Indian Journal of Science and Technology, *Vol 7(1), 19–24, January 2014*

A New Group Key Transfer Protocol using CBU Hash Function

Mohit Kumar[1*] and S. P. Ghrera[2]

[1]Asst. Prof. Department of Computer Science and Engineering, Cambridge Institute of Technology, Tatisilwai, Ranchi (Jharkhand), India; mohitsmailbox13@gmail.com

[2]Prof & Head Department of Computer Science and Information Technology, Jaypee University of Information Technology, Waknaghat, Distt. Solan, (H.P), India; sp.ghrera@juit.ac.in

Abstract

In this paper we have proposed and successfully implemented a new group key transfer protocol based on CBU hash function. The proposed scheme relies on mutually trusted Key Distribution Center (KDC) to generate and distribute session keys to all communicating entities secretly. In this scheme the key information is broadcasted at once to all the participating entities, but with the available information only the authorized user will be able to recover the actual session key. The advantage of this protocol lies in the fact that even if the pre-shared master key gets compromised, the attacker will still will not be able to recover the actual session key. Furthermore, our protocol makes use of CBU hash function along with Advanced Encryption Standards (AES) to provide confidentiality.

Keywords: Group Key Transfer Protocol, CBU Hash Function, Key Distribution Centre (KDC), Advance Encryption Standards (AES), Confidentiality.

1. Introduction

Cryptography can be used as a security mechanism to provide confidentiality, integrity, and authentication, but not if the keys are compromised in any way. There are lots of threats that are associated with the keys. They can be captured, modified, or corrupted. Hence we can say that key management is the most challenging part of cryptography. Now-a-days group oriented applications have become very popular, so securing group communication have become very important task. A group key is needed when a secure communication is taking place between more than two entities (group members). Group key should be transferred to group members in a secure and authenticated manner to achieve secure group communication[1–3]. There are various ways in which a group key is transferred but they can be broadly classified into three categories[4]:-

- Centralized Group Key Management protocol (CGKM)-In this protocol, a group key generation center manages the entire group. In CGKM there is a central trusted authority that is engaged in managing the entire group[5].
- Distributed Group Key Management protocol (DGKM)–In this scheme, group is divided further into smaller sub-groups and for each sub-group there is a sub-group controller that is engaged in managing the key management of that particular group[4].
- Contributory Group Key Agreement protocol (CGKA)-In this scheme each group member contributes to the key generation and distribution. In CGKA every

*Author for correspondence:

group member is treated equally and the workload among the group members are distributed evenly.

The class of centralized group key management protocol is most widely used key management protocol. Most of the group key transfer protocol relies on mutually trusted Key Generation Center (KGC) to select session keys, and then transport session keys to all communicating entities secretly. Most often KGC encrypts session keys under another secret key (master's key) shared with each entity during registration[2, 5]. The problem with this kind of approach is that a unique master key is required to be pre-shared between KGC and each group members and if the master's key is captured during initial distribution then whole communication will become unsecured. Harn and Changlu[6] proposed an authenticated group key transfer protocol based on secret sharing. A secret sharing scheme is a method which distributes shares of secret to a set of participants in such a way that only specified group of participants can reconstruct the secret by polling their secret. But the problem with this scheme is that even if one part of secret is compromised or corrupted, then the session key will not be recovered at any cost[4, 5, 7–9]. In this paper, we have proposed a new group key transfer protocol. The security of session key in our protocol doesn't solely depend upon the pre-shared master key. The strength of this protocol lies in the fact that even if the master key gets compromised the attacker will still not be able to recover the session key. In this protocol the encrypted session key is transferred to all the group members at once, but only those members (authorized) who also have the group key information will be able to recover the session key correctly. Another advantage of using this protocol over other protocols is that the actual session key is not send as cipher text, it is send as simple plain text. So any eavesdropper, who is continuously monitoring the channel, will not get suspicious about the message being passed.

2. Proposed Work

Key management is an important service for providing secure communication when the network is large, or the topology is undergoing frequent changes[10]. In most of the key transfer protocol, pre-sharing of key is required between the KDC and the group members. If any group member wants to establish a communication with another group member, then it has to first send a request to KDC for a session key. On receiving such requests the KDC selects a random session key and sends it to the group member (initiator) by encrypting the session key with the pre-shared key. In our protocol, the confidentiality of group key is ensured by using CBU hash function and AES and the security of our protocol does not solely depends upon the pre-shared key between the KDC and the group members. We have used CBU hash function that is a one-way hash function[7]. It has the advantage that it is fast to compute and it works well for an ASCII character string domain. In this approach each user is required to register first at KDC to subscribe to group key transfer service. As soon as all the information about the node gets registered in the KDC, it generates a set of secret session keys for each user. These session keys get stored in the database of the KDC. When the KDC receives any request for session key it randomly selects a session key from its database and gives it to the initiator and responder. The responder can recover the actual session key by using the group key information, which will be provided by the KDC. Our protocol works in the following steps:-

2.1 Registration

Each user first registers its information with the KDC, and then only they can participate in communication. The KDC generates a set of session keys on registration of each user. KDC initialises the keys.

2.2 Request for Session Keys

A (initiator) issues a request to the KDC for session key to protect a logical connection to B and C (responder). The message includes the name of A merged with B and C (ABC). The KDC then first checks whether the request has come from a legitimate user or not.

2.3 Response

The KDC responds by broadcasting the session key to both the initiator, and the responder. Here, the KDC randomly selects one of the session key from all the keys and makes it the session key for that particular session. The session key sends to A, B and C in encrypted form, and these group members can recover the actual session key by making use of the information provided by the KDC.

2.3.1 Process of Encryption of the Session Key

Figure 1 shows how the session key is encrypted. The session key is randomly selected from list of session keys by the KDC. Once the KDC gets to know which nodes will be participating in the communication, it takes that value as input for hash. By using CBU hash function on this value, we will get a hash value of fixed size. How a CBU hash function works is explained below.

```
Unsigned long PROC cbuhash (string key);
Unsigned long h = 0;
For I from 0 to size(Key) – 1
h = h<<2 + key[i];
return h;
```

Then, this hash value is merged with the session key. The merged data goes through AES encryption. After that we will get a cipher text. Subsequently, this cipher text passes through Base64 Encoder to convert the cipher text into some textual data. This text is merged again with the hash value, and the whole process is repeated. After performing the second iteration, we will get the encrypted session key, and this key is then broadcasted to initiator as well as the responders. Figure 1 shows the encryption process of session key.

2.4 Encryption

The initiator then encrypts the file content using the actual session key. Following this, the initiator broadcasts the encrypted file to the responders.

2.5 Authorized/Unauthorized

On receiving the encrypted session key, the responders sends a request to the KDC to send the group key information (Input for hash) so that it can recover the actual session key with the help of that information. The KDC first checks whether the request is coming from a legitimate user or not by checking the input for hash value that was ABC. Since B is present in it, the KDC will authenticate B and will give this value to B.

2.6 Decryption Process of Session Key

At the responders end, the normal plain text is first decoded to a cipher text by using the Base64 Decoder. Afterwards, we will get a cipher text. Then after performing the AES decryption algorithm on it, we get a value that is a combination of hash value and plain text. The responders then by using CBU hash function will get the hash value used and can then separate it from the whole value to get the plain text. The whole process is repeated, and finally the receivers will get the original session key used. Figure 2 shows the decryption process of the session keys.

2.7 Decryption

Once the responders gets this session key, they will easily decrypt the encrypted session key by using this session key.

3. Conclusion

In this paper we proposed an efficient group key transfer protocol based on CBU hash function. The proposed protocol is simple, robust, secure, and scalable. Our scheme improves the security of key management, as its security doesn't depend solely upon the pre-shared master key with the KDC. The beauty of this protocol lies in the fact that KDC uses broadcast channel for distributing the session keys, and only the authorized receivers alone are able to recover the session keys with the help of key information, they received from KDC. Here, backward and forward secrecy is maintained by KDC, and confidentiality is provided by using CBU hash function. We

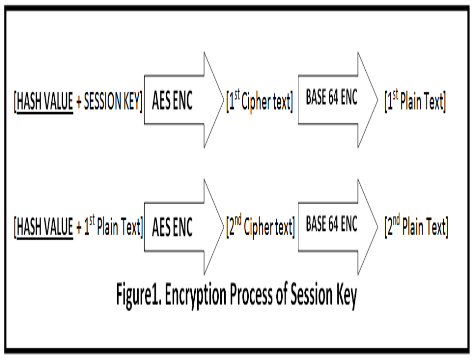


Figure 1. Encryption process of session key.

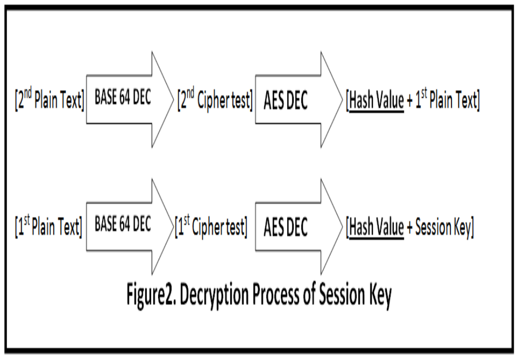
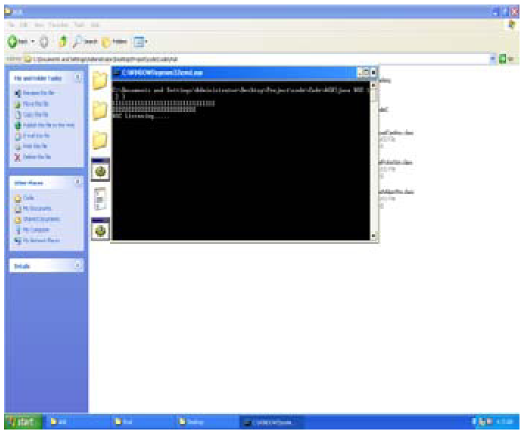
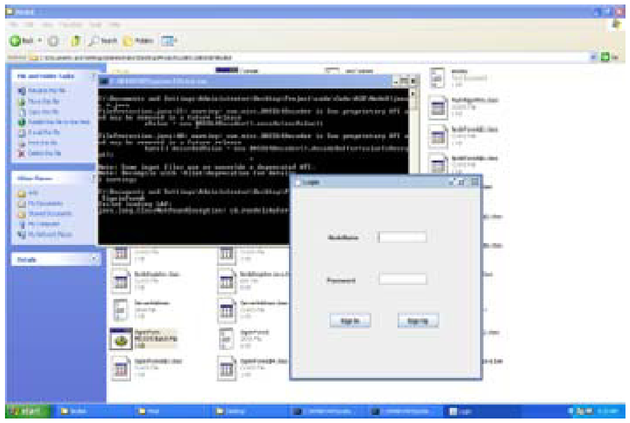


Figure 2. Decryption process of session key.

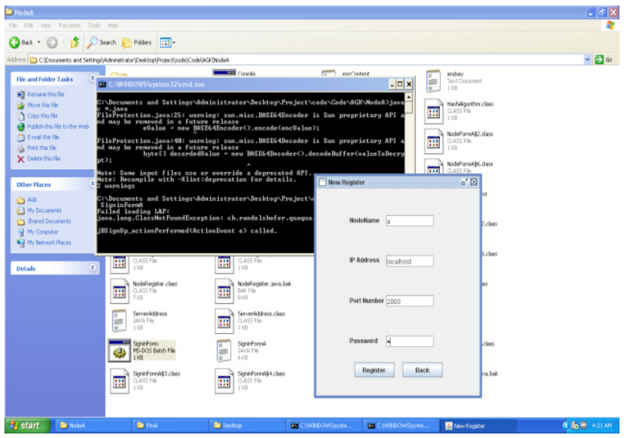


```
KGC Listening.....
```

Snapshot-1. KGC stats listening.



Snapshot-2. Login form for existing user.



```
C:\Documents and Settings\Administrator\Desktop\Project\code\Code\AGK\NodeA>java
c *.java
FileProtection.java:25: warning: sun.misc.BASE64Encoder is Sun proprietary API a
nd may be removed in a future release
            eValue = new BASE64Encoder().encode(encValue);
                         ^
FileProtection.java:40: warning: sun.misc.BASE64Decoder is Sun proprietary API a
nd may be removed in a future release
            byte[] decordedValue = new BASE64Decoder().decodeBuffer(valueToDecry
pt);
                                                    ^
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2 warnings

C:\Documents and Settings\Administrator\Desktop\Project\c
 SigninFormA
Failed loading L&F:
java.lang.ClassNotFoundException: ch.randelshofer.quaqua.

jBSignUp_actionPerformed(ActionEvent e) called.
```

New Register

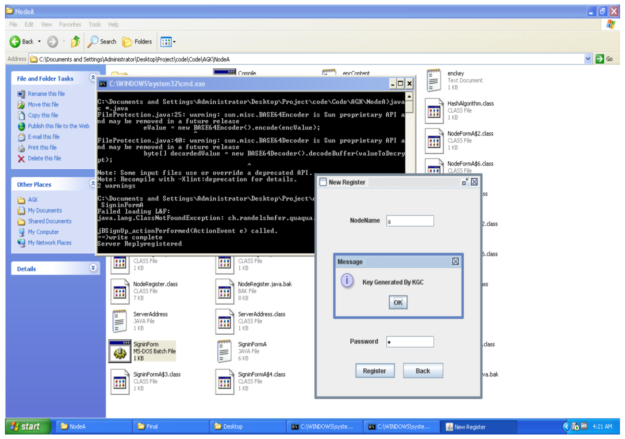
NodeName: a

IP Address: localhost

Port Number: 2000

Password: •

Snapshot-3. Node register form for new user.



```
C:\Documents and Settings\Administrator\Desktop\Project\code\Code\AGK\NodeA>java
c *.java
FileProtection.java:25: warning: sun.misc.BASE64Encoder is Sun proprietary API a
nd may be removed in a future release
            eValue = new BASE64Encoder().encode(encValue);
                         ^
FileProtection.java:40: warning: sun.misc.BASE64Decoder is Sun proprietary API a
nd may be removed in a future release
            byte[] decordedValue = new BASE64Decoder().decodeBuffer(valueToDecry
pt);
                                                    ^
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2 warnings

C:\Documents and Settings\Administrator\Desktop\Project\c
 SigninFormA
Failed loading L&F:
java.lang.ClassNotFoundException: ch.randelshofer.quaqua.

jBSignUp_actionPerformed(ActionEvent e) called.
==>write complete
Server Replyregistered
```

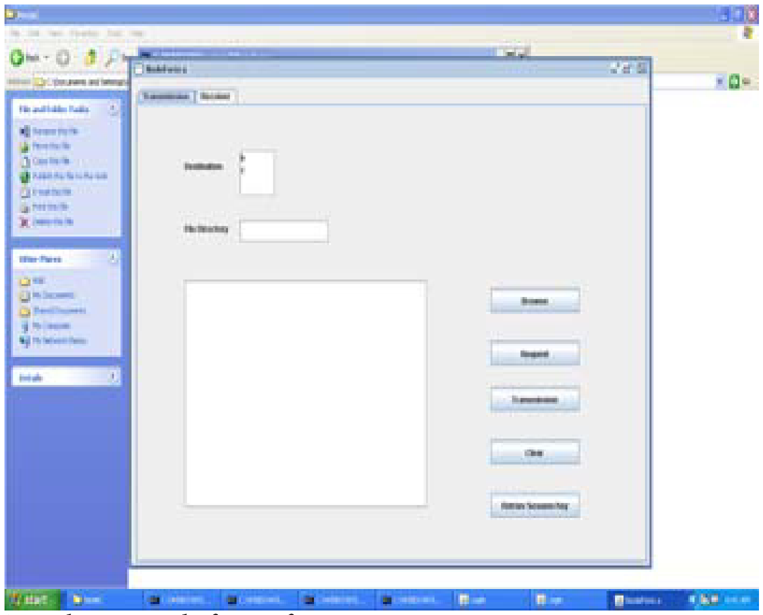
New Register

NodeName: a

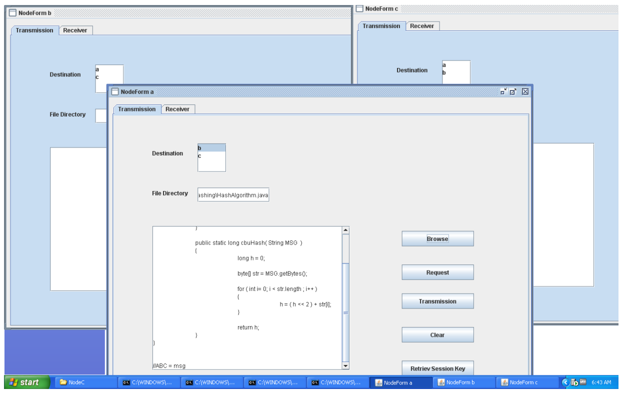
Message: Key Generated By KGC

Password: •

Snapshot-4. Session key generated for new user.



Snapshot-5. Node form of user A.

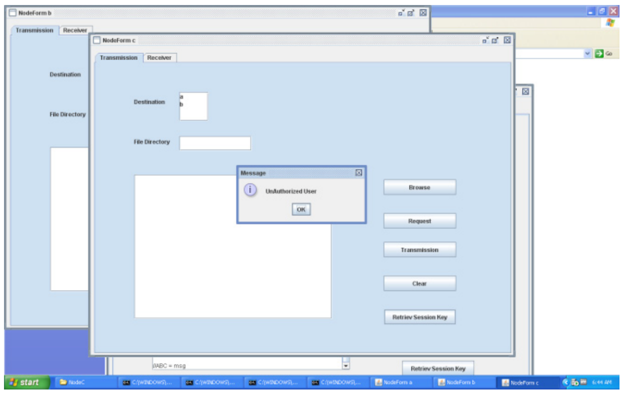


Destination: b, c

File Directory: ashing\HashAlgorithm.java

```
    }

    public static long cbuHash( String MSG  )
    {
            long h = 0;

            byte[] str = MSG.getBytes();

            for ( int i= 0; i < str.length ; i++ )
            {
                    h = ( h << 2 ) + str[i];
            }

            return h;
    }
}

//ABC = msg
```

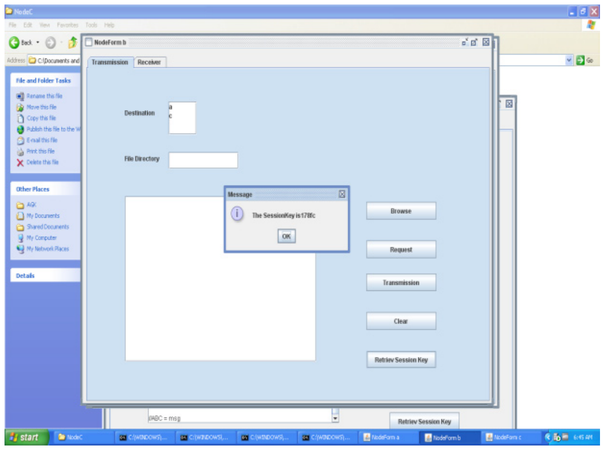
Snapshot-6. User A selects the destination.



Destination: a, b

Unauthorized User

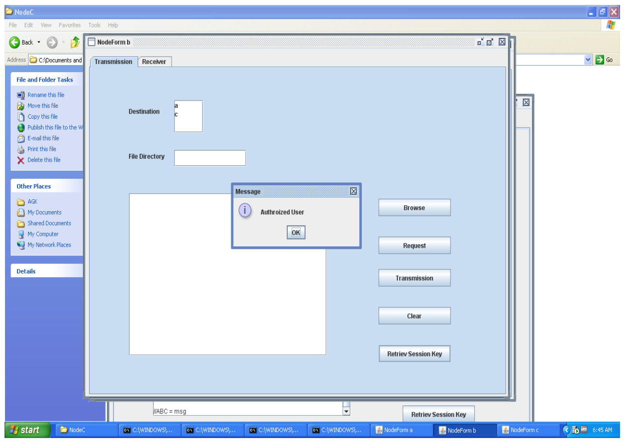
Snapshot-7. If an unauthorized user tries to access the message it gets an error message(Unauthorized user).



Destination: a, c

The SessionKey is 178fc

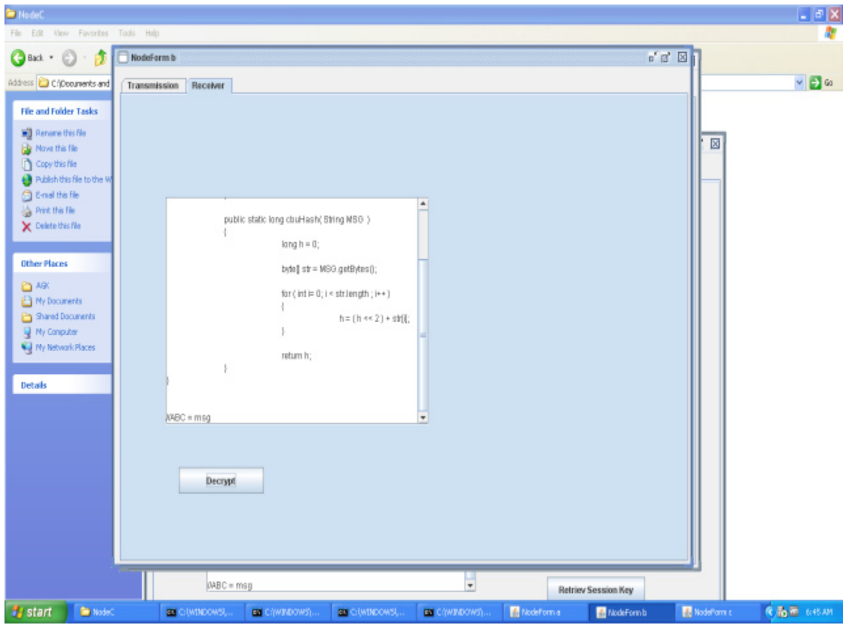
Snapshot-8. Authorized user.



Destination: a, c

Authroized User

Snapshot-9. Session key.



```
    public static long cbuHash( String MSG  )
    {
            long h = 0;

            byte[] str = MSG.getBytes();

            for ( int i= 0; i < str.length ; i++ )
            {
                    h = ( h << 2 ) + str[i];
            }

            return h;
    }
}

//ABC = msg
```

Snapshot-10. Decrypted message.

have also successfully implemented this protocol. The implementation can be seen in the snapshots from [1–10].

4. References

1. Blakely GR. Safeguarding cryptographic keys. Proc Am Federation of Information Processing Soc (AFIPS '79). Nat'l Computer Conf. 1979; 48:313–317.
2. Berkovits S. How to broadcast a secret. Advances in Cryptology–CRYPTO '91. Lecture Notes in Computer Science. 1991; 547:536–541.
3. Stalling W. Cryptography and network security. 4th ed.; 2005.
4. Adusumilli P, Zou X, Ramamurthy B. DGKD: Distributed Group Key Distribution with authentication capability. Workshop on Information assurance and Security. IEEE; 2005 Jun.
5. Zhang Q, Wang Y. A centralized key management scheme for hierarchical access control. IEEE Communication Society. 2004; 4.
6. Harn L, Lin C. Authenticated group key transfer protocol based on secret sharing. IEEE Transactions on Computers. 2010 Jun; 59(6).
7. Sherman AT, McGrew DA. Key establishment in large dynamic groups using one-way function trees. IEEE Transactions on Software Engineering. 2003 May; 29(5) : 444–458.
8. Song R, Korba L, Yee GOM. A scalable group key management protocol. IEEE Communication Letters. 2008 July; 12(7).
9. Chang C-C , Lin C-H, Chen C-Y. A conference key distribution scheme using interpolating polynomials. IEEE International Conference on Multi-media and Ubiquitous Engineering; 2007; 1(2):963–967.
10. Metal M. A new secure multi-cast key distribution protocol using combinatorial boolean approach. Int J Netw Secur. 2009 Jan; 8(1).