XTEA CRYPTOGRAPHY IMPLEMENTATION IN ANDROID CHATTING APP

Irfan Syamsuddin[1], Siska Ihdianty[2], Eddy Tungadi[3], Kasim[4], Irawan[5]

[1,2,3,4,5] Politeknik Negeri Ujung Pandang , Indonesia
irfansyam@yahoo.com

***Abstract*-- Information security plays a significant role in information society. Cryptography is a key proof of concept to increasing the security of information assets and has been deployed in various algorithms. Among cryptography algorithms is Extended Tiny Encryption Algorithm. This study aims to describe a recent Android Apps to realize XTEA Cryptography in mobile form. In addition, a thorough example is presented to enable readers gain understanding on how it works within our Android Apps.**

***Keywords*: information security; cryptography; XTEA; mathematics visualization**

I. INTRODUCTION

The role of information security has gained more serious concerns particularly in the last two decades. The more dependent organizations on access, store, and transfer their information through the internet, the more probability of cyber security attacks they could face [1].

This concern is what causes obstruction in the delivery of information, therefore methods and techniques is strongly needed to securely manage the security of information. One of the paramount solutions is the application of cryptography[2].

Cryptography is used to randomize messages between two communicating parties, so that the other party who gets the encrypted message cannot decrypt it without the correct password. Cryptography disguises the meaning of a message, for example by scrambling or encoding the message. With cryptography, if the message falls into the wrong hands, it is hoped that the person will not get the desired information [3].

Cryptography has been applied in many ways although generally it is grouped into symmetric and asymmetric cryptography. While symmetric cryptography uses the same key in performing encryption and decryption, asymmetric cryptography creates different key for encryption and decryption [4][5][6].

Among many symmetric cryptography, Extended Tiny Encryption Algorithm (XTEA) was proposed by Wheeler and Needham in 1997 [1][2]. It is designed to cover weaknesses in the TEA which was previously introduced but then some serious bugs found.

It is a simple but powerful cryptography algorithm which was considered applicable to be implemented in today's mobile communication era.

In this paper, we introduce a work in progress Secure Chat application based on Android environment and then demonstrate the way message encrypted and decrypted using XTEA algorithm.

The rest of paper is organized in the following structure. Section 2 presents literature review of XTEA. In section 3, Secure Chat application based on Android is presented. Then, the next section presents the actual encryption and decryption mechanisms are exemplified in section 4. Finally, concluding remarks are given in section 5.

TEA is a symmetrical algorithm. TEA is a block cipher algorithm created by David J. Wheeler and Roger M. Needham of Cambridge University in 1994. TEA operates in a size of 64 bits and a key length of 128 bits[1][2].

The TEA is based on a feitsel network and has 32 turns. The key of K will first be divided into 4 internal keys, namely K [0..3] each 32 bits long. Each TEA round consists of two Feitsel rounds (see figure 2.4). The scheduling of the TEA key is very simple, namely K [0] and K [1] are used for odd rounds, while K [2] and K [3] are used for even rounds (William, 2009). The form of one round of encryption in the Feitsel network belonging to the Tiny Encryption (TEA) Algorithm, can be seen in Figure 1.

In 1997, David Wagner and Kesley found TEA susceptibility to equivalent key and related key attacks due to the simplicity of key scheduling. The equivalent key that TEA has is that for each key there are three other keys that produce the same ciphertext. These keys are obtained by reversing the Most Significant Bit (MSB) values in K [0] and K [1] or K [2] and K [3] so that the 128-bit key length will only produce 2126 different keys (William, 2009).

In the algorithm, there is a DELTA number which is obtained from the golden number formula: delta = which produces the number 0x9E3779B9. The following is an example of evidence of the existence of the equivalent key in the TEA.$(\sqrt{5}-1)2^{31}$.

As a result, Wheeler and Needham released the XTEA which was designed to cover weaknesses in the TEA in 1997.

TABLE I
Equivalent key in TEA (Source: William, 2009)

Plaintext	Key	Ciphertext
00000000 00000000	80000000 00000000 00000000 00000000	9327c497 31b08bbe
00000000 00000000	00000000 80000000 00000000 00000000	9327c497 31b08bbe
00000000 00000000	80000000 00000000 80000000 80000000	9327c497 31b08bbe
00000000 00000000	00000000 80000000 80000000 80000000	9327c497 31b08bbe

Extended Tiny Encryption Algorithm(XTEA) also operates in a block size of 64 bits and a key length of 128 bits. The form of the Feitsel network is still the same, only the difference between the Feitsel function and the key scheduling used. The key scheduling in XTEA is odd rounds using K [sum & 3], while even rounds use K] sum >> 1 & 3].

XTEA which is a derivative of the TEA algorithm is Feitsel cryptography and uses operations that are included in mixed algebra (orthogonal) such as XOR, ADD, and SHIFT. The form of one loop in the Feitsel network belonging to the Extended Tiny Encryption (XTEA) algorithm, can be seen in Figure 2.5

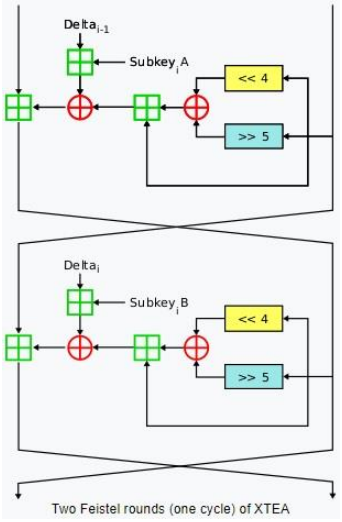

Fig 1. One round of encryption in XTEA's Feitsel network

According to William [1][2], the explanation of Figure 2.5 which is one rotation in XTEA's Feitsel network is as in the formula below:

- The inputted 128-bit key will be partitioned into 4 sublocks consisting of s [0] = 32 bits, s [1] = 32 bits, s [3] = 32 bits and s [3] = 32 bits.
- The plaintext block = 64 bits, then partitioned into 2 sub-blocks v0 = 32 bits and v1 = 32 bits. Then initialize the process into a variable i = 1.
- The encryption process uses plaintext v0 sub-blocks and keys with the formula v0 + = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3]). Use plaintext sub-blocks v1 and key v1 + = ((((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3]).
- The decryption process uses sub-block ciphertext v1- and key with the formula v1- = (((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3])). Use the plaintext sub-block v0- = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3])).
- Then add the process i = i + 1.
- If I have not reached 32 processes (32 rounds), then it is repeated to step 5.

The Delta number used by XTEA is the same as the TEA, which is 0x9E3779B9. In the description, the variable sum is initialized with the value: -957401312 (0xC6EF3720) which is obtained from the result of the DELTA number multiplied by 32 round = DELTA << 5.

Therefore, it is claimed that while keep using a very limited calculation process, XTEA provides strong encryption and decryption mechanisms adequate for mobile requirements currently [6][7].

This has motivated us to implement XTEA into a simple Secure Chat apps in Android environment in order to show its powerful and simplicity in securing messages [8][9].

II. METHODE

Secure Chat Application is implemented in Android environment. The basic concept of how it works is presented as follows before sending, messages can be encrypted. Then the chat is sent and received by the recipient's cellphone. In order for it to be read by the recipient, the reverse process must be carried out, namely if it is encrypted, it is decrypted after which the new message can be read according to the original message.

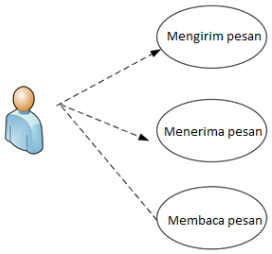

Fig 2. Main Use Case

In Figure 3.4, the Use Case illustrates that the user types a message in the textbox which will later be taken by the characters in it. Then the message will be received by the designated number. Recipients can read messages normally if they have the same application.

For more details, an explanation will be made at the next stage of the process. Before the sender sends the message, when you finish typing the message, the sender can encrypt the message. A detailed explanation can be seen in Figure 3.4.

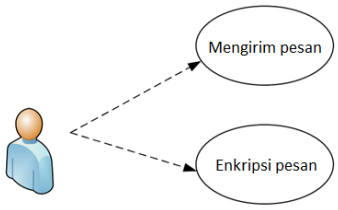

Fig 3. Use Case Send

In the next process, after the sender sends the message, the application in the recipient will receive a notification in the form of an incoming alert. When there is an incoming alert, the message will be automatically stored in the android sms container then the application will access this message when it is decrypted in the inbox. For more details, see Figure 3.5.

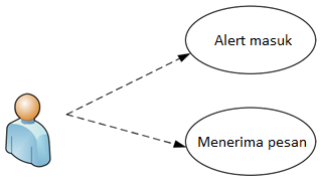

Fig 4. Use case receive

Fig 5. Secure Chat interface

The main interface Secure Chat App in Android is presented in figure 5 while figure 6 shows an incoming chat from other party. In addition, figure 7 shows the actual data in firebase.

Fig 6. Incoming secure message

Fig 7. Visualization of secure message in firebase

III. Result and Discussion

As mentioned in previous section, the Android apps enable secure chat between two parties using XTEA to secure chatting information. The case study is Besse chats with Amini and Amini sent message “poltek upandang” to Besse. In this section, we discuss in details how XTEA handles the message given example.

a. XTEA Encryption

First, it search for subkey s [0] -s [3]. If it is known that the key used for encryption is 16 bytes = 128 bits, namely: Chiperkey: 1234567890123456. The initial stage is to change the cipherkey to decimal form by looking at the ASCII table.

TABLE II
The cipherkey in ASCII

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
49	50	51	52	53	54	55	56	57	48	49	50	51	52	53	54

Chiperkey: 1234567890123456. The initial stage is to change the cipherkey to decimal form by looking at the ASCII table.The colored table is the decimal value of the ascii

cipherkey table: 1234567890123456.

The next step is to enter the decimal value of the cipher key into the key randomization function by performing shift and OR operations to generate subkeys. Enter the cipherkey into a block of 16 bytes, where one randomization process requires 4 bytes of the cipherkey.

S [0] = ((49 AND 255) Shl 24) OR ((50 AND 255) Shl 16) OR ((51 AND 255) Shl 8) OR ((52 AND 255)) S [0] = ((49 Shl 24) OR (50 Shl 16) OR (50 Shl 8) OR (52))
S [0] = 822083584 OR 3276800 OR 13056 OR 52
S [0] = 825373492

S [0] is a key generator used for the encryption and decryption process in the XTEA algorithm, then it is repeated so that the values S [0] to S [3] are obtained.

S [1] = ((53 AND 255) Shl 24) OR ((54 AND 255) Shl 16) OR ((55 AND 255) Shl 8) OR ((56 AND 255))
S [1] = ((53 Shl 24) OR (54 Shl 16) OR (55 Shl 8) OR (56))
S [1] = 889192448 OR 3538944 OR 14080 OR 56
S [1] = 892745528

S [2] = ((57 AND 255) Shl 24) OR ((58 AND 255) Shl16) OR ((49 AND 255) Shl 8) OR ((50 AND 255))
S [2] = ((57 Shl 24) OR (58 Shl 16) OR (49 Shl 8) OR (50))
S [2] = 956301312 OR 3145728 OR 12544 OR 50
S [2] = 959459640

S [3] = ((51 AND 255) Shl 24) OR ((52 AND 255) Shl16) OR ((53 AND 255) Shl 8) OR ((54 AND 255))
S [3] = ((51 Shl 24) OR (52 Shl 16) OR (53 Shl 8) OR (54))
S [3] = 855638016 OR 3407872 OR 13568 OR 54
S [3] = 859059510

So that we get the key scheduling value (subkey), as follows:
S [0] = 825373492
S [1] = 892745528
S [2] = 959459640
S [3] = 859059510

This subkey is used for the encryption and decryption process in the XTEA algorithm. The next stage is the calculation of the encryption process. The encryption process in the XTEA algorithm is done by taking each plaintext per 8 byte block and breaking it into odd and even rounds. An example of encryption in the XTEA algorithm, if it is known that the key is 16 bytes long and plaintext will be used for encryption with 16 bytes.

Chiperkey : 1234567890123456
Plaintext : "poltek Upandang"

The initial stage is to change the plaintext into decimal form by looking at the ASCII table

TABLE III
Plaintext into ASCII

p	**o**	**L**	**t**	**e**	**k**	**space**	**u**
112	111	108	116	101	107	32	117
p	a	n	d	a	n	g	space
112	97	110	100	97	110	103	32

Solve the plaintext per 8byte block. Each block is broken down into odd rounds and even rounds, each of which has 4 byte characters.

TABLE IV
Plaintext into ASCII

p	o	l	t	e	k	space	u
112	111	108	116	101	107	32	117

v0 = ((112 AND 255) Shl 24) OR ((111 AND 255) Shl 16) OR ((108 AND 255) Shl 8) OR ((116 AND 255))
v0 = ((112 Shl 24) OR (111 Shl 16) OR (108 Shl 8) OR (116))
v0 = (1879048192 OR 7274496 OR 27648 OR 116)
v0 = 1886350452
v1 = ((101 AND 255) Shl 24) OR ((107 AND 255) Shl 16) OR ((32 AND 255) Shl 8) OR ((117 AND 255))
v1 = ((101 Shl 24) OR (107 Shl 16) OR (32 Shl 8) OR (117))
v1 = (1694498816 OR 7012352 OR 8192 OR 117)
v1 = 1701519477

TABLE V
Plaintext into ASCII

p	a	n	d	a	n	g	space
112	97	110	100	97	110	103	32

v0 = ((112 AND 255) Shl 24) OR ((97 AND 255) Shl 16) OR ((110 AND 255) Shl 8) OR ((100 AND 255))
v0 = ((112 Shl 24) OR (97 Shl 16) OR (110 Shl 8) OR (100))
v0 = (1879048192 OR 6356992 OR 28160 OR 100)
v0 = 1885433444
v1 = ((97 AND 255) Shl 24) OR ((110 AND 255) Shl 16) OR ((103 AND 255) Shl 8) OR ((32 AND 255))
v1 = ((97 Shl 24) OR (110 Shl 16) OR (103 Shl 8) OR (32))
v1 = (1627389952 OR 7208960 OR 26368 OR 32)
v1 = 1634625312

Initialize DELTA with the value 0x9E3779B9 (-1640531527 in integer) and 32 rounds (n). Calculate the value of v0 with the initial value sum = 0. Calculations are performed using integer values that have a limit of -2147483648 to 21447483647.

The value of S [0..3] is taken from the key generator that has been obtained from the key generation process. The calculation below is the calculation of the first sub-block from the plaintext “poltek u”. The value of v1 is obtained from the sub-block calculation for the word "poltek u", namely v1 = 1701519477

v0 + = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3])
v0 + = ((((1701519477 Shl 4) XOR (1701519477 Shr 5) +1701519477) XOR (0 + S [0 AND 3])
v0 + = (((1454507856 XOR 53172483) + 1701519477) XOR (0 + S [0])
v0 + = (((1454507856 XOR 53172483) + 1701519477) XOR S [0])
v0 + = (1436114515 + 1701519477) XOR 825373492
v0 + = -1157333304 XOR 825373492
v0 + = -1976152580
= 1886350452 + (- 1976152580)
v0 = -89802128
Sum + = DELTA
Sum = 0 + (- 1640531527)
= -1640531527
v1 + = (((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3])
v1 + = (((-89802128 Shl 4) XOR (-89802128 Shr 5) + (-89802128)) XOR (-1640531527 + S (-1640531527 Shr 11 AND 3])
v1 + = (((-1436834048 XOR (131411411) + (- 89802128)) XOR (-1640531527 + S [3])
v1 + = ((-1383167277) + (- 89802128)) XOR (-1640531527+ 859059510)
v1 + = -1472969405 XOR -781472017
v1 + = 2036329388
= 1701519477 + 2036329388
v1 = -557118431
While the calculation below is the calculation of the second sub-block from the plaintext "view"
v0 + = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3])
v0 + = (((1634625312 Shl 4) XOR (1634625312 Shr 5) + 1634625312) XOR (-1640531527 + S [-1640531527 AND 3])
v0 + = ((((384201216 XOR 51082041) + 1634625312) XOR (-1640531527 + S [1])
v0 + = ((((384201216 XOR 51082041) + 1634625312) XOR (-1640531527 + 892745528)
v0 + = ((((384201216 XOR 51082041) + 1634625312) XOR (-747785999)
v0 + = (367853881 + 1634625312) XOR (-747785999)
v0 + = 2002479193 XOR -747785999
v0 + = -1539909464
= 1885433444 + -1539909464
v0 = -1228224047
Sum + = DELTA Sum = 0 + (- 1640531527) = -1640531527
v1 + = (((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3])
v1 + = (((-1228224047 Shl 4) XOR (-1228224047 Shr 5) + (1228224047)) XOR (-1640531527 + S [-1640531527 Shr 11 AND 3])
v1 + = (((1823251728 XOR (95835726) + (- 1228224047)) XOR (-1640531527 + S [3])
v1 + = ((1763363678) + (- 1228224047)) XOR (-1640531527+ 859059510)
v1 + = 535139631 XOR -781472017
v1 + = -829539392
= 1634625312 + (-829539392)
v1 = 805085920
Repeat the calculation until 32 rounds so that the final values of v0 and v1 are obtained, namely:
The first sub-block v0 = -539453652 v1 = -283483773
Second sub block v0 = -1857333815 v1 = 1111744780
Convert the final values v0 and v1 to ASCII characters by shifting. The decimal value taken is in the form of a byte (8 bits) value.
The first sub block
v0 Shr 24 = -539453652 Shr 24 = -33
v0 Shr 16 = -539453652 Shr 16 = -40
v0 Shr 8 = -539453652 Shr 8 = -105
v0 Shr 0 = -539453652 Shr 0 = 44
v1 shr 24 = -283483773 Shr 24 = -17
v1 Shr 16 = -283483773 Shr 16 = 26
v1 Shr 8 = -283483773 Shr 8 = 97
v1 Shr 0 = -283483773 Shr 0 = -125
Second sub block
v0 Shr 24 = -1857333815 Shr 24 = -111
v0 Shr 16 = -1857333815 Shr 16 = 75
v0 Shr 8 = -1857333815 Shr 8 = 85
v0 Shr 0 = -1857333815 Shr 0 = -55
v1 shr 24 = 1111744780 Shr 24 = 66
v1 Shr 16 = 1111744780 Shr 16 = 67
v1 Shr 8 = 1111744780 Shr 8 = -31
v1 Shr 0 = 1111744780 Shr 0 = 12

The decimal value taken from the shift result above is then converted to hex as in the table below

TABLE VI
Encryption result

Decimal (byte)	-33	-40	-105	44
Binary	11011111	11011000	1—1-111	00101100
Character	β	∞	-	,
Decimal (byte)	-17	26	97	-125
Binary	11101111	00011010	01100001	10000011
Character	ï		a	*f*
Decimal (byte)	-11	75	85	-55
Binary	10010001	01001011	01010101	11001001
Character	‘	K	U	É
Decimal (byte)	66	67	-31	12

Binary	01000010	01000011	11100001	00001100
Character	B	C	á	

From the encryption process above, it will obtain 8 characters of ciphertext. So that the ciphertext derived fom plaintext "upandang poltek" after encryption process is
"**B ∞—, ï a *f* 'I ÉBC á**"

b. XTEA Decryption Process

The steps to change the encrypted data (ciphertext) in the database to plaintext in the chat room are as follows:
The decryption process in the XTEA algorithm is carried out by taking each ciphertext per 8 byte block and breaking it into odd and even rounds. Repeat these steps for 32 rounds. Example of decryption on the XTEA algorithm. For example the Ciphertext of "upandang poltek": "ßØ—, ï a*f*'KUÉBCá". As for the conversion of ciphertext characters into decimal form as in the table below

TABLE VII
Chipertext into decimal

β	∞	-	,	ï		A	*f*
223	216	151	44	239	26	97	131
'	K	U	É	B	C	Á	
145	75	85	201	66	67	225	12

After that enter the same decryption key as the encryption key, namely: 1234567890123456 so that the key or subkey scheduling value can be obtained. Subkey S [0..3] is obtained when performing the key generation process and is also used to perform the decryption process. The value of the S subkey [0..3] is:
S [0] = 825373492 S [2] = 959459640
S [1] = 892745528 S [3] = 859059510

Breaks the ciphertext per 8-byte block. Each block is broken down into odd rounds and even rounds, each of which has 4 byte characters.

Breaks the ciphertext per 8-byte block. Each block is broken down into odd rounds and even rounds, each of which has 4 byte characters.

TABLE VIII
Decryption process

B	∞	-	,	ï		A	*f*
223	216	151	44	239	26	97	131

v0 = ((223 AND 255) Shl 24) OR ((216 AND 255) Shl 16) OR ((151 AND 255) Shl 8) OR ((44 AND 255))
v0 = ((223 Shl 24) OR (216 Shl 16) OR (151 Shl 8) OR (44))
v0 = (-553648128 OR 14155776 OR 38656 OR 44)
v0 = -539453652
v1 = ((239 AND 255) Shl 24) OR ((26 AND 255) Shl 16) OR ((97 AND 255) Shl 8) OR ((131 AND 255))
v1 = ((239 Shl 24) OR (26 Shl 16) OR (97 Shl 8) OR (131))
v1 = (-285212672 OR 1703936 OR 24832 OR 131)
v1 = -283483773

TABLE IX
Decryption process

'	K	U	É	B	C	á	
145	75	85	201	66	67	225	12

v0 = ((145 AND 255) Shl 24) OR ((75 AND 255) Shl 16) OR ((85 AND 255) Shl 8) OR ((201 AND 255))
v0 = ((145 Shl 24) OR (75 Shl 16) OR (85 Shl 8) OR (201))
v0 = (-1862270976 OR 4915200 OR 21760 OR 201)
v0 = -1857333815
v1 = ((66 AND 255) Shl 24) OR ((67 AND 255) Shl 16) OR ((225 AND 255) Shl 8) OR ((12 AND 255))
v1 = ((66 Shl 24) OR (67 Shl 16) OR (225 Shl 8) OR (12))
v1 = (1107296256 OR 4390912 OR 57600 OR 12)
v1 = 1111744780

Then, initialize DELTA number with value 0x9E3779B (-1640531527 in integer) and round (n) of 32 rounds. Calculate the value of v0 and v1 as many as 32 rounds. Calculate the value of v0 with the initial value sum = DELTA * round = -957401312 (0xC6EF3720). Calculations are also performed using an integer value that has a limit of -2147483648 to 2147483647. The calculation below is the calculation of the first sub-block of the ciphertext "ßØ—, ï a*f*"
v1- = (((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3)])
v1- = (((-539453652 Shl 4) XOR (-539453652 Shr 5) + (-539453652)) XOR (-957401312 + S (-957401312 Shr 11 AND 3])
v1- = (((-41323840 XOR 117359801) + (- 539453652)) XOR (-957401312 + S [2])
v1- = (((-76040583 + (- 539453652)) XOR (-957401312 + 959459634)
v1- = -615494235 XOR 2058322
v1- = -615569929 = -283483773 - (-615569929)
v1 = 332086156
Sum - = DELTA
Sum = (-957401312) - (-1640531527)
= 683130215
v0- = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3]))
v0- = (((332086156 Shl 4) XOR (332086156 Shr 5) + 332086156) XOR (683130215 + S [683130215 AND 3]))
v0- = (((1018411200 XOR 10377692) + 332086156) XOR (683130215 + S [3]))

v0- = ((((1009639708 + 332086156) XOR (683130215 + 859059510)
v0- = 1341725864 XOR 1542189725
v0- = 336784949
= (- 539453652) - 336784949
v0 = -876238601

Whereas in the calculation below, is the second sub-block calculation of the ciphertext "'KUÉBCá"
v1- = (((v0 Shl 4) XOR (v0 Shr 5) + v0) XOR (sum + S [sum >> 11 AND 3)])
v1- = (((-1857333815 Shl 4) XOR (-1857333815 Shr 5) + (-1857333815)) XOR (-957401312 + S (-957401312 Shr 11 AND 3])
v1- = (((347430032 XOR 76176046) + (- 1857333815)) XOR (-957401312 + S [2])
v1- = ((((272565822 + (-1857333815)) XOR (-957401312 + 959459634)
v1- = -1584767993 XOR 2058322
v1- = -1584057259
= 1111744780 - (-1584057259) v1 = -1599165257
Sum - = DELTA
Sum = (-957401312) - (-1640531527)
= 683130215
v0- = (((v1 Shl 4) XOR (v1 Shr 5) + v1) XOR (sum + S [sum AND 3]))
v0- = (((-1599165257 Shl 4) XOR (-1599165257 Shr 5) + (-1599165257) XOR (683130215 + S [683130215 AND 3]))
v0- = ((((183159664 XOR 84243813) + (-1599165257) XOR (683130215 + S [3]))
v0- = (((267370005 + (-1599165257) XOR (683130215 + 859059510)
v0- = -1331795252 XOR 1542189725
v0- = -344614831
= (-1857333815) - (-344614831)
v0 = -1512718984

Next, repeat the calculation until 32 rounds so that the final values of v0 and v1 are obtained, namely:
The first sub-block v0 = 1886350452 v1 = 1701519477
Second sub block v0 = 1885433444 v1 = 1634625312
After that, change the integer value from the calculation result to ASCII characters by moving it. The decimal value taken is in the form of a byte (8 bits) value.

First sub block v0 Shr 24 = 1886350452 Shr 24 = 112v0 Shr 16 = 1886350452 Shr 16 = 111v0 Shr 8 = 1886350452 Shr 8 = 108v0 Shr 0 = 1886350452 Shr 0 = 116v1 shr 24 = 1701519477 Shr 24 = 101v1 Shr 16 = 1701519 107v1 Shr 8 = 1701519477 Shr 8 = 32v1 Shr 0 = 1701519477 Shr 0 = 117
Second subblock v0 Shr 24 = 1885433444 Shr 24 = 112v0 Shr 16 = 1885433444 Shr 16 = 97v0 Shr 8 = 1885433444 Shr 8 = 110v0 Shr 0 = 1885433444 Shr 0 = 100v1 shr 24 = 1634625312 Shr 24 = 97
v1 Shr 16 = 1634625312 Shr 16 = 110v1 Shr 8 = 1634625312 Shr 8 = 103v1 Shr 0 = 1634625312 Shr 0 = 32

The decimal value taken from the shift result above is then converted to hex as in the table.

TABLE X
Decryption result

Decimal (byte)	112	111	108	116
Binary	01110000	01101111	01101100	01110100
Character	p	o	l	t
Decimal (byte)	101	107	32	117
Binary	01100101	0101011	00100000	01110101
Character	e	k	space	u
Decimal (byte)	112	97	110	100
Binary	01110000	01100001	01101110	01100100
Character	p	a	n	d
Decimal (byte)	97	110	103	32
Binary	01100001	01101110	01100111	00100000
Character	a	n	g	space

Finally the plaintext derived from Ciphertext: "ßØ—, ï a*f*'KUÉBCá" is "**poltek upandang**".

IV. Conclusion

We have introduced an implementation of XTEA cryptography in the form of Secure Chat application in Android environment. A simple chat between two parties also presented followed by in details calculation procedure of XTEA for both encryption and decryption process.

This approach is useful for students to learn mathematical procedures behind XTA cryptography which may lead to applying more complicated algorithm in different application.

V. Acknowledgment

Gratefully acknowledge the contributions to all authors. Also, Politeknik Negeri Ujung Pandang and all support.

VI. References

[1.] D. Wheeler and R. Needham, TEA, a Tiny Encryption Algorithm (1994)

http://www.ftp.cl.cam.ac.uk/ftp/papers/ djw-rmn/djw-rmn-tea.html

[2.] D. Wheeler and R. Needham, XTEA (eXtended TEA), Technical report (1997)

[3.] Y.Ko, H. Seokhie, W.Lee, S.Lee, and JS Kang. "Related key differential attacks on 27 rounds of XTEA and full-round GOST." In International Workshop on Fast Software Encryption, pp. 299-316. Springer, Berlin, Heidelberg, (2004).

[4.] G.N. Khan, X. Yu, and F.Yuan. "A novel XTEA based authentication protocol for RFID systems." In 2011 XXXth URSI General Assembly and Scientific Symposium, pp. 1-4. IEEE, (2011).

[5.] J.P. Kaps, "Chai-tea, cryptographic hardware implementations of xtea." In International Conference on Cryptology in India, pp. 363-375. Springer, Berlin, Heidelberg, (2008).

[6.] M.H.AlMeer, "FPGA implementation of a hardware XTEA light encryption engine in co-design computing systems." In 2017 Seventh International Conference on Innovative Computing Technology (INTECH), pp. 26-30. IEEE, (2017).

[7.] P.Israsena, "On XTEA-based encryption/ authentication core for wireless pervasive communication." In 2006 International Symposium on Communications and Information Technologies, pp. 59-62. IEEE, (2006).

[8.] T.Isobe, and K.Shibutani. "Security analysis of the lightweight block ciphers XTEA, LED and Piccolo." In Australasian Conference on Information Security and Privacy, pp. 71-86. Springer, Berlin, Heidelberg, (2012).

[9.] S.Maitra, and K.Yelamarthi. "Rapidly deployable IoT architecture with data security: Implementation and experimental evaluation." Sensors 19, no. 11 (2019.