Logical Cryptography in Modern Cryptosystems

Dr.Kamal Shah[1], Mr.Vikas Kaul[2], Pratik Kanani[3]
[1]Professor, [2]Asst. Professor, [3]M.E.I.T (pursuing)
[1,2,3] Department of Information Technology, Thakur College of Engineering and Technology
[1,2,3] University of Mumbai, Mumbai – 400101, India

Abstract

Due to advancement in the wireless networks the data transfer speed has increased, where high end applications are demanding more data rates with high level of security, it is causing threat to Information Technology. In such scenarios there is a need for efficient crypto system, which will establish secured connections in less time with more key length. The traditional security algorithms available so far are able to generate the ciphers till certain bits of key lengths not higher than that. In this paper we describe some security functions which help to generate ciphers of higher length by making use of logical operations. Moreover we use inbuilt features of java programming language to implement different kinds of security algorithms and mechanisms that are discussed here with their comparative experimental studies.

***Keywords*—**Logical Operations, Dafun(), IDafun(), ExOR, HMAC, A-RSA, RNG, stringToBigInteger, bigIntegerToString, Secret Key.

I. INTRODUCTION

Cryptographic science always attempts to find a better and efficient way to encrypt and decrypt the messages. Such methods should be easy to understand and implement but impossible to reverse. Till now many security algorithms have been suggested and implemented. e.g. RSA, Diffie-Hellman, AES, DES and many more. Some of these are based on symmetric and asymmetric key cryptography. The symmetric key cryptography is considered faster than asymmetric key cryptography. Most of these algorithms make use of prime numbers, which makes it very difficult to recover it by permutation and combination. Moreover all these algorithms are dealing at bit levels to generate its cipher text from the plain text and vice-versa. Some of these algorithms are irrevocable, that's why they are useful to check integrity of the messages. These algorithms are used as MAC functions.

Diffie-Hellman and RSA algorithms were breakthroughs to 1970s public-key cryptography [1]. Diffie-Hellman algorithm is used to share a secret key in an unsecured channel and determines the private secret key from the publicly known parameters. Preliminary Number theory helps to understand the mystery of numbers implemented in such algorithms. It includes Chinese Remainder theorem, Euler's Theorem and Fermat's Little Theorem [1]. While implementing such algorithms one has to instigate the logic which will work at bit levels. Due to advancement in networking technologies and computing powers the applications are demanding higher data rates and more levels of security. The key sizes needs to be longer to ensure required security ranks. In each algorithm, going at bit level and inverting again increases the level of abstractions, while use of logical operations avoids the same. In this paper we propose some logical operations and its backend needed for its implementation. All the functions described here are implemented and the results are discussed in each section. One of the class called as

BigInteger [2] in java is used here to implement such logical cryptographic functions and works at bit level.

All functions mentioned in this paper are implemented on the following test bed configurations. The resulting time shown in different time units [3] based on its feasibility.

TABLE I. HARDWARE AND SOFTWARE CONFIGURATION OF TEST BED

Platform	Windows XP, 32-bit
RAM	1472 MB
CPU	Intel CPU 2140, 1.66 GHz
IDE	NetBeans IDE 7.2

Rest of the paper is organized as follows. Section 2 refers to logical operations and their uses, Section 3 discusses proposed Advance RSA algorithm using BigInteger class. Section 4 talks about needed type conversions. Section 5 specifies performance analysis for random numbers. Section 6 spells out conclusion with future work.

II. LOGICAL OPERATIONS IN MODERN CRYPTOGRAPHY

In this section we put forward different types of logical operations [4] and how they can be used in modern cryptography to generate plain-cipher text and vice-versa.

A. *Logical Operations*

The operations used in this approach and their meanings are as follows.

1) $X +_2 Y$: where $+_2$ indicates Logical OR operation which ignores most significant carry.

2) $X \oplus Y$: where $\oplus$ represents Logical Ex-OR function.

3) $X+Y$: where + represents addition.

First two operations are used such that the number of resulting bits are same as number of operand bits.

B. *Dafun() and IDafun()*

Dafun() and IDafun() is used for data authentication [4]. Dafun() is used to send a secret data where as inverse of it, IDafun() is used to regain the secret data sent in respective Dafun(). The functions are defined as follows.

$$\text{Dafun}(a,b,c) = (a \oplus b) + c$$

On receiving side

$$a = \text{IDafun}(a,b,c) = \begin{cases} (x - c) \oplus b, & \text{if } x \geq c \\ (x + \bar{c} + 1) \oplus b, & \text{if } x < c \end{cases}$$

Where x= Dafun(a,b,c). User p wants to send a secret key 'a' to user q, provided both has 'b' and 'c'. User p will send encrypted 'a' by using 'b', 'c' and Dafun(). On the other hand user q receives the message and using IDafun(), 'b' and 'c' it recovers the secret key 'a'. So Dafun() provides sharing of secret message and its decryption only by legitimate users. The time taken by Dafun() and IDafun() is given in Table 1.

TABLE II. TIME TAKEN BY DAFUN() AND IDAFUN() IN MS

Bit Lengths	Dafun()	IDafun()
512	0.02626	0.02737
768	0.02821	0.04274
1024	0.04525	0.03827
1280	0.05129	0.04567
1536	0.06062	0.04889

C. Exclusive-OR Operation

The logical Exclusive – OR operation called as ExOR and is represented by ⊕ sign. The process of encryption and decryption by using this operation is shown is below [5].

Encryption : Cipher = Text ⊕ Key
Decryption : Text = Cipher ⊕ Key

When the plain – text message is ExORed with the Key it generates the Cipher Text. And when the same key is ExORed with the cipher text it produces the plain text again. The plain text and keys of different bit lengths are taken and their respective time is measured. Comparison is made with the existing Cipher class of java to produce the same length ciphers using AES [6,7]. Table 3 shows the performance analysis.

TABLE III. ENCRYPTION AND DECRYPTION TIME FOR EXOR OPERATION AND CIPHER CLASS IN MS

Bit Lengths	ExOR		Cipher Class	
	Encryption	**Decryption**	**Encryption**	**Decryption**
512	0.02078	0.02564	0.625219	0.453689
768	0.02318	0.02862	0.820216	6.619556
1024	0.02598	0.03017	7.418261	0.845079
1280	0.03073	0.03156	7.400661	0.874972
1536	0.03464	0.03296	4.872965	12.59042
1792	0.03631	0.04162	5.212115	12.30519
2048	0.05112	0.04469	3.402388	15.14019

Though Cipher class takes varying time for variable message lengths but the time taken by ExOR operation for encryption and decryption is very less.

D. *Secret Key Generation from other keys*

The Secret Key can be generated with the help of other keys present or by using prime numbers and their combinations. It can be defined as follows.

$$SK = (P)^{Q} \bmod R$$

Where P,Q and R are the keys used in encryption and decryption or they can be a long bit next probable prime numbers. The operations can be implemented by modPow() of BigInteger class. Required time to generate variable length SK is as shown.

TABLE IV. SK TIME REQUIREMENTS IN MS

Bit Lengths	Secret Key
256	0.292
512	0.769
768	2.277
1024	4.653
1280	6.948

E. *Hash Based Message Authentication Codes*

Hash Based Message Authentication Codes called as HMAC in short, are used to find the message integrity of the transmitted messages and guarantees that messages are not tempered. The advantage of this function is attacker even if decrypts it , he will get a sum of keys but not actual keys. In BigInteger class sufficient methods are implemented by using them, powerful HMACs can be generated.e.g.

$$HMAC((AK_9 \oplus P_{R3}) +_2 AK_8)$$
$$HMAC((IMEI +_2 P_{M1}) \oplus P_{R2})$$

The resulting time is shown in Table 5.

TABLE V. TIME TAKEN BY HMAC IN MS

Bit Lengths	HMAC
768	0.03408
1024	0.03911
1280	0.04749
1536	0.05225
1792	0.05587
2048	0.06257

III. BIGINTEGER CLASS AND ADVANCE RSA ALGORITHM

The RSA algorithm is a public-private key based algorithms. Here it is called as Advance RSA because this RSA algorithm helps user to generate variable long bit public and private key lengths of any sizes. This algorithm is implemented without using any inbuilt libraries of java.

A. *Big Prime Numbers*

Big prime numbers are required to generate the bigger key length, e.g. the length of the primes are almost equal to half of the key lengths [8]. The BigInteger class helps to generate two prime numbers p and q required to find the e, d and n as

```
BigInteger p = BigInteger.probablePrime(size/2,rnd);
BigInteger q = p.nextProbablePrime();
```

Where rnd is object of Random class, helps to generate random prime numbers. Their time taken in ms is shown in table below.

TABLE VI. TIME TAKEN BY PRIME NUMBERS

Key Bit Lengths	Prime Numbers p and q
512	47
768	125
1024	156
1280	454
1792	625
2048	641

B. *Advance RSA Algorithm*

According to CISCO guidelines, the RSA-2048 is considered to be the minimum security level needed for today. With the help of available methods and classes present in math.BigInteger class [2,8], the A-RSA is implemented. The resulting time to generate pair of public and private key is shown in the table below.

TABLE VII. TIME COMAPRISON FOR RSA IN MS

Key Bit Lengths	A-RSA
256	47
512	125
768	610
1024	766
1280	850

IV. TYPE CONVERSIONS

The java BigInteger class has Number as a parent class. So it works with numeric data types. But in communications a string data type is used most frequently. Here two algorithms are suggested which converts a String to BigInteger and vice-versa [9].

A. *stringToBigInteger*

This method suggests a conversion mechanism which converts a String type to a BigIntegertype. And the given output explains the steps in flowchart which is as follows

1) *Output for stringToBigInteger Conversion*

Let the input is s=”pratik”;

code for p is 112
b before shifting is 0
b after shifting by 8 left is 0
b after OR with c is 112

code for r is 114
b before shifting is 112
b after shifting by 8 left is 28672
b after OR with c is 28786

code for a is 97
b before shifting is 28786
b after shifting by 8 left is 7369216
b after OR with c is 7369313 ……for k it is

code for k is 107
b before shifting is 482955326569
b after shifting by 8 left is 123636563601664
b after OR with c is 123636563601771

2) *Flowchart*

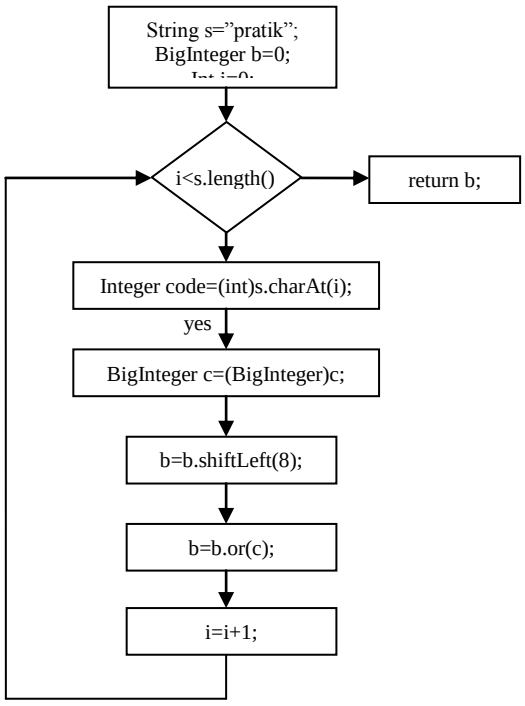

Figure 1. Flowchart for String to BigInteger Conversion

B. *bigIntegerToString*

The String converted to BigInteger needs to be switched back to the string again after the operations are over. The suggested next flowchart converts BigInteger to respective string. And the given output explains the operations.

1) *Output for bigIntegerToString Conversion*

Let the input is b=123636563601771.

c is 255
cb is 107
cv is k
s is k
B after shifting 8 bytes is 482955326569

c is 255
cb is 105
cv is i
s is ik
B after shifting 8 bytes is 1886544244

c is 255
cb is 116
cv is t
s is tik
B after shifting 8 bytes is 7369313…….for last it is

c is 255
cb is 112
cv is p
s is pratik
B after shifting 8 bytes is 0

2) Flowchart

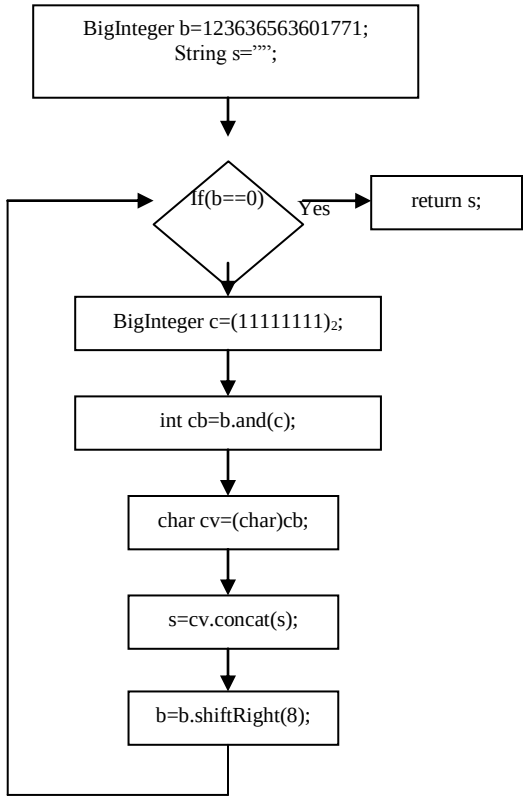

Figure 2. Flowchart for BigInteger to String Conversion

The time duration for variable length conversions are as follows. Time is measured in ms.

TABLE VIII. TIME REQUIREMENT FOR DATA TYPE CONVERSIONS

Message Bit Lengths	stringToBigInteger conversion	bigIntegerToString
256	<1ms	<1ms
512	<1ms	<1ms
768	<1ms	<1ms
1024	<1ms	<1ms
1280	<1ms	<2ms
1792	<1ms	<2ms
2048	<1ms	<2ms

V. RANDOM NUMBERS

A. *Characteristics of Random Numbers*

The random numbers are used in cryptographic applications because they possess some characteristics as follows [10,11].

1) *Uniformity:* Random numbers are equally probable everywhere.

2) *Independence:* The next random variable does not depends on the previous random variable.

3) *Unpredictability:* They do not follow any patterns.

4) *Long Repeating Periods:* It requires such a long time period to generate the same number generated before.

5) *Stability:* It generates same random number for the same seed inputed.

6) *Initial Sensitivity:* Two seeds with very less difference makes large possible difference in random variables.

B. *Practical Generation of Random Numbers*

The random numbers are generated by using Random Number Generators (RNG). In java this is possible by the Random class and to hold the big random numbers it requires BigInteger class.

```
BigInteger b = new BigInteger(SIZE, new Random());
```

The time of generation is as follows.

TABLE IX. TIME IN MS REQUIRED BY RANDOM NUMBER GENERATOR

Random Number length	Time
512 bit	0.781384
768 bit	1.151384
1024 bit	1.152096
1280 bit	1.433982
1792 bit	1.208020
2048 bit	1.251835

VI. CONCLUSION AND FUTURE WORK

Various available JCA frameworks and its available data types are studied and implemented to establish security functions of different key lengths. Their respective time consumption shows that they are feasible and can be implemented for real time applications. But the use of logical functions in cryptography is more efficient, flexible and from results shown it is clear that they are feasible too. They can be extended for higher bit key lengths and complies the use of future

cryptosystem needs. Moreover they take less time to establish a good security schemes. In future more such functions can be derived and by using multiple keys, an algorithm can be designed where not only the number of keys but also the order of the keys can be taken in consideration. So that attacker need all keys in desired order to decrypt the messages and it reduces the possible attacks by permutations and combinations.

REFERENCES

[1] RSA and Diffie Hellman key exchange. (2013,September,28). Vanilla47. [Online]. Available: http://vanilla47.com/PDFs/Cryptography/RSA%20Cryptography/RSA%20In%20General/RSA_ENCRYPTION_AND_DIFFIE_HELLMAN_KEY_EXCHANGE.pdf

[2] BigInteger. (2014,January,10). CIS. [Online]. Available: http://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.math.BigInteger.html

[3] Time Converter. (2014,February,09). Unitconversion. [Online]. Available: http://www.unitconversion.org/unit_converter/time.html

[4] Y. Huang, F. Leu and Y.Sun, "A Secure Communication System by Integrating RSA and Diffie-Hellman PKDS in 4G Environments and an Intelligent Protection-key Chain with a Data Connection Core," IEEE, International Symposium on Industrial Electronics, 2013, pp. 1-6.

[5] Detection of Eavesdropping in Quantum Key Distribution using Bell's Theorem and Error Rate Calculations. (2014,March,15). Raysforexcellaence. [Online]. Available: http://www.raysforexcellence.se/wp-content/uploads/2013/02/Detection-of-Eavesdropping-in-Quantum-Key-Distribution-using-Bells-Theorem-and-Error-Rate-Calculations.pdf

[6] Java Cryptography Architecture standard algorithm name documentation. (2014,March,20). [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#Cipher

[7] Java Cryptography Architecture. (2014, February,20). Oracle. [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

[8] RSA Key Generator. (2014,January,20). Herongyang. [Onlie]. Available: http://www.herongyang.com/Cryptography/RSA-BigInteger-RsaKeyGenerator-java.html

[9] XOR.java. (2014,January, 15). [Online]. Available: http://www.wiu.edu/UsableSecurity/cs395/XOR.java

[10] Properties of Random Numbers. (2014, April,2). Bucknell. [Online]. Available: http://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node37.html

[11] Y. Huang, F. Leu and J. Liu, "A Secure Wireless Communication System Integrating PRNG and Diffie-Hellman PKDS by Using a Data Connection Core," IEEE International conference on Broadband Wireless Computing, Communication and Applications, 2013, pp.360-365.