discussion have been presented along with their subsections. Eventually, the article closes by the conclusion summary followed by the list of up to date references.

Material and methods

This section gives a comprehensive insight into cryptography and encryption algorithms used in this research.

1.1 Cryptography

Cryptography is the practice and study of techniques for securing communication and information by converting it into a format that is unreadable to unauthorized users. It involves creating codes and ciphers to protect data from adversaries. Here are some key concepts in cryptography:

Table 1: key concepts in cryptography [5]

key concepts	Features
Encryption and Decryption	• Encryption: The process of converting plain text into cipher text using a specific algorithm and a key. • Decryption: The reverse process, converting cipher text back into plain text using a key.
Keys	A key is a piece of information used in algorithms to encrypt and decrypt data. The security of the cryptographic system often relies on the secrecy and complexity of the key.
Symmetric vs. Asymmetric Cryptography	• **Symmetric Cryptography**: Uses the same key for both encryption and decryption. Example: AES (Advanced Encryption Standard). • **Asymmetric Cryptography**: Uses a pair of keys — a public key for encryption and a private

	key for decryption. Example: RSA (Rivest-Shamir-Adleman).
Hash Functions	Functions that convert an input (or 'message') into a fixed-size string of bytes. The output is typically a digest that is unique to each unique input. They are commonly used in storing passwords and ensuring data integrity. Example: SHA-256 (Secure Hash Algorithm).
Digital Signatures	A method of validating the authenticity and integrity of a message or document by using asymmetric cryptography. A sender can sign a message with their private key, which can then be verified by others with their public key.
Public Key Infrastructure (PKI)	A framework that provides a set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.
Applications	Cryptography is used in securing online communications (SSL/TLS), digital currencies (like Bitcoin), secure messaging apps, and more.

1.2 Existing Encryption Algorithm in Literature

Three symmetric encryption methods are reviewed, implemented, and evaluated so that they can be compared in terms of efficiency in time. It is important to review a few kinds of literature about symmetric encryption.

A new Symmetric Algorithm called Dripto Jee Symmetric Algorithm (DJSA) using extended Mallick Saima Asoke (MSA)was introduced by Chatterjee et al. In this solution, for generating key the ideal is to use a random key generator. Keys generated will be used for encrypthe tion of required source file. Basically, a substitution process is employed in which four characters is taken from files that have input values then look for corresponding characters in the key matrix. After the whole process, ciphertext is found, it is then stored in another file.

Table 2: Summary of Key Algorithms [6].

Category	Algorithm	Key Size	Use Case
Symmetric-Key Encryption	AES	128, 192, 256 bits	Data encryption, SSL/TLS
	DES/3DES	56/168 bits	Legacy systems
	Blowfish/Twofish	32–448/128–256 bits	General-purpose encryption
	Salsa20/ChaCha20	256 bits	High-speed encryption
Asymmetric-Key Encryption	RSA	1024–4096 bits	Key exchange, digital signatures
	ECC	160–521 bits	Modern systems, blockchain
	Diffie-Hellman	1024–4096 bits	Key exchange
Hash Functions	SHA-256	256 bits	Data integrity, blockchain
	SHA-3	224–512 bits	Future-proofing
	BLAKE2	Variable	Password hashing, data integrity

1.3 Symmetric or Private Key Encryption

It is a type of encryption where the same key is used for both encrypting and decrypting data as tabulated in Table 3. It is one of the oldest and most widely used encryption methods due to its simplicity, speed, and efficiency, especially for encrypting large amounts of data.

Table 3: Comparison with Asymmetric-Key Encryption [7].

Feature	Symmetric-Key Encryption	Asymmetric-Key Encryption
Key Usage	Same key for encryption/decryption	Different keys for encryption/decryption
Speed	Faster	Slower
Key Distribution	Challenging	Easier (public keys can be shared)
Use Case	Bulk data encryption	Key exchange, digital signatures
Examples	AES, DES, Blowfish	RSA, ECC, Diffie-Hellman

1.3.1 AES or Rijndael Cipher

Equation 1 presented the AES calculation [8].

$$nr = \max\{nb, nk\} + 6 \tag{1}$$

Calculate the subkeys as in equation 2 :

$$K_0, K_1, \ldots. K_n, from\ t\square e\ key\ K \quad (2)$$

Compute the state by adding the plaintext block *B* and the key *K* as in equation 3:

$$S = B \oplus K_0 \quad (3)$$

For *i*= 1 to *nr*– 1

For the Sub-bytes round, each byte of the block is replaced by its substitute in an S-box.as in equation 4:

$$S = SubBytes(S) \quad (4)$$

For the Shift-Row round, the block are made up of bytes 1 to 16 and shifted as in equation 5.

$$S = ShiftRow(S) \quad (5)$$

For the Mix-Column round, the matrix multiplication is performed as in equation 6:

$$S = MixColumn(S) \quad (6)$$

XOR the Add-RoundKey in the subkey as in equation 7:

$$S = K_i \oplus S \quad (7)$$

Repeat the Sub-Bytes and the Shift-Row rounds in order, respectively as in equations 4 and 5 the finally XOR the Add-Round Key in the subkey as in equation 8:

$$S = K_n \oplus S \quad (8)$$

The transformation for the inverse can be described by considering the following steps: Calculate the subkeys as in equation 9:

$$K_0, K_1, \ldots. K_n, from\ t\square e\ key\ K \quad (9)$$

Compute the state by adding the plaintext block *B* and the key *K* as in equation 10:

$$S = B \oplus K_n \quad (10)$$

Performed the inverse for the Shift-Row as in equation 11:

$$S = InvShiftRow\ (S) \quad (11)$$

Performed the inverse for the Sub-Bytes as in equation 12:

$$S = InvSubBytes(S) \quad (12)$$

XOR the Add-RoundKey in the subkey as in equation 13:

$$S = K_n \oplus S \quad (13)$$

For *I* = *nr* – *1* to 1

XOR the Add-RoundKey in the subkey as in equation 14:

$$S = K_i \oplus S \quad (14)$$

Performed the inverse for the Mix-Column as in equation 15:

$$S = InvMixColumn(S) \quad (15)$$

Repeat the inverse for Shift-Row and the Sub-Bytes rounds in order respectively as in equation 10 and 11 then finally XOR the Add-RoundKey in the subkey as in equation in 16:

$$S = K_n \oplus S \quad (16)$$

Transformation process for each round consists of four functions but for the final round, it consists of three.

Figure 1 shows the flow chart of the encryption process of Rinjdael Cipher for key size of 128 bit. For 192 and 256 key sizes, the rounds are increased from 12 to 14 respectively. While Figure 2 presented the Flowchart for AES 128 Decryption.

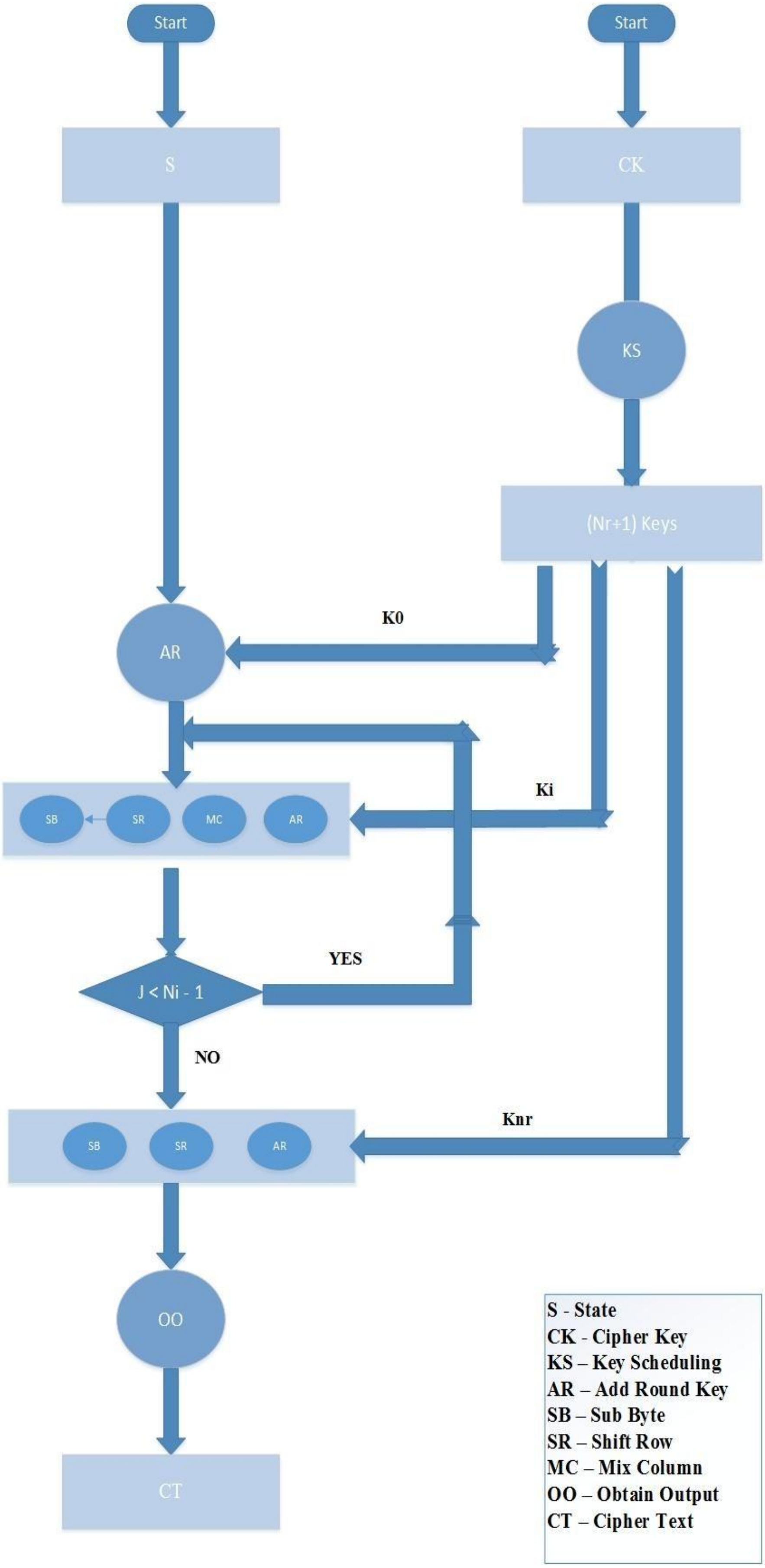

Figure 1: Flowchart for AES 128 encryption [9].

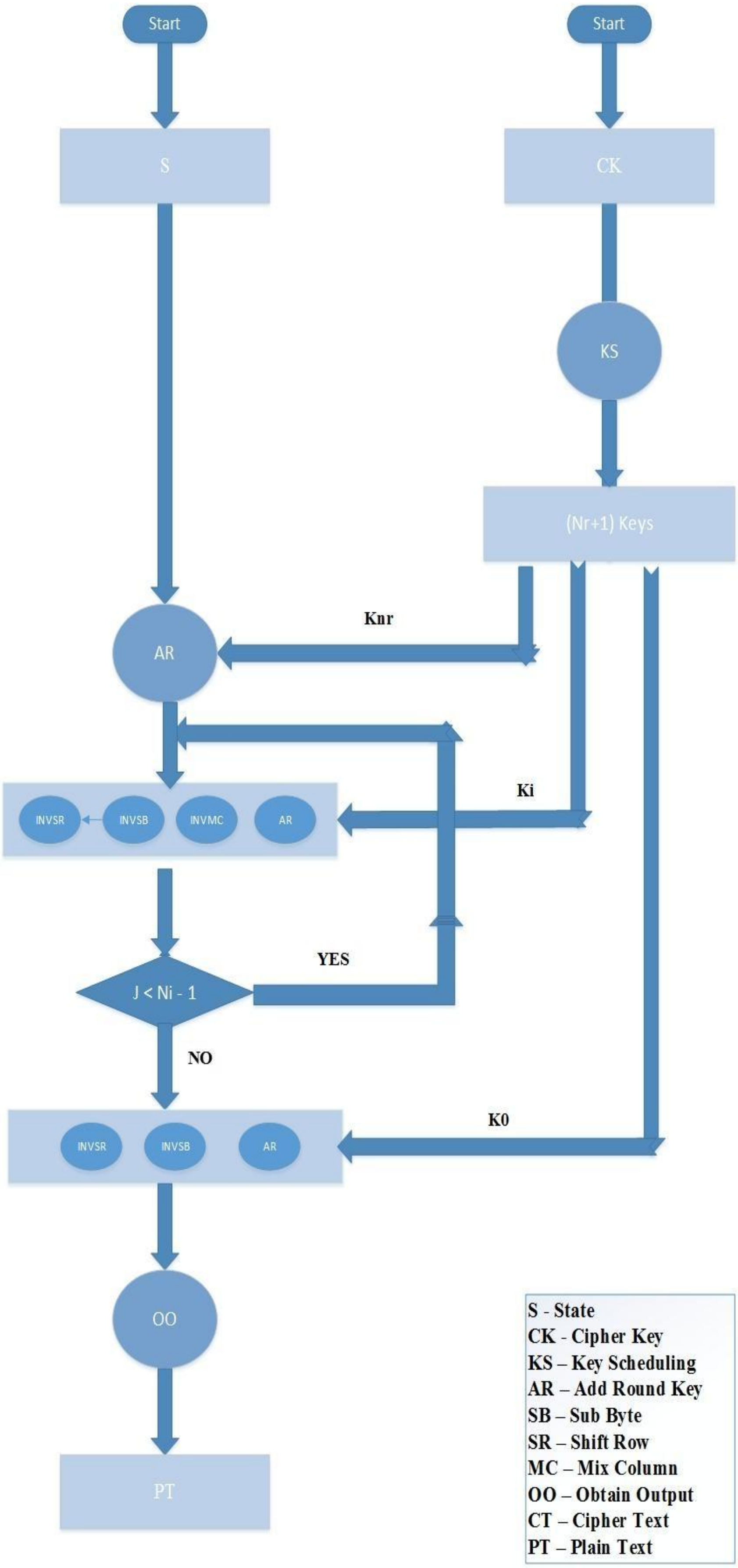

Figure 2: Flowchart for AES 128 Decryption [8]**.**

1.3.2 RC6 Cipher

Figure 3 shows the Flowchart for RC6 Encryption Process. RC6 is a block cipher with 128 bits per block b. RC6 allows works with three key lengths of 128, 192, and 256 bits. It is designed to improve RC5 and was submitted to NIST to be considered as an Advanced Encryption Standard and eventually got to the finals in the competition. RC6 uses four registers each one of 32 bit and thus more secure than the RC5.

RC6 makes use of three key algorithm elements: key expansion, encryption and decryption. RC6 use key expansion to widen the key supplied by the user filling an expanded array which is denoted E so that E looks like array of random binary character's g (Rives et al, 1998). RC6 cipher uses 44 cells of subkeys that are derived from the keys and called E [0] to E [43]. The length of each subkey is 32 bits.

In RC6, much of its characters are obtained from key that user supplies. The user supplies key to bytes, where $0 \leq l \leq 255$ as l represents bytes and characters of (2g+4) are obtained then kept in a round key array E to be encrypted and decrypted later. Key bytes are put in an array c w-bit words E[0]…. E[c-1]. The first byte of the key is placed as in E [0], The second byte in E[1] and so on. 2g + 4 are the number of u-bit characters generated for round keys and these are stored in the array E [0… 2t +3]. To make key length and non-zero integer number equal, zero bytes are joined. When e = 0, c = 1 and E[0] = 0 the key bytes are loaded into an array E of size c.

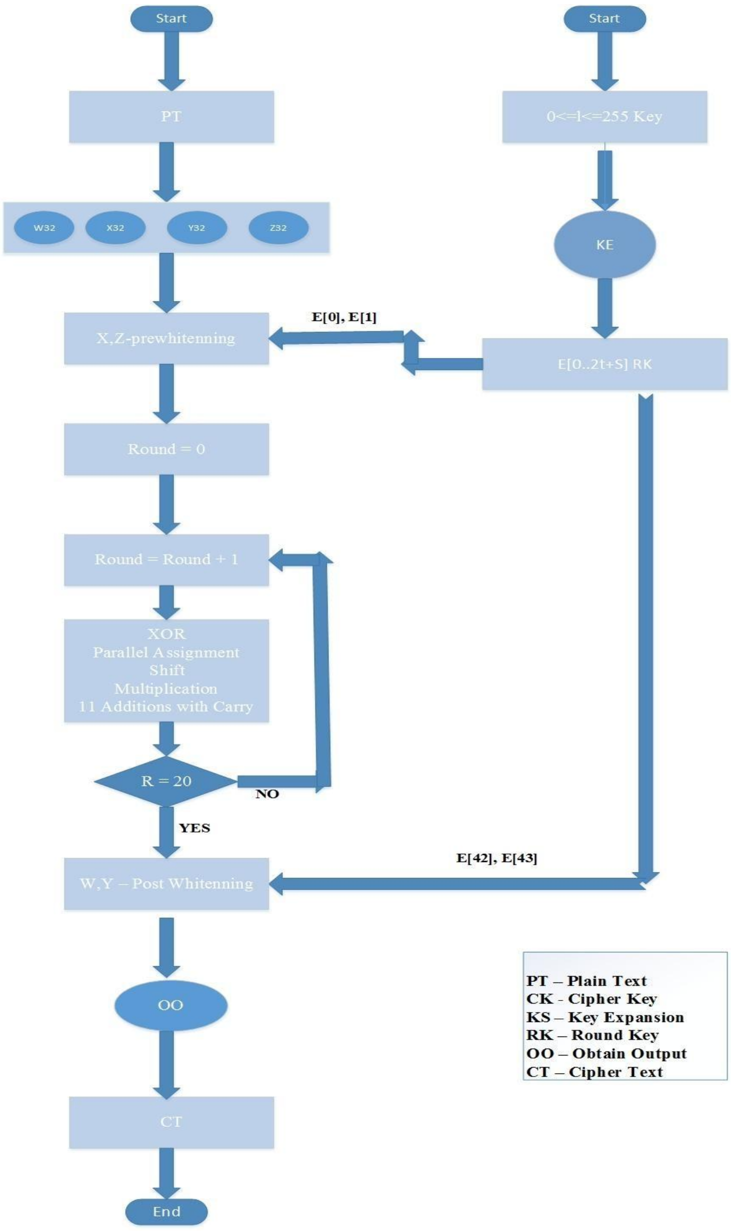

Figure 3: Flowchart for RC6 Encryption Process [10].

1.3.3 Camellia Cipher

Figure 4 presents the Flowchart for RC6 Decryption Process. While Figure 5: Flowchart for Camellia 128 Encryption Process. Additionally, Figure 6: Flowchart for Camellia 128 Decryption Process. Developed in 2000in Japan by Mitsubishi and NTT companies, Camellia is a block cipher with 128 bit per block b. Camellia allows works with three key lengths of 128, 192, and 256 bit. Camellia is known for its efficiency for software and hardware implementations which makes it very good for low-cost smart cards to mobile devices. The most important elements of Camellia are the F-functions which are used to encrypt, decrypt and create helper variables of the key. 128 input bits are grabbed by the F-function, mixes them with subkeys bits and then returns 128 new bits. The F-function calls are arranged in blocks and each of these blocks comprises of six rounds.

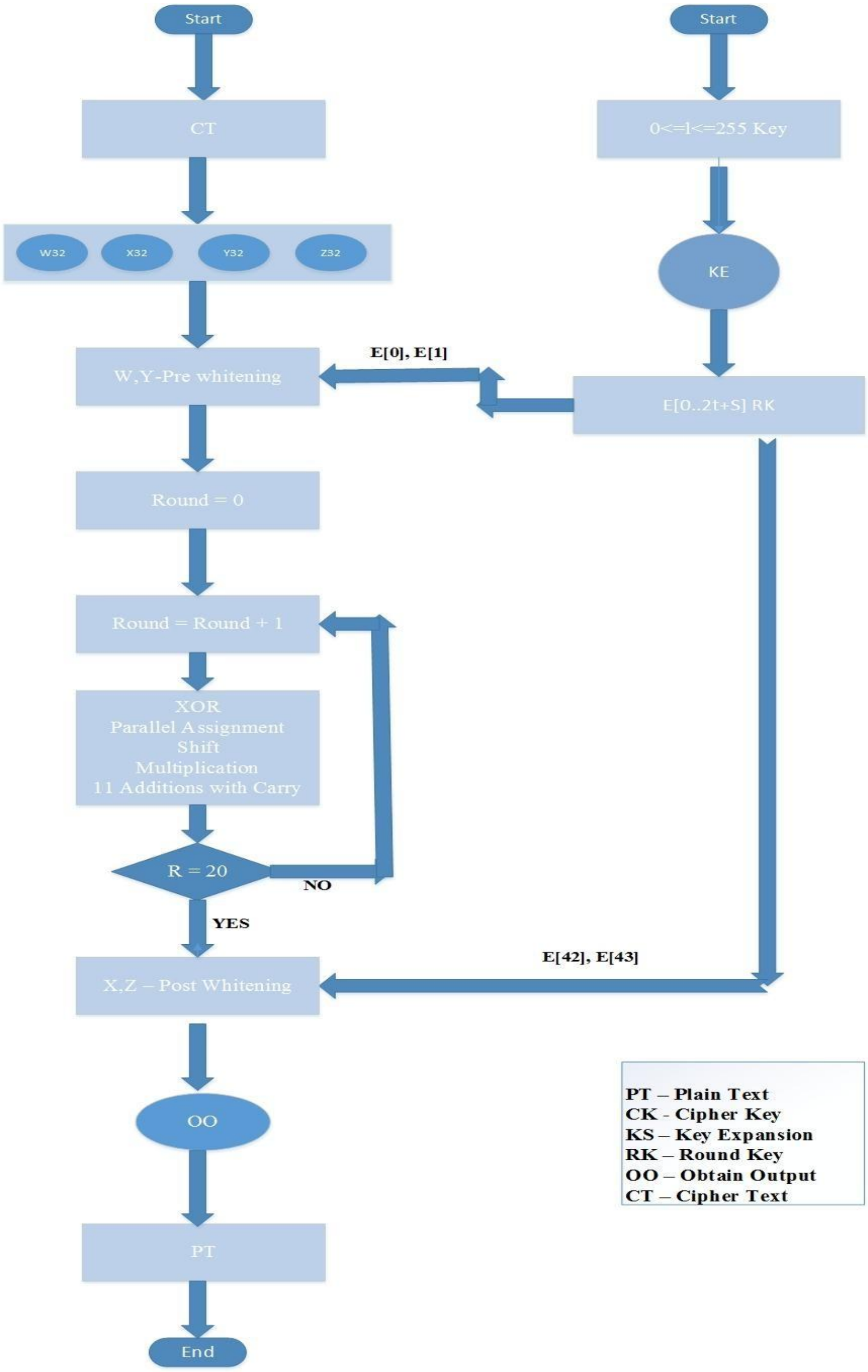

Figure 4: Flowchart for RC6 Decryption Process [11].

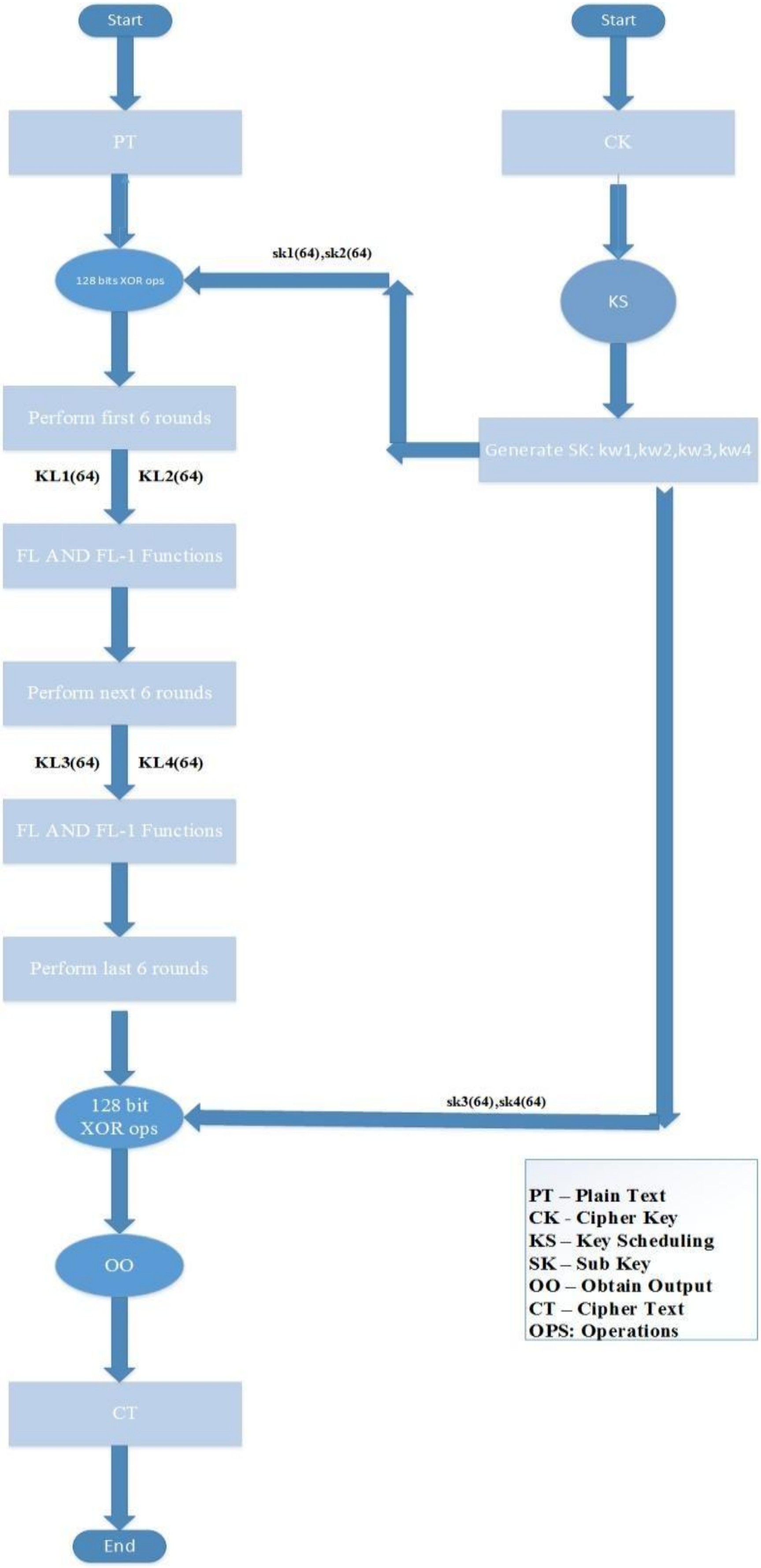

Figure 5: Flowchart for Camellia 128 Encryption Process [12]**.**

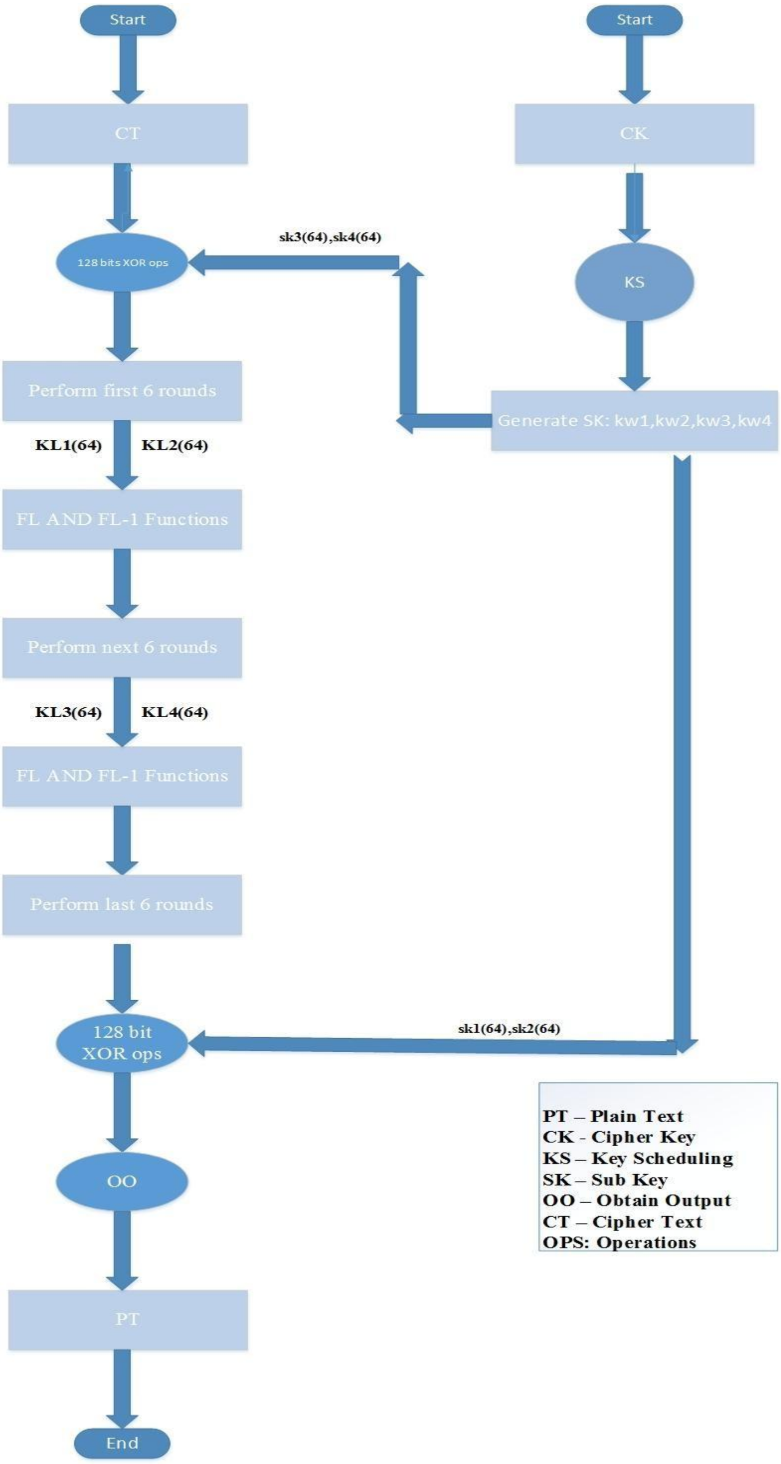

Figure 6: Flowchart for Camellia 128 Decryption Process**.**

Results and discussion

The user interface above shows every process of the application. From the moment the user opens the application, composes, encrypts, and sends a secured message to when the user decrypts the secret message. The application generates a cypher key for the user by obtaining a passphrase of equivalent key length, with which the user feeds to it. This dynamic passphrase is used to derive a secret key, which is used for the encryption of the message. After the encrypted message is sent to the receiver, the application uses the same passphrase which was embedded in both users' applications to derive a key that matches the secret key that the sender used to send the message. This process helps in decrypting the message to make it readable. Thus, both encryption and decryption take place in the application not within the network. In addition, the secret key is not transferred via the network but kept secret inside the application. Some permission in the application prevents reverse engineering of the application.

Key Generation Time (Encryption)

For encryption, key generation time for each symmetric algorithm was calculated. Table 4 shows key generation time for each algorithm for encryption. Calculations are in nanoseconds for generating keys while text sizes are in bytes. The time for generating a key was obtain for each algorithm by calculating the average of the time in nanoseconds. These averages were used to determine the speed of key generation for each algorithm.

Table 4: Key Generation Time for Each Algorithm (encryption)**.**

Plaintext	AES 128	AES 192	AES 256	CAM 128	CAM 192	CAM 256	RC6 128	RC6192	RC6256
10	129307	163030	233077	36385	49000	48538	32538	38307	43923
20	160269	235847	233692	41869	49154	50769	34847	41308	41077
30	276615	201507	212923	55923	49846	61077	36385	40154	43923
40	335154	271231	347923	56154	45153	50539	31923	40000	59000
50	310769	290769	249385	58384	48308	54615	46000	49077	53308
60	240616	213000	287230	42308	54077	61077	48077	53385	44539
70	284462	341077	392007	52154	57462	57231	50077	70077	56692
80	400769	451154	405308	54770	65077	79154	61924	64615	81077
Key Gen. Enc.Avg	267245	270952	295193	49743	52260	57875	42721	49615	52942

Figure 7 shows that for key sizes of 128, 192 and 256 bits, AES is four times slower to produce cipher keys when compared to Camellia and RC6. Camellia and RC6 just about have the same speed to generate keys.

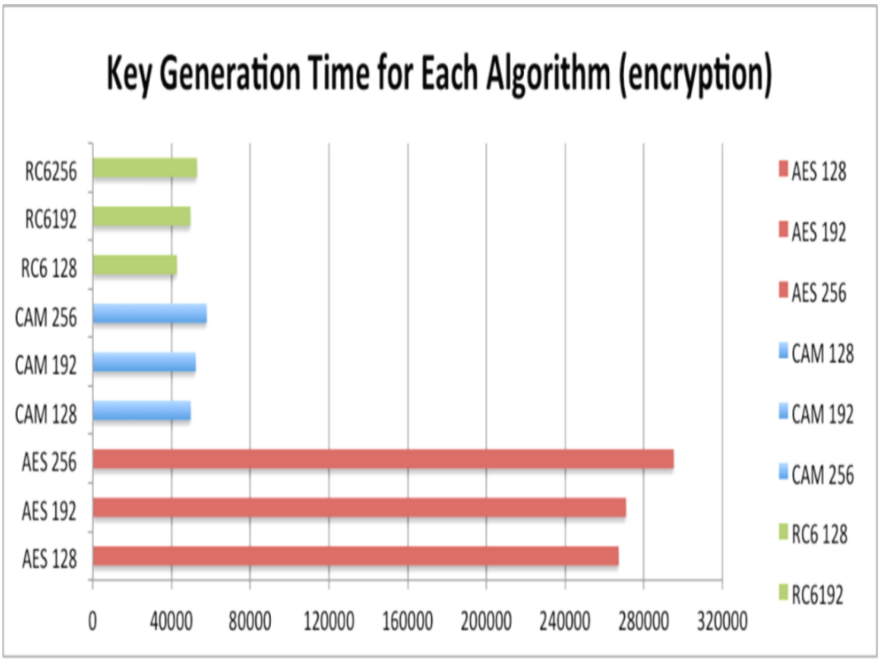

Figure 7: Key Generation Time for Each Algorithm (encryption)

Encryption Time

Encryption time is how long it takes to convert plaintext to ciphertext. Time of encryption was measured for each algorithm. Table 5 shows encryption timing for each algorithm. Calculations are in nanoseconds for the timings while text size is in bytes. The time for encrypting a message was obtain for each algorithm by calculating the average of the time in nanoseconds. These averages were used to determine the speed of encryption for each algorithm. Figure 8 shows that for key sizes of 128, 192 and 256 bit, AES is slightly slow to encrypt a text when compared to Camellia and RC6. There is also not much in terms of speed of encrypting a text between Camellia and RC6.

Table 5: Encryption Time for Each Algorithm**.**

Plaintext	AES 128	AES 192	AES 256	CAM 128	CAM 192	CAM 256	RC6 128	RC6192	RC6256
10	146323	203769	229231	109777	109561	114154	102385	116532	110461
20	243007	220154	222923	122311	110138	114538	104846	133769	111500
30	232308	227461	309000	108385	111616	120154	131961	132693	113308
40	274000	232153	291462	114538	111770	215961	148308	124846	138615
50	265847	271692	271077	125616	113539	226077	151507	146539	154381
60	268615	281976	285076	117693	127000	287077	151692	180539	307231
70	273007	287230	327539	122311	149231	289000	171461	194077	324815
80	318000	300385	403770	120077	151323	377753	163846	196276	365382
Encryption Avg	252638	253103	292510	117588	123022	218089	141376	153159	203211

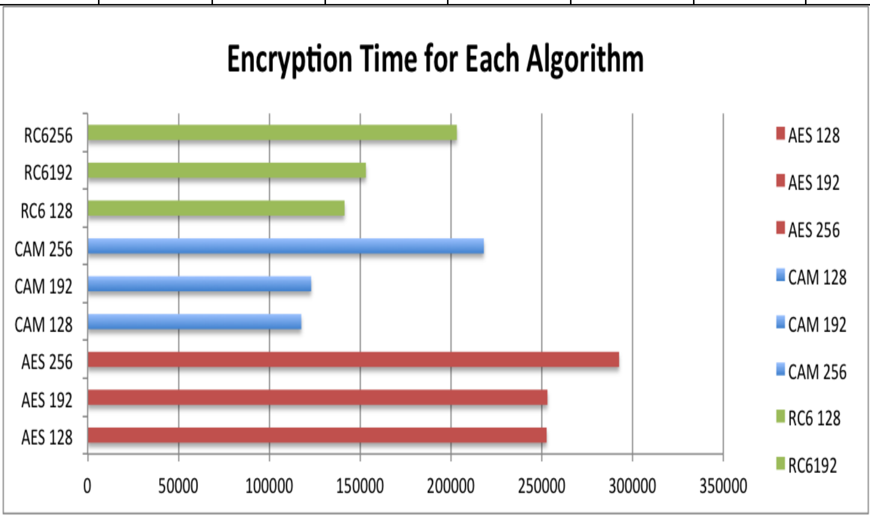

Figure 8: Encryption Time for Each Algorithm.

Key Generation Time (Decryption)

Key generation time for each symmetric algorithm for decryption was calculated. Table 6 shows key generation time for each algorithm for encryption based on decryption. Calculations are in nanoseconds for generating keys while text sizes are in bytes. The time for generating a key was obtain for each algorithm by calculating the average of the time in nanoseconds. These averages were used to determine the speed of key generation for each algorithm.

Figure 9 shows that for key sizes of 128, 192 and 256 bit, AES is about eight times slower to generate cipher keys. Camellia and RC6 are about the same in terms of speed of generating ciphertext.

Table 6: Key Generation Time for Each Algorithm (decryption)**.**

Plaintext	AES 128	AES 192	AES 256	CAM 128	CAM 192	CAM 256	RC6 128	RC6192	RC6256
10	280016	429536	107446	35847	68923	51307	13015	38845	35539
20	350231	296308	296769	35924	36923	35307	29462	39307	37301
30	233385	235308	264230	46000	51385	51230	34538	39307	35538
40	234616	249446	383615	35385	37153	35923	28924	39539	40538

50	289000	257461	306769	50461	35847	50154	38307	38846	36923
60	317230	200769	339461	49538	41154	49769	39077	40307	49231
70	359000	333846	401769	49230	49769	49462	36461	39307	48693
80	471462	463847	378769	54154	51007	50077	43308	41153	59230
Key Gen.Dec.Avg	306968	308315	309858	44569	46520	46654	32887	39576	42874

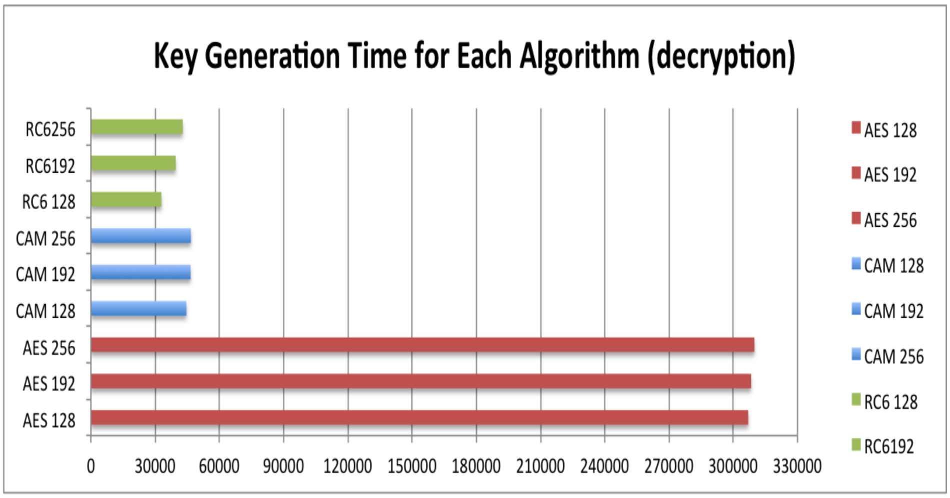

Figure 9: Key Generation Time for Each Algorithm (decryption)

Decryption Time

Decryption time is how long it takes to convert ciphertext to plaintext. Time of decryption was measured for each algorithm. Table 7 shows decryption timing for each algorithm. Calculations are in nanoseconds for the timings while text size is in bytes. The time for decrypting a message was obtain for each algorithm by calculating the average of the time in nanoseconds. These averages were used to determine the speed of decryption for each algorithm.

Table 7: Decryption Time for Each Algorithm.

Plaintext	AES 128	AES 192	AES 256	CAM 128	CAM 192	CAM 256	RC6 128	RC6192	RC6256
10	175769	246769	233385	35307	62154	49192	30077	103615	56616
20	179385	245000	234616	35923	67693	60230	38846	10800	58385
30	206461	205769	280616	49769	68077	66692	39307	59461	83692
40	204184	235000	289000	49462	73815	68000	39307	63539	99000
50	304153	237692	350231	50077	76386	87693	39539	74692	78077
60	327385	232462	317230	51230	78231	97077	38846	80693	102154
70	409692	415231	359000	50154	95384	101231	40307	83077	110000
80	412692	504769	471462	51307	101154	114308	41153	168692	169531
Decryption Avg	277465	294837	316943	46653	77862	80553	38423	80576	94682

Figure 10 shows that for key sizes of 128, 192 and 256 bits, AES is about three times slower to decrypt ciphertext when compared to Camellia and RC6. Camellia and RC6 are just about the same in speed for decrypting a text**.**

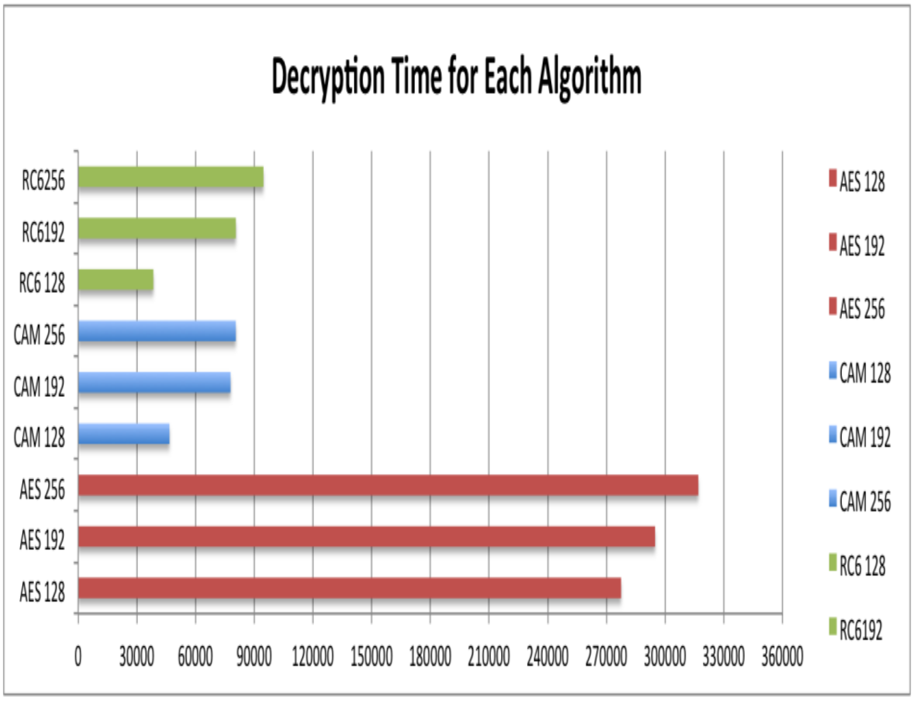

Figure 10: Decryption Time for Each Algorithm

Plaintext and Ciphertext Length

Table 8 shows text count of plaintext and ciphertext. According to findings, plaintext is increased when encryption is performed by all algorithms. Camellia plaintext length increased more when compared to AES and RC6. In addition, for key sizes 128-, 192- and 256-bits cipher length increment is constant for AES and RC6. For Camellia, cipher length changes as key size changes.

Table 8: Plaintext and Ciphertext Length**.**

Plan Text	AES 128	AES 192	AES 256	CAM 128	CAM 192	CAM 256	RC6 128	RC6 192	RC6256
10	25	25	25	25	21	25	25	25	25
20	45	45	45	41	41	41	45	45	45
30	45	45	45	65	61	65	45	45	45
40	65	65	65	86	82	86	65	65	65
50	90	90	90	106	102	106	90	90	90
60	90	90	90	118	118	126	90	90	90
70	110	110	110	138	139	142	110	110	110
80	130	130	130	159	159	160	130	130	130

As key sizes increase from 128 to 192 to 256 bits, time to generate cipher keys increases for all three algorithms. Also, as key sizes increase from 128 to 192 to 256 bits time of encryption and decryption increases for all three algorithms**.**

Conclusion

The objectives and questions of this thesis have been met and answered. This thesis was able to meet the goals of securing SMS by employing symmetric encryptions. The encryption keys generated by each algorithm were embedded on the application. Both applications for sender and receiver have a passphrase embedded inside them.

That passphrase is used to generate a key and encrypt a message so that when the message is sent to receiver, receiver is able to regenerate the key because it also has the required passphrase embedded in it. It uses the passphrase to regenerate the key and matches it for decryption of message. Therefore, encryption keys are not transmitted over the network.
This thesis also determined which of the symmetric methods is suitable for securing SMS when it boils to how fast secured SMS will be sent and read. In summary, after studying a handful of existing solutions in literature three symmetric algorithms were selected. These algorithms were based on how effective they are when it comes to mobile devices that have low storage space and computational resources. The symmetric algorithms were analyzed, implemented and evaluated based on encryption, decryption and key generation time.

From the experimental results, it was found that Camellia and RC6 of key length 128, 192 and 256 bit is faster generating encryption keys, encrypting and decrypting text when compared to AES of equivalent key length. On the other hand, Camellia and RC6 of key size 128 and 256 are just about the same in terms of generating encryption keys, encrypting and decrypting text. Plaintext increases in length after encryption for all three algorithms. The length of cipher text of AES and RC6 are constant for key sizes 128, 192 and 256 bits while that of Camellia changes for equivalent key sizes.

In conclusion, when it comes down to efficiency in time for key generation, encryption and decryption of Camellia and RC6 is faster than AES. So, it is fair to conclude that both Camellia and RC6 should be use more often as it provides security, authentication, integrity, fast in generating keys and also fast in encryption and decryption of SMS than AES. This thesis is only implemented for Android mobile phones. Future work should extend it to other operating systems that will make it work on other operating systems other than Android. In addition, access control like biometrics should be used by authorized users of the application to boost security of the application.

References

[1] N. M. Coe and C. Yang, "Mobile Gaming Production Networks, Platform Business Groups, and the Market Power of China's Tencent," Ann. Am. Assoc. Geogr., vol. 112, no. 2, pp. 307–330, Feb. 2022, doi: 10.1080/24694452.2021.1933887.

[2] L. Lo Presti, G. Maggiore, V. Marino, and R. Resciniti, "Mobile instant messaging apps as an opportunity for a conversational approach to marketing: a segmentation study," J. Bus. Ind. Mark., vol. 37, no. 7, pp. 1432–1448, May 2022, doi: 10.1108/JBIM-02-2020-0121.

[3] I. Alam, S. Khusro, A. Rauf, and Q. Zaman, "Conducting Surveys and Data Collection: From Traditional to Mobile and SMS-based Surveys," Pakistan J. Stat. Oper. Res., vol. 10, no. 2, p. 169, Aug. 2014, doi: 10.18187/pjsor.v10i2.758.

[4] A. Singhal, A. Jain, and L. Kharb, "HacXBear: An Android App to Safeguard Mobile Theft," 2023, pp. 487–499. doi: 10.1007/978-981-99-3963-3_37.

[5] F. F. M. Yahia and A. M. Abushaala, "Cryptography using Affine Hill Cipher Combining with Hybrid Edge Detection (Canny-LoG) and LSB for Data Hiding," in 2022 IEEE 2nd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), IEEE, May 2022, pp. 379–384. doi: 10.1109/MI-STA54861.2022.9837714.

[6] M. Shahrier and A. Hasnat, "Route optimization issues and initiatives in Bangladesh: The context of regional significance," Transp. Eng., vol. 4, no. February, p. 100054, 2021, doi: 10.1016/j.treng.2021.100054.

[7] P. Jindal, A. Kaushik, and K. Kumar, "Design and Implementation of Advanced Encryption Standard Algorithm on 7th Series Field Programmable Gate Array," in 2020 7th International Conference on Smart Structures and Systems (ICSSS), IEEE, Jul. 2020, pp. 1–3. doi: 10.1109/ICSSS49621.2020.9202114.

[8] K. Muttaqin and J. Rahmadoni, "Analysis And Design of File Security System AES (Advanced Encryption Standard) Cryptography Based," J. Appl. Eng. Technol. Sci., vol. 1, no. 2, pp. 113–123, May 2020, doi: 10.37385/jaets.v1i2.78.

[9] A. Altigani, S. Hasan, B. Barry, S. Naserelden, M. A. Elsadig, and H. T. Elshoush, "A Polymorphic Advanced Encryption Standard – A Novel Approach," IEEE Access, vol. 9, pp. 20191–20207, 2021, doi: 10.1109/ACCESS.2021.3051556.

[10] S. A. Ajagbe, O. D. Adeniji, A. A. Olayiwola, and S. F. Abiona, "Advanced Encryption Standard (AES)-Based Text Encryption for Near Field Communication (NFC) Using Huffman Compression," SN Comput. Sci., vol. 5, no. 1, p. 156, 2024, doi: 10.1007/s42979-023-02486-6.

[11] X. Li et al., "Thermal-Triggered Phase Separation and Ion Exchange Enables Photoluminescence Tuning of Stable Mixed-Halide Perovskite Nanocrystals for Dynamic Display," Laser Photon. Rev., Jan. 2024, doi: 10.1002/lpor.202301244.

[12] B. Langenberg, H. Pham, and R. Steinwandt, "Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit," IEEE Trans. Quantum Eng., vol. 1, pp. 1–12, 2020, doi: 10.1109/TQE.2020.2965697.