International Journal of Information & Network Security (IJINS)
Vol.3, No.2, April 2014, pp. 98~115
ISSN: 2089-3299

Privacy Amplification in Quantum Cryptography BB84 using Combined *Univarsal*$_2$-Truly Random Hashing

Ahmed Mahmoud Abbas, Amr Goneid, Sherif El-Kassas
Department of Computer Science and Engineering, The American University in Cairo, Egypt

Article Info

Article history:

Received Jan 28th, 2014
Revised Feb 26th, 2014
Accepted Mar 15th, 2014

Keyword:

BB84
Quantum Cryptography
Privacy Amplification
Hash Functions

ABSTRACT

The Quantum Cryptography BB84 protocol is considered to be the most widely used in Quantum Key Distribution (QKD) to create secure keys that are shared between communicating parties. Major phases in this protocol are the correction of discrepancies between keys of sender and receiver as well as the privacy amplification phase where two communicating parties distill highly secret shared key from a larger body of a shared key, which is only partially secret. In the present paper, we propose enhancing the performance of the privacy amplification phase by introducing message digests (MD) hash functions combined with truly random functions. The resulting Hash-Mod and Hash-Div functions are used to compress reconciled keys resulting from the error correction phase. Experiments were conducted using three different simulators to assess the performance of these combined functions using entropy and information measures. The newly introduced functions were tested against the plain versions of Mod and Div functions as well as the permutated Mod and Div functions using hash function signature size (MD4-128). The results supported the use of Hash-Mod and Hash-Div over the plain versions as well as their permuted versions in enhancing the entropy behavior and minimizing information content within reconciled key strings.

Ahmed Mahmoud Abbas
Department of Computer Science and Engineering,
The American University in Cairo, Egypt
aabbas@aucegypt.edu

1. INTRODUCTION

It is now evident that when quantum computers become operational, public key cryptography would no longer be secure. Quantum Cryptography (QC) offered a solution to this problem and became an interesting issue in cryptography during the past few years. Being a combination of ideas in quantum physics and information theory, QC is based on the Heisenberg Uncertainty Principle which states that "measurements of unknown quantum states can not be accomplished without disturbance" [1]. This means eavesdropping on the quantum communication channel causes unavoidable disturbance that can be recognized by the communicating channel legitimate users. QC is not used to encrypt a message itself, yet it is rather used to generate and distribute random secret keys between sender and receiver. The key created is used with symmetric cryptosystems to encrypt the messages between sender and receiver. (QC) is a combined technology of quantum mechanical phenomena and classic cryptographic schemes with the target of generating a secret key, or extending a short key, between two communicating parties [2].

The scheme for quantum key distribution is composed of the following three steps [1]:

1. *Key Creation* between two communicating parties known as Alice and Bob using one of several techniques such as polarization methods or phase methods.
2. *Error Correction* for the errors and discrepancies between keys of Alice and Bob.

3. *Privacy Amplification* which is concerned with reduction of eavesdropper's information that might be gained during the initial creation of the key.

A realization of such scheme is the BB84 which is considered to be the most widely used in Quantum Key Distribution (QKD) or Quantum Key Exchange systems. In the presented paper, we proposed to enhance the Privacy Amplification phase by introducing message digest (MD) hash functions combined with truly random functions to compress reconciled keys resulting from the BB84 error correction phase. The use of such combined functions is expected to enhance the entropy behavior and minimize the information content in the reconciled key strings.

The paper is organized as follows. The BB84 protocol is discussed in section (2), where key creation, error correction and privacy amplification algorithms are explained. Section (3) discusses in details the basic privacy amplification model. The proposed combined function strategy and the privacy amplification functions are presented in section (4). Section (5) explains the three simulators of privacy amplification. Section (6) shows the proposed three experimental comparisons of simulators analysis. Finally, section (7) concludes the paper outcomes.

2. The BB84 Protocol

The (BB84) is details a procedure for Quantum Key Distribution (QKD). This protocol creates a strongly secured key which is shared between two communicating parties over a quantum channel. In 1984, Charles Bennett and Gilles Brassard had proposed the (BB84) protocol using four polarization states that built a (QKD) system. The (BB84) is composed of: key creation, error correction or reconciliation and privacy amplification phases. [3]

2.1 Key Creation

The scheme shown in **Figure (1)** describes the key creation steps between Alice and Bob until both agree on a common shared binary string in order to proceed with second phase of BB84 protocol. The Key creation steps of (BB84) protocol are as follows [3]:

1. (Alice) sends a random sequence of photons polarized horizontal (↔), vertical (↕), right-circular (↻) and left-circular (↺).
2. (Bob) measures photons' polarization in random sequence of bases, rectilinear (+) and circular (○).
3. Results of (Bob's) measurements as some photons may not be received at all.
4. (Bob) tells (Alice) which basis he used for each photon he received.
5. (Alice) tells him which bases were correct.
6. (Alice) and (Bob) keep only the data from these correctly-measured photons, discarding all the rest.
7. This data is interpreted as a binary sequence according to the coding scheme:

[↔ = ↺ = 0] and **[↕ = ↻ = 1].**

The previous steps results are referenced as the quantum transmission or the raw quantum transmission to ensure that it was recently gained in the process as shown in **Figure (1)** of QKD protocol [3].

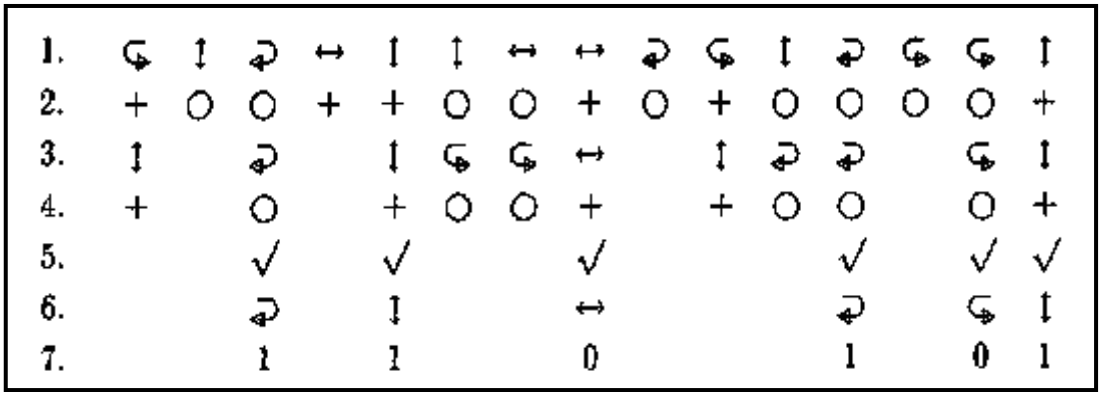

Figure (1): Basic Quantum Key Distribution (QKD) Protocol [3]

2.2 Error Correction

As long as the quantum transmission is done with very dim light pulses utilized in place of single photons, (Alice) and (Bob's) first job is to switch public messages allowing them to reconcile the differences between (Alice) and (Bob's) data. It is assumed that (Eve) spy on all transmitted public messages between (Bob) and (Alice), such exchange should be carried out in a manner that shows as little possible information

on this data, while (Eve) is incapable of damaging these public messages contents. An efficient manner for (Alice) and (Bob) to execute reconciliation is first to consent an arbitrary permutation of the bit positions in their strings to randomize the error locations then partition the permuted strings into blocks of size (***k***) where single blocks are only supposed to have no more than one error. The optimal block size is assumed to be a function of expected error rate [3]. Concerning that block, (Alice) and (Bob) compare the blocks' parity. Matching parity blocks are cautiously approved correct, yet those of harsh parity are subject to a bisective search, disclosing [log (k)] further parities of sub-blocks till error is found and corrected. If initial block size was far too large or too small as of "a bad a priori guess of the error rate" [3], the procedure could be tried again using an appropriate block size [3].

In order to secure information leakage to (Eve) during the reconciliation process, (Alice) and (Bob) consent to reject the last bit of each block or sub-block which parity they just have disclosed. In spite of using proper block size, some errors staying undetected that happened in blocks or sub-blocks with an even number of errors [3]. For eliminating additional errors, the arbitrary permutation and block parity disclosure are repeated many times while using greater block sizes till (Alice) and (Bob) guess that some errors stay in the data as whole. Here, the block parity disclosure approach turns to be less because it makes (Alice) and (Bob) give up at least one bit in every block for the sake of privacy [3].

2.3 Privacy Amplification

Privacy Amplification is a technique of distilling highly secret shared information that could be used as a cryptographic key from a larger body of shared information which is only partially secret. Consequently, (Alice) and (Bob) are able to apply Privacy Amplification using the following assumptions. Assume (**x**) denotes the reconciled string and (**n**) denotes its length. Assume a deterministic bit of information about (**x**) evaluated as **e(x)** of a randomly function [$e:\{0,1\}^n \rightarrow \{0,1\}$] [3].

For example, physical and parity bits are deterministic bits, yet bits of information in the sense of Shannon's information theory are not. Charles Bennett and Gilles Brassard showed that if (Eve's) information about (**x**) is only (ℓ) deterministic bits, a randomly hash function (**h**) and publicly chosen from a proper class of functions [$\{0,1\}^n \rightarrow \{0,1\}^{n-\ell-s}$] will map (x) to h(x) about which (Eve's) expected knowledge is less than [$2^{-s}/\ln 2$] bit, where (s>0) is an arbitrary security parameter [3]. Later, this paper will show how to modify the use of hash functions to improve the output of privacy amplification phase.

2.3.1 How to Control Eavesdropping Using Privacy Amplification

Realistically, (Eve) is not capable from generating errors while spying on communication to gain as maximum information as she can in a noisy channel where (Alice) and (Bob), the legitimate parties, could not differentiate either the induced errors were because of noise or (Eve). Hence, legitimate users need to the guarantee key distribution security within their network. (Alice) and (Bob) achieve such goal through focusing on "how to reduce the information leakage to (Eve) instead of distinguish noise and (Eve)" [1].

(Alice) and (Bob) can select one of these approaches given an estimation of the channel Error Rate. First, (Alice) and (Bob) can measure the gained information by (Eve) in the raw key using Shannon information and quantum information theory. (Alice) and (Bob) can depend on using the mutual information to measure (Eve's) expected information about transferred messages on the quantum channel as expressed in Equation (1) as $I(x;y)$ where (**e**) is the Error Rate of the communicating channel [1].

$I(x;y)$ **OR** $I_e = 1+[e\log_2 e+(1-e)\log_2(1-e)]$ **Equation (1)** [1]

Secondly, (Alice) and (Bob) can delete the gained information by (Eve) from the raw key by calculating the probability that (Eve) can guess the correct key given her knowledge about it. It is achieved when (Eve) tries to measure a photon's state correctly, then she will not be detected by (Alice) and (Bob), yet if (Eve) chose a wrong measurement, she will produce disturbance to the state of the photon as stated by *Heisenberg Uncertainty Principle*. The solution is meant to apply the Privacy Amplification framework of (BB84) protocol in order to lessen the leaked information to adversaries while the legitimate channel participants are communicating [1].

3. The Basic Privacy Amplification Model in the BB84 Protocol

(Alice) and (Bob) need to agree on a shared secret random bit string to be used as secret key for securing their message transmissions over an imperfect private channel, known as main channel, and an unauthenticated public channel, known as wire-tap channel [4].

The private channel is an example of Binary symmetric Channel (BSC) that is imperfect because it allows an eavesdropper to reasonably spy on some of transmitted bits of exchanged keys without much

disturbance to the channel and protecting (Eve) from being detected. Such problem could be solved through applying Privacy Amplification. [5] .Concerning the public channel, it transfers information correctly since it is supported with error correcting code schemes, yet transmissions are exposed to (Eve) to read and not change. Such problem could be solved through applying authentication schemes [5].

Hash functions are functions that map from larger domains to smaller ones. It is depicted as "assigning an abbreviation to a name" [6]. A $Universal_2$ hash class is a collection of hash functions instead of one function. Each time an application is run using $Universal_2$ hash functions, a hash function is randomly chosen from the hash collection [6]. In case the set of functions is cautiously selected to what is named as a $Universal_2$ class, thus many applications of hashing are estimated to have good performance for any distribution of inputs and not just the uniform distribution [6].

3.1 Basics of Information Theory

The Binary Symmetric Channel (BSC) is a famous model for depicting communication scenario between two parties sending and transmitting data over a binary channel. It is required to find the Mutual Information $I(x;y)$ of the channel where messages are delivered via binary digits of (0's) and (1's) [7]. The Mutual Information $I(x;y)$ could be expressed as: $I_{\max}(x;y) = 1-\Omega(P)$ (bits/binits.) **Equation (2)** [7]

The Maximum Information transmitted in bits per binary digits (binits)
$I(x;y) = 1-[P\log_2 1/P + \overline{P}\log_2 1/\overline{P}]$ (bits/binits.) **Equation (3)** [7]
The Error Probability is defined as [$\Omega(P) = P\log 1/P + \overline{P}\log 1/\overline{P}$] where [$\overline{P} = 1-P$] **Equation (4)** [7]

The maximum $\Omega(P)$ value of (**1**) was found at (**P=1/2**), yet the minimum $\Omega(P)$ value of (**0**) was found at (**P=0**) and (**P=1**) [7]. The information transmitted over a channel per symbol, $I(x;y)$, is a function of channel matrix having probability of $P(y_j \mid x_i)$, and the input symbol probabilities $P(x_i)$ only. For one set of input-symbol probabilities, $I(x;y)$ reaches a maximum value [7]. This represents the maximum amount of transmitted information over the channel per symbol. Hence, the maximum value of $I(x;y)$ gained for a specific set of input symbols probabilities is known as Channel Capacity per Symbol transmitted [7].

- Channel Capacity per symbol transmitted expressed as follows:
  $C_s = \max_{P(x_i)} I(x;y)$ (units/symbol) **Equation (5)** [7]
- Binary Symmetric Channel Capacity expressed as follows:
  $C_s = 1-\Omega(P)$ (bits/binit) **Equation (6)** [7]
  where: $\Omega(P) = P\log 1/P + \overline{P}\log 1/\overline{P}$; replacing ($\log$) with ($\log_2$) as of binary bits.
- Binary Symmetric Channel Capacity expressed as follows:
  $C_s = 1-[P\log_2 1/P + \overline{P}\log_2 1/\overline{P}]$ (bits/binit) **Equation (7)** [7]

The Channel Capacity $C_s = 1-\Omega(P)$ minimum value of (**0**) was found at (**P=1/2**), yet the maximum $C_s = 1-\Omega(P)$ value of (**1**) was found at (**P=0**) and (**P=1**) [7].

The relation Channel Capacity [$C_s = \max_{P(x_i)} I(x;y)$] (units/symbol) gives maximum possible rate of transmitted information when one symbol is transmitted. In case there are (**K**)-symbols being transmitted per second, then Maximum Rate of Information Transmission per second expressed [$C = (K.C_s)$] **Equation (8)**, where (**K**) is number of transmitted symbols over channel between sender and receiver [7].

3.2 Using Truly Random Function to Minimize Eavesdropper's Activity

The Error Detection function [$f : \{0,1\}^N \rightarrow \{0,1\}^K$] was selected randomly among all functions from [$\{0,1\}^N \rightarrow \{0,1\}^K$] requires $(K2^N)$ bits to transmit function (**f**) over the public channel. (Eve's) information about string (**x**) is described by the set (**C**) of possible candidates that is known to (Alice) and (Bob). Let the string (**x**) of length (**N**) transmitted between (Alice) and (Bob) where (**K<N**) be the safety parameter for Error Detection and the value of (**K**) is bounded by $(0 \le K \le N)$ [5].

- [$\pi : \{0,1\}^N \rightarrow \{0,1\}^N$]: Randomly selected permutation of string(**x**) [5]

- [$f:\{0,1\}^N \rightarrow \{0,1\}^K$]: Defined as [$f(x)=\pi(x)Mod(2^K)$] [5]
- [Mod]: Length of (**K**)-bit string consisting of rightmost (**K**)-bits of string (**x**) [5]
- [$g:\{0,1\}^N \rightarrow \{0,1\}^{N-K}$]: Defined as [$g(x)=\pi(x)Div(2^K)$] [5]
- [Div]: Length of (**N-K**)-bit string obtained from string (**x**) by deleting its rightmost (**K**)-bits and proceeding with rest of string (**x**) [5]

Both functions (**f**) and (**g**) are equitable functions. Thus, knowledge of π [as **f( )** and **g( )**] and **f(x)** gives no information on **g(x)**; except that of $(R \le N-K)$ that is the desired length of the final compressed string. Assume (**S**) to be any non-negative integer as $(S < N-K)$ and $(R = N-K-S)$. Let [$g:\{0,1\}^N \rightarrow \{0,1\}^R$] be any fixed equitable function, then the expected amount of Shannon Information given on **g(x)** by **f(x)**: $I(F;G)=[2^{-S}/\ln 2]$ [5].

3.3 Privacy Amplification by Public Discussion in BB84 Protocol

The Privacy Amplification has an Information Measure framework listed in below steps [8]:

- The reconciled key (**x**) is a random (**N**)-bit string of uniform distribution over binary alphabet $\{0,1\}^N$.
- The [**V =e (x)**] is defined as a random eavesdropper function [$e:\{0,1\}^N \rightarrow \{0,1\}^t$]; where $(t < N)$.
- The Error Probability is calculated via [$\Omega(e)=e\log 1/e+\bar{e}\log 1/\bar{e}$] where [$\bar{e}=1-e$] to calculate (**t**) in terms of Error Rate (**e**) [8].
- The Channel Capacity is defined as [$I_{\max}(x,y)=1-\Omega(e)$] and defined as [**V (e)**] [8].
- The (**V**) is a random (**t**)-bit string of leaked bits by (Eve) on (**x**) [8].
- The (**S**) is a positive safety parameter defined as $(S < N-t)$ and $(N \ge S > 0)$ that is used to minimize number of leaked bits [8].
- The [**K =g (x)**] is a random arbitrary key selected by (Alice) or (Bob) to be their secret key from a $Universal_2$ class of hash functions as [$g:\{0,1\}^N \rightarrow \{0,1\}^R$] [8].
- The secret key (**K**) string is of length (**R**) which is generated by (Alice) and (Bob) where $(R \le N)$ and (**R**) is computed as $(R=N-t-S)$ [8].
- The secret key (**K**) is (**R**)-bit string generated by (Alice) or (Bob) where $(0 \le R \le N)$ [8].
- (Eve's) expected information on secret key (**K**) given (**G**), (**V**) to be expressed as [$I(K;GV) \le \frac{2^{-S}}{\ln 2}$]
- Information Measure $I(K;GV)$ is defined on average over the values of (**V**) [8].
- (Alice's) and (Bob's) tactics does not count on **e ( )** since the Privacy Amplification model operates if (Alice) and (Bob) have nothing about **e ( )** given that they have an upper bound on the value of (**t**) leaked bits on (**W**) [8].

Finally, (Alice) and (Bob) are able to have a secret key which is utilized into hybrid encryption techniques "for use as a cryptographic key from a larger body of shared information that is only partially secret" [8] during their message exchange communications. Therefore, using Privacy Amplification by public discussion enables the legitimate users to "distill a secret key about which (Eve) has arbitrary little information" [8].

4. The Proposed Functional Model for Privacy Amplification

Our proposed model discusses the input matrix model, entropy model formulation, matrix permutation model, matrix message digest hashing model and deployed functions. The model introduces a new solution to enhance the Privacy Amplification of (BB84) protocol through combining the set of hashing functions with the truly random functions of Mod and Div to have Hash-Mod and Hash-Div functions. Both new functions are used to hash, compress or shrink the reconciled key generated from (BB84) second phase. A set of experiments were conducted over truly random functions in combination with permutation and message digest hashing in order to evaluate the performance measure of Privacy Amplification using a proposed Entropy metric and the introduced Information Measure [9].

4.1 [2D]-Input Matrix Model

It is a random key 2D matrix generation of desired keys number as (rows) by message digest hash signature key size as (columns). It is used as a loading key set to start running every function in order to calculate the Entropy value metric for every used (R-Value) scanned per every key size into the 2D matrix. The matrix generation algorithm is known for the following [9]:

- Sizing the key buffer to the desired message digest hash signature size and saving every sized key buffer into matrix column while looping over the rest of desired keys in the matrix rows until the 2D-matrix is built [9].
- Every key buffer is stored as binary array into matrix for later Entropy value metric calculations and having a 2D binary array of (**1's**) and (**0's**) [9].

4.2 Entropy Model Formulation

The "entropy of (**m**) is a measure of uncertainity, and then the probability distribution which generates the maximum uncertainty will have maximum entropy". The Entropy equation could express as:

$H(m) = \sum_{i=1}^{n} P_i \log_2 1/P_i = -\sum_{i=1}^{n} P_i \log_2 P_i (bits)$ **Equation (9)** [7]

The Entropy measurement metric mechanism depends on having a 2D-input matrix made of keys of different (**R**)-values where $(0 \le R \le N)$. The (**R**)-values could be of (**0**)-bit length, where its entropy is zero, and ranges to (**N**)-bit length, where its entropy equals or grater than zero. All experiments generated relative entropy for every (**R**)-value. The input matrix has rows of (**M**)-key of range belongs to $[1 \le i \le M]$ and has columns of (**N**)-bit key length of range belongs to $[1 \le j \le N]$. Every element in the matrix is represented by $[b_{ij}]$ bit value of {0, 1} binary domain. The applied experiments aim to calculate the probability of having (**1's**) in the matrix columns using **Equation (10)** and further illustrated into **Figure (2)**.

[$p_j = \sum_{i=1}^{M} b_{ij} \Big/ M$]; where $[p_j]$ the probability that bit (**j**) is {**1**}. **Equation (10)** [9]

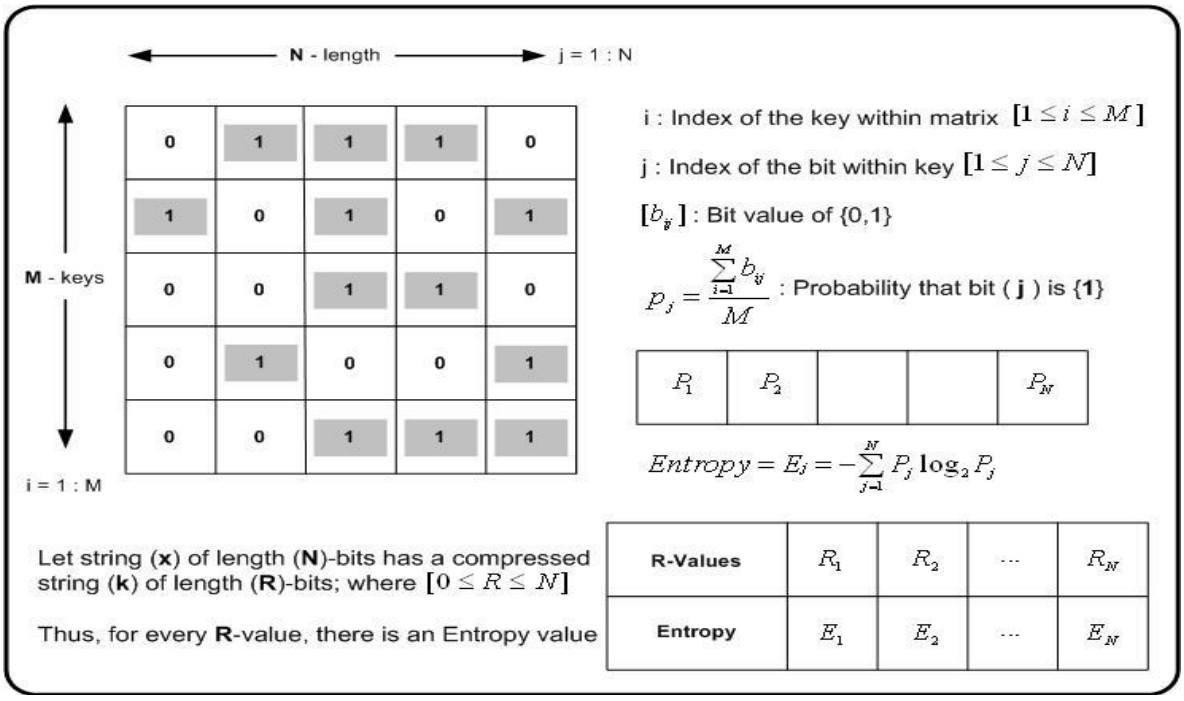

Figure (2): Practical Demonstration of Proposed Entropy Model [9]

4.3 Matrix Permutation Model

It is a permutated 2D-matrix applied to 2D-input matrix of desired keys number as (rows) by message digest hash signature key size as (columns). It maintains permutation of key bits' locations, made of (**1's**) and (**0's**), randomly and scattered; while keeping same number of (**1's**) and (**0's**) per key into the 2D-matrix. Such scheme is iterated over every key into 2D-matrix until all keys are successfully scattered [9].

4.4 Matrix Message Digest Hashing Model

The hashing model is based on the message digest hashing algorithms supported by the FlexiProvider package. FlexiProvider has been developed by the Theoretical Computer Science Research Group of Professor Johannes Buchmann at the department of Computer Science at Technische Universität Darmstadt in Germany [10]. The hashing model applies (Big) Endian or (Little) Endian over fourteen defined message digest hashing algorithms according to experiment configurations that shapes how the 2D-Input

Matrix keys are hashed according to the Endian type configuration [10]. The hashing functions maintain scattering of input keys made of (**1's**) and (**0's**) randomly through the use of the Endian storing mechanism; however they do not keep the same number of (**1's**) and (**0's**) per key into the 2D-matrix. Such scheme is iterated over every key into the 2D-matrix until all keys are successfully scattered and compressed. The defined fourteen message digest hashing algorithms implemented by FlexiProvider package and their corresponding signature size in bits are shown in **Table (1)** [10]:

Table (1): Flexiprovider 14 Message Digest Hashing Algorithms with Sizes [10]

[MD4] signature size = 128 bits	[MD5] signature size = 128 bits
[RIPEMD128] signature size = 128 bits	[RIPEMD160] signature size = 160 bits
[RIPEMD256] signature size = 256 bits	[RIPEMD320] signature size = 320 bits
[SHA1] signature size = 160 bits	[SHA224] signature size = 224 bits
[SHA256] signature size = 256 bits	[SHA384] signature size = 384 bits
[SHA512] signature size = 512 bits	[Tiger] signature size = 192 bits
[DHA256] signature size = 256 bits	[FORK256] signature size = 256 bits

4.5 Deployed Functions

The presented work defines six functions where the main two are Mod and Div functions. Another two functions are a combination of permutation function with Mod and Div. The last two functions are a combination of message digest hash functions with Mod and Div. All six functions take the reconciled keys generated from (BB84) second phase as 2D-input matrix to generate shorter compressed keys as 2D-output matrix. The 2D- input matrix is made of some number of randomly generated binary keys, known as (Key Number), while every key has a fixed length equals to the selected message digest hash function signature size, known as (Key Size) [9]. The performance measure metric is using our customized Entropy model as well as adopting the Information Measure. Through our model, we use a string (**x**) which is a reconciled random binary key of length (**N**)-bit resulted from a uniform distribution $\{0,1\}^N$. A function [$K = g(x)$] is a random hash function selected from a $Universal_2$ hash functions that outputs a key string of (**R**)-bit string which length belongs to $(0 \le R \le N)$. The proposed work defines eight functions [9]:

1. Permutation function (π) applied to binary string (**x**) of length (**N**) and outputs a permutated string of same length (**N**) which preserves the number of (**1's**) and (**0's**), while randomly shuffle their locations.
   Permutation Function: $\pi : \{0,1\}^N \rightarrow \{0,1\}^N$ **Equation (11)** [5]
2. Message digest hash function (ϕ) applied to binary string (**x**) of length (**N**) and outputs a hashed string of selected hash function signature length which changes the number of (**1's**) and (**0's**) and randomly shuffle their locations via using big or little endian types.
   Hash Function: $\phi : \{0,1\}^N \rightarrow \{0,1\}^{Signature}$ **Equation (12)** [5]
3. Mod function ($g_{Mod}(x)$) applied to binary string (**x**) of length (**N**) and outputs a compressed string of length (**R**)-bit which is made of rightmost (**R**)-bits of string (**x**).
   Mod: $g_{Mod} : \{0,1\}^N \rightarrow \{0,1\}^R$ **OR** $g_{Mod}(x) = (x) Mod (2^R)$ **Equation (13)** [5]
4. Div function ($g_{Div}(x)$) applied to binary string (**x**) of length (**N**) and outputs a compressed string of length (**N-R**)-bit which is made of string (**x**) by deleting its rightmost (**R**)-bits and proceeding with rest of bits of string (**x**).
   Div: $g_{Div} : \{0,1\}^N \rightarrow \{0,1\}^{N-R}$ **OR** $g_{Div}(x) = (x) Div (2^{N-R})$ **Equation (14)** [5]
5. Perm-Mod function ($g_{Perm-Mod}(x)$) applied to binary string (**x**) of length (**N**) where permutation function is applied first to string (**x**) and then apply the Mod function.
   Perm-Mod: $g_{Perm-Mod} : \{0,1\}^N \rightarrow \{0,1\}^R$ **OR** $g_{Perm-Mod}(x) = \pi(x) Mod (2^R)$ **Equation (15)** [9]
6. Perm-Div function ($g_{Perm-Div}(x)$) applied to binary string (**x**) of length (**N**) where permutation function is applied first to string (**x**) and then apply the Div function.
   Perm-Div: $g_{Perm-Div} : \{0,1\}^N \rightarrow \{0,1\}^{N-R}$ **OR** $g_{Perm-Div}(x) = \pi(x) Div (2^{N-R})$ **Equation (16)** [9]
7. Hash-Mod function ($g_{Hash-Mod}(x)$) applied to binary string (**x**) of length (**N**) where digest hash function is applied first to string (**x**) and then apply the Mod function.
   Hash-Mod: $g_{Hash-Mod} : \{0,1\}^N \rightarrow \{0,1\}^R$ **OR** $g_{Hash-Mod}(x) = \phi(x) Mod (2^R)$ **Equation (17)** [9]

8. Hash-Div function ($g_{Hash-Div}(x)$) applied to binary string (**x**) of length (**N**) where digest hash function is applied first to string (**x**) and then apply the Div function.
   Hash-Div: $g_{Hash-Div} : \{0,1\}^N \rightarrow \{0,1\}^{N-R}$ **OR** $g_{Hash-Div}(x) = \phi(x) Div(2^{N-R})$ **Equation (18)** [9]

5. Privacy Amplification Simulators

The presented work developed three simulators that demonstrate the relationship between the (**R**)-bit secret key string denoted as (**K**) and the Entropy (**E**). This relationship shows how far using permutation function and message digest hashing function had influenced the performance of Mod and Div functions in the generation of a compressed secret key from the reconciled random (**N**)-bit string key denoted as (**W**). The hash function **g ( )** denoted as [**K = g (W)**] is a random hash function selected by (Alice) and (Bob) from a $Universal_2$ class of hash functions in order to generate their secret key. Meanwhile, (Eve) can spy over their transmitted key bits using a random eavesdropper function **e ( )** denoted as [**V =e (W)**] where (**V**) is a random (**t**)-bit string of leaked bits resulted from listening on string key (**W**) [8]. The Privacy Amplification defined the (**R**-Value) in terms of (**N**), (**t**) and a positive safety parameter denoted as (**S**), where (**R**-Value) is computed as (**R=N-t-S**). Also, the model defined (Eve's) expected information on secret key (**K**) given (**G**) and (**V**) and known as Information Measure defined into **Equation (19)** [9].

Information Measure on (**K**) given (**G**), (**V**) is [$I(K;GV) \leq \frac{2^{-S}}{\ln 2}$] **Equation (19)** [8]

The simulations were designed to run on three forms of (R-Value) in order to show the development stages of (R-Value) calculation as secret key string length where it is formulated: (**R=N**), (**R=N-t**) and (**R=N-t-S**) [9].

5.1 (R=N) Simulator Model

This simulator uses a simple structure of (R-Value) to conduct a study on its multiple values by applying the six types of Mod and Div combined with permutation and message digest hash functions. The simulator applies the range of (**N**) for the (R-Value) and formulated as (**R=N**) through setting some parameters within configuration file of our simulator developed by Java programming language [9].

5.2 (R=N-t) Simulator Model

This simulator uses another structure of (R-Value) to conduct a study on its multiple values by applying the six types of Mod and Div combined with permutation and message digest hash functions. The secret key (**K**) of length (**R=N-t**) is computed through setting some parameters within configuration file of our simulator developed by Java programming language [9] as follows:

- Using the Binary Symmetric Channel Capacity defined before in **Equation (7)**.
- The (C_s) is refined using [$V(e)$] and defined in terms of (**e**) as (Error Rate) defined **Equation (20)**.
- $V(e) = [1 + (e \log_2 e) + ((1-e)\log_2(1-e))]$ **Equation (20)** [1]
- Using Channel Capacity in Information Units (per second) was defined before in **Equation (8)** [7]
- The transmitted bits (**K**) in (C) is refined to be (**N**) that is the secret key size in bits. The (C_s) is refined as [$V(e)$] which is defined in terms of (**e**) or (Error Rate). The (C) is refined to be (**t**) that is leaked bits by (Eve) and defined into **Equation (21)**.
  tValue = (**t**) = [N * V (e)] **Equation (21)** [9]
- The (R-Value) is computed in terms of (**N**) which is the secret key (**K**) size in bits and (**t**) the leaked bits by (Eve). Hence, the (R-Value) is defined as (**R=N-t**) [9].

5.3 (R=N-t-S) Simulator Model

This simulator uses a third structure of (R-Value) in order to conduct a study on its multiple values by applying the six types of Mod and Div combined with permutation and message digest hash functions. The secret key (**K**) of length (**R=N-t-S**) is computed through setting some parameters within configuration file of our simulator developed by Java programming language [9] as follows:

- Using the Binary Symmetric Channel Capacity defined before in **Equation (7)**
- Using the [$V(e)$] defined before in **Equation (20)** [1]
- Using the Channel Capacity in Information Units (per second) was defined before in **Equation (8)**
- Using the [tValue] defined before in **Equation (21)** [9]
- The positive Security Parameter (**S**) is introduced which lies in range of [$0 < S \leq N - t$]. The Security Parameter (**S**) value is calculated as [**S=N–t**] in terms of (**N**) secret key size in bits and (**t**) leaked bits by (Eve). **Equation (22)** [9]

- The (R-Value) is computed in terms of (**N**) which is the secret key size in bits, (**t**) the leaked bits by (Eve) and the positive Security Parameter (**S**). Hence, (**R**-Value) is defined as (**R=N-t-S**). [9]
- The Information Measure introduced by Bennett and Brassard which is known as (Eve's) expected information on secret key (**K**) given (**G**) and (**V**) is calculated in terms of (**S**) and defined as

$$[\, I(K;GV) \le \frac{2^{-S}}{\ln 2} \,]$$ **Equation (19)** [8]

The experiments were conducted based on a set of parameters shown in **Table (2)** that shaped the results output and were used to sketch the experimental graphs later [9].

Table (2): Set of Parameters for Privacy Amplification Phase [9]

Parameter	Description	R=N	R=N-t	R=N-t-S
KeysNum	The number of reconciled keys loaded into random 2D-Input matrix	✓	✓	✓
KeySize	The key length of every reconciled key known as (**N**)-bit string key where its value equals selected message digest hash signature size	✓	✓	✓
HashFunction	The message digest hash function selection out of (14) hash functions developed by FlexiProvider. Our experimental runs used (MD4-128 bits).	✓	✓	✓
EndianType	The Endian type that is either (Big) or (Little) as defined by Flexiprovider	✓	✓	✓
LoopsNum	The number of loops within simulator that sets how many Entropy iteration needed to calculate the average Entropy via dividing by (LoopsNum)	✓	✓	✓
ErrorRateLimit	Maximum value for (Error Rate) vector size generation that defines possible error value used in calculations that ranges from (0) multiplied by (0.01) to the maximum determined value multiplied by (0.01)		✓	
ChannelErrorRate	The maximum value for the Channel Error Rate that is a fixed value out of [0.0, 0.25, 0.50, 0.75, 1.0] in order to allow variation of Security Parameter vs. (R-Value). Our experimental runs used (0.50) value			✓

6. Experimental Comparison of Proposed Models

A systematic approach was applied to conduct experimentations and extract the results for both the Mod and the Div implementations of key. The target is assessing the performance of both implementations with respect to their Entropy measure under different configurations that influence the outcome of experiments for the (R=N), (R=N-t) and (R=N-t-S) simulators [9].

6.1 Experimental Scenarios for (R=N)

The experiment was configured to use (KeysNum=100), (KeySize = 128) of same (MD4) signature size, (LoopsNum = 10), (HashFunction = MD4) and (EndianType = Little) for both implementations of Mod and Div functions. The three Mod functions recorded maximum Entropy value for each at (R-Value) of (128) and the average Entropy values were close for the three algorithms for (R-Value) from (1) to (108). However, continuing (R-Value) from (109) to (128), the Entropy values of Perm-Mod algorithm became higher than those of the Mod function. The Entropy values of Hash-Mod algorithm are also higher than those of Mod function and Perm-Mod function in that region. Consequently, the knee-shaped of Mod function at high (R-Value) from (109) to (128) was corrected by Perm-Mod function and further rectified by Hash-Mod function. Both Mod and Perm-Mod Entropy increasing curves resulted from implementing the standard (BB84) Privacy Amplification which demonstrated the permutation function would produce more Entropy improvement because of the permutation scattering effect. The knee-shaped curve, behaved as a constant function in its region and demonstrated by Mod function, was later cured by the effect of using Perm-Mod function at high (R-Value) as a straight line-shape as shown in **Figure (3)** [9].

The Hash-Mod function proved a better performance over the Perm-Mod and Mod functions. The knee-shaped curve appearing at high (R-Value) ranging from (109) to (128) had diminished by using Perm-Mod function. Similarly, Hash-Mod function managed to produce enhanced results because of the hashing compression and scattering effects. Such correction of knee-shaped of Mod function in both of Perm-Mod and Hash-Mod functions had preserved the whole range of (R-Value) to be fully utilized instead of removing

such defected knee-shaped area [9].

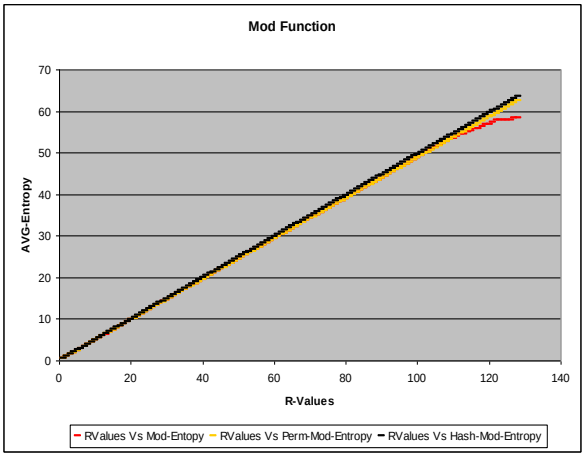

Figure (3): Scenario (R=N) for Mod-Function using (MD4-128) of Little Endian [9]

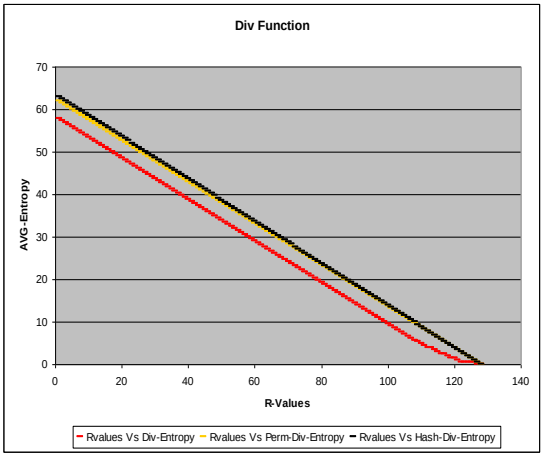

Figure (4): Scenario (R=N) for Div-Function using (MD4-128) of Little Endian [9]

The three Div functions recorded zero Entropy value for every one at (R-Value) of (128) and the average Entropy values of Hash-Div function are slightly higher than those of Perm-Div function over the entire range of (R-Values). On the other hand, the Entropy readings of Div function are far below those of Perm-Div and Hash-Div functions. Consequently, the knee-shaped of the Div function at high (R-Values) from (109) to (128) was corrected by the Perm-Div function and further rectified by the Hash-Div function. Both Div and Perm- Div results were obtained by implementing the standard (BB84) Privacy Amplification that demonstrated the permutation function would show much entropy improvement because of the permutation scattering effect. In addition, the knee-shaped curve, behaved as a constant function in its region and demonstrated by the Div function, which was obtained for the Div function was later cured by the effect of using Perm-Div function at (R-Value) in range of (109) to (128) as a straight line-shape as shown in **Figure (4)**. The knee-shaped curve appearing at high (R-Value) ranging from (109) to (128) had diminished by using Perm-Div function. Similarly, Hash-Div function managed to produce enhanced results because of the hashing compression and scattering effects. Such correction of knee-shaped of Div function in both of Perm-Div and Hash-Div functions had preserved the whole range of (R-Value) to be fully utilized instead of removing such defected knee-shaped area [9].

6.2 Experimental Scenarios for (R=N-t)

The experiment was configured to use (KeysNum=100), (KeySize = 128) of same (MD4) signature size, (LoopsNum = 10), (HashFunction = MD4), (EndianType = Little) and (ErrorRateLimit = 100) for both implementations of Mod and Div functions. The three Mod functions recorded zero Entropy value for each (R-Value) of (0) for (Error Rate) of (0) and maximum Entropy value for each (R-Value) of (128) for (Error Rate) of (0.5) and zero Entropy value for each at (R-Value) of (0) for (Error Rate) of (1). The average Entropy values were close for the three algorithms for (R-Value) from (1) to (108). However, continuing (R-Value) from (109) to (128), the Entropy values of Perm-Mod algorithm become higher than those of the Mod function. The Entropy values of Hash-Mod algorithm were also higher than those of Mod function and Perm-Mod function in that region [9].

Consequently, the knee-shaped of the Mod function at high (R-Value) from (109) to (128) was corrected by the Perm-Mod function and further rectified by the Hash-Mod function. The (R-Value) was computed via (R-Computed) since it is a function in (Error Rate). Therefore, the (R-Computed) values were not sequentially generated as they resulted from an equation as shown in **Figure (5)**. Both Mod and Perm-Mod Entropy curves resulted from implementing the standard (BB84) Privacy Amplification and demonstrates that the permutation function would produce more Entropy improvement because of the permutation scattering effect [9]. In addition, the knee-shaped curve, behaved as a constant function in its region, which was demonstrated by the Mod function, was later cured by the effect of using Perm-Mod function at high (R-Computed) as a straight line-shape. There is a mirror axis appeared around the Error Rate of (**0.50**) value where (R-Computed) of (128) and the three functions could be graphed using any data either above or below the mirrored axis. The Hash-Mod function showed more increasing curve behavior over the Perm-Mod and Mod functions. Moreover, the knee-shaped appeared at high (R-Computed) ranging from (109) to (128) had diminished later by the effect of using Perm-Mod function. The Hash-Mod function

managed to show such progress because of the hashing compression and scattering effects [9].

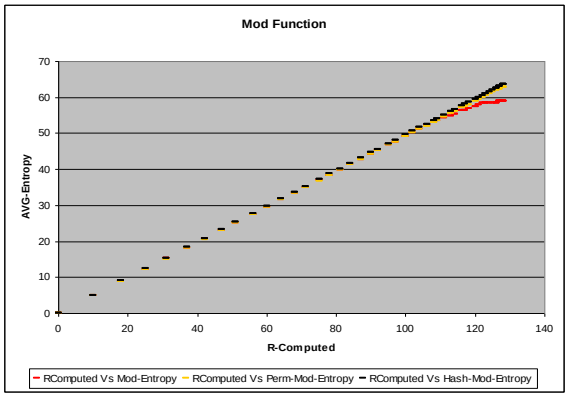

Figure (5): Scenario (R=N-t) for Mod-Function using (MD4-128) of Little Endian [9]

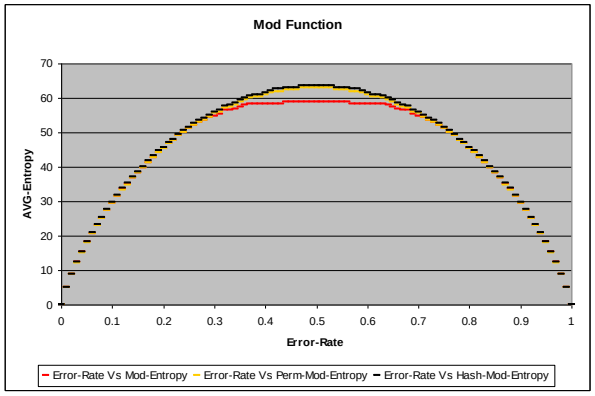

Figure (6): Scenario (R=N-t) of Error Rates for Mod-Function using (MD4-128) of Little Endian [9]

Similar to graphing the (R-Computed) across the average Entropy of the three Mod functions, the (Error Rate) is also graphed across the average Entropy to compare behavior of functions as shown in **Figure (6)**. The new graphs of the three Div functions showed a similar incrementing behavior with knee-shaped to prove that the Div function is maintaining an increasing graphical behavior. The behavior of the three Div algorithms is shown graphically in **Figure (7)**. The Perm-Div and Hash-Div Entropy functions showed a decreasing behavior, while the Div Entropy function showed more decreasing behavior than the other. In addition, the knee-shaped curve, behaved as a constant function in its region, which was obtained for the Div function was later cured by the effect of using Perm-Div function at (R-Computed) in range of (109) to (128) as a straight line-shape. The Perm-Div Entropy curve showed a notable development over the Div Entropy curve to prove the importance of permutations. As of the mirror axis that appeared around the Error Rate of (0.50) value and around the (R-Computed) of (128) generating (AVG-Entropy) of (0) for the three types of Div function. Thus, the three functions could be graphed using any data either above or below the mirrored axis. Later, the current thesis introduced new function named Hash-Div function that apply the message digest hashing functions by hashing the key to a fixed hash signature then apply the Div function. The Hash-Div function showed more increasing curve behavior over the Perm-Div and Div functions. Moreover, the knee-shaped appeared at high (R-Computed) ranging from (109) to (128) had diminished later by the effect of using Perm-Div function. The Hash-Div function introduced in the present work showed such progress because of the hashing compression and scattering effects. Thus, hashing had improved Entropy values, which proved to be more promising during experimentations [9].

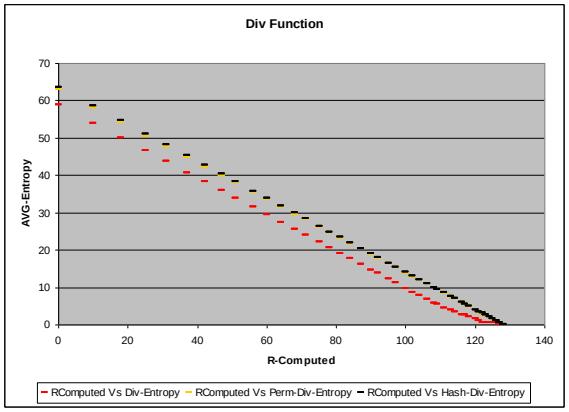

Figure (7): Scenario (R=N-t) for Div-Function using (MD4-128) of Little Endian [9]

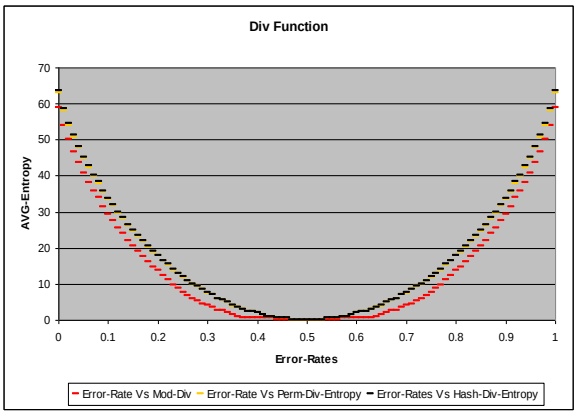

Figure (8): Scenario (R=N-t) of Error Rates for Div-Function using (MD4-128) of Little Endian [9]

Such correction of knee-shaped of Div function in both of Perm-Div and Hash-Div functions had preserved the whole range of (R-Computed) to be fully utilized instead of removing such defected knee-shaped area in Div function. Moreover, the permutation and hashing had improvements over the Div curve because they generated elevated curves for both Perm-Div and Hash-Div than the Div function curve [9].

Similar to graphing the (R-Computed) across the average Entropy of the three Div functions, the (Error Rate) is also graphed across the average Entropy to compare behavior of functions as shown in **Figure (8)**. The new graphs of the Perm-Div and Hash-Div functions showed a similar decrementing behavior while the Div Entropy function showed more decreasing behavior than the others. Hence, such decrementing behavior proves that the Div function is maintaining a decreasing graphical behavior. The Channel Capacity model verification using V(e) could be achieved to get the Binary Symmetric Channel Capacity as defined into **Equation (7)** in section (3) and defined in terms of (P) as the (Error Probability).

The (C_s) is refined to be V(e) which is defined in terms of (e) as the (Error Rate) and declared the following substitutions [9]:

- Error Probability of value (P = 0) generated (C_s) of value (1)
- Error Probability of value (P = 1/2) generated (C_s) of value (0)
- Error Probability of value (P = 1) generated (C_s) of value (1)

The (Error Rate) was graphed against V(e) as shown in **Figure (9)** and defined as $[V(e) = 1 + [e\log_2 e + (1-e)\log_2(1-e)]]$ which applied the following readings [9]:

- Error Rate of value (e = 0) generated V(e) of value (1)
- Error Rate of value (e = 1/2) generated V(e) of value (0)
- Error Rate of value (e = 1) generated V(e) of value (1)

Throughout our work, we managed to prove the applied equations of the Privacy Amplification model are in compliance with the Information Theory model concerning the Binary Symmetric Channel Capacity (C_s). All results generated by the three Mod and Div functions are successfully verified because graph equations in **Figure (9)** used (C_s) within their equation formulation [9].

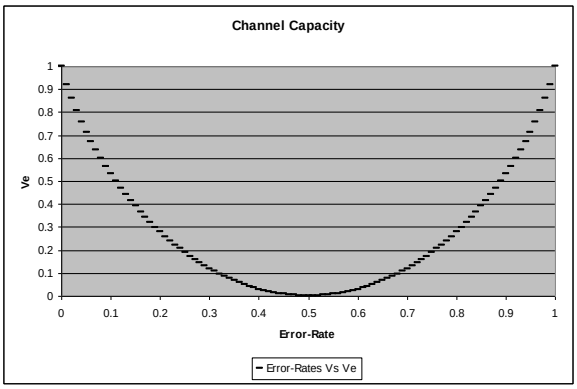

Figure (9): Scenario (R=N-t) of Channel Capacity having Error Rates Vs [V (e)] [9]

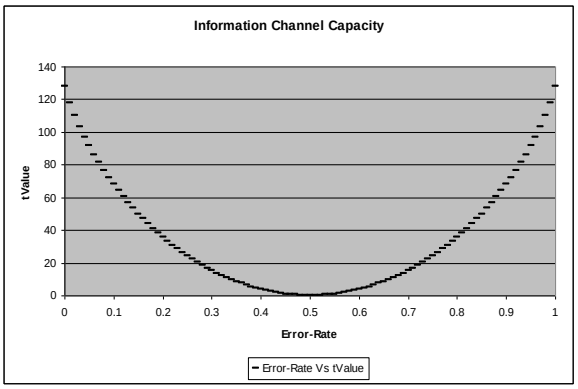

Figure (10): Scenario (R=N-t) of Information Channel Capacity having Error Rates Vs (t-Value) [9]

Information Channel Capacity Model Verification using [t-Value] could be achieved get the Channel Capacity in Information Units (per second) as defined into equation Equation (8) in section Equation (2) where the transmitted bits (K) in (C) is refined to be (N) [9].

The (C_s) is refined as V(e) which is defined in terms of (e) or (Error Rate). The (C) is refined to be (t) and defined to be (t-Value) or [(t) = N * V (e)]. The (Error Rate) was graphed against (t-Value) as shown in **Figure (10)** and concluded the following readings [9]:

- Error Rates of value (e = 0) generates V(e) of value (1)
- Error Rates of value (e = 1/2) generates V(e) of value (0)
- Error Rates of value (e = 1) generates V(e) of value (1)

Applying the (t-Value) rule by multiplying the V(e) by the (N) represented by (KeySize) parameter value that happened to be (128) as configured into the experiment, we get [9]:

- Error Rates of value (e = 0) generates [N*V(e)] of value (128*1 = 128)
- Error Rates of value (e = 1/2) generates [N*V(e)] of value (128*0 = 0)
- Error Rates of value (e = 1) generates [N*V(e)] of value (128*1 = 128)

Throughout our work, we managed to prove the applied equations of the Privacy Amplification model are in compliance with the Information Theory model concerning the (C) or the Channel Capacity in Information Units (per second). All results generated by the three Mod and Div functions are successfully verified because graph equations in **Figure (10)** are using (C) within their equation formulation [9].

6.3 Experimental Scenarios for (R=N-t-S)

The experiment was configured to use (KeysNum=100), (KeySize = 128) of same (MD4) signature size, (LoopsNum = 10), (HashFunction = MD4), (EndianType = Little) and (ChannelErrorRate = 0.50) used to apply the mirrored case of having (V (**e**) = 0, where **e = 0.50**) and (tValue = 0) in order to study the full range of (R-Computed) through the full range of Security Parameter (**S**) for both implementations of Mod and Div functions. The Mod and Perm-Mod functions return the same input key size since both functions are preserving the key size, yet the Hash-Mod function compresses the key to its signature size which is (128) bits. This helped in applying fair comparisons of Entropy values for equal (R-Computed) of the three Mod functions. The experiment aimed to adjust the key size configuration parameter to be of the same hashing signature size. Therefore, the experiment deployed the (MD4) hashing algorithm of (KeySize) equals to (128). By inspecting the results for the three Mod functions implementations, they recorded maximum Entropy value for every one at (R-Value) of (128) for Security Parameter (S) of (0) and zero Entropy value for every one at (R-Value) of (0) for Security Parameter (S) of (128) value. Also, the results show that the average Entropy values are close for the three algorithms for (R-Value) from (1) to (108) [9].

However, continuing (R-Value) from (109) to (128), the Entropy values of Perm-Mod algorithm become higher than those of the Mod function. The Entropy values of Hash-Mod algorithm are also higher than those of Mod function and Perm-Mod function in that region. Consequently, the knee-shaped of Mod function at high (R-Value) from (109) to (128) corrected by Perm-Mod function and further rectified by Hash-Mod function [9].

Finally, the (R=N-t-S) simulator computes the (R-Value) through the (R-Computed) because it is a function in Security Parameter variable (S), yet having (e) of fixed value determined via the (ChannelErrorRate = 0.50) configuration parameter. In turn, this shows that the (R-Computed) values were sequentially generated as they resulted from an equation that depends on (S). The (R-Computed) is calculated as (R=N-S) where (t = 0) and this shows that (R-Computed) depends on (N) and Security Parameter variable (S). Also, the Security Parameter variable (S) lies in range of $(0 < S \leq N)$ instead of $(0 < S \leq N - t)$ where (t = 0). The Information Measure introduced by Bennett and Brassard defined as [$I(K;GV) \leq \frac{2^{-S}}{\ln 2}$] is applied to every value of (R-Computed) generated in terms of (S) and (N) [9].

The behavior of the three Mod algorithms is shown graphically in **Figure (11)**. Both Mod and Perm-Mod Entropy functions resulted from implementing the standard (BB84) Privacy Amplification and demonstrates that the permutation function would produce more Entropy improvement because of the permutation scattering effect. In addition, the knee-shaped curve, behaved as a constant function in its region, which was demonstrated by the Mod function, was later cured by the effect of using Perm-Mod function at high (R-Computed) as a straight line-shape. Later, the current work introduced new function named Hash-Mod function that apply the message digest hashing functions by hashing the key to a fixed hash signature then apply the Mod function. The Hash-Mod function showed more increasing curve behavior over the Perm-Mod and Mod functions [9].

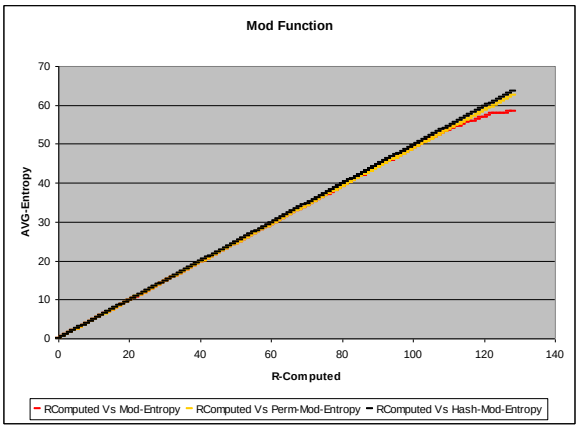

Figure (11): Scenario (R=N-t-S) for Mod-Function using (MD4-128) of Little Endian [9]

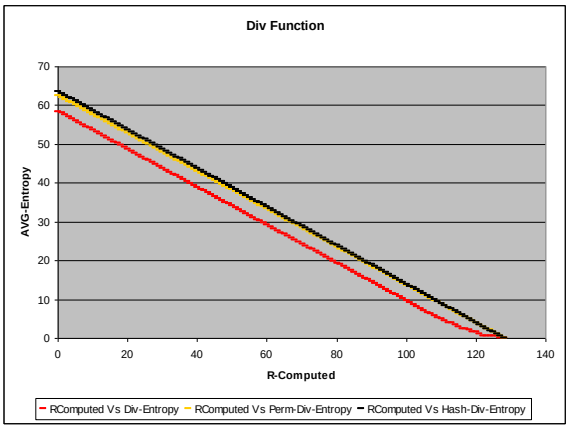

Figure (12): Scenario (R=N-t-S) for Div-Function using (MD4-128) of Little Endian [9]

Moreover, the knee-shaped appeared at high (R-Computed) ranging from (109) to (128) had diminished later by the effect of using Perm-Mod function. The Hash-Mod function managed to show such progress because of the hashing compression and scattering effects. Therefore, the hashing technique had

improved Entropy values which proved to be more promising during experimentations.Such correction of knee-shaped of Mod function in both of Perm-Mod and Hash-Mod functions had preserved whole range of (R-Computed) to be fully utilized instead of removing such defected knee-shaped area in Mod function [9].

The Div and Perm-Div functions return the same input key size since both functions are preserving the key size. However, the Hash-Div function compresses the key to its signature size which is (128). This aided in applying fair comparisons of Entropy values for equal (R-Computed) of the three Div functions. Similar to the behavior of Mod function, the experiment adjusted the key size to be of the same hashing signature size. The experiment deployed the (MD4) hashing algorithm and in turn the used (KeySize) for the three Div function implementations was (128). Also, the experiment hashing used the (Little) Endian type as a configuration parameter [9].

By examining the results for the three Div functions implementations, they recorded zero Entropy value for every one at (R-Value) of (128) for Security Parameter (S) of (0) and maximum Entropy value for every one at (R-Value) of (0) for Security Parameter (S) of (128) value. Also, the results show that the average Entropy values of Hash-Div function are slightly higher than those of Perm-Div function over the entire range of (R-Value). On the other hand, the Entropy readings of Div function are far below those of Perm-Div and Hash-Div functions. Consequently, the knee-shaped of the Div function at high (R-Value) from (109) to (128) was corrected by Perm-Div function and further rectified by Hash-Div function [9].

Finally, the (R=N-t-S) simulator computes the (R-Values) through the (R-Computed) because it is a function in Security Parameter variable (S), yet having (e) of fixed value determined via the (ChannelErrorRate = 0.50) configuration parameter. In turn, this shows that the (R-Computed) values were sequentially generated as they resulted from an equation that relies on (S). The (R-Computed) is computed as (R=N-S) where (t = 0) and this shows that (R-Computed) relies on (N) and the Security Parameter variable (S). Also, the Security Parameter variable (S) lies in the range of $(0 < S \le N)$ instead of $(0 < S \le N - t)$ where (t = 0). The Information Measure introduced by Bennett and Brassard formulated as [$I(K;GV) \le \frac{2^{-S}}{\ln 2}$] is applied to every value of (R-Computed) generated in terms of (S) and (N) [9].

The behavior of the three Div algorithms is shown graphically in **Figure (12)**. The Perm-Div and Hash-Div Entropy functions showed a decreasing behavior, while the Div Entropy function showed more decreasing behavior than the others. Both Div and Perm- Div results were obtained as suggested by the standard (BB84) Privacy Amplification that demonstrated the permutation function would show much entropy improvement because of the permutation scattering effect. In addition, the knee-shaped curve, behaved as a constant function in its region, which was obtained for the Div function was later cured by the effect of using Perm-Div function at (R-Computed) in range of (109) to (128) as a straight line-shape. The Perm-Div Entropy curve showed a remarkable enhancement over the Div Entropy curve that proved the importance of the permutations [9].

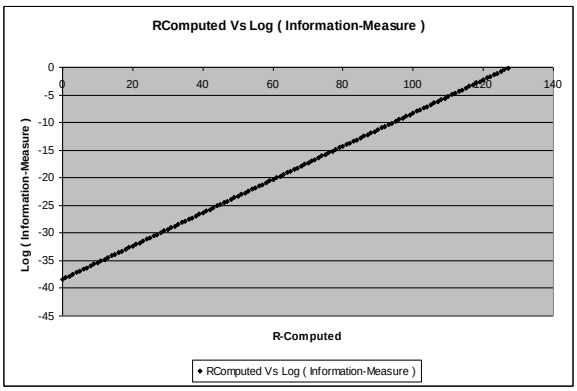

Figure (13): Scenario (R=N-t-S) for R-Computed Vs Log (Information-Measure) [9]

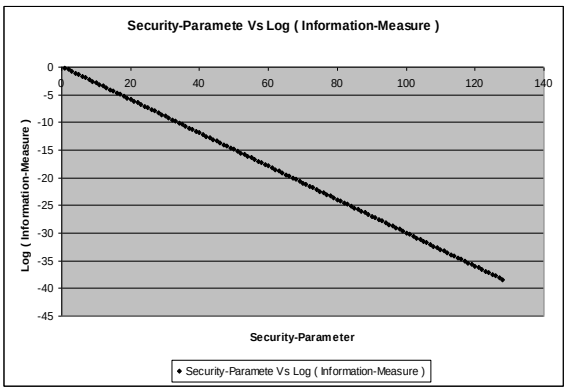

Figure (14): Scenario (R=N-t-S) for Security-Parameter Vs Log (Information-Measure) [9]

Later, the current work introduced new function named Hash-Div function that apply the message digest hashing functions by hashing the key to a fixed hash signature then apply the Div function. The Hash-Div function showed higher curve over both Perm-Div and Div functions. Moreover, the knee-shaped appeared at high (R-Computed) ranging from (109) to (128) had diminished later by the effect of using Perm-Div function. The Hash-Div function introduced in the present work showed such progress because of the hashing compression and scattering effects. Therefore, the hashing technique had improved Entropy values which proved to be more promising during experimentations. Such correction of knee-shaped of Div function

in both of Perm-Div and Hash-Div functions had preserved the whole range of (R-Computed) to be fully utilized instead of removing such defected knee-shaped area in Div function. Moreover, the permutation and hashing had improvements over the Div curve because they generated elevated curves for both Perm-Div and Hash-Div than the Div function curve [9].

The Information Measure Metric shows the results obtained for the Mod and Div three function forms. By inspecting the columns of Security Parameter (S), (R-Computed) and Information Measure in both tables, it is concluded that the (R-Computed) values are directly proportional with the Information Measure, yet the Security Parameter (S) values are inversely proportional with the Information Measure. The highest value of (R-Computed) is (128) has an Information Measure of (1.442695041) and the lowest value of (R-Computed) is (0) has an Information Measure of (4.2397E-39). However, the lowest value for the Security Parameter (S) is (0) has an Information Measure of (1.442695041) and the highest value of Security Parameter (S) is (128) has an Information Measure of (4.2397E-39). The Information Measure, introduced by Charles Bennett and Gillis Brassard who developed the (BB84) protocol, is the second metric used to ensure the restriction of an eavesdropper's knowledge upper bound about the compressed or secret key transmitted between the two legitimate users over the communication channel. The Information Measure formulated equation states that (Eve's) expected information on secret key (K) given (G), (V) as [$I(K;GV) \leq \frac{2^{-s}}{\ln 2}$] which is a function in Security Parameter (S) [8]. Hence, the Information Measure equation shows that the more the quantity [$\frac{2^{-s}}{\ln 2}$] is small, then the lesser information (I) of the eavesdropper happens. This implies that the Privacy Amplification phase of (BB84) protocol's security depends on minimization of the quantity [$\frac{2^{-s}}{\ln 2}$] in order to ensure (Eve's) expected information on secret key (K) is diminishing and having a distillated secret key free from leaked bits. Thus, the conducted experiments for (R=N-t-S) simulator proved such security notion where the least Information Measure value was accomplished for the secret key length or (R-Computed) equals (0), however, the Information Measure value kept on increasing as the secret key length or (R-Computed) was increasing in value. The behavior of such direct proportional relation is graphed in **Figure (13)** showing (R-Computed) Vs Log_{10}(Information-Measure). The Information Measure values were very small to be graphically represented; hence we applied Logarithm function of (base-10) to the Information Measure values. Such [Log_{10}] function maintained having a direct proportional relationship through an increasing behavior between (R-Computed) and (Information-Measure) [9].

Moreover, the conducted experiments for (R=N-t-S) simulators had also verified the security notion where the least Information Measure value was accomplished for the secret key Security Parameter (S) equals (128), however, the Information Measure value kept on increasing as the secret key Security Parameter (S) was decreasing in value. The behavior of such inversely proportional relation is graphed in **Figure (14)** showing the Security Parameter (S) Vs Log_{10}(Information-Measure). Again, the Information Measure values were very small to be graphically represented; hence we applied Logarithm function of (base-10) to the Information Measure values. Such [Log_{10}] function maintained having an inversely proportional relationship through a decreasing behavior between Security Parameter (S) and (Information-Measure) [9].

6.4 Enhancements of Hash over Permutation Functions

The behavior of Perm-Mod and Hash-Mod algorithms demonstrating the average Entropy progress which are deployed within the three defined types of simulators and configured with (MD4-128) message digest hash function is shown in **Figure (15)**. The hashing effect of Hash-Mod function showed enhanced values over the permutation effect of Perm-Mod function. By examining **Figure (15)**, we noticed that the (**R=N-t**) simulator showed higher values of permutation and hashing over (**R=N**) and (**R=N-t-S**) ones which were nearly equal in values. The difference in average Entropy progress percentage for [(Hash-Mod) – (Perm-Mod)] % applied to **Figure (15)** [9]:

- (**R=N**): (4.556918239) - (2.729937918) = (1.826980321 %)
- (**R=N-t**): (4.95530088) - (3.931554798) = (1.023746082 %)
- (**R=N-t-S**): (4.456260295) - (2.686937993) = (1.769322302 %)

Also, the behavior of Perm-Div and Hash-Div algorithms demonstrating the average Entropy progress which are deployed within the three defined types of simulators and configured with (MD4-128)

message digest hash function is shown in **Figure (16)**. The hashing effect of Hash-Div function showed enhanced values over the permutation effect of Perm-Div function. By examining **Figure (16)**, we noticed that the (**R=N-t**) simulator showed higher values of permutation and hashing over (**R=N**) and (**R=N-t-S**) ones which were nearly equal in values. The difference in average Entropy progress percentage for [(Hash-Div) – (Perm- Div)] % applied to **Figure (16)** [9]:

- (**R=N**): (177.7187055) - (176.7871512) = (0.9315543 %)
- (**R=N-t**): (198.3982817) - (194.8842781) = (3.5140036 %)
- (**R=N-t-S**): (176.1395125) - (173.241891) = (2.8976215 %)

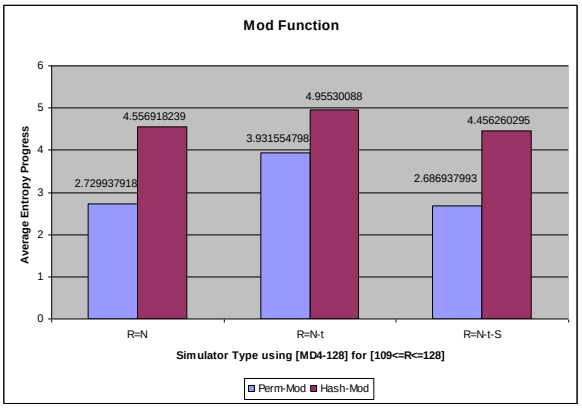

Figure (15): Average Entropy Progress Bar-graph Mod Func of [MD4-128] of Little Endian [9]

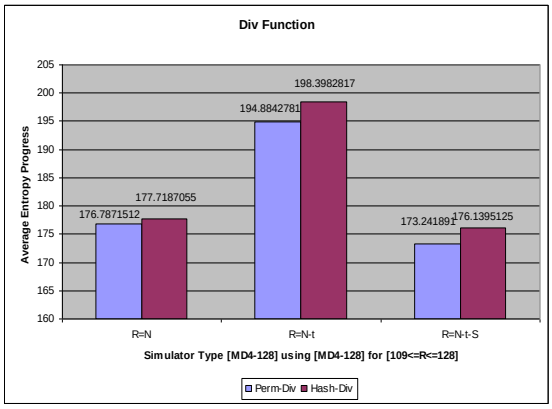

Figure (16): Average Entropy Progress Bar-graph Div Fun of [MD4-128] of Little Endian [9]

7. Conclusions

This work aimed to acess the performance of traditional $Universal_2$ Hash functions, recommended by (BB84), through the combination of the message digest hash functions with the truly random functions of Mod and Div to have Hash-Mod and Hash-Div functions in distilling the secret key in the Privacy Amplification. All results were supporting the use of Hash-Mod and Hash-Div over the Perm-Mod and Perm-Div in rectifying the Entropy behavior of Mod and Div functions. In order to be able to apply the new functions, an implementation was made to the standard (BB84) introduced functions as: Mod, Perm-Mod, Div and Perm-Div functions. The performance was measured on the basis of Entropy and Information Measure. In addition, an implementation was made to boost the performance with new enhancement of Hash-Mod and Hash-Div functions which reported much enhanced results when being compared to the Entropy values of the former functions [9].

The study succeeded to categorize three types of simulators, (**R=N**), (**R=N-t**) and (**R=N-t-S**) where every one was configurable through configuration parameters. Such variation in parameter configuration managed to prove that the three simulators agreed on generating a similar function graph behavior produced by experiments. The Mod function preserved an incrementing knee-shaped in every experiment at high (R-Value) values. Such problem was corrected by permutation that generated an incrementing straight line graph. The hashing has generated enhanced incrementing graph that rectified the knee-shaped problem. Similarly, the Div function preserved the knee-shaped in every experiment at high (R-Value) values. Such problem was corrected by permutation which generated a decrementing straight line graph. The hashing has generated enhanced decrementing graph which rectified the knee-shaped problem [9].

This paper demonstrated set of experiments execution using a set of customized configuration parameters using a small hash function signature size (MD4) through practical investigations in section (6) for the three types of simulators. More investigations were applied for another set of customized configuration parameters using bigger hash function signature size (RIPEMD320). Both results proved that the Mod and Div function implementations had maintained the same function behavior proving the success of combing the hash functions with the truly random functions. Throughout the three defined simulators which used the message digest hash function (MD4-128) for high (R-Value), the average Entropy progress of permutation combined with Mod function and deployed with (**R=N-t**) showed higher values of (3.93) over (**R=N**) and (**R=N-t-S**) ones which are closely equal in values. Also, the average Entropy progress of hashing combined with Mod function used within (**R=N-t**) showed higher values of (4.95) over (**R=N**) and (**R=N-t-S**) ones that are nearly equal. The average Entropy progress percentage for Mod function is (1.02 %) [9].

Similarly, the average Entropy progress of permutation combined with Div function and deployed with (**R=N-t**) showed once more higher values of (194.88) over (**R=N**) and (**R=N-t-S**) ones that were nearly the same. Also, the average Entropy progress of hashing combined with Mod function used within (**R=N-t**)

displayed higher values of (198.39) over (**R=N**) and (**R=N-t-S**) ones which are nearly equal. The average Entropy progress percentage for Div function is (3.51 %). Such behavior of results was maintained when deploying higher signature of message digest hash function such as (RIPEMD-320).

This proves that the hashing effect produced more enhanced Entropy progress over that of permutation when combined with Mod or Div functions and can be recommended to improve the Privacy Amplification model. Therefore, the use of the Hash-Mod and Hash-Div has influenced the key distillation process of the Privacy Amplification phase of (BB84) protocol. Moreover, the performance measure of the new functions was remarkable since hashing had improved the Entropy measurements over the permutation ones because not only the hashing provides key bits random scattering but also provides key compression. The choice of hash functions could be influenced according to the complexity and speed of the selected hash function in order to achieve the distillation process in lesser processing time [9].

In conclusion, the newly implemented functions were proved to be superior to the standard implemented ones in terms that they generated shorter keys with randomized bits with much enhanced Entropy than the standard ones that generate keys of same length with randomized bits [9].

ACKNOWLEDGEMENTS

We are grateful to Professors Gillis Brassard, Charles Bennett, Artur Ekert, Louis Salvail, Claude Crépeau and Mr. Christian Knopf for many helpful discussions and sending us many missing papers. In particular, Professor Brassard (University of Montréal) had uploaded some of his crucial papers on his homepage to download and we had some short discussions about Privacy Amplification applications through emails. Professor Bennett (IBM researcher) was enthusiastic enough through our discussions about quantum cryptography. Besides, Professor Salvail sent us a (BB84) java simulator source code which was developed by him and his colleagues in publishing some of their papers. Professors Artur Ekert and Claude Crépeau emailed us some rare papers that were not posted or downloaded from the Internet.

REFERENCES

[1] Zhou Xu, "An introduction to Quantum Key Distribution", ACM Journal Name, Vol. V, No. N, (October 2002), www.comp.nus.edu.sg/~xuzhou/reports/quantum-cryptography-survey-xuzhou-11-2002.pdf.

[2] Ch. Kollmitzer, Ch. Monyk, M. Peev, M. Suda, "An Advance towards Practical Quantum Cryptography". ARC Seibersdorf research Ltd. Austrian Research Centers (2002). http://www.arcs.ac.at/quanteminfo

[3] Charles. H. Bennett, Francois Bessette, Gilles Brassard, Louis Salvail, and John. A. Smolin, "Experimental Quantum Cryptography", Advances in Cryptology - Proceedings of Eurocrypt '90, Aarhus, Springer – Verlag, (pp. 253 - 265), (September 1991).

[4] Charles. H. Bennett, Gilles Brassard and Jean-Marc Robert, "How to Reduce you Enemy's Information (Extended Abstract)", Advances in Cryptology - Proceedings of Crypto '85, Santa Barbara, Springer – Verlag, (pp. 468 – 476), (August 1998).

[5] Charles. H. Bennett, Gilles Brassard and Jean-Marc Robert, "Privacy Amplification by Public Discussion", SIAM J. COMPUT. Vol (17), NO (2), (pp. 210 – 229), (April 1988).

[6] Mark N. Wegman, and J. Lawrence Carter, "New Hash Functions and Their Use in Authentication and Set Equality". Journal of Computer and System Sciences-JCSS (22), (pp.265-279), (1981).

[7] B. P. Lathi, "An Introduction to Random Signals and Communication Theory". International Text Book Company. Scranton, Pennsylvania. (pp.428-450), (1968).

[8] Charles. H. Bennett, Gilles Brassard, Claude Crepeau, and Ueli M. Maurer, "Generalized Privacy Amplification", IEEE International Symposium on Information Theory - Trondheim, Norwat, (page 350), (May 1995).

[9] Ahmed Mahmoud Abbas Abd-Allah, " Privacy Amplification in Quantum Cryptography Protocols," M.Sc. Thesis 320 pages, Department of Computer Science, The American University in Cairo, 2009.

[10] FlexiProvider, www.flexiprovider.de, The Computer Science Department at Technische Universität Darmstadt in Germany.

BIOGRAPHY OF AUTHORS

Mr. Ahmed Mahmoud Abbas is a software project manager. Mr. Ahmed received his BSc in 2001 and MSc in 2009 from the Department of Computer Science and Engineering at the American University in Cairo. His research interests are concerned with software project management, enterprise portal development, computer security and cryptography and open source software development.
Email : aabbas@aucegypt.edu

Dr. Amr Goneid is a professor in the Department of Computer Science and Engineering at the American University in Cairo. Dr. Amr joined AUC in 1990 and served as the chair of department from 1994 to 2002. Also, he served as the director of graduate programs from 2007 to 2009. Dr. Amr's research interests are: image processing, pattern analysis, Human Computer Interaction and cryptography.
Email : goneid@aucegypt.edu

Dr. Sherif El-Kassas is a professor in the Department of Computer Science and Engineering at The American University in Cairo. Dr. Sherif's research interests are focused on security management, application of formal methods in software engineering and computer security. He also works in the interdisciplinary area concerned with open source development and innovation.
Email : sherif@aucegypt.edu