IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Performance Assessment of RSA, ElGamal and Proposed DHOTP for File Security in Pervasive Computing Environment

Dr. Mohammed Najm Abdullah[1], Atheer Marouf Al-Chalabi[2]

Computer Engineering Dept., University of Technology, Baghdad, Iraq[1]

Research Scholar, Informatics Institute for Postgraduate Studies, UITC, Baghdad, Iraq[2]

Abstract: Pervasive computing is a new technology which will be integrated into all the objects that interact with peoples, to enhancement and make people's routine lives easier through making the process of those objects interactive automatically to anything and everything in every place and at any moment of time in their environment without human intervention as possible. Security is very important element in pervasive computing environment to protect data that is transferred between devices connected with each other via (Wi-Fi or Bluetooth …etc.). This papers proposed a new secure method called HDOTP which is determine on Diffie-Hellman key exchanging to exchange and establish secret key and use this secure key as an initial key in One-Time pad algorithm with new algorithm for steam key generator to generate a random key and compared it with RSA and ElGamal according to parameter performance (runtime, memory usage, avalanche Effect, throughput) and concluded that HDOTP is more efficient than RSA and ElGamal.

Keywords: Pervasive computing environment, cryptography, RSA, ElGamal, Diffie-Hellman Key Exchange, One Time Pad, key stream.

I. INTRODUCTION

The increase in the development of technical devices that entered in all details of people's daily lives (mobile devices, wireless networks, sensors and communications technology) carrying the information and telecommunications revolution to new dimension in the field of telecommunications named pervasive computing. Pervasive computing, the new generation of personal computing that operate in every place and at any moment of time [1].

The term of Pervasive Computing Environment refer to the combination of mobile devices and wireless networking technologies to detect and access services to nearby devices [2].

This environment is susceptible to several challenges, one of these challenges is the security problem, especially the problem of data protection when the data transferred between devices connected with each other via (Wi-Fi or Bluetooth …etc.).

Recently data security has become an important element for many applications that are related to networks, communications and embedded systems to hold out attacks through providing the applications in strong encryption algorithms with a key owned by authorized parties to protect the important data when it is transferred from the sender to the recipient [3, 4].

Fig.1 displays the general flow of usual used encryption algorithms.

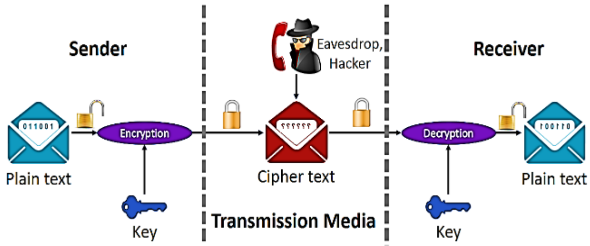

Fig. 1. Encryption/decryption general flow

Literatures have depicted the previous work done in the topics of data encryption which is implemented the security algorithm. Goshwe [5] presented a successfully implementation of data encryption and decryption using RSA algorithm in a network environment. Okeyinka [6] implement the algorithm of RSA and ElGamal using c# language and environment. He concluded that the RSA is better than the ElGamal on the total assessment except in the rate of data decryption. Some other Literatures presented the difference between the encryption algorithms to determine the most efficient algorithm in performance. In [6-8] they implement and compare between RSA ElGamal algorithm using factor analysis, they conclude that RSA is better than ElGamal in time consumption but ElGamal is more secure RSA algorithm in complex cipher-text. In [9-11] they compare between asymmetric, symmetric algorithm. They have concluded that the ratio of encryption of each algorithm is considered high, and that the technique of asymmetric key cipher is more secure than the technique of symmetric key cipher.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

This paper is propose new method by using Diffie-Hellman Key Exchanging algorithm and One-Time pad algorithm with new algorithm to generate key stream and compare the propose algorithm with RSA and ElGamal using performance parameter.

II. CRYPTOGRAPHIC TECHNIQUES

There are two basic techniques for encrypting information: asymmetric encryption (also called public key encryption) and symmetric encryption (also called secret key encryption). Asymmetric encryption uses two different keys, but they linked in some mathematical way together so, it is possible to use one of them as a public key to encrypt the data and the other is used as private key to decrypt the encryption like RSA algorithm. Symmetric encryption uses the same key to encrypt and decrypt data like AES and DES algorithm [12]. It is divided into two types: block cipher and stream cipher. Block cipher encrypt all bits in block at the same time using the same key (i.e. encrypt any bit depends on the rest of the bits in the same block). Stream cipher encrypt every bit in block independently using key stream. Vernam cipher (invented by Gilbert Vernam in 1917) is symmetric stream encryption which is used in encryption the exclusive or function between plaintext and key stream [13]. Key stream is classified into two type: synchronous key (key stream depend on only the key to generate new bit) and asynchronous key (key stream depend on cipher-text too as well as the key) [14]. Key management is an element of public key cryptography, that responsible to share (one or many) public key in order to use it in encryption or decryption process depend on public key encryption technique [15].

III. DIFFIE–HELLMAN KEY EXCHANGE

In 1976 Whitfield Diffie and Martin Hellman have written Diffie Hellman algorithm.

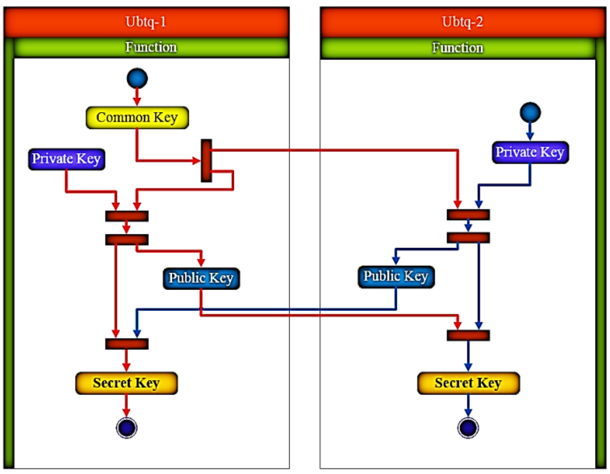

Fig 2. The Diffie-Hellman key exchange UML Activity Diagram between ubiquity_1 and ubiquity_2

It solved the challenges that facing the private key using the management of public keys, so it was taken advantage of public keys to generate a private key for the two devices which are not connected with each other previously as shown in Fig.2, and Algorithm 1 and Algorithm 2 illustrate the exchanging process between two ubiquities [16].

Algorithm 1 Ubiquitous_1 DH

```
function UBTQ_1PROCESS()
    P ← GeneratePrimeNumberRandomly()
    g ← GeneratePrimitiveRootRandomly(P)          ▷ g < P − 2
    a ← GenerateRandomNumber()                    ▷ a < P
    A ← g^a Mod P
    B ← UBTQ_2PROCESS(P, g, A)   ▷ share and exchange public key
    SharedSecretKey ← B^a Mod P
    return (True)
```

Algorithm 2 Ubiquitous_2 DH

```
function UBTQ_2PROCESS(P, g, A)
    b ← GenerateRandomNumber()        ▷ b < P
    B ← g^b Mod P
    SharedSecretKey ← A^b Mod P
    return (B)
```

IV. ONE TIME PAD ENCRYPTION

One-Time pad is a cryptography algorithm proposed by Joseph Mauborgne to improved security of Vernam cipher when uses random and independent (non-repeated) key stream as shown in Fig. 3, and Algorithm 3 and Algorithm 4 illustrate the encryption/decryption process between two ubiquities using the same initial key stream [17].

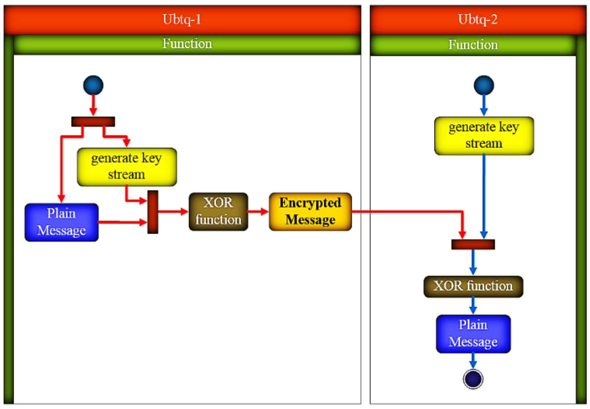

Fig. 3. Encryption/Decryption process in One-time Pad algorithm UML Activity Diagram between ubiquity_1 and ubiquity_2

Algorithm 3 One_Time Pad Encryption Process

```
function ENCRYPTIONXOR(PlainText, KeyStream)
    index ← 0
    repeat
        CipherText[index] ← PlainText[index] ⊕ KeyStream[index]
        index ← index+ 1
    until (KeyStream.Length ≤ index)
    return (CipherText)
```

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Algorithm 4 One_Time Pad Decryption Process

function DECRYPTIONXOR($CipherText, KeyStream$)
  $index \leftarrow 0$
  repeat
    $PlainText[index] \leftarrow CipherText[index] \oplus KeyStream[index]$
    $index \leftarrow index + 1$
  until ($KeyStream.Length \leq index$)
  return ($PlainText$)

V. RSA

The RSA is an asymmetric encryption, it developed by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977. It is used tow type of keys, public key that send to other side to encryption process and private key to decryption which is kept secret as shown in Fig. 4, and Algorithm 5 illustrate the generation key , Algorithm 6 and Algorithm 7 illustrate the encryption/decryption process using RSA algorithm between two ubiquities [18].

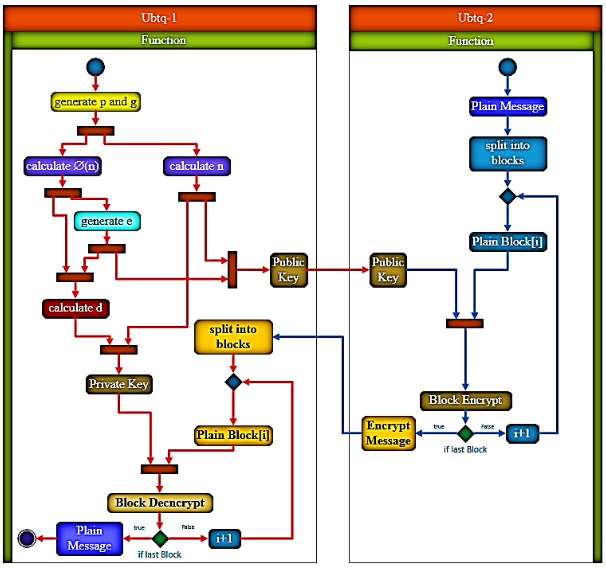

Fig. 4. RSA encryption/decryption UML Activity Diagram between ubiquity_1 and ubiquity_2

Algorithm 5 Generate RSA parameters

function GETPARAMETER()
  ▷ Generate $p \neq q$ prime number randomly
  $p \leftarrow$ GeneratePrimeNumber()
  $q \leftarrow$ GeneratePrimeNumber()
  $n \leftarrow p \times q$
  $\emptyset n \leftarrow (p-1) \times (q-1)$
  ▷ *Generate e where* $1 < e < \emptyset n$ *and* $GCD(\emptyset n, e) = 1$
  $e \leftarrow$GeneratePrimeNumber()
  $d \leftarrow$ ModInverse$(e, \emptyset n)$
  return (n, e, d)

Algorithm 6 RSA Encryption

function ENCRYPTIONRSA($plaintext, e, n$)
  $ciphertext \leftarrow plaintext^e$ mod n
  return ($ciphertext$)

Algorithm 7 RSA Decryption

function DECRYPTIONRSA($ciphertext, d, n$)
  $plaintext \leftarrow ciphertext^d$ mod n
  return ($plaintext$)

VI. ELGAMAL

The ElGamal encryption is public key cryptographic algorithms submitted by Taher Elgamal as an extension of the Diffie-Hellman Key Exchange in 1985. Its security depend on the intractability of the (discrete logarithm and Diffie-Hellman) [14]. Fig.5 describe the ElGamal mechanism for exchanging key and encryption/decryption process, , and Algorithm 8 illustrate the generation key , Algorithm 9 and Algorithm 10 illustrate the encryption/decryption process using RSA algorithm between two ubiquities.

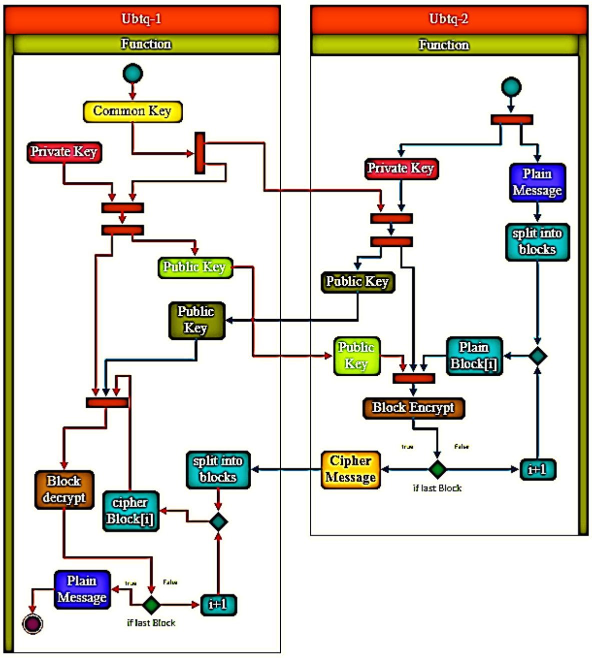

Fig. 5. ElGamal algorithm encryption/decryption UML Activity Diagram between ubiquity_1 and ubiquity_2

Algorithm 8 Ubiquitous_1 Key Generation

function UBTQ_1KEYGENERATION()
  $P \leftarrow$ GeneratePrimeNumberRandomly()
  $g \leftarrow$ GeneratePrimitiveRootRandomly(P) ▷ $g < P - 2$
  $a \leftarrow$ GenerateRandomNumber() ▷ $a < P$
  $A \leftarrow g^a$ Mod P ▷ A is a public key
  return (P, g, A, a)

Algorithm 9 Encription Process

function UBTQ_2ENCRIPTIONPROCESS($P, g, PK1$)
  ▷ $PK1$ is a sender public key
  ▷ PT is a Plain-Text and $PT < P - 1$
  $b \leftarrow$ GenerateRandomNumber() ▷ $b < P$
  $B \leftarrow g^b$ Mod P ▷ B is a public key
  $CipherText \leftarrow (PK1^b \times PT)$ Mod P
  return $(CipherText, B)$

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Algorithm 10 Decryption Process

```
function UBTQ_2ENCRIPTIONPROCESS(P, Pr1, PK2, CT)
    ▷ PK2 is a receiver public key
    ▷ CT is a Cipher-Text
    ▷ Pr1 is a sender private key
    PlainText ← CT / PK2^Pr1 Mod P
    return (PlainText)
```

VII. PROPOSE MODEL

The subject of this paper depends on the construction of the integrated security system in pervasive computing environment by establishing a connection between two parties and exchanging a key between them, and uses the product key (secret key) to encrypt a file that will transmit over an environment that might be not secure using solid algorithm. Diffie-Hellman key exchanging algorithm being used in our contribution to exchange and establish secret key and use this secure key as an initial key in One-Time pad algorithm. And also propose a generation key method to generate a random key depends on the extracted value from the plain-text and the key which is not give any indication on the plain-text and the key as shown in Fig. 6.

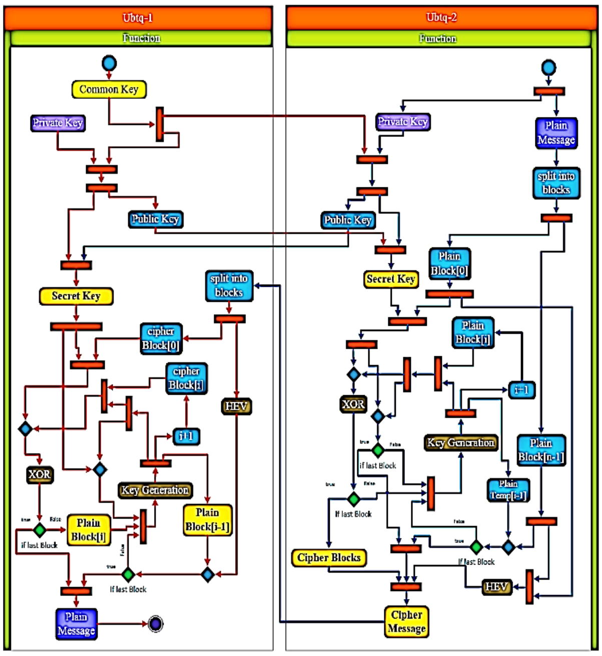

Fig.6 HDOTP encryption/decryption UML Activity Diagram between ubiquity _1 and ubiquity _2

The propose method process called HDOTP (Diffie-Hellman and One-Time Pad). It begins when the Ubiquity_1 and Ubiquity_2 exchanging the key between them and use the size of the key to divided the plain-text into blocks (n).

$$n = \frac{\text{Plaintext size}}{\text{key Size}} \qquad \text{———} \qquad (1)$$

The plain-text block encrypt/decrypt with its specified key using exclusive OR function as it illustrated in Algorithm 11 and Algorithm 12. The new key is generated depend on many concepts:

Shift value (SV): is the value of shifting bit.

$$SV = \text{Random}(15) + 5 \qquad \text{———} \qquad (2)$$

C1b: its mean (count 1's bit).

Extract Value (EV): is the value that is calculated from the key and plain-text as it's shown in equation (5). The key extract value (KEV) that's refer to the amount of 1's in the key and it's calculated as equation (3). The plain-text extra value (PEV) refer to the amount of 1's in each the current and previous plain-text block (not that the second key is depend on the extract value from the $\text{block}_{[0]}$ and $\text{block}_{[n-1]}$, where n refer to the total blocks) as it's shown in equation (4).

$$KEV = C1b_{key}^{2} \bmod 2^{SV} \qquad \text{———} \qquad (3)$$

$$PEV = C1b_{i} \times C1b_{(i-1)} \bmod 2^{SV} \qquad \text{———} \qquad (4)$$

$$EV = PEV \oplus KEV \qquad \text{———} \qquad (5)$$

Temp Cipher Value (TCV): is a value that controlled on the continuation to generate different bit even if the PEV, KEV, and the out bit from the key is still the same in all encryption process, the initial TCV = 1.

$$TCV = (TCV \oplus Key_{lastbit}) \oplus EV \qquad \text{———} \qquad (6)$$

Algorithm 11 HDOTP Encription Process

```
function ENCRIPTIONPROCESS(Key, Blocks)
    ▷ Blocks is a plain-text blocks
    ▷ Key is a secret key generated from Diffie-Hellman Key Exchange
    ▷ C1bi is count a plain-text 1's bit for a block
    ▷ C1bj is count a plain-text 1's bit for a previous block
    ▷ C1bk is count a Key 1's bit for a block
    ▷ HEV is a Hash Extract Value
    ▷ SV is a Shift Value
    n ← GetSize(Blocks)
    C1bi ← GetbitCount(Blocks[0])
    C1bj ← GetbitCount(Blocks[n − 1])
    SV ← ((C1bi + C1bj) MOD 15) + 5
    HEV ← C1bi ⊕ C1bj
    i ← 0
    repeat
        CipherBlock[i] ← Blocks[i] ⊕ Key
        C1bk ← GetbitCount(Key)
        Key ← GenerateNewKey(C1bi, C1bj, C1bk, Key, SV)
        C1bi ← GetbitCount(Blocks[i + 1])
        C1bj ← GetbitCount(Blocks[i])
        i ← i + 1
        if (i == n − 1) then
            SV ← KeyLength / 2
        endif
    until (i > n − 1)
    CipherText ← CipherBlock + HEV
    return (CipherText)
```

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Algorithm 12 HDOTP Decription Process

```
function DecriptionProcess(Key, Blocks, HEV)
    n ← GetSize(Blocks)
    PlainBlock[0] ← Blocks[0] ⊕ Key
    C1bi ← GetbitCount(PlainBlock[0])
    C1bj ← C1bi ⊕ HEV
    SV ← ((C1bi + C1bj) MOD 15) + 5
    i ← 1
    repeat
        C1bk ← GetbitCount(Key)
        Key ← GenerateNewKey(C1bi, C1bj, C1bk, Key, SV)
        PlainBlock[i] ← Blocks[i] ⊕ Key
        C1bj ← C1bi
        C1bi ← GetbitCount(PlainBlock[i])
        i ← i + 1
        if (i == n − 1) then
            SV ← KeyLength / 2
        endif
    until (i > n − 1)
    PlainText ← PlainBlocks
    return (PlainText)
```

Fig. 7 and Algorithm 13 illustrate the key generation process in DHOTP.

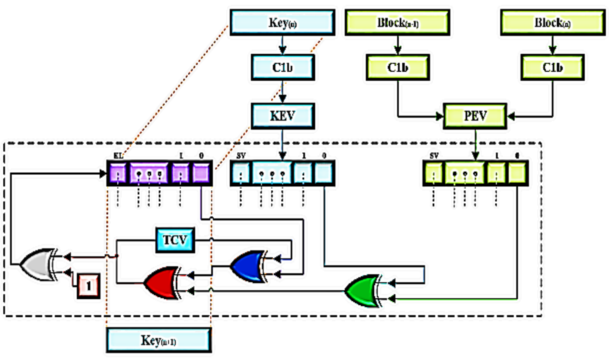

Fig. 7. Logic scheme to explain the new key generation

Algorithm 13 HDOTP Generate Key

```
function GenerateNewKey(C1bi, C1bj, C1bk, Key, SV)
    ▷ PEV is a binary matrix plain-text extract value
    ▷ KEV is a binary matrix key extract value
    ▷ TCV is a Temp Cipher value
    PEV ← (C1bi × C1bj) MOD 2^SV
    KEV ← C1bk^2 MOD 2^SV
    TCV ← 1
    i, j ← 0
    repeat
        EV ← PEV[j] ⊕ KEV[j]
        Result ← TCV ⊕ Key[n − 1]
        Result ← Result ⊕ EV
        TCV ← Result
        Result ← Result ⊕ 1
        Key ← RightShiftKey
        Key ← InsertOnTheRight(Result)
        i ← i + 1
        j ← j + 1
        if (j > SV − 1) then
            j ← 0
        endif
    until (i > SV − 1)
    return (Key)
```

VIII. ANALYSIS AND TESTS

A. Key Bit Random Tests:

The first stage in this work is to generate a sequence keys from the private key. This sequence must be a random. So, to measure the generator quality that is propose to be a random bit generator by detect if the generator may have a kinds of weaknesses. If a set of bit sequence is $S = s_0, s_1, \ldots, s_{n-1}$. The general type of tests are [13]:

▪ Monobit test (Frequency test): This test purpose is to determine if 0's and 1's number in S are nearly equivalent to the same according to the following equation:

$$x = \frac{(n_1 - n_0)^2}{n}$$

Where: n_0 → 0's count.
n_1 → 1's count.
n → total bits count.

▪ Two-bit test (Serial test): This test purpose is to determine if the occurrences number of 00, 01, 10, and 11 are nearly equivalent to the same according to the following equation:

$$x = \left[\frac{4}{n-1} \times \left(\sum_{i=0}^{1}\sum_{j=0}^{1} n_{ij}^2\right)\right] - \left[\frac{2}{n} \times (n_0^2 + n_1^2)\right] + 1$$

▪ Poker test: This test purpose is to determine if the sequences of length m (where m is a positive number), each appear nearly equivalent to the same number of times in S according to the following equation:

$$x = \frac{2^m}{k} \times \left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

B. Algorithm Analysis Performance:

The second stage in this work is to compare the proposal work than RSA and ElGamal algorithm with three key size (512, 1024 and 2048) to determine the efficient algorithm with best key size to use it in pervasive computing by using algorithm performance parameter which is:

▪ Runtime: is the time that the program need it dynamically after the program successfully compile to execute a code statements [19].

▪ Memory usage: is the amount of memory which every value precisely required when you run a program [20].

▪ Avalanche Effect: is the characteristic which is important for encryption algorithm. Its concept is when any bit change in attribute of metadata will change the outcome [21].

$$\text{Avalanche Effect} = \frac{\text{Number of Bits Change}}{\text{Total Bits}}$$

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

▪ Throughput: is a set of items which is processed in a unit per time [22].

$$\text{Throughput} = \frac{\text{Total Bits}}{\text{Total Time}}$$

IX. RESULTS

All the algorithms which mentioned in the above was implement and test using android studio v1.5 with two mobile have the same specification as shown in Table 1.
We test the random key stream with three type of keys (512, 1024 and 2048) using Key Bit Random. The result shown in Table 2 which is the average of 50 time generated different key to encrypt/decrypt a file size 10KB. In each time take the average of all keys that is generated along the blocks of file to encryption/decryption process as shown in Fig. (8, 9, 10, 11, 12, 13, 14, 15 and 16).

TABLE 1 MOBILE SPECIFICATION USE IN TEST AND ANALYSIS THE ALGORITHMS

Model	Samsung I9300I Galaxy S3 Neo
CPU	Quad-core 1.4 GHz Cortex-A7
OS	Android v4.3
Total RAM	1.5 GB
Internal Storage	16 GB

TABLE 2 KEY BIT RANDOM TESTS WITH DIFFERENT THREE KEYS

key size	test type	average statistic value	Threshold
512	Frequency	0.2971	3.8415
	serial	1.0852	5.9915
	poker	1.9137	7.8147
1024	Frequency	0.364	3.8415
	serial	1.0795	5.9915
	poker	1.936	7.8147
2048	Frequency	0.622	3.8415
	serial	1.3831	5.9915
	poker	1.856	7.8147

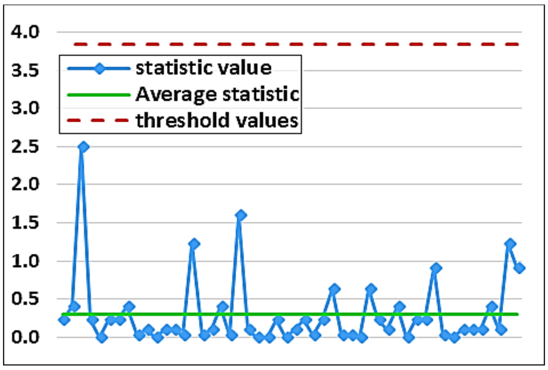

Fig. 8. Frequncy test for 50 keys (512bit) generated in HDOTP

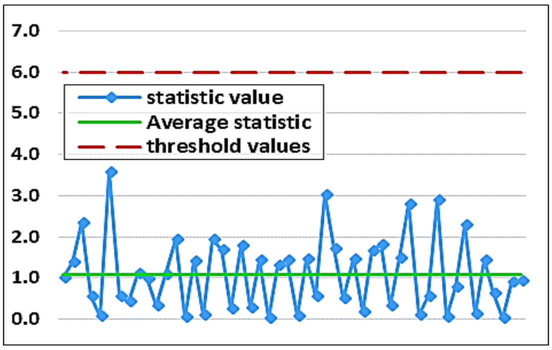

Fig. 9. Serial test for 50 keys (512bit) generated in HDOTP

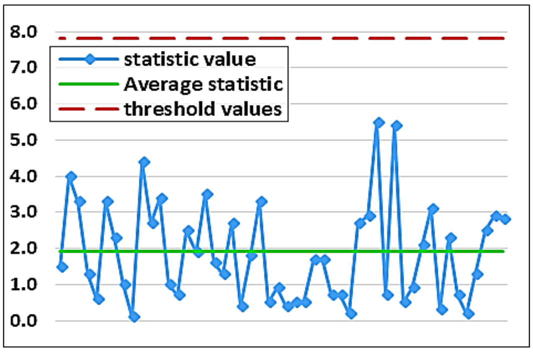

Fig. 10. Poker test for 50 keys (512bit) generated in HDOTP

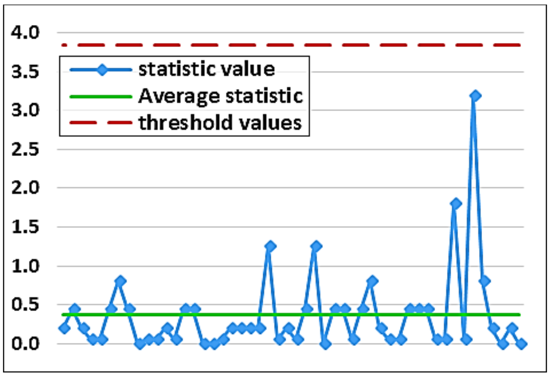

Fig. 11. Frequncy test for 50 keys (1024bit) generated in HDOTP

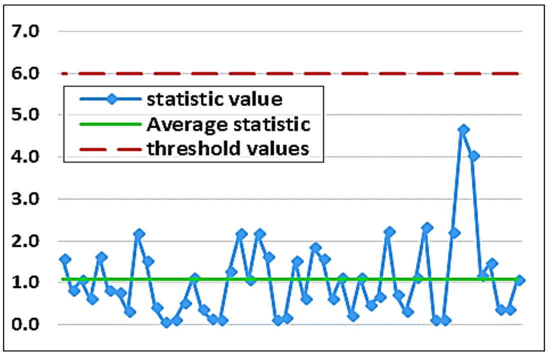

Fig. 12. Serial test for 50 keys (1024bit) generated in HDOTP

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

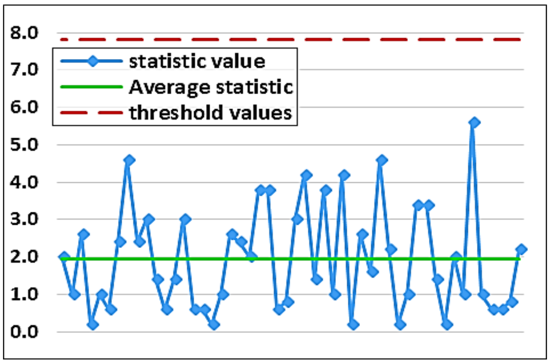

Fig. 13. Poker test for 50 keys (1024bit) generated in HDOTP

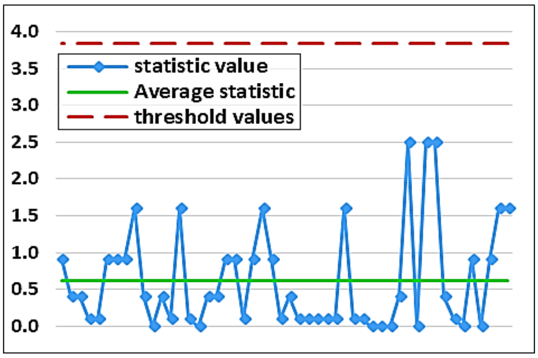

Fig. 14. Frequncy test for 50 keys (2048bit) generated in HDOTP

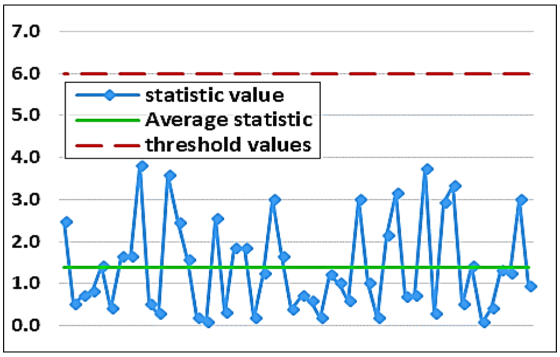

Fig. 15. Serial test for 50 keys (2048bit) generated in HDOTP

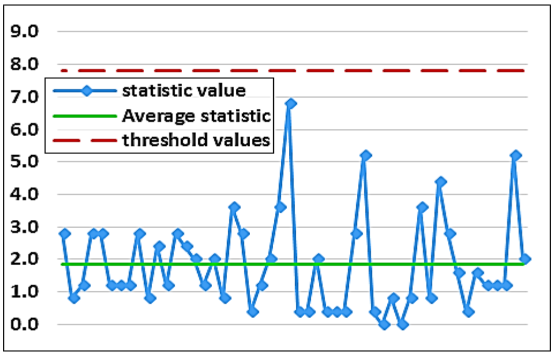

Fig. 16. Poker test for 50 keys (2048bit) generated in HDOTP

The comparison between DHOTP, RSA and AlGamal with three key sizes (512, 1024 and 2048) bit and three different sizes of file (100, 500 and 1000) KB to encryption and decryption using: runtime factor is shown in Table 3 and Fig. 17, memory usage factor shown in Table 4 and Fig. 18, avalanche effect factor is shown in Table (5, 6 and 7) and Fig. (19, 20 and 21), and throughput factor is shown in Table 8 and Fig. 22.

TABLE 3 ALGORITHM RUNTIME (MILLISECOND)

		File size	Key length (bit)		
			512	1024	2048
RSA	Encryption	100	728	791	1315
		500	3124	3906	6631
		1000	6117	7899	13325
	Decryption	100	4038	11722	42641
		500	18855	58158	246752
		1000	37428	117152	427669
DHOTP	Encryption	100	2004	1071	647
		500	9594	5398	3168
		1000	19176	10773	6302
	Decryption	100	2000	1063	626
		500	9554	5297	3067
		1000	18961	10689	6145
ElGamal	Encryption	100	1416	1889	2816
		500	6606	8928	13125
		1000	14290	19613	30343
	Decryption	100	2206	3104	4893
		500	10736	14675	24339
		1000	21940	29554	48827

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

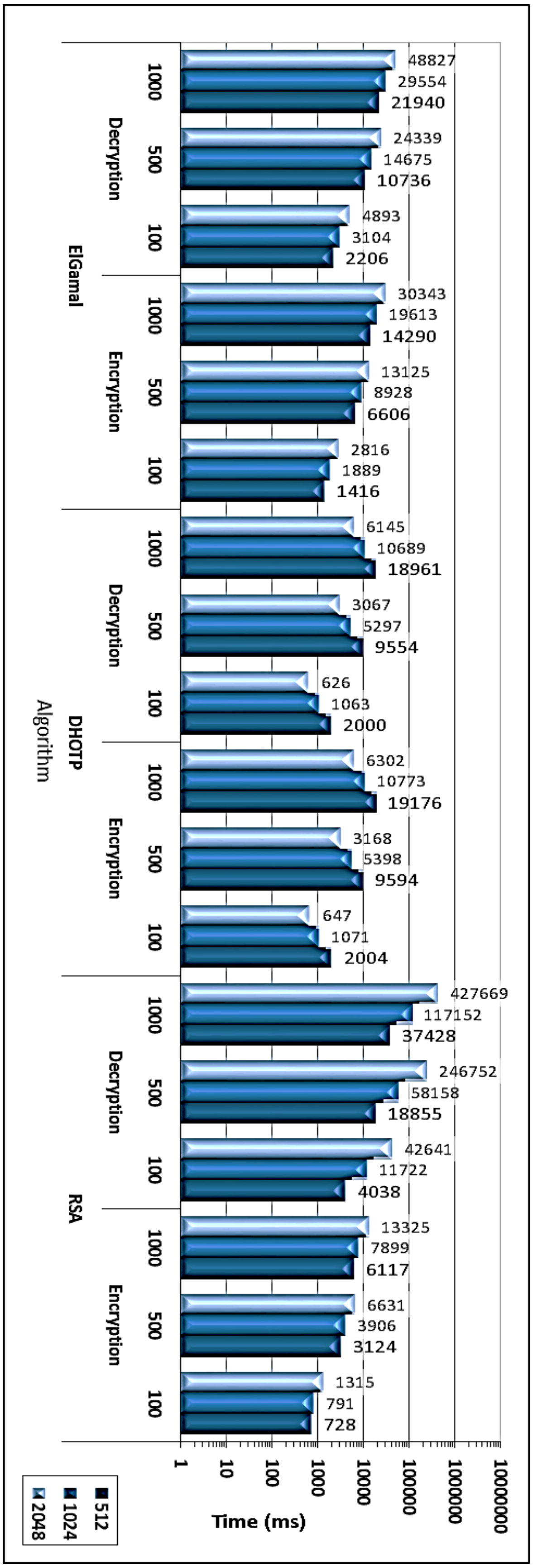

Fig. 17. Algorithm runtime

TABLE 4 ALGORITHM MEMORY USAGE (MB)

		File size	Key length (bit)		
			512	1024	2048
RSA	Encryption	100	1.8	0.8	0.4
		500	9.1	4.1	1.9
		1000	18.2	8.25	3.86
	Decryption	100	1.8	0.8	0.4
		500	8.7	3.8	1.8
		1000	17.5	7.56	3.51
DHOTP	Encryption	100	17.3	10.1	6.15
		500	86.9	50	30.9
		1000	173.7	99.88	61.67
	Decryption	100	17.3	10.1	6.08
		500	86.7	49.9	30.9
		1000	173.4	99.71	61.58
ElGamal	Encryption	100	5.1	4.5	2.9
		500	37	21.8	12
		1000	73.4	32.5	18.1
	Decryption	100	18.5	18.7	20.3
		500	89.36	89.58	104.7
		1000	200.4	205.1	221.2

DOI 10.17148/IJARCCE.2016.51201

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

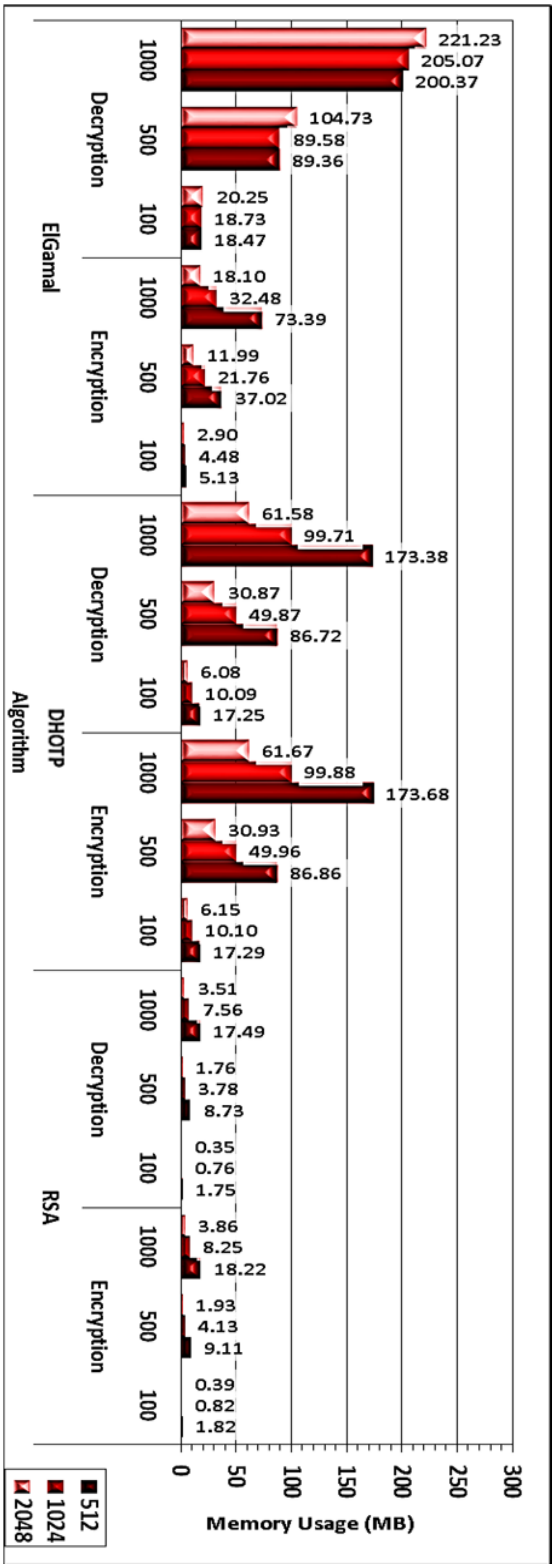

Fig. 18. Algorithm memory usage

TABLE 5 Algorithm avalanche effect key 1 bit change (%)

Algorithm	File Size	Key length (bit) 512	1024	2048
RSA	100	49.95	49.97	50.00
	500	49.96	49.99	50.00
	1000	49.98	50.00	50.00
DHOTP	100	49.98	49.99	50.00
	500	50.00	50.03	50.03
	1000	50.02	50.03	50.04
ElGamal	100	49.96	49.98	49.97
	500	49.97	49.99	49.99
	1000	50.00	50.00	50.00

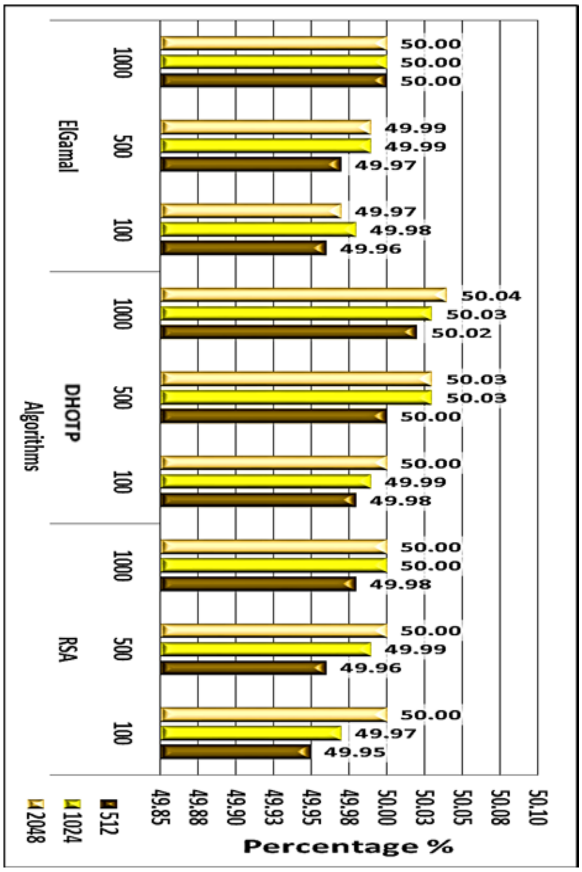

Fig. 19. Algorithm avalanche effect for key 1bit change

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

TABLE 6 Algorithm avalanche effect key and text 1 bit change (%)

Algorithm	File Size	key Length (bit) 512	1024	2048
RSA	100	49.95	49.97	49.97
	500	49.96	49.99	50.00
	1000	49.98	50.00	50.00
DHXOR	100	50.01	50.02	50.02
	500	50.02	50.03	50.03
	1000	50.05	50.05	50.05
ElGamal	100	49.96	49.98	49.98
	500	49.98	49.99	49.99
	1000	50.00	50.00	50.00

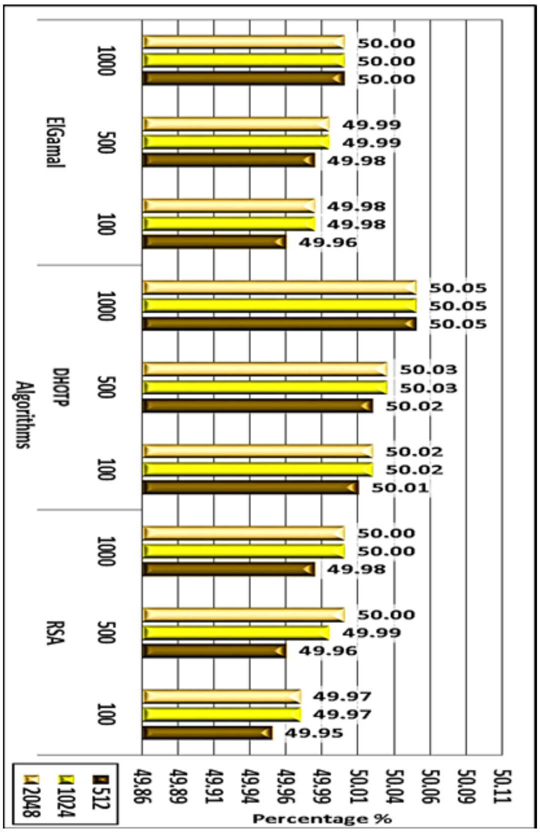

Fig. 20. Algorithm avalanche effect for key and text 1bit change

TABLE 7 Algorithm avalanche effect text 1 bit change (%)

Algorithm	Change	File Size	key Length (bit) 512	1024	2048
RSA	Any place	100	0.025	0.057	0.119
		500	0.005	0.011	0.024
		1000	0.003	0.006	0.012
DHXOR	1st block	100	49.941	49.972	49.981
		500	49.986	49.989	49.991
		1000	49.990	49.989	49.986
	Mid-block	100	24.980	25.000	25.108
		500	24.979	25.001	25.117
		1000	25.016	25.108	25.123
	Prior last block	100	0.032	0.061	0.126
		500	0.006	0.013	0.025
		1000	0.003	0.006	0.012
ElGamal	Any place	100	0.026	0.059	0.120
		500	0.005	0.011	0.024
		1000	0.003	0.006	0.012

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

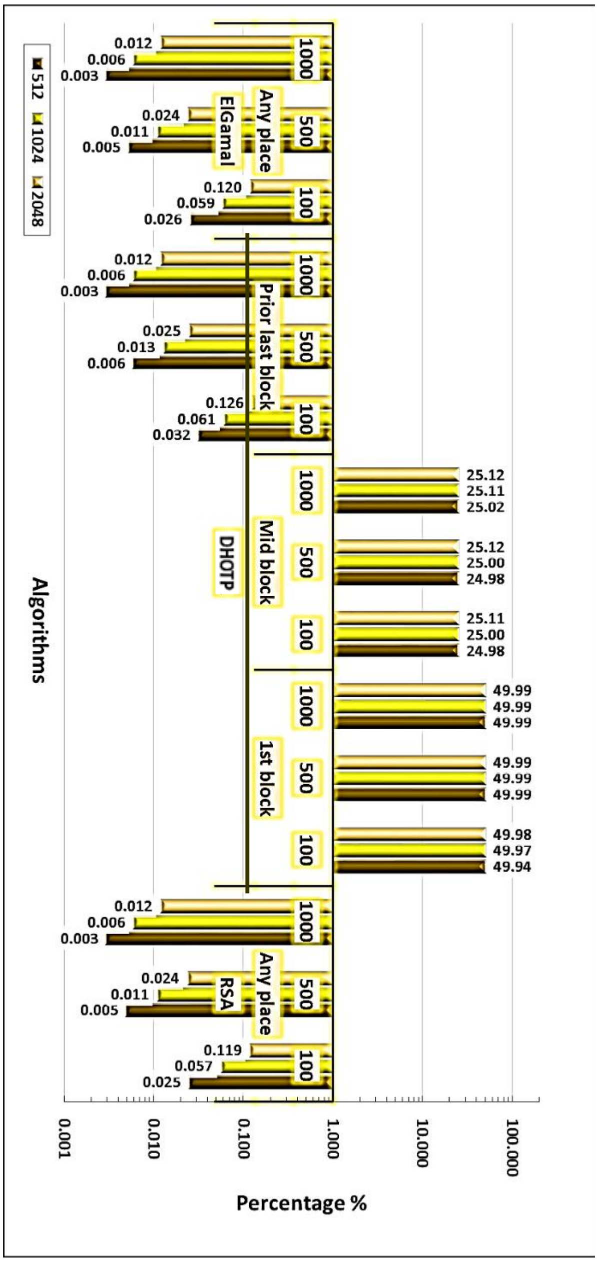

Fig. 21. Algorithm avalanche effect for text 1bit change

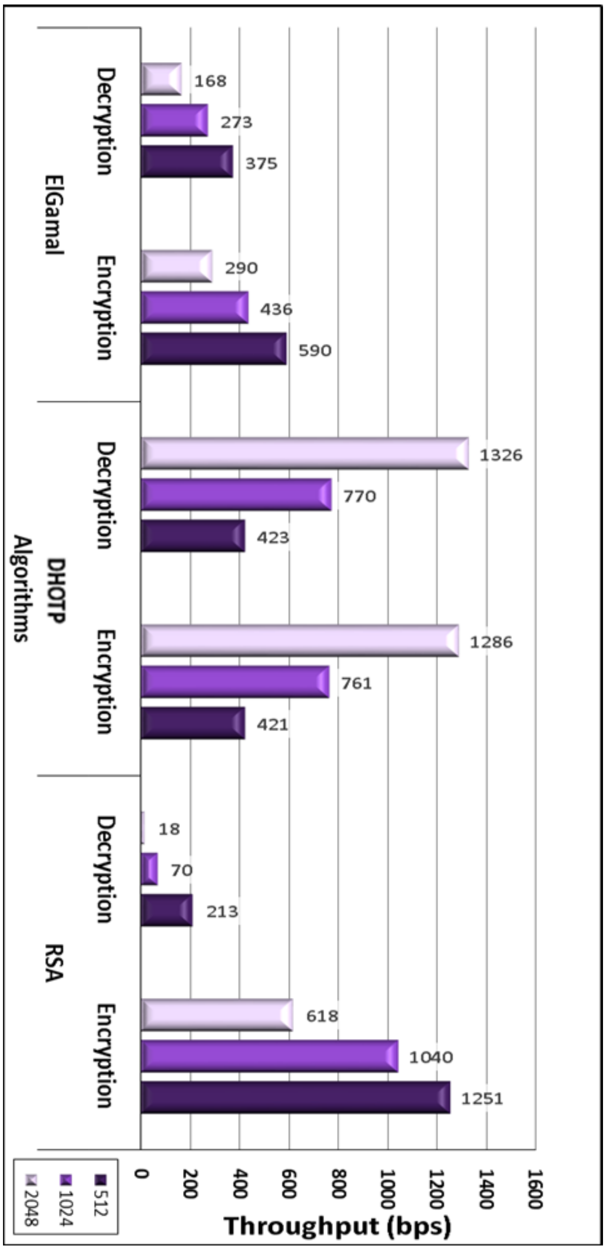

Fig. 22. Algorithm throughput

TABLE 8 Algorithm throughput (bit per second)

		key Length (bit)		
		512	1024	2048
RSA	Encryption	1251	1040	618
	Decryption	213	70	18
DHXOR	Encryption	421	761	1286
	Decryption	423	770	1326
ElGamal	Encryption	590	436	290
	Decryption	375	273	168

X. CONCLUSION

In this paper, we propose new method which is called HDOTP, and the conclusion is divided into three considerations. The first one is about the random of key generation, and all results accepted in the three tests: frequncey test, serial test and poker test. The second consideration is about the key size of HDPOTP, and all the comparision performance test is shown that 2048 bit key is better than the other key size (i.e. 1024 bit and 512 bit). The last consideration is about the algorithm analysis performance among HDOTP, RSA and Elgamal. The runtime and the throughput factors show that HDOTP using 2048 bit key size is better than RAS (with the same key size around 68% in encryption and 99% in decryption)

DOI 10.17148/IJARCCE.2016.51201

and ElGamal (with the same key size is around 82% in encryption and 89% in decryption). The avalanche effect factor shows that HDOTP is better than RSA and ElGamal algorihms in all situations: change just one bit from a key, change one bit from a key and plain text, and change just one bit from plain text. The memory usage factor shows that HDOTP need more memory than the other algorithms. From all the concluded points here above show that HDOTP is the efficient method when its use in pervasive computing environments, especially that all devices in pervasive environment must have a huge memory size.

REFERENCES

[1] J. Burkhardt, T. Schaeck, H. Henn, S. Hepper, And K. Rindtorff, Pervasive Computing: Technology And Architecture Of Mobile Internet Applications: Addison-Wesley Longman Publishing Co., Inc., 2001.

[2] I. Mezgár And S. Grabner-Kräuter, "Privacy, Trust, And Business Ethics For Mobile Business Social Networks," Handbook Of Research On Business Ethics And Corporate Responsibilities, P. 390, 2015.

[3] K. Sireesha and S. M. Rao, "A Novel Approach Of Area Optimized And Pipelined FPGA Implementation Of AES Encryption And Decryption," Int. J. Sci. Res. Publ, Vol. 3, Pp. 1-5, 2013.

[4] R. Hosseinkhani And S. H. H. S. Javadi, "Using Image As Cipher Key In AES," International Journal Of Computer Science Issues (IJCSI), Vol. 9, 2012.

[5] N. Y. Goshwe, "Data Encryption And Decryption Using RSA Algorithm In A Network Environment," International Journal Of Computer Science And Network Security (IJCSNS), Vol. 13, P. 9, 2013.

[6] A. Okeyinka, "Computational Speeds Analysis Of RSA And Elgamal Algorithms On Text Data," In Proceedings Of The World Congress On Engineering And Computer Science, 2015.

[7] A. Shetty and K. Shravya Shetty, "A Review On Asymmetric Cryptography–RSA And Elgamal Algorithm," International Journal Of Innovative Research In Computer And Communication Engineering, 2014.

[8] A. Sharma, J. Attri, A. Devi, And P. Sharma, "Implementation & Analysis Of RSA And Elgamal Algorithm," Presented At The National Conference On 'Advances In Basic & Applied Sciences', 2014.

[9] G. Singh, "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) For Information Security," International Journal of Computer Applications, Vol. 67, 2013.

[10] R. Tripathi and S. Agrawal, "Comparative Study Of Symmetric And Asymmetric Cryptography Techniques," International Journal Of Advance Foundation And Research In Computer (IJAFRC), Vol. 1, Pp. 68-76, 2014.

[11] A. Roy, "Brief Comparison of RSA and Diffie-Hellman (Public Key) Algorithm," ACCENTS Transactions On Information Security, Vol. Vol 1(1), 2016.

[12] J. A. Carlson, "METHOD FOR SECURE COMMUNICATION USING ASYMMETRIC & SYMMETRIC ENCRYPTION OVER INSECURE COMMUNICATIONS," Ed: US Patent 20, 150, 326, 547, 2015.

[13] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook Of Applied Cryptography: CRC Press, 1996.

[14] C. Paar And J. Pelzl, Understanding Cryptography: A Textbook For Students And Practitioners: Springer Berlin Heidelberg, 2009.

[15] G. Al-Aali, B. Boneau, And K. Landers, "Diffie-Hellman Key Exchange," D. O. C. S. A. E. U. O. N. Dame, Ed., Ed. Notre Dame, Indiana 46556-0309, 2000, Pp. 67-74.

[16] K. Palmgren, "Diffie-Hellman Key Exchange: A Non mathematicians Explanation," ISSA J, 2006.

[17] P. ŠTIKA, "Unconditional Security in Classical Cryptography," Masarykova Univerzita, Fakulta Informatiky, 2010.

[18] R. Laboratories, RSA Laboratories' Frequently Asked Questions about Today's Cryptography, Version 4.1: RSA Security Inc., 2000.

[19] I. Alsamadi, Advanced Automated Software Testing: Frameworks For Refined Practice: Frameworks For Refined Practice. USA: Information Science Reference, 2012.

[20] R. Sesgewick and K. Wayne, Algorithms, 4th Ed. USA: Pearson Education, 2011.

[21] D. B. Endicott-Popovsky, Proceedings of the International Conference on Cloud Security Management: ICCSM-2013. USA: Academic Conferences and Publishing International Limited, 2013.

[22] P. Killelea, Web Performance Tuning: Speeding Up the Web, 2nd Ed. USA: O'Reilly Media, Incorporated, 2002.