Performance Comparison of AES-CCM and AES-GCM Authenticated Encryption Modes

Levent Ertaul, Anup Mudan, Nausheen Sarfaraz
CSU East Bay, Hayward, CA, USA.
levent.ertaul@csueastbay.edu, amudan@horizon.cueastbay.edu, nsarfaraz@horizon.csueastbay.edu

***Abstract* — In data security, especially in mobile devices, it has long been understood that to meet the highest compliance standards, Authenticated Encryption is required. Encryption alone is not enough to provide the utmost level of security in various mobile applications. This paper proposes the implementation of Authenticated Encryption Mode, CCM in our application. This work also shows a comparison of the performance analysis of AES-CCM and AES-GCM modes. The choice to use Android Java programming language was made in order to create an Android application which sends and receives text messages between two parties by sharing a secret key, and uses an authentication feature. The execution of the algorithm is performed in Android Studio and the implementation of the code is accomplished using Android API.**

I. Introduction

Authenticated Encryption is a process of ensuring that both ends of a connection are completely secure. Mobile operating systems in today's world are vulnerable when it comes to hackers. Eighty percent of the world's cellphones use an open source operating system such as Android [8]. Open source allows third parties to change the original code according to their own requirements and needs. This can often leave open loop holes and back doors that hackers will try to exploit. For this, it is not sufficient to use basic encryption techniques to protect information. Authenticated Encryption, or AE, addresses these issues by creating a more secure and bulletproof connection. AE security protects the user and service being used from the inherent flaws in open source software, and ensures that information within a session is not being compromised. In addition to providing authentication and confidentiality, AE provides a strong protection from various attacks like replay attacks, chosen cipher-text attacks, and the man-in-the-middle attack.

Over the last decade, there has been significant amount of research and effort involved to invent the dedicated AE modes CCM, GCM, EAX, and OCB [7]. Rather than using the authenticity and privacy techniques separately, these AE schemes provide more proficient results and have very few chances of being incorrect. In order to ensure safety of the information, it was suggested to combine authentication mechanism such as MAC with the encryption algorithms. The combinations were applied in multiple secure and insecure ways [6]. This paper talks about the comparison of the CCM (CTR + CBC-MAC) mode and the GCM (Galois Counter Mode) [9] mode of operation, which are symmetric key block cipher algorithms defined and used in security systems. The key features like authenticity, integrity and confidentiality are achieved by these modes [18]. CCM is defined in IEEE 802.11i, IPsec [19], TLS 1.2 [20] and uses Advanced Encryption Standard [13] [5] as its cryptographic algorithm. AES is the standard recognized by National Institute of Standards and Technology (NIST) in 2001 [2] and specified in Federal Information Processing Standard (FIPS) [4].

GCM is used in various security standards such as the IEEE 802.1AE for frame data encryption in the Ethernet [15], the IEEE P1619.1 for encrypting hard disks [16], IEEE 802.11AD, and RFC 4106 IPsec [17]. It is based on a parallelization process which generates ciphertexts and an authentication tag simultaneously. It uses counter mode and a hash function over Galois Field (2 ^ 128) to generate a tag. It consists of Galois Field (GF) multiplier adders. In counter mode, the counter blocks are numbered in a sequential manner and the encryption function is performed on these blocks. The output of this function is XORed with the plaintexts to produce ciphertexts. A hash function is used to generate the tag by combining the ciphertext and an authentication code to check the integrity of the data [9].

This paper is organized into the following sections: Section II proposes the approach we followed using the AES-CCM algorithm. Section III discusses the AES-CCM algorithm. Section IV represents the implementation of this algorithm using Android API. Section V shows the performance analysis of CCM and GCM mode and the comparison results. Section VI proposes the conclusion of this paper.

II. Traditional Encryption v/s AES-CCM

Companies sometimes prefer traditional encryption methods when it comes to using mobile applications. The main focus of the traditional encryption schemes is to provide confidentiality, but they do not protect from malicious tampering or data being modified intentionally by the attackers. These methods do not provide the level of security required, and in fact they can be vulnerable to an informed hacker. Authenticated encryption schemes are the alternative methods. Using AE schemes, many different approaches are taken into consideration, i.e. Encrypt-then-MAC, Encrypt-and-MAC and MAC-then-Encrypt [14].

The solution proposed in this paper uses MAC-then-encrypt scheme in which the MAC value is generated first, and then the data and MAC are encrypted using counter mode. This would make it hard for the attacker to obtain the MAC value in order to perform attacks. The following figure 2.1 depicts the approach being implemented.

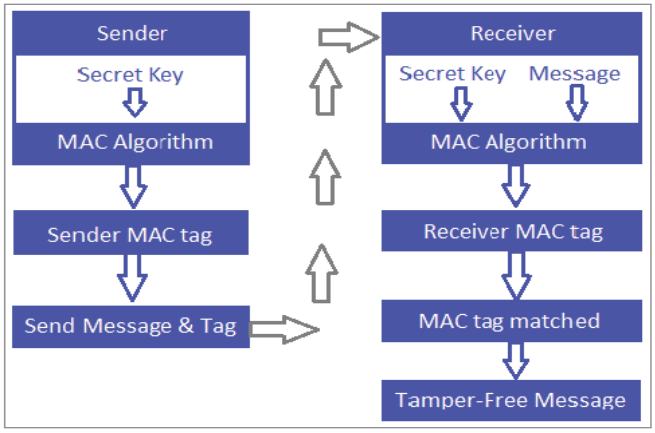

Figure 2.1 MAC-then-encrypt mechanism

III. The AES-CCM Algorithm

Advanced Encryption Standard, or AES, [13] is the standard known for a symmetric block cipher mechanism that uses 128 bits, 192 bits and 256 bits of key sizes. CCM is an Authenticated Encryption Standard which is based on a key management structure. In this algorithm, the plaintext is divided into block ciphers of 128 bits size. The modes of operations used in AES-CCM are counter mode (CTR) with Cipher Block Chaining and Message Authentication Code (CBC-MAC). They perform generation-encryption and decryption-verification functions [3]. The confidentiality feature is achieved in CTR mode by AES and the authentication is achieved in CBC-MAC with the MAC value generated.

In AES-CBC-MAC, the encryption function is applied to the first block to generate a cipher. Then the cipher result is XORed with the second block to obtain the next result. The process keeps going on for all the remaining blocks until the final value MAC is obtained, which is used in CTR mode encryption. The following 3.1 shows the block diagram of AES-CBC-MAC.

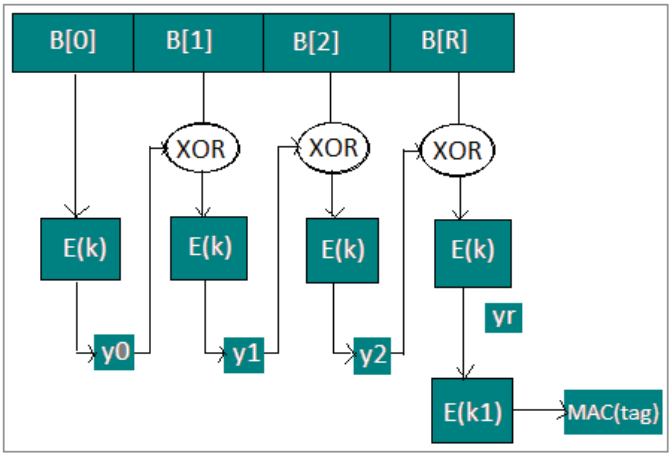

Figure 3.1 Block diagram of AES-CBC-MAC

In AES-CTR, different cipher blocks are produced which are dependent on nonce value. The CTR mode is applied to MAC and the payload to obtain the cipher-text [1]. CCM is not compatible with steam ciphers and does not work with the Data Encryption Standard which supports a 64 bits of block size. It works in the packet environment where all of the data is available in storage beforehand [3]. The following figure 3.2 shows the block diagram of AES-CTR mode.

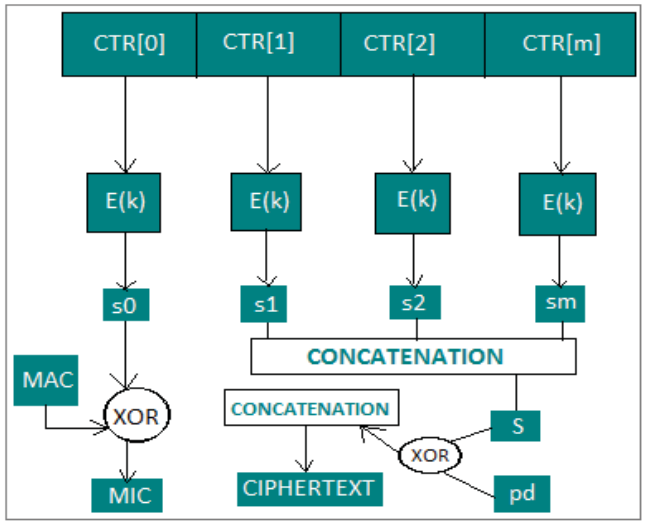

Figure 3.2 Block diagram of AES-CTR

The input elements of CCM are: the valid payload (*pd* < *2^64*) (data which is authenticated and encrypted), the valid nonce (*nc* < *2^61*) (must be unique), and the valid associated data (*ad* ≤ *256 bits*) (which is authenticated but not encrypted). The nonce is applied to the payload and the associated data. The secret key (*k*) to the block cipher is generated uniformly at random whose size is 128 bits. CCM only works with the forward cipher function [3].

A. Generation-Encryption

In generation-encryption mechanism, cipher block chaining is applied to the payload (*pd*), the nonce (*nc*), and the associated data (*ad*), to generate MAC. The MAC length (*Mlen*) is always greater than or equal to 64 bits. Then the counter mode encryption is applied to the MAC and payload to convert it into cipher-text [3].

Prerequisites:
The various prerequisites that are required are as follows: the cipher block algorithm, key *k,* counter generation function, formatting function, MAC length *Mlen.*

Input:
The input values required are: valid payload *pd* of length *pdlen* bits; valid associated data *ad;* valid nonce *nc.*

Output:
The output will be cipher-text *C.*

Steps:

1. Apply the formatting function to *(nc, ad, pd)* to produce the blocks *B0, B1,....., Br*
2. Set *Y0 = CIPH k(B0)*
3. For *I = 1 to r,* do *Yi = CIPH k(Bi XOR Yi-1)*
4. Set *MAC = MSBMlen (Yr)*
5. Apply the counter generation function to generate the counter blocks *CTR0, CTR1, ..., CTRm,* where *m = pdlen/128*

6. For *j = 0 to m,* do *Sj = CIPH k(CTRj)*
7. Set *S = S1 || S2 || .... || Sm*
8. Return *C = (pd XOR MSB pdlen(S)) || (MAC XOR MSB Mlen(S0))*

B. Decryption-Verification

In decryption-verification mechanism, counter mode decryption is performed to get the MAC value and its corresponding payload. Cipher block chaining is applied to the payload, the nonce received, and the associated data received to check if the MAC is correct. If the verification succeeds that means that inputs are generated from the source and have access to the key [3]. MAC plays the most important role as it can keep away security threats and can protect data from being modified.

Prerequisites:
The various prerequisites that are required are as follows: Cipher block algorithm; Key *k;* Counter generation function; Formatting function; and Valid MAC length *Mlen.*
Input:
The main input values required are: associated data, *ad;* nonce, *nc;* ciphertext *C* of length *cplen* bits.

Output:
The output will be either payload *pd* or *INVALID*.

Steps:

1. If *cplen* ≤ *Mlen,* then return *INVALID*
2. Apply the counter generation function to generate the counter blocks *CTR0, CTR1, ..., CTRm*
3. For *j = 0 to m,* do *Sj = CIPH k(CTRj)*
4. Set *S = S1 || S2 || ..... || Sm*
5. Set *pd = MSB cplen – Mlen(C) XOR MSB cplen – Mlen(S)*
6. Set *MAC = LSBMlen (C) XOR MSB Mlen(S0)*
7. If *nc, ad or pd* is not valid, then return *INVALID,* else apply the formatting function to *(nc, ad, pd)* to produce the blocks *B0, B1, ..., Br*
8. Set *Y0 = CIPH k(B0)*
9. For *I = 1 to r,* do *Yj = CIPH(Bi XOR Yi-1)*
10. If *MAC ≠ MSBMlen(Yr),* then return *INVALID,* else return *pd*

IV. Implementation

The following tables I and II show the hardware and software specifications of the device we used.

A. Specification

Table I. Hardware Specification

Type	**Specification**
Device Type	Mac OS X
Processor	2.5 GHz Intel Core i7
RAM	16 GB
Operating System	Android version 5.1 Lollipop

Table II. Software Specification

Type	**Specification**
Android Programming Language	Java
Android Studio	Version 2.0
Android API	Spongy Castle
Android Virtual Machine	Genymotion

B. Screen Shots

We created and executed our CryptUtil application using Android Studio. We made two Android Virtual Machines for the sender and the receiver side, up and running. The screen shots are taken from the emulator.

Once the application is launched, it shows the main page for sender and receiver (Figure 4.1) which contains buttons to send and receive a message.

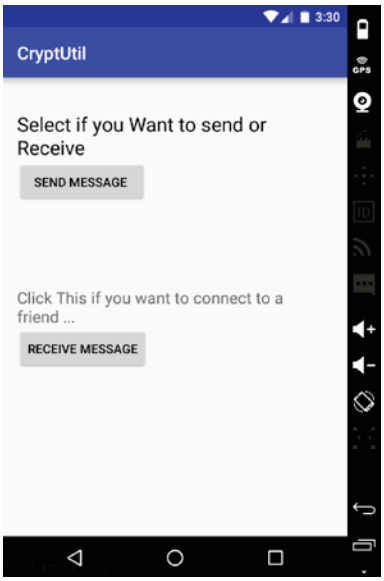

Figure 4.1 Main access page

Once the sender clicks on the Send Message button, the next page is displayed which tells the user to enter the IP address of the receiver side to get connected (Figure 4.2).

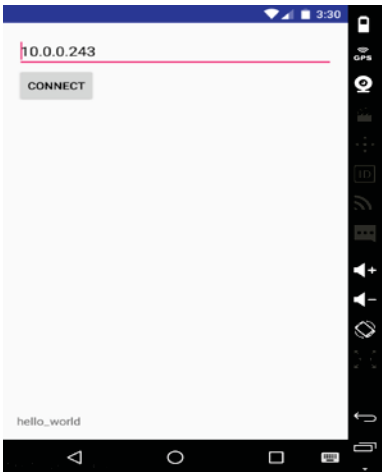

Figure 4.2 Entering IP address to connect to the Receiver

Once the sender enters the IP address, a page is displayed showing that the connection has been established (Figure 4.3).

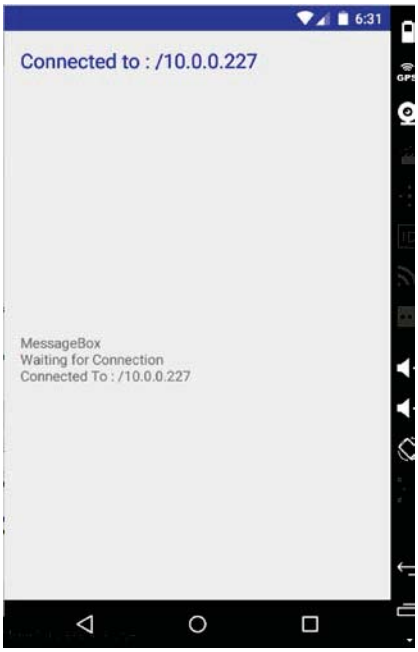

Figure 4.3 Connection established page

After the sender and receiver are connected, the page showing "Enter Your Message To Encrypt" field, and "Enter Your Password Key", field is displayed (Figure 4.4). This page also includes a text field to enter a AEAD value. The AEAD value has to match the default value already set in order to avoid an error.

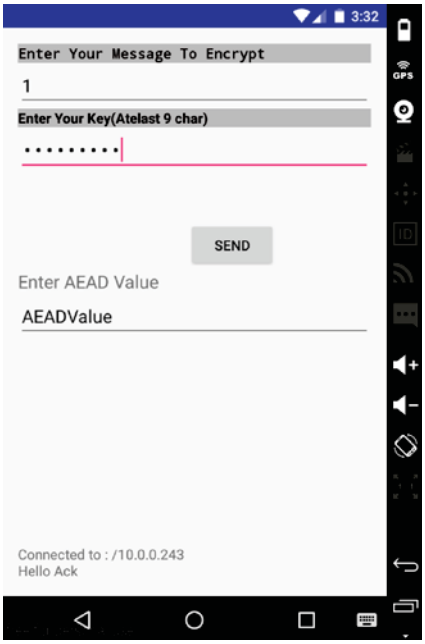

Figure 4.4 Enter message and key page on sender's side

Once the sender clicks on the Send button, the receiver receives the encrypted text i.e. cipher-text. Also, a dialog box appears for the receiver to enter the matching AEAD value and the password key in order to decrypt the text message (Figure 4.5).

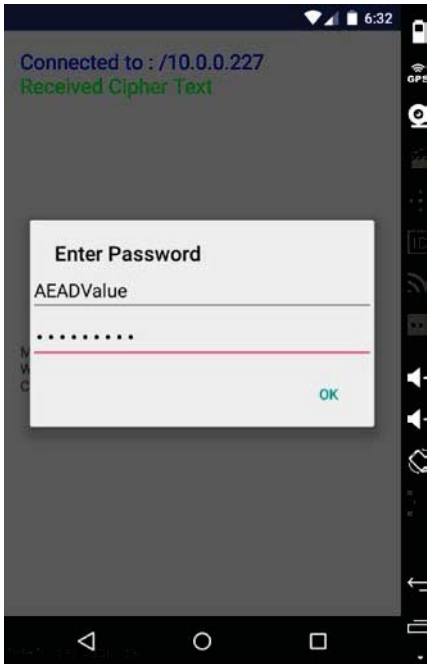

Figure 4.5 Enter password page on receiver's side

If the entered AEAD value and the password key both match the shared value and the key of the sender, then the decryption process is successful on the receiver's end (Figure 4.6). This page also shows the time it took to decrypt the text message in microseconds.

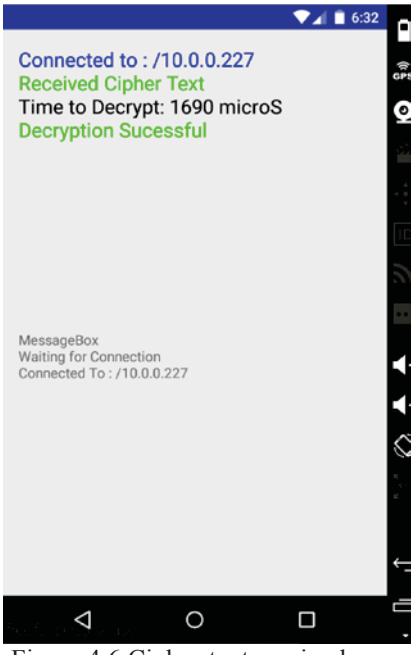

Figure 4.6 Cipher-text received page

If the password key or AEAD value entered do not match the shared key and the value, then the decryption process cannot be performed and it will end up displaying an error message (Figure 4.7).

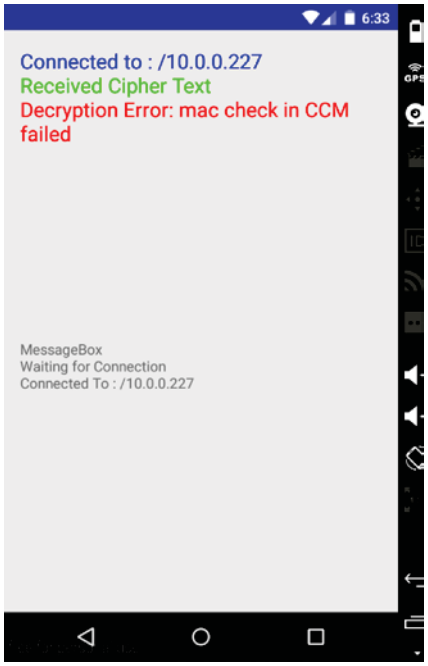

Figure 4.7 Decryption error page

V. Performance Analysis and Results

A. Comparison using different AEAD values of 9 chars, 16 chars and 24 chars in AES-CCM

In the following graphs (figures 5.1, 5.2 and 5.3), we can see that as we increase the number of characters in AEAD value, the encryption and decryption time increases.

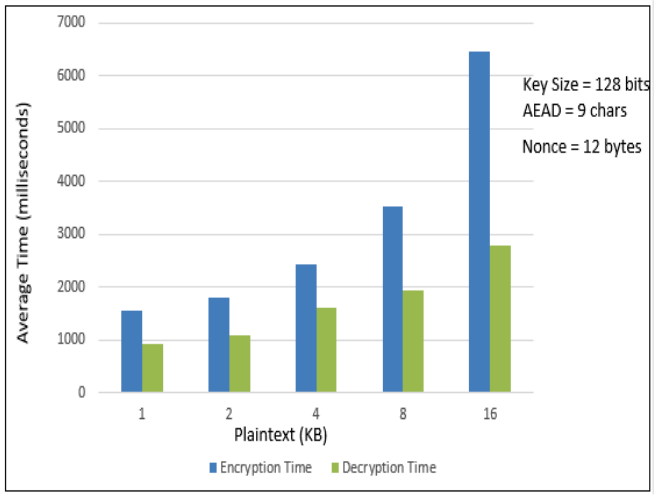

Figure 5.1 128 bits Key, 9 chars AEAD, 12 bytes Nonce

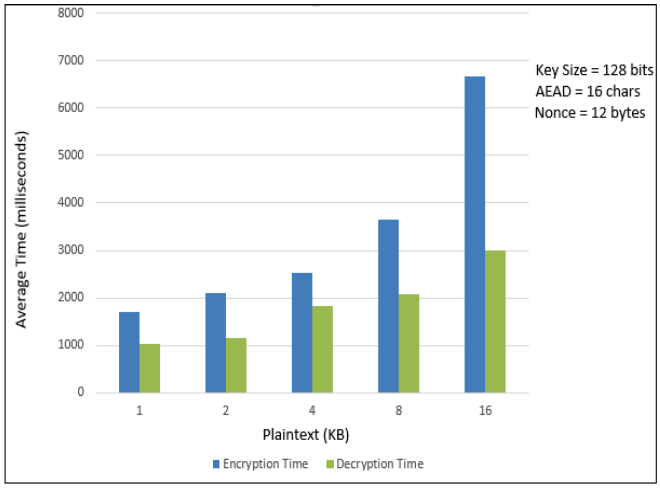

Figure 5.2 128 bits Key, 16 chars AEAD, 12 bytes Nonce

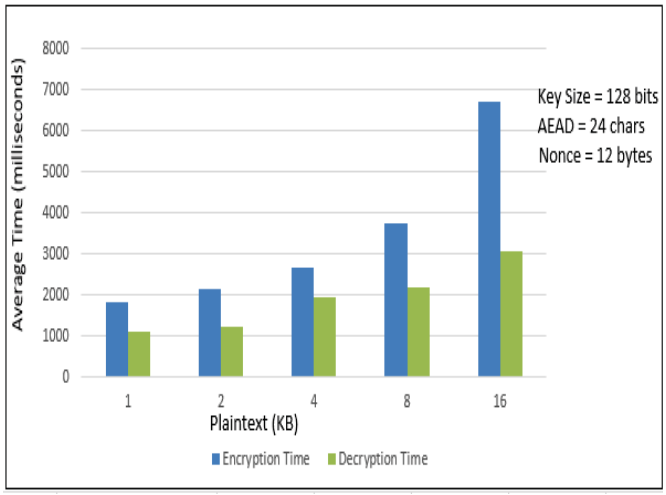

Figure 5.3 128 bits Key, 24 chars AEAD, 12 bytes Nonce

B. Comparison using different key sizes of 128 bits, 192 bits and 256 bits in AES-CCM

We can see the differences in the encryption time in the following figures with the variable key sizes. . As the key size increases, the encryption time rises. On the other hand, it shows a drop in the decryption time when the key size is changed from 128 bits to 192 bits.

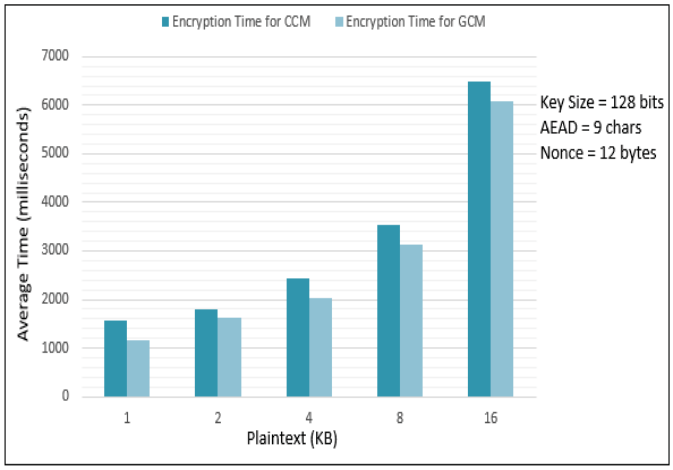

Figure 5.4 128 bits Key, 9 chars AEAD, 12 bytes Nonce

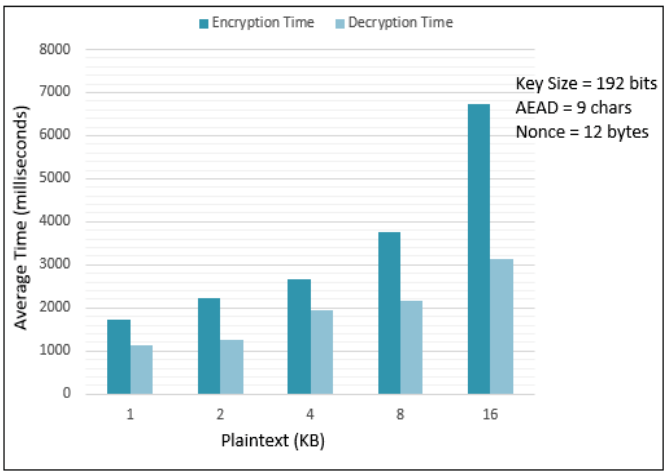

Figure 5.5 192 bits Key, 9 chars AEAD, 12 bytes Nonce

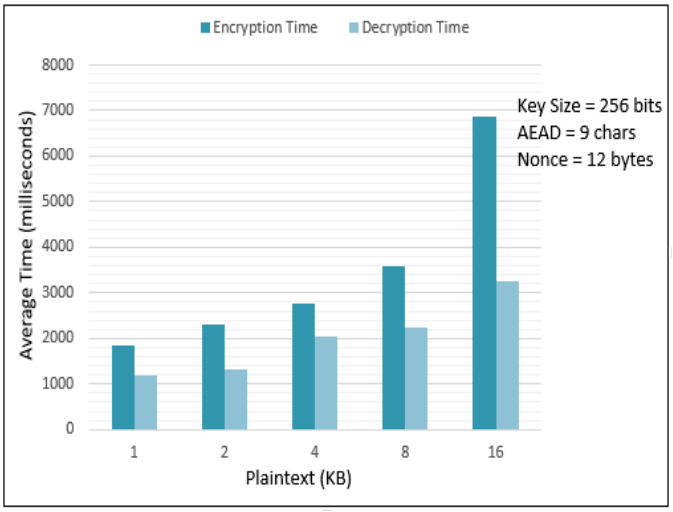

Figure 5.6 256 bits Key, 9 chars AEAD, 12 bytes Nonce

C. Comparison between different key sizes using Nonce values as 8 bytes and 10 bytes in AES-CCM

In this section, we compare the performance by taking nonce values as 8 bytes and 10 bytes (Figures 5.7 and 5.8), respectively. The AEAD value used is constant and the key sizes were compared using a 16 KB plaintext in both cases.

We can see in the following figures that as the nonce value is increased from 8 bytes to 10 bytes, there is a vast difference in the encryption time for the key sizes 128 bits and 192 bits. The time to encrypt plaintext increases at great speed. For a key size of 256 bits, there is no substantial difference in the encryption time as compared to other key sizes. Furthermore, the time to encrypt plaintext is dependent on the nonce value; as the higher the latter becomes, the higher the former is.

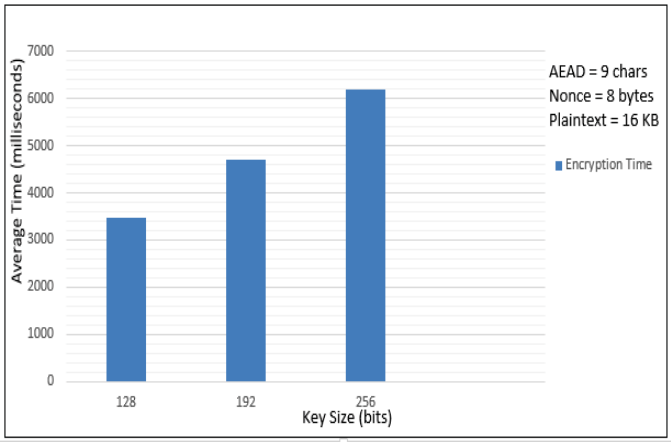

Figure 5.7 Nonce = 8 bytes, AEAD = 9 chars

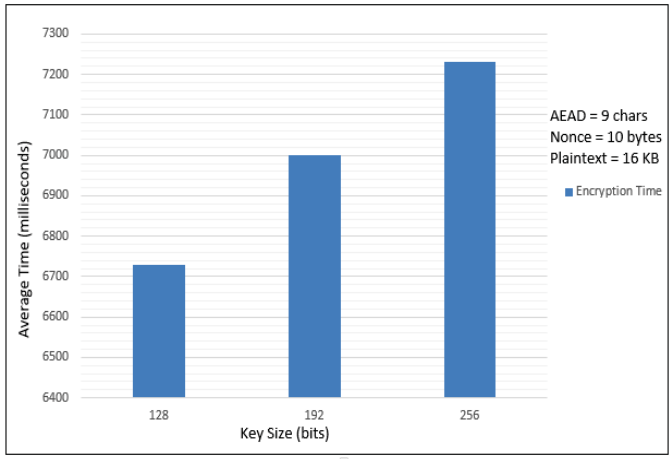

Figure 5.8 Nonce = 10 bytes, AEAD = 9 chars

D. Comparison of AES-CCM and AES-GCM

The main idea behind this paper was to compare the performance of AES-CCM and AES-GCM mechanisms in terms of time taken to encrypt the plaintext. The key size taken is 128 bits as it was kept fixed in the AES-GCM mode. As we can observe in graphs (Figures 5.7, 5.8 and 5.9), the AES-GCM has better performance than AES-CCM as it is taking less time to encrypt the plaintext.

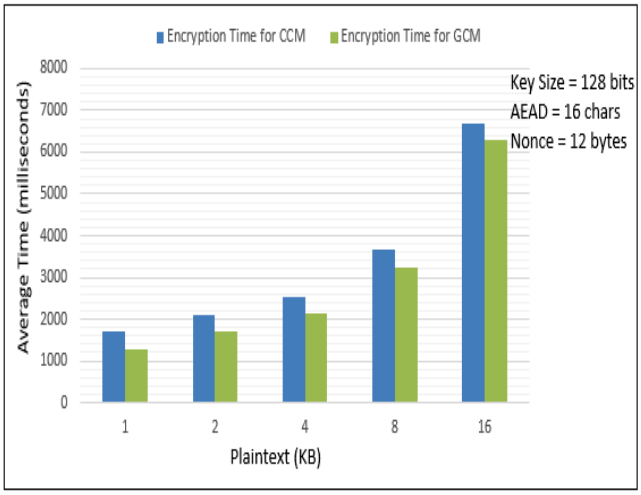

Figure 5.7 128 bits Key, 16 chars AEAD, 12 bytes Nonce

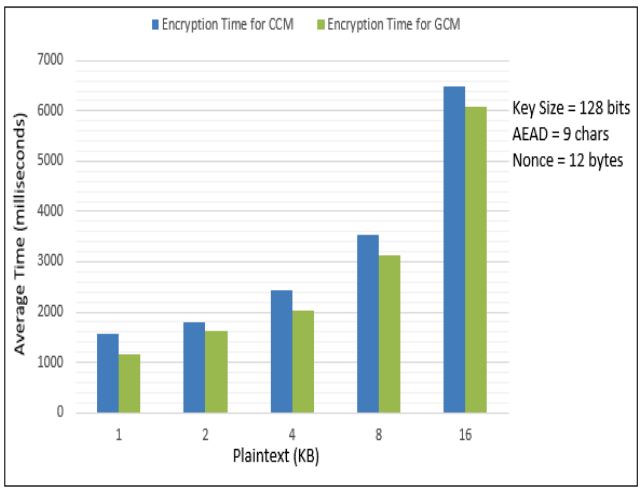

Figure 5.8 128 bits Key, 9 chars AEAD, 12 bytes Nonce

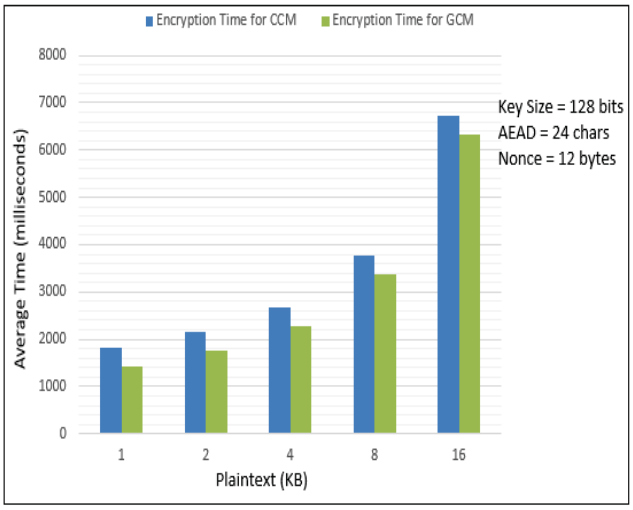

Figure 5.9 128 bits Key, 24 chars AEAD, 12 bytes Nonce

VI. Conclusion

In this paper, we have shown the implementation of an Authenticated Encryption scheme, AES-CCM, on Android application to check if this algorithm is feasible in terms of performance. We did performance analysis for AES-CCM to make comparisons by taking various parameters (key size, AEAD and nonce). These comparisons show that the performance of AES-CCM goes down when the key size, AEAD value and nonce lengths are increased. We have not noticed any major fluctuations in the encryption and decryption times of the plaintext when the key size and AEAD value were changed. However, it was noticeable that when the nonce value was increased, the encryption time rose at greater speed.

We have made another comparison between AES-CCM and AES-GCM to check which mode of operation is better performance wise. Our results show that AES-GCM is faster than AES-CCM when it comes to performance. We have come to the conclusion that the AES-GCM is more feasible to be used in applications where performance is the main concern.

VII. Acknowledgement

We respectfully acknowledge the assistance and support of Bhanuchandra Siddam and Sai Krishna Reddy Siripuram and their contribution towards the implementation and testing of the AES-GCM Authenticated Encryption mode of operation.

VIII. References

[1] M. Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. NIST Special Publication 800-38C., 2004.

[2] Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001.

[3] National Institute of Standards and Technology Special Publication 800-38C Natl. Inst. Stand. Technol. Spec. Publ. 800-38C 25 pages (May 2004)

[4] FIPS Publication 197, Advanced Encryption Standard (AES). U.S. DoC/NIST, November 26, 2001. Available at http://csrc.nist.gov/publications/.

[5] J Daemen, V Rijmen. The design of Rijndael: AES--the advanced encryption standard. Springer Verlag, 2002.

[6] Nyberg, K., & Heys, H. (2003). Selected areas in cryptography: 9th annual international workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002: Revised papers (Vol. 2595). Berlin: Springer-Verlag.

[7] NIST Computer Security Division's (CSD) Security Technology Group (STG) (2013). "Block cipher modes". Cryptographic Toolkit. NIST.

[8] http://venturebeat.com/2013/08/01/android-reaches-massive-80-market-share-windows-phone-hits-global-high-iphone-languishes/

[9] Lemsitzer, Wolkerstorfer, Felber, Braendli, Multi-gigabit GCM-AES Architecture Optimized for FPGAs. CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems, 2007.

[10] "Android Studio" http://developer.android.com/sdk/index.html

[11] Genymotion – Fast and easy Android Emulation https://www.genymotion.com/

[12] Cryptography for mobile security. Mitchell, Chris 1. Security for Mobility. Ed. Chris J Mitchell. TEE Press, 2004

[13] Daemen, Joan; Rijmen, Vincent (March 9, 2003). "AES Proposal: Rijndael" (PDF). National Institute of Standards and Technology. p. 1. Retrieved 21 February 2013.

[14] "Information technology -- Security techniques -- Authenticated encryption". 19772:2009. ISO/IEC. Retrieved March 12, 2013.

[15] IEEE: 802.1AE-media access control (MAC) security, draft 3.5. http://www.ieee802.org/1/pages/802.1ae.html (2005)

[16] IEEE: P1619.1/d12astandard for authenticated encryption with length expansion for storage devices. http://grouper.ieee.org/groups/1619/email/bin00084.bin (2006)

[17] Viega, J., McGrew, D.: The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (EPS). http://www.faqs.org/rfcs/rfc4106.htm (2005)

[18] B. Schneider. Applied Cryptography. JohnWiley Sons, NY, 1996.

[19] RFC 4309 Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)

[20] RFC 6655 AES-CCM Cipher Suites for Transport Layer Security (TLS)