MCSAuth: A New Authentication Mechanism for Cloud Systems

François S. N. Athanasios,
Electrical & Computer Engineering, HTI
Tenth of Ramadan City, Egypt

Sherif F. Fahmy
Computer Engineering, AASTMT
Cairo, Egypt

Gamal I. Selim
Computer Engineering, AASTMT
Cairo, Egypt

ABSTRACT
Cloud Computing has attracted a lot of attention in both academia and industry lately. With its focus on scaling, collaboration, agility, availability and cost reduction, cloud computing offers a compelling alternative to in-house IT solutions. However, by "outsourcing" the computing infrastructure, it introduces a number of security issues. Specifically, since cloud computing is a shared computing platform, it needs to provide strong mechanisms for authenticating it's users and ensuring that no confidential information stored on the cloud is compromised. This paper addresses a number of vulnerabilities in existing authentication mechanisms and proposes enhancements to mitigate these vulnerabilities. In addition, the paper studies how these enhancements affect performance. The results show that the existing vulnerabilities can be overcome by the proposed mechanisms. However, this results, in the worst case, in a six-fold increase in execution time. This can be considered relatively small when security is prime important.

Keywords
Communication networks, Cloud Computing, Authentication, Hybrid, Encryption, Keyed hash.

1. INTRODUCTION
Cloud computing is an evolution of traditional computing resources into shared utility-like services. Instead of each organization or business building its own in-house computing infrastructure, cloud computing allows a third party to provide this infrastructure and to rent out its services on an on-demand basis. This provides a number of important benefits, among which are increased scalability, availability, agility, collaboration, and cost reduction.

Thus, cloud computing can be considered a new computing model where resources are provided as a service over the Internet or any other networking infrastructure. A cloud user is offered a variety of services on the cloud, including operating an entire operating system, storing data and using applications that are hosted off-site. It is based on utility pricing. This means that the users can get resources simply by paying a monthly fee, or pay per using these resources, like you do with utility bills (e.g. renting an apartment for a monthly payment).

Cloud computing can deliver a vast array of its capabilities in real time using many different types of resources such as hardware, software, and virtual storage, once one logs onto a cloud [1]. In general, it describes a new delivery model of services based on the Internet. It uses virtualized computers that are dynamically provisioned and is presented as a service over the Internet. This is based upon SLA "Service Level Agreements" established between users and service providers through negotiations [2].

Currently, the three types of cloud computing offered are public, private and hybrid clouds. As their names imply, a public cloud is hosted by a third party, a private cloud is hosted by the entity that needs cloud services, and a hybrid is a combination of both.

Three primary models of cloud computing services have emerged, namely SaaS "Software as a Service", PaaS "Platform as a Service", and IaaS "Infrastructure as a Service". SaaS is the model in which an application is hosted as a service by customers who access it via the Internet [3]. PaaS is a model in which a platform, an OS and supporting APIs and software development environment, are provided [4]. IaaS is a model in which only the underlying infrastructure, networking and processing nodes, are provided. Everything else is up to the customer [5].

Cloud computing environment consists of two main entities: the users and the CSP "Cloud Service Provider". Both need to establish a secure channel for either storing or retrieving data. While information is being transmitted, a secure channel protects data from malicious activities by an adversary trying to impersonate either the CSP or the user.

Several papers such as [6, 7, 8 and 9] focus on information and resources protection from unauthorized users. It is of vital importance that a secure channel be established between the end user and the CSP to maintain trust and confidentiality.

The objective is to design a secure authenticated communication channel between the user and the CSP. In order to feel safe and enjoy the usefulness and preference of cloud computing over many types of networks like wired and wireless, information should be secured taking into consideration CIA "Confidentiality, Integrity and Availability" [10, 11 and 12].

2. EXISTING AUTHENTICATION METHODS AND THEIR LIMITATIONS
Cloud sessions must be secured. Businesses hesitate to join the cloud because they are not sure that it is 100% secured from any malicious activities. Cloud sessions can be secured via encryption and authentication, though authentication is another issue, with many options available. Many authentication schemes have been introduced to deal with secret data over insecure networks, remote login systems and computer networks, all of which are found within cloud computing systems. In this section, there is the result of our survey through a group of the current available authentication schemes.

Most of the existing methods are vulnerable to many attacks, which cause insecurity to users who use systems dealing with these schemes. Choosing the proper authentication method depends mainly on the type of the entities being authenticated

whether public, private or hybrid and the degree of trust that the entities have.

1.1. Design Methodology:

Strong authentication guarantees the secrecy without revealing it and is indispensable when two entities communicate in untrustworthy environment like that of the cloud computing system [12]. Now, a group of scenarios and their limitations will be discussed.

1.2. Notation:

The following notations have been used in the coming figures:

U:	Refers to user.
C:	Refers to cloud service provider.
+:	For public key.
-:	For private key.
Ku+:	Information and data owner's public key.
Ku-:	Information and data owner's private key.
K_{c+}:	CSP'S public key
K_{c-}:	CSP'S private key
⊕:	Bit-wise XOR.
$SK_{u,c}$:	Secret key shared by user and CSP.
[U,C]:	User's identity and CSP'S identity.
KDC:	"Key Distribution Centre".
Token:	A pair of username, password and session key.
$K_{U,KDC}$:	Secret key between user and KDC.
$K_{C,KDC}$:	Secret key between CSP and KDC.
R:	Challenge (Random number).
Shuffler[X,Y]:	Shuffling that takes place between X and Y bits.
[X]HMAC:	X is hashed by HMAC "Hash-based Message Authentication Code" algorithm.

1.3. Scenarios:

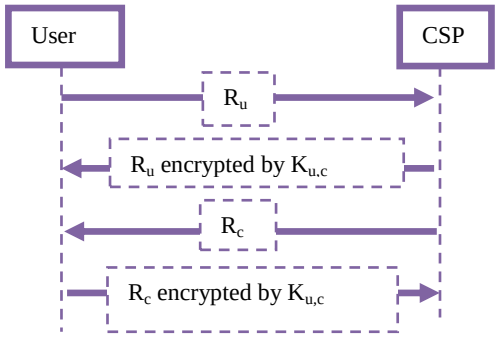

Fig 1: User and CSP use challenge – response authentication.

Ru and Rc are random numbers chosen by user and CSP respectively as a challenge. In the second step, Ru is encrypted by Ku,c (shared secret key between user and CSP) and is called response to the challenge Ru. The previous process is repeated but in reverse by the cloud provider to perform mutual authentication. If the last step is successful, then user and CSP are authenticated.

If the remainder of the conversation is not encrypted, an adversary can act as user after authentication is over, by using user's address. Clocks of user and CSP need to be synchronized. An advantage is that CSP doesn't need to maintain R. Shared secret mechanisms have one problem which is vulnerability if CSP's data base can be read. In PKC "Public Key Cryptography" mechanism, encryption takes place using public key, and decryption using private key. Therefore, no shared secret key is necessary.

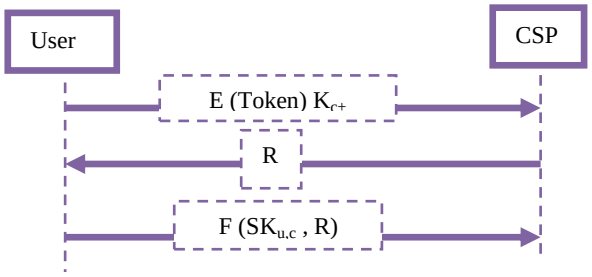

Fig 2: User is authenticated by CSP based on encryption or hash.

The above mechanism token is encrypted using CSP's public key and is sent to CSP. CSP decrypts the previous message by using its K_{c-}, and sends a challenge to user to be sure whether he/she is the real one or not. User sends back the challenge with $SK_{u,c}$ in a function form F ($SK_{u,c}$, R), where R is encrypted or hashed by $SK_{u,c}$. However, this above scenario has some limitations:

- Here authentication is not mutual; the user doesn't authenticate the CSP. Through address spoofing an adversary can try to convince the user that the adversary's address is the CSP's address. The adversary can send any old challenges and ignore user's response.
- If the remainder of the conversation is not under cryptographic protection, an adversary can hijack the channel.
- An attacker can mount off-line password guessing attack, if $SK_{u,c}$ is generated from password, and both R and F($SK_{u,c}$,R) are known.

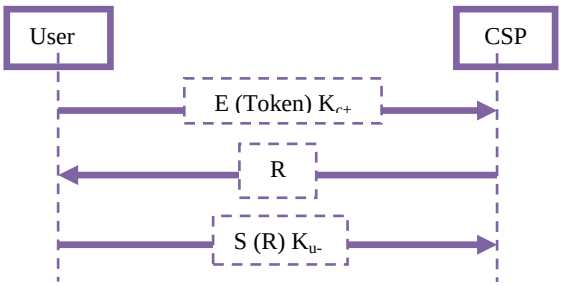

Fig 3: CSP authenticates user based on one-way PKC (a).

For the above scenario, user sends his/her information (Token) encrypted by CSP's public key (where K_{c+} is known to user). CSP decrypts the previous step by its K_{c-} and sends a challenge R to user. User signs R using user's private key and sends it to CSP. Using the user's public key K_{u+}, CSP decrypts the signed challenge and compares it with the one held in the CSP.

Thereby, the CSP authenticates the user. In the above scenario, it is called one-way public key. There is a problem: an attacker can trick a user into signing something (forging),

act as CSP, wait for user to send a login request, and send a certain quantity as the challenge R. User will sign it using his/her private key, thinking it is R !

In the following scenario, in order to know what was sent earlier in an encrypted message by someone to a user, act as CSP, wait for user to send login request, and have the user decrypt the encrypted message for you. Same as the previous scenario, user can send encrypted token with CSP's public key. CSP decrypts it by CSP's private key. Here the negotiation is reversed: CSP signs a challenge with user's public key and sends it to user. User decrypts the previous step by his/her private key and retrieves the challenge in order to send it back to CSP. Now, CSP authenticates user.

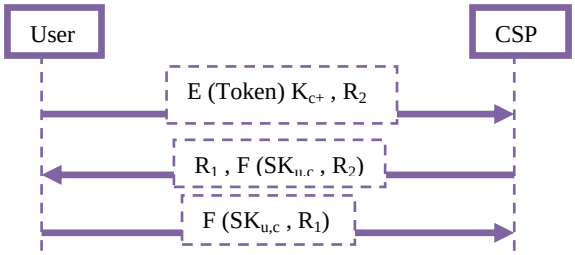

Fig 4: Mutual authentication between CSP and user based on shared secret key.

The above scenario is initialized by public key cryptography and the rest of it is achieved by shared secret key. User sends his/her credentials to CSP encrypted by CSP's public key followed by challenge R_2. CSP sends challenge R_1 appended to the function containing the challenge R_2 using shared secret key.

User decrypts and verifies challenge R_2 to authenticate CSP and sends function F ($SK_{u,c}$, R_1) containing challenge R_1. CSP decrypts it and verifies R_1. Now both user and CSP authenticate each other. This above scenario has a security pitfall known as reflection attack.

Reflection attack is a way of attacking a challenge-response mutual authentication mechanism through which an adversary tries to impersonate user to CSP and makes the CSP think that it is communicating with the user.

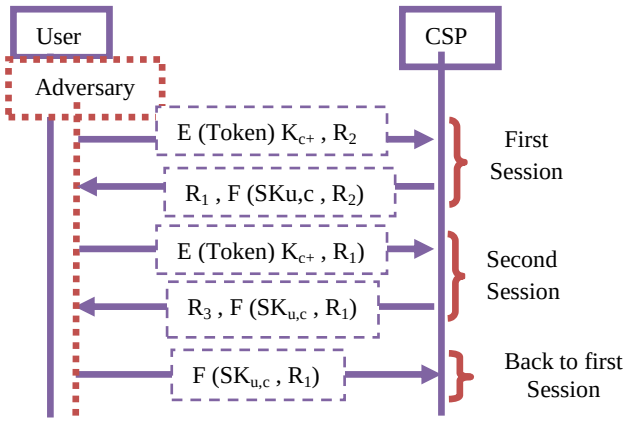

Fig 5: Reflection attack on mutual authentication.

In step 1 "First session or channel": an adversary sends user's identity along with R_2 to CSP. CSP sends back encrypted R_2 and R_1 as a challenge. The attacker needs to encrypt R_1 and send it back to CSP, but he/she cannot! Adversary cannot continue any more as he/she does not have a shared secret key. Adversary sets up a new session.

In step 2 "second session", the attacker does the same as the first step with R_1 instead R_2. The CSP encrypts R_1 and sends it with another challenge R_3 back to the attacker. At this point, the adversary gets the encrypted R_1, abandons the second session and resumes the first session, and then it resends the encrypted R_1 which was originally received from in the second session.

In order to solve the previous problem, there should be different keys and challenges, two shared keys: for example $SK_{u,c}$ and $SK_{c,u}$. Therefore, CSP cannot encrypt using user's key now. The challenges sent by user and CSP to each other should be different. Another solution is to use different time stamps, different keys and add identity names before encryption.

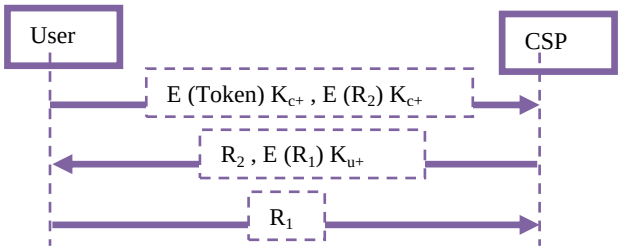

Fig 6: Optimized mutual authentication based on PKC

In Fig. 6, user sends his/her credentials encrypted by K_{c+} followed by a random number R_2 chosen by user as a challenge which is also encrypted by K_{c+}. CSP receives the encrypted message from user and starts to decrypt it using K_{c-} and retrieves R_2. CSP sends R_2 back followed by encrypted random number R_1 using K_{u+}. User receives this message and verifies R_2 so as to authenticate CSP. User starts to decrypt the encrypted part using K_{u-} to get R_1 and forwards it back to CSP. CSP receives it and authenticates user.

PKC has some limitations: neither user nor user's workstation are going to remember the CSP's public keys stored respectively. This problem can be solved by making a trusted entity to handle all users' domain and that works also for the CSP. Another form of mutual authentication using PKC is as follows:

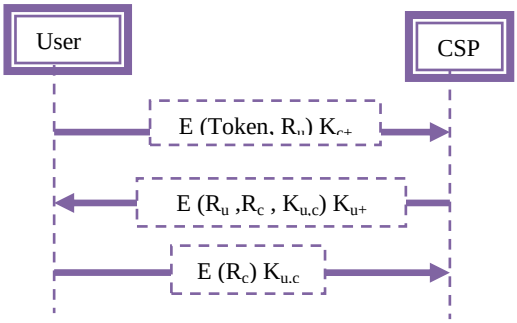

Fig 7: Improved mutual authentication between user and CSP based on PKC and session key

To achieve the above mechanism, user sends his/her credentials followed by R_u random number as a challenge, all encrypted by K_{c+}. CSP decrypts the previous message by K_{c-} and sends R_u, R_c and session key $K_{u,c}$, all encrypted by K_{u+} . (Where Ru is a challenge chosen by user and R_c is chosen by CSP, both Ru and Rc should be different from each other). User will decrypt E (R_u , R_c , $K_{u,c}$) K_{u+} by user's K_{u-} and return the response of the CSP's challenge R_c to the CSP encrypted by $K_{u,c}$ session key.

The problem faced when dealing with Cloud Computing Environment is not only the huge number of keys but also the distribution of keys. If a user wants to communicate with a hundred people, how can he/she manage or exchange a hundred keys with them? Using the web is definitely not a secure way of dealing with a huge number of secret keys without hassle.

A centralized approach by means of a Key Distributed Centre, manages n keys instead of n (n-1) / 2, which is clearly an improvement over the situation handled without any entity like KDC. [11] KDC will act as a trusted third party. Each user will establish a shared secret key with it.

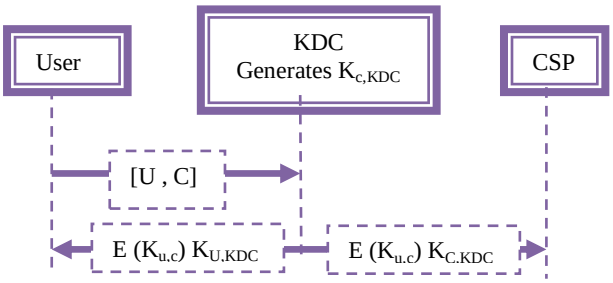

Fig 8: KDC idea

In Fig. 8, user sends his/her identity and CSP's identity to KDC. KDC sends to both user and CSP session key $K_{U,C}$ encrypted by $K_{U,KDC}$ respectively. There is a problem when user instantly sends a message after getting $K_{U,C}$ to the CSP before the CSP gets session key from KDC. This problem can be solved if the KDC just passes $E(K_{U,C})$ $K_{C,KDC}$ back to the user. This message $E(K_{U,C})$ $K_{C,KDC}$ is called "ticket". That will help users to access CSP. User will send this ticket to the CSP while communicating as shown in the next scenario K.

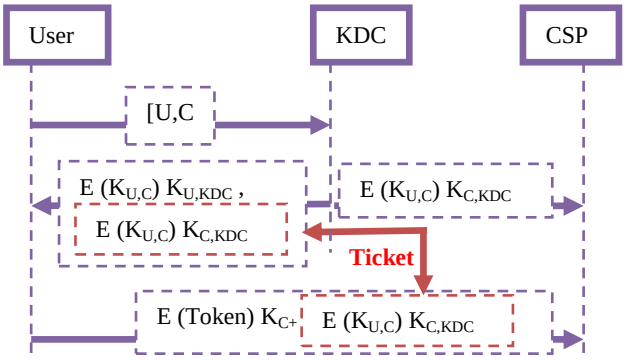

Fig 9: KDC using ticket

3. PROPOSED MECHANISM "MCSAuth"

Cloud computing is a SSO "Single Sign On" system for cloud users. Only registered users who have valid tickets are accepted by cloud systems.

An authentication mechanism was proposed in [9] for cloud computing to avoid any misuse of cloud systems. This mechanism is said to be an improvement over SSL "Secure Socket Layer", as SSL has some fallbacks like session hijacking and Man-in-the-Middle.

For SSL, an adversary will only need to have a symmetric key, so he/she can cause replay attacks whereas in the mechanism proposed for cloud systems in [9], an adversary will need to have a ticket and a session key. This can be done exceeding and surpassing the protection supplied by cryptographic algorithms.

The mechanism cited in [9] is more secure than the previous scenarios. In the next section, the mechanism in [9], together with its pitfalls, will be discussed in detail. Next figure shows the mechanism cited in [9] and the negotiations that go on between the nodes. There are four nodes: cloud users, AS "Authentication Server/Service", TGS "Ticket Granting Server/Service" and CSP respectively. This system is based on KDC with MA "Mediated Authentication". The mechanism cited in [9] can be named as "modified Kerberos".

AS accepts login request from cloud users. It is responsible for authentication between servers and cloud users by providing a key that can be used for securing communication channels. TGS is responsible for setting up secure channels by providing tickets which are used to convince service providers that the user is the one who claims to be. In the beginning, a user types his/her user's ID at a workstation. Then it is sent to AS which will go through a group of steps.

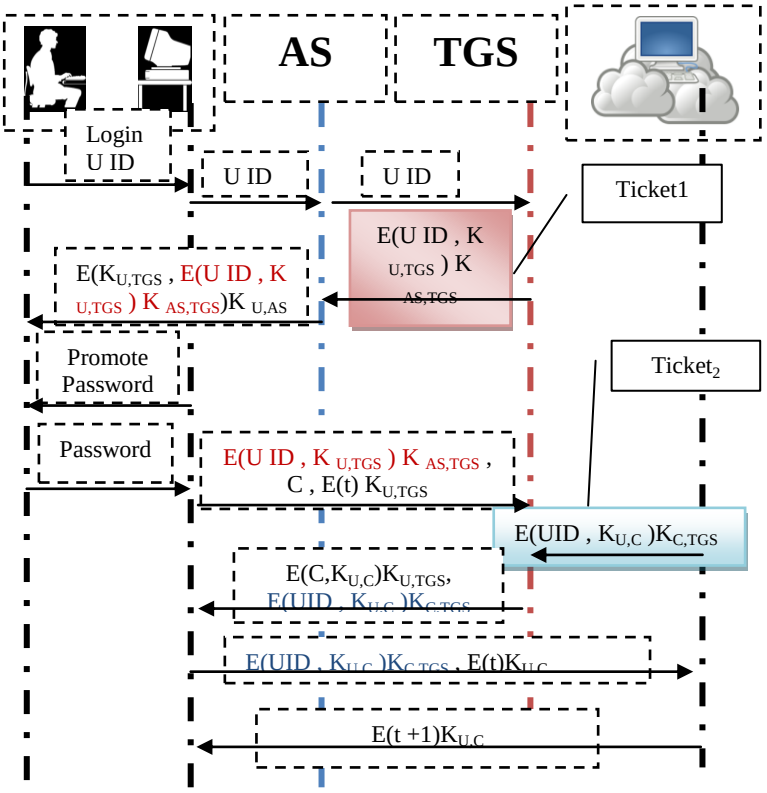

Fig 10: Modified Kerberos proposed in [9]

A shared secret key (Key $_{u,TGS}$) is used for confidentiality of message and discarded when the communication channel is no longer used. A ticket (E(U_{ID}, Key $_{u,TGS}$) Key $_{AS,TGS}$) is formed by TGS which represents ticket $_1$ and is forwarded to the user.AS encrypts ticket $_1$ appended by Key $_{u,TGS}$ by using the shared secret key between AS and cloud user Key $_{u,AS}$ to form (E(Key $_{u,TGS}$, Ticket $_1$) Key $_{u,AS}$).

By receiving the encrypted message from AS, user is asked to provide a password. Entering the right password will help in generating Key $_{u,AS}$. The workstation now can decrypt the received message from AS and distinguish Key $_{u,TGS}$. Now, the user's workstation has two shared keys: Key $_{u,TGS}$ and Key $_{u,AS}$. Phase 1 of authentication is finished: user is now connected by the workstation that he/she is currently using.

Ticket $_1$ is stored temporarily at the workstation; it proves that the users are genuine. User will send a request to TGS in order to set up a secure communication channel with CSP.

The request helps to prove that the cloud user is genuine; it will contain ticket $_1$ and cloud ID. To avoid replay attacks by an adversary, a timestamp is added to the request. The time stamp is verified by TGS. Then it forms and sends ticket $_2$ ($E(U_{ID}, Key_{u,C}) Key_{C,TGS}$) to CSP. Ticket $_2$ informs CSP about forthcoming communication from user. TGS sends ticket $_2$ to cloud user who forwards it back with the encrypted time stamp to CSP.

CSP checks time stamps and ticket $_2$ which are forwarded by cloud user. After successful verification, CSP authenticates itself to user by responding with ($E(t+1) Key_{u,C}$) to prove that the responder is CSP.

3.1.Vulnerabilities in the above mechanism

New attacks have emerged making the mechanism cited in [9] vulnerable. It is considered a malleable encryption system and part of it is subjected to unauthenticated encryption. The mechanism in [9] is subjected to an encryption oracle attack, MIC "Message Integrity Check" bits attack, and PO "Padding Oracle" attacks. It was proven in [2] that the following pitfalls affect the mechanism severely and open a chance for an adversary to attack easily.

3.2.MCSAuth "Multiple Crypto Shuffler Authentication" Mechanism

Solution must be found to counter the problems that were mentioned in the modified Kerberos mechanism. The goal is to enhance the overall security of the cloud not to counter the main vulnerabilities only. The more complex the mechanism becomes, the smaller the odds that an adversary can break or tamper it. MCSAuth design will be explained in the following section. A new node will be added called shuffler node. It sends information only to authorized and authenticated cloud workstations and AS/TGS nodes. Shuffler node is responsible for the transmission of shuffling tables and certain codes. It informs the other nodes how to reveal the shuffled message. Shuffler node will be connected to AS/TGS, cloud user's workstation and cloud systems.

Every user in a registered cloud will have three identifiers that act together as his/her credentials to help him/her log to the cloud system. The three elements are, User Identity, Password, and Registration codes given only to registered cloud users by cloud systems.

MCSAuth Mechanism passes through four steps at each node during negotiation, based on E-M then E-S "Encode and MAC "Message Authentication Code" then Encrypt and Shuffle".

3.2.1. Encoding data: It is done in this way: Data is given a form: a confounder is added at the beginning before data and then a random padding is appended at its end. A confounder is one block of random bytes. It is like a challenge between nodes during negotiation to authenticate each other. Destination doesn't know what random challenge to expect. It prevents cut-and-paste attacks, chosen plaintext attacks and using ciphertext as oracle since an adversary has no knowledge of the value of challenge.

Random padding is done so as to disguise the size of the encoded data by adding a random value of random bytes as padding, and to indicate the value of the random bytes added by the last byte in padded input block. Random padding avoids oracle padding.

3.2.2. MAC: It is a piece of information added to the encoded data in the end and this is used to authenticate the encoded data. In MCSAuth, HMAC-SHA-1 is used for integrity checking, that can't be forged by encryption oracle. An adversary can't construct a valid ciphertext since he/she doesn't know the signature key K that is used for HMAC-SHA-1.

3.2.3. MCS "Multiple Crypto Systems": To provide confidentiality, a hybrid encryption scheme for the encoded data is used. MCS scheme uses four Encryption/Decryption algorithms RC4, DES "Data Encryption Standard", IDEA "International Data Encryption Algorithm" and AES "Advanced Encryption Standard". All the gathered data about encryption algorithms are taken from [13, 14, 15, 16, 17, 18, and 19].

A hidden code during negotiation between nodes in MCSAuth mechanism will benefit in using only one of four encryption/decryption algorithms called CS "Crypto Slot". CS is 8-bit long. It uses 256 slots. Four algorithms are inserted in CS code in a random way where 2-bit codes discriminate between each algorithm and the other of the four algorithms instead of their name. The key K is used exclusively for both encryption/decryption algorithms and HMAC-SHA-1.

3.2.4. Shuffling Effect: To provide more complexity to the mechanism, diffusion takes place between the HMAC bits and the encrypted plaintext in a random way. The interspace between bits is controlled by the shuffling table. Shuffler node generates the shuffling tables randomly, a thing which causes much more confusion to any attacker.

HMAC bits are being scrambled with ciphertext bits to avoid MIC static replacement attack. An attacker will not be able to find either HMAC or ciphertext bits. Only authorized cloud users and cloud system nodes will be able to de-shuffle the diffused message back to its original form and detach the encrypted plaintext from the HMAC bits. Each shuffling row has random values generated from a random value generator.

SC "Shuffler Code" is a code added to the header. It is 4-bit long to select between 16 rows found in the shuffling table each with length N. The shuffling row is used in both shuffling and de-shuffling the HMAC bits inside the encrypted messages and back to their original place.

Figures 11 and 12 show the negotiation mechanism for any cloud user using SSO. Servers empower cloud user authentication to AS/TGS and Shuffler node. Requests are accepted from a registered user having a valid user ID, registration code, password and a valid ticket.

Shuffler node and AS/TGS are designated for handling a login request from a cloud user. A cloud user is authenticated and provided with a key that can be used to set up a secure channel with servers. Tickets, provided by TGS, are used to convince servers that the cloud user is the one who really he/she claims to be. Here, login is a process like typing a user ID and registration code at a workstation node that is found anywhere. Then user's ID and registration code are sent to AS/TGS and Shuffler node. Note that all the messages between nodes are encrypted during the transmission.

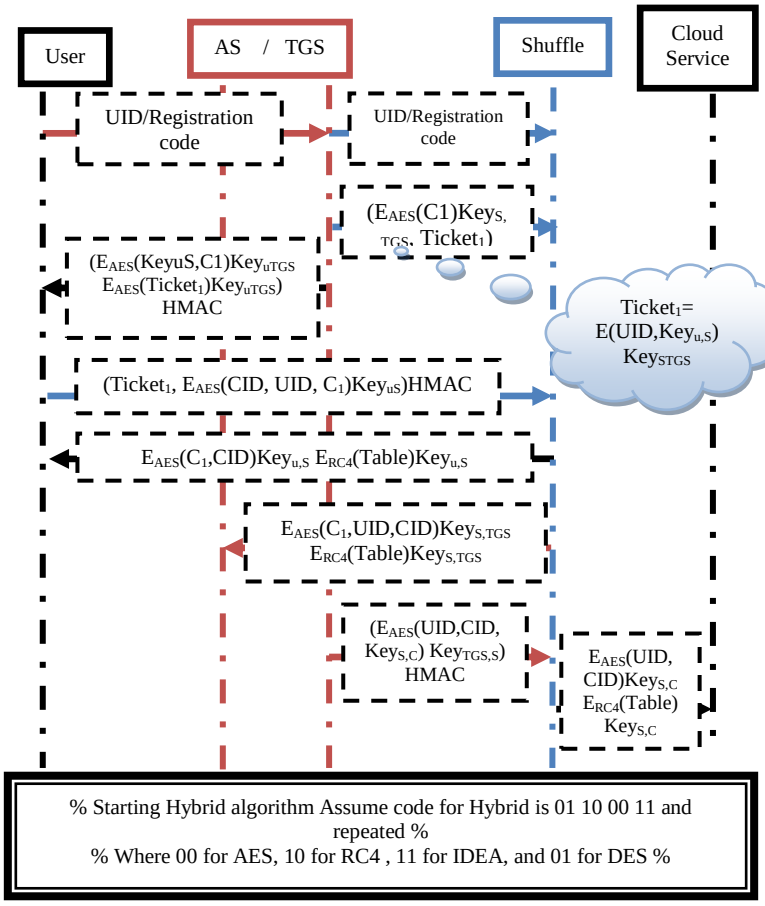

Fig 11: MCSAuth mechanism phase 1.

3.2.5. Encryption/Decryption stages:

3.2.5.1. Stage one (Default Stage): AES is the beginning stage. It is used at the beginning of negotiation as default which is found in phase 1.

3.2.5.2. Stage two (Middle Stage): RC4 with CTR "Counter"-mode is used in table transmission between shuffler node and the other nodes which are found in phase 1and 2.

3.2.5.3. Stage Three (Hybrid Stage): It is the hybrid stage where encryption schemes hop randomly between RC4, IDEA, 3-DES and AES through the negotiation between nodes which is found in phase 2.

TGS generates $ticket_1$ and transmits it to Shuffler node supported with keyed hash function HMAC-SHA-1. If the user logs with both valid user ID and registration code at the same time, TGS will transmit to the user the same $ticket_1$ that was transmitted to shuffler node. This registration code is used to generate the shared secret key $Key_{u,TGS}$ by taking a character string code and applying cryptographic hash MD5 and then taking 128-bit string values which will be the shared secret key $Key_{u,TGS}$.

After user decrypts the received message, he/she will be having $ticket_1$, $Key_{u,S}$, and C_1 ($Confounder_1$). Confounders are used as a type of challenge between nodes. Note that almost all negotiated messages are checked by comparing the received hashed function with the generated one to ensure integrity. User transmits the received $ticket_1$ and sends it with Cloud ID, token and C_1 encrypted by $Key_{u,S}$ towards shuffler node.

Shuffler node decrypts, checks integrity and compares the received $ticket_1$ and $confounder_1$ from user holding the old one which was received by TGS for that user. With all the previous checking passed, shuffler node generates a unique random table for that user and sends the table encrypted to user.

This table is used only for that user. So as to continue the rest of negotiation, this table must be found at the other nodes. Shuffler node sends the same table encrypted to AS/TGS and CSP for such user. Cloud user, AS/TGS and CSP decrypts the encrypted table by their shared secret keys $Key_{u,S}$, $Key_{S,AS/TGS}$, and $Key_{S,C}$ respectively. Now all the nodes have the same table for such user.

The hybrid stage also called phase 2, as shown in next figure, starts when the negotiations between nodes are encrypted randomly based on MCS code which was hidden and transmitted to all nodes through the shuffled tables.

TGS sends $ticket_2$ to AS. AS, in return, sends a shuffled message holding $ticket_2$ encrypted with the keyed hash function. AS prompts a password which will help in generating $Key_{u,AS}$, same as before like with registration code and $Key_{u,TGS}$.

This password is used to generate the shared secret key $Key_{u,AS}$ by applying it to MD5 and then taking 128-bit string values which will be the shared secret key, $Key_{u,AS}$.

User rearranges, decrypts message and checks its integrity. He/she now holds $ticket_2$. User sends $ticket_2$ back with Cloud ID and encrypted C_2 hashed then shuffled to TGS requesting the cloud service.

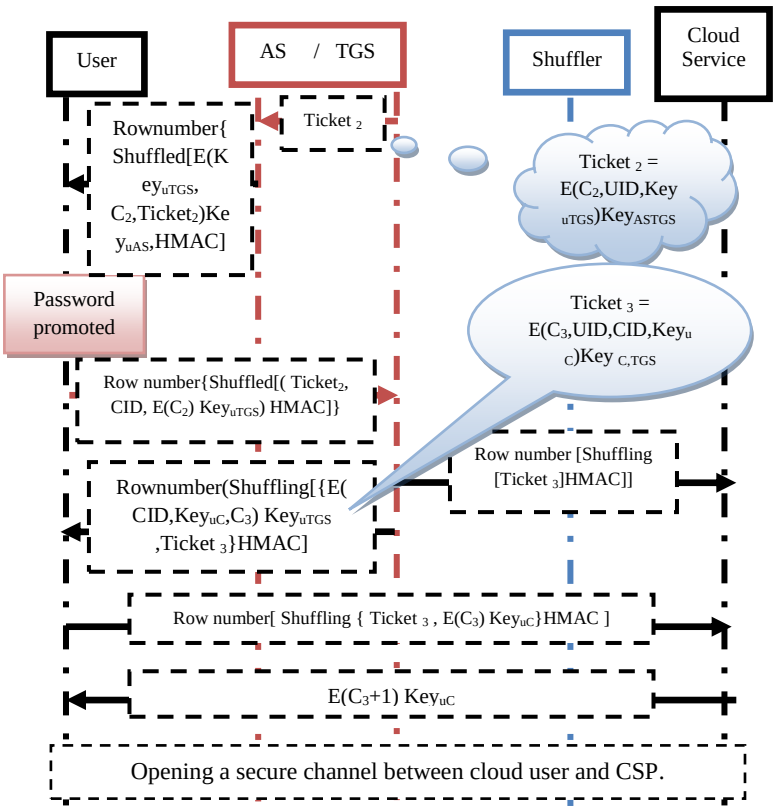

Fig 12: MCSAuth mechanism phase 2.

TGS rearranges, decrypts message and checks hashing. It compares $ticket_2$ received from user and the one held by it. If all passes, it transmits a request to CSP for that user holding $ticket_3$, Key $_{u,C}$ and C_3 to user.

User and CSP rearrange, decrypt received messages and check their integrity. User sends $ticket_3$ and C_3 to CSP encrypted by Key $_{u,C}$, CSP decrypts the message sent by user and compares C_3 and $ticket_3$ with the other received by TGS. If CSP comparing passes, it will open a secure communication channel with that user.

4. DISCUSSION AND RESULTS

Next section discusses MCSAuth test results which will be presented and discussed in detail compared with the previous proposed mechanism.

4.1.Testing and Results

The following is an explanation of experiments that were applied on MCSAuth and the previous proposed mechanism. The experiments were performed using a computer simulation; the simulation package that was used is Matlab simulink. In this section, we will report the security parameters results. First, we discuss some theoretical properties of the proposed mechanism, after which we discuss the actual experiments conducted.

4.1.1. Injected message attack result: An adversary will have no chance to attack, if the shuffling table is still not compromised. Assuming the shuffling had been compromised, an attacker doesn't know which one of encryption algorithms is being used, so MCSAuth mechanism is considered not vulnerable to injected message attack.

4.1.2. MIC vulnerability parameter result: The proposed mechanism uses HMAC-SHA-1 keyed hash function in addition to that; the hashed bits are defused inside the encrypted plaintext. Assuming that the shuffling had been compromised, the adversary will not know the key used for hashing. If the shuffling table is still not compromised, an attacker should try to overcome the shuffling effect by going through all feasible shuffling values that are equal to a factorial of N-bit length.

4.1.3. Brute force attack result: The value of brute force permutation concerning the proposed mechanism is equal to $\{(2^{RC4\ Key} + 2^{IDEA\ Key} + 2^{DES\ Key} + 2^{AES\ Key})\}*(\text{factorial } (N))\}$ for every single message transmitted between the nodes. While for the other mechanisms presented in this paper has a key permutation field equal to 2^{128}.

So, it is obvious that, compared with the time consuming brute force, the proposed mechanism MCSAuth is at least four multiplied by factorial N times longer than the other mechanisms. The enormous time consuming brute force attack for MCSAuth is due to the four encryption algorithms being used combined with the diffusion effect caused by shuffling the HMAC bits with the encrypted plaintext and this adds factorial N trials for each key.

4.1.4. Unauthenticated encryption results: Certain messages transmitted between nodes in MCSAuth mechanism contain confounders which work as challenges between nodes.

The idea of using confounders is to provide mutual authentication between nodes during negotiations through the encoded data prior to encryption to avoid man-in-the-middle attack and an unauthenticated encryption attack. In addition to, AS/TGS, Shuffler and cloud service nodes have the authority to ban any compromised nodes.

4.1.5. Encryption Oracle result: The proposed mechanism is no longer malleable due to the use of diffusion effect. Assuming that an attacker has compromised the shuffling table, if he/she tries to use chosen-plaintext attack, it can be easily recognized through the change of HMAC bits.

4.1.6. Oracle padding result: Two solutions are made in the proposed mechanism to avoid such attacks. First solution is by using random padding. The second one is by using hybrid encryption schemes and shuffling effect which are going to be an obstacle for an adversary.

4.1.7. Maintainability parameter test result: It is the ability of the mechanism to be capable of surviving after its encryption algorithm has been compromised and broken mathematically but still renders security. The proposed mechanism uses four hybrid encryption schemes and hops randomly between them. These are considered more maintainable than the other traditional mechanisms introduced in this paper. These use only one discrete encryption scheme. Assuming that the shuffling effect has been compromised and one of the four encryption schemes was broken mathematically, MCSAuth will still maintain its security.

4.2.Mechanism Metrics

The most measurable and encountered metrics of a mechanism are speed, memory, message overhead and bottleneck of the mechanism.

4.2.1. Speed: The computing time of the previous algorithm and MSCAuth are measured. Table 1, shows the results of the measured execution time for the previous mechanism and MCSAuth. For the previous mechanism, the execution time was measured using different encryption schemes like AES, DES, IDEA and RC4. The next table is a normalized table by the average output of AES used on the previous mechanism compared with the other encryption schemes and with the new proposed MCSAuth mechanism. The error margin was also calculated to represent the confidence interval; specifically the 95% confidence interval was calculated.

Mechanism cited in [9] for cloud system was implemented with different encryption schemes: RC4, IDEA, DES and AES. RC4 is the fastest one compared with other schemes. However, it has several weaknesses in the keys that it uses. IDEA is second best in speed after RC4. It is good but patented. DES is faster than AES, and is considered second best choice, but weaker than AES. AES is the best choice for securing a mechanism. It is strong but complex and takes long in execution compared to RC4, IDEA and DES.

Table 1, Normalized table for performance level comparing between modified Kerberos mechanism for cloud system and MCSAuth (using n= 45 and confidence level = 95%).

	Previous proposed protocol				New Proposed Protocol
	RC4	IDEA	DES	AES	MCS
Average	0.0123	0.0643	0.2569	1	6.1558
Error Margin	0.0020	0.0034	0.0042	0.0045	0.0402

Lower Bound	0.0103	0.0608	0.2526	0.9954	6.1156
Upper Bound	0.0143	0.0678	0.2611	1.0045	6.1961

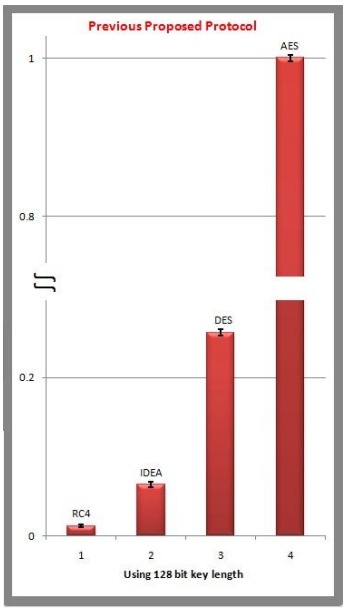

Fig 13: Performance level Compared with different encryption schemes (RC4, IDEA, DES, and AES) implemented on the modified Kerberos mechanism.

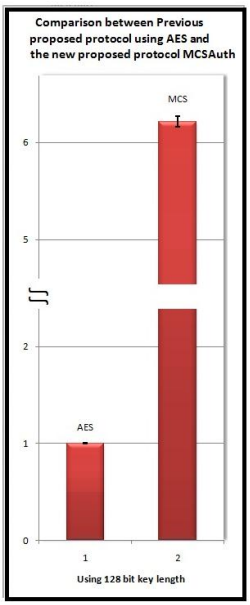

Fig 14: Performance level Compared to modified Kerberos mechanism using AES and the new proposed MCSAuth mechanism.

For MCSAuth, execution time took approximately six times that of AES used with the previous mechanism as shown in previous Fig. 14. The cause of this huge increase is due to more negotiation presented in MCSAuth and due to the use of three parameters: multiple crypto systems, shuffling tables and keyed hash function.

4.2.2. Memory Analysis: Table 2 is a comparison presented between the memory of the previous authentication mechanism found at each node and the new proposed mechanism MCSAuth.

The additional memory requirements of MCSAuth are due to the extra space for holding shuffling tables, the addition of more tickets, more shared secret keys between nodes presented, and user's token parameter. Table 2, the memory analysis comparison is made for only a single cloud user and a single cloud service provider, the same pattern is expected to hold as the system scales up.

4.2.3. Message overhead of a mechanism: In this experiment, we measured the message overhead of the previous mechanism and MSCAuth.

Tables 3 and 4, show the number of messages transmitted and received for each node in the previous mechanism and MCSAuth mechanism. Besides, the total length of messages transmitted and received in bit length is added. As we can observe, the new proposed authentication mechanism MCSAuth has a large number of messages plus their length increased dramatically compared to the previous mechanism, because several issues were added: First, we added a new node called shuffler node which caused more negotiations. Second, we used shuffling tables. Third, more tickets are used. Fourth, more shared secret keys are presented, and last the use of keyed hashing function HMAC-SHA-1 is applied. All the issues that were added to MCSAuth give more security strength.

Table 2, Memory analysis is compared between modified Kerberos and MCSAuth mechanism.

	Modified Kerberos		**MCSAuth**
Cloud User	168 bytes	Cloud User	7968 bytes
AS/TGS	192 bytes	AS/TGS	8016 bytes
		Shuffler node	7808 bytes
Cloud Service	96 bytes	Cloud Service	7824 bytes

Table 3, Message overhead for modified Kerberos

	Modified Kerberos			
	Messages transmitted	bits	Messages transmitted	bits
Cloud user	3	960	3	1024
AS/TGS	3	1280	3	576
Cloud service	1	128	2	768
Total	7	2377	7	2368

Table 4, Message overhead for MCSAuth.

	MCSAuth			
	Messages transmitted	bits	Messages transmitted	bits
Cloud User	4	2246	5	64006
AS/TGS	7	3785	3	62531
Shuffler Node	3	185216	4	1728
Cloud Service	1	128	3	63110
Total	15	191375	15	191375

4.2.4. Bottleneck of the mechanism: In this experiment, we attempted to identify the performance bottleneck of the mechanism. The purpose of this experiment is to identify an aspect of the algorithm that can be subjected to further optimization in order to improve performance. Next Table 5, Fig. 15 and Fig. 16 show the comparison of time taken during execution for each node between the previous mechanism and MCSAuth mechanism.

We can see that processing time for cloud user in MCSAuth mechanism increased dramatically. The main cause of this increase is mainly keyed hashing function HMAC-SHA-1 that is used for integrity check during negotiation between nodes. The previous test and simulated results proved that MCSAuth was able to counter most of the previously mentioned vulnerabilities.

Table 5, Comparing the execution time at each node for each mechanism.

Nodes	RC4	IDEA	3-DES	AES	Nodes	MCS Auth.
Cloud User	0.0588	0.2589	1.0295	4.9282	Cloud User	25.752
AS/ TGS	0.1322	0.5245	1.5576	6.2061	AS/ TGS	29.584
CSP	0.0267	0.1931	0.849	3.0995	Shuffler Node	20.261
					CSP	12.256

5. CONCLUSION

This thesis introduces and discusses a new authentication mechanism for cloud computing. Our design focuses on blocking any malicious attacks. This mechanism has, so far, proved to be effective against almost all security attacks.

MCSAuth was introduced in an effort to counter most of the known vulnerabilities that were found in a previously proposed mechanism. For the sake of clarity, a comparison is presented below between MCSAuth and the previously proposed mechanism. From table 6, MCSAuth has proved to be more resilient to attacks than the previously proposed mechanism. Yet it has some disadvantages that were evident after testing, and they are:

- MCSAuth needs a new node called shuffler node with powerful processing capabilities.
- Nowadays, there are many encryption schemes that can replace one or two of the chosen schemes to provide better results.
- All nodes especially AS/TGS and Shuffler nodes must have a considerable memory size, to be able to store a copy of tickets, shared keys, and shuffling tables.
- Time taken by MCSAuth to open a secure communication channel is long compared with the other mechanism. The cause of this problem is the use of HMAC-SHA-1 during negotiations between nodes.

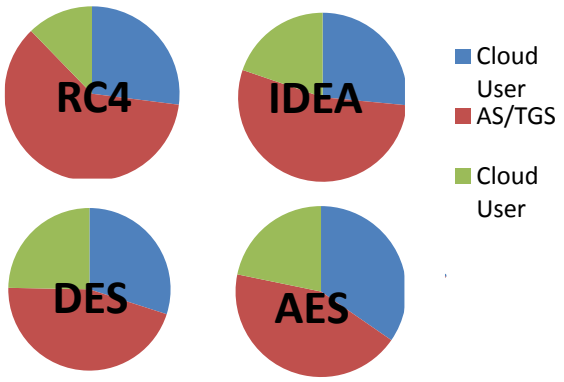

Fig 15: Bottleneck is compared with different encryption schemes used on the previous mechanism.

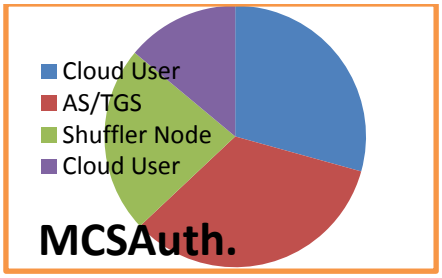

Fig 16: Bottleneck for MCSAuth

Table 6, Comparison between the previously proposed mechanism and MCSAuth.

Points	Modified Kerberos	MCSAuth
Time attack for a single packet using brute force Let all encryption schemes use same key size called K.	Let the brute force permutation of the previous proposed mechanism be called P $=2^K$ Permutation	Let the brute force permutation of MCSAuth be called F = $\{(2^K + 2^K + 2^K + 2^K)\}$*(factorial (N)) = (2^K)*(4)*(factorial (N)) = 4* factorial (N) of P.
MIC Problem	Not introduced as there is no MIC used.	Doesn't exist
Execution time	Best	Least
MIC	Not using	Best using HMAC-SHA-1
Malleability	Malleable	Non-Malleable
Unauthenticated Encryption	Exist	Doesn't Exist
Encryption Oracle	Exist	Doesn't Exist
Padding Oracle	Exist	Doesn't Exist

6. FUTURE WORK

Further research is needed to optimize our design model. In particular, further studies should apply the model to other security model standards once available, apply any new security attacks, once available, on the designed MCSAuth model, design software for AS/TGS and Shuffler nodes to enhance and speed up the execution time of MCSAuth, intrusion detection algorithms can be added to MCSAuth, which will help in finding and eliminating any intruders. Besides, an updated version of MCSAuth will use more encryption schemes. However, a foundation for such future research has been established.

7. REFERENCES

[1] Ramgovind, Eloff, and Smith, "The Management of security in Cloud Computing", Information Security for South Africa (ISSA), 2010.

[2] Gamal Ibrahim Selim, Sherif Fadel Fahmy, and François Samir Nicola, "A New Authentication Mechanism for Cloud Systems", Helwan University Engineering Journal 138, Mars 2013.

[3] Anthony Velte, Toby Velte, and Robert Elsenpeter, "Cloud Computing, A Practical Approach", McGraw Hill, 2012.

[4] Bhaskar Prasad Rimal, Eunmi Choi, and Jan Lumb, "A Taxonomy and survey of Cloud Computing System", INC, IMS and IDC, 2009.

[5] "Cloud Computing Alliance V2.1", web site: http: // www. cloud security alliance. Org /.

[6] Bradley, Tony and Carvey, Harlan, "Essential Computer Security", Elsevier Science Ltd, 2006.

[7] James Haskett, "Pass-algorithms: a user validation scheme based on knowledge of secret algorithms", communications of the ASM, 27, No 8, PP 777 – 781, (1984).

[8] Anne Adams, Martina Angela Sasse and Peter Lunt: "Making passwords secure and usable", HCI 97 Proceedings of HCI on People and Computers XII, Springer-Verlag London. 1997.

[9] Mahbub Ahmed, Yang Xiang and Shawkat Ali, "Above the Trust and Security in Cloud Computing: A Notion towards Innovation", Embedded and Ubiquitous Computing (EUC), IEEE/IFIP 8th, 2010.

[10] Charles Paul Pfleeger and Shari Lawrence Pfleeger, "Security in Computing", Third Edition Prentice Hall Professional, 2003.

[11] Andrew Tanenbaum and Maarten Van Steen, "Distributed Systems: Principles and Paradigms", Second Edition Pearson Prantice Hall, 2007.

[12] Charlie Kaufman, Radia Perlman and Mike Speciner, "Network Security: private communication in a public world", Prentice Hall Press Upper Saddle River, NJ, USA, 2002.

[13] William Stallings, "Cryptography and Network Security Principles and Practices", Fourth Edition, Prentice Hall, 2005.

[14] Paul Garrett, "Making, Breaking Codes: An Introduction to Cryptology". Upper Saddle River, NJ, Prentice Hall, 2001.

[15] Doug Stinson, "Cryptography: Theory and Practice" Boca Raton, FL: CRC Press, 2005.

[16] Susan Landau, "Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard." American Mathematical Monthly, 2004.

[17] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, "Report on the Development of the Advanced Encryption Standard", National Institute of Standards and Technology, 2000.

[18] William Stallings, "The Advanced Encryption Standard." Cryptologia, 2002.

[19] Pardeep, Pushpendra Kumar Pateriya, "PC-RC4 Algorithm: An Enhancement Over Standard RC4 Algorithm" , International Journal of Computer Science and Network (IJCSN) Volume 1, Issue 3, 2012.