Implementation and Performance of Threshold Cryptography for Multiple Escrow Agents in VoIP

Abdullah Azfar

Department of Computer Science and Information Technology (CIT),
Islamic University of Technology (IUT), Board Bazar, Gazipur 1704, Bangladesh
azfar@iut-dhaka.edu

Abstract. This paper focuses on improving the accessibility and security of multiple escrow agents by dividing the session master key into M chunks and escrowing the chunks with M escrow agents. Using threshold cryptography the key can be regenerated by gathering any N-out-of-M chunks. This N-out-of-M approach increases the security of the session master key as at least N chunks of the session key are needed to regenerate the session key. Disclosure of less than N chunks does not threat the security of the session key. On the other hand, failure of a single escrow agent does not affect the availability of the session key as long as N escrow agents are working. For a highly sophisticated session, the user might define a higher value for M and N. For a less confidential or less important session, the value of M and N might be smaller.

Keywords: Key escrow, VoIP, Escrow Agents, Threshold Cryptography, Shamir's Secret Sharing.

1 Introduction

An escrow agent is a trusted third party (TTP) with whom users store their session master key. The term key escrow refers to storing the cryptographic key with a TTP or escrow agent [1]. Using a Key escrow agent in conjunction with Voice over IP (VoIP) communication ensures that law enforcements agencies (LEAs) can retrieve the session key used to encrypt data between two users in a VoIP session. However, the use of a single escrow agent has some drawbacks. A fraudulent request by an evil employee from the LEA can lead to improper disclosure of a session key. The problem with a single escrow agent becomes even more critical as a failure of the escrow agent can delay or even make it impossible to reveal the session key.With threshold cryptography [2] approach, the session key is divided into several parts and each part is stored in different escrow agents. For (N, M) threshold cryptography, there will be M escrow agents with each escrow agent stores their own secret part. If at least N escrow agents reveal their secret parts, then the session key can be generated. The secret shares in threshold cryptography do not have any explicit relation with each other. If one escrow agent is compromised by an attacker, only the secret portion stored by that escrow agent is revealed. The complete key can be

V.V. Das, E. Ariwa, and S.B. Rahayu (Eds.): SPIT 2011, LNICST 62, pp. 143–150, 2012.

generated only if N escrow agents' keys are compromised. Additionally, if one or more escrow agents are not functioning, the key can still be generated as long as N escrow agents are functioning. This gives extra reliability for key retrieval.In this paper, we have split the session key of a VoIP session into five chunks and escrowed them to five escrow agents. The threshold value is set to three as the session key can be regenerated by combining any of the three chunks out of the five chunks. This is the first approach of threshold cryptography in the context of the escrowing of VoIP session key.The rest of the paper is organized as follows: related works are reviewed in section 2. The design and implementation issues of splitting the session key are discussed in section 3. The performance measurements and discussion are done in section 4. Finally, some conclusions are drawn in section 5.

2 Related Works

RomanidisEvripidis at the Royal Institute of Technology (KTH) addressed the issues of key escrow [3]. The escrow agent stores the session key and some related data. The LEA is assumed to have recorded a communication session between two users. The escrow agent reveals the key to the LEA upon a legal request from the LEA. He pointed out that there is a problem when using a single escrow agent as any evil person either in the LEA or in the escrow agent can reveal the session key.The problem of forgery by a single escrow agent can be overcome by signing the hashes of the data with the user's private key and storing the final hash with the escrow agent. This proposed solution has been implemented by Md. SakhawatHossen at the Royal Institute of Technology (KTH) [4]. Md. SarwarJahanMorshed addressed the issues of a LEA when retrieving a session key and performing decryption of a captured session [5].

3 Design and Implementation Issues

As we are interested in splitting the key into M chunks and then retrieve the key from N-out-f-M chunks, a suitable algorithm for this would be Shamir's Secret Sharing Algorithm [7][7]. Shamir's secret sharing is an N-out-of-M threshold scheme based on polynomial interpolation. At least N participants must provide their shares in order to decrypt the secret. Shamir's Secret Sharing algorithm is scalable because the number of chunks can be changed. As a result, it is possible to increase or decrease the value of M or N for an implementation.We have used Minisip [8] as our user agent. Minisip is an open source Voice over Internet Protocol (VoIP) user agent (UA). It is based on the Session Initiation Protocol (SIP) [9] and special security features are included in it. The user agent is responsible for splitting the keys. The escrow agents will only store the escrowed information. The splitting operation is performed in the user agent and the user agent stores the values in files. Upon successful connection with the escrow agent, the user agent escrows the split values. This is shown in Fig. 1.

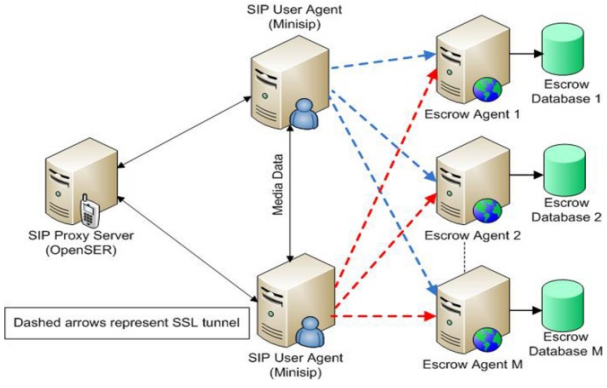

Fig. 1. General architecture of M Escrow agents

According to the thesis of Hossen [4] and Morshed [5], they have escrowed the session master key, i.e., the Traffic Encryption Key (TEK) Generation Key (TGK) along with the pseudo-random number (Rand), last signed hash and Crypto Session Bundle (CSB) ID value. This key is exchanged by the key agreement protocol MIKEY [10]. This TGK along with some security parameters are used to generate the session keys for encryption and integrity protection.In our case, we have extended the escrow operation from one escrow agent to multiple escrow agents. Thus, the parameters being escrowed remained the same. But question arises which parameters should be split and then escrowed. The reason behind splitting the key is to enhance security and availability of the key. Splitting the Rand, last signed hash and CSB ID value enhances the security, but it is not really necessary to do so. If we rather only split the TGK and replicate the Rand, last signed hash and CSB ID value to M escrow agents, then our purpose is served. As this implementation is an N-out-of-M system, no one can retrieve the TGK value without having at least N chunks. This increases the security and confidentiality. Again, the availability increases as even if few escrow agents are not working, the key is retrievable from N escrow agents.

3.1 Implementation of Split Operation

The procedure of splitting the key into chunks and escrowing them can be depicted by a general algorithm. The algorithm works according to the following steps:

Step 1: Create five files for temporarily storing the key chunks.

Step 2: Divide the TGK into two parts.

Step 3: Invoke the split function for each of the parts.

Step 4: Each part is split into five subparts. Store each of these split parts into the files created in step 1. The five subparts created from the first part will be the first entries in the five files. Five subparts created from the second part will be the next entry in the files. Separate them with a separator "%".

Step 5: Create a temporary string with the Rand, signed hash, CSB ID value and the names of the escrow agents. Separate them with "%" symbol.

Step 6: Read the contents of the first file created in step 1 into a string.

Step 7: Form a string with the IP address of the escrow agent.

Step 8: Append the user id, password and strings created in steps 5 and 6 to the string created in step 7.

Step 9: Escrow the parameters by creating a curl object with the string formed in step 8.

Step 10: Repeat steps 6 to 9 four times, read values from different files each time in step 6.

There are 256 bytes in the base 64 value of the TGK. These 256 bytes are divided into 2 equal (128 bytes each) parts. The split function is called for each of these parts and each part is divided into 5 chunks by the split function according to Shamir's Secret Share algorithm. Each of these 5 parts is written into the 5 files. At the end of the split operation, we get 5 files. Each of the files contains 2 subparts of the TGK. These 2 parts are separated by a "%" symbol.An URL is formed with the IP address of the first escrow agent, user name and password of the user appended by the contents of the first file. The signed hash value, rand value, CSB ID values are appended into the URL separated by a "%" symbol without any modification. We have used secure HTTP (HTTPS) to escrow the session master key. The key is transferred along with the URL of the escrow agent by appending a key value pair in addition to the key value pairs used to provide the user name and password for authentication to the escrow agent. To escrow the session master key with the escrow agent from the user agent we have used libcurl [11]. We have used third party code written by B. Poettering [12] for splitting the key licensed under the GNU General Public License [13]. We have modified the code according to our need to integrate with Minisip.

3.2 Implementation of Combine Operation

This subsection discusses about the general approach of how we have designed our system to retrieve the key chunks from the escrow agents and combine them in order to get the TGK. The general algorithm is as follows:

Step 1: Login to the escrow agent by providing user id and password by a web based form.

Step 2: Provide the target user id, start time and end time of target session for which the session key has to be retrieved.

Step 3: For authenticated user, read the first escrow agent database and fetch the two parts of the split TGK, Rand, CSB ID value, last signed hash value and names of the escrow agents.

Step 4: Write the values read in step 3 into a temporary file. Separate the values by a "%" symbol.

Step 5: Repeat step 3 and 4 for four other escrow agents. For each escrow agent, write the retrieved values in separate files.

Step 6: Read any three files until the first separator "%" is found.

Step 7: Invoke the combining function with the values fetched in step 6. At theend of this step we get the first half of the TGK.

Step 8: Read the same three files read in step 6 starting after the first "%" symbol until the next "%" symbol is found.

Step 9: Invoke the combining function with the values fetched in step 8. At the end of this step we get the second half of the TGK.

Step 10: Merge the two halves of the TGK to get the session key.

A php script has been written in order to fetch the information from the escrow agents. The php script runs five times and invokes the escrow agents one after another. Upon successful invocation on the escrow agents the script fetches the two split parts of the TGK, Rand, CSB ID, the signed hash value and the names of the escrow agents and writes them into files separated by a "%" symbol. After invoking five escrow agents, the LEA is provided with five text files. This text files are stored and are inputs to the combining function.

4 Performance Measurements and Discussion

For the purpose of measuring the performance of the escrow operations we used the experimental setup as shown in fig. 2.

Fig. 2. Experimental setup for escrow operations

We have used one machine as SIP user agent and another machine as the SIP proxy server and other user agent. We have used a third machine configured as escrow agents along with escrow databases. The escrow agents machine was connected to the user agent via SSL tunnel. The escrow agents machine had five databases and five escrow agents defined in it. We have used SuSE version10.3 of Linux,on DellT7570with Intel® Pentium® D CPU processor clocked at 2.80GHz and configured with2048MB of memory as the SIP user agents and proxy server.The base 64 TGK value consists of 256 bytes. We have divided this value into two equal halves. Then we executed the split function ten times for each of the halves and each time the split function was executed hundred times. In a normal execution, the split function is called twice, once for the first half of the TGK, and again for the remaining half of the TGK value. In our experiment, we executed the split function one hundred times for the same TGK value. The procedure was repeated ten times with ten different TGK values. As a result we have ten measurements for each of these hundred calls.Fig. 3 shows a box plot of the measured execution time of the split function for the first half of the base 64 TGK value. The reason for this large number of outliers in round 2 is unknown. But we can make some assumptions based on the

following facts: The time to compute the split is data dependent - i.e., for different values of a key it takes different amounts of time, the computer is multitasking - thus the CPU is being allocated to other processes, and the computer is also servicing interrupts from various devices.

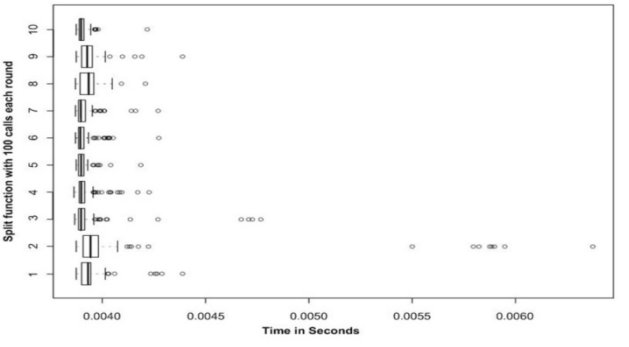

Fig. 3. Execution time for split function applied to the first 128 bytes of the TGK

Fig. 4 shows the box plot for the execution time of the split function for the second half of the TGK value encoded in base 64 (i.e., the last 128 bytes). The scale for this figure is quite different for the previous figure, due to the one extreme outlier at 0.1 seconds in round 10.

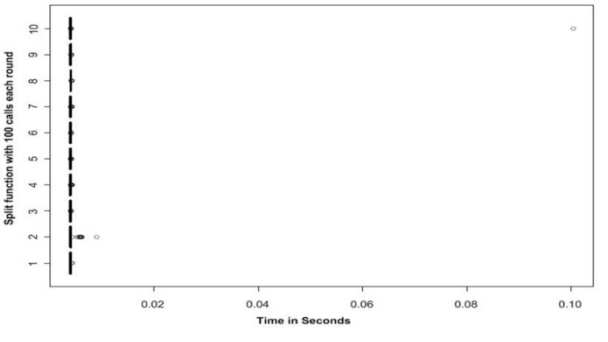

Fig. 4. Execution time for split function applied to the last 128 bytes of the TGK

Table 1 shows the statistical data found from the time required to split the first 128 bytes and last 128 bytes of the base 64 TGK value using 1,000 calls of the split function. Here, the unit of time is seconds.

Some observations can be made from table 1. First, the mean time to execute the split function for the first half of TGK encoded in base 64 is 0.0039 seconds and the mean time to execute the split function for the second half of the TGK encoded in base 64 is 0.004 seconds. So, the mean time to execute the split function for escrowing the TGK encoded in base 64 is the sum of these two values i.e. 0.0079

Table 1. Statistical data to split two halves of TGK encoded in base 64

	First 128 bytes	Last 128 bytes
Count	1000	1000
Mean	0.0039	0.004
Median	0.0039	0.0039
Mode	0.0038	0.0038
Standard Deviation	0.00019	0.003
Sample Variance	3.72056E-08	9.38E-06
Minimum	0.0038	0.0038
Maximum	0.0063	0.1
Confidence Level (95.0%)	3.82489E-07	6.07E-06

seconds or 7.9 milliseconds. The next observation is that the median values are identical in both experiments. The third and final observation is that the minimum time required to execute the split operation is same for both experiments. This means that the time to split the original 256 byte based 64 encoded TGK is 2 * 0.0038 seconds or 7.6 milliseconds.

5 Conclusions

This paper focused on a proposal, implementation, and evaluation of a multiple key escrow agent model that allows escrowing the session keys into M escrow agents. Shamir's secret sharing algorithm was used to implement threshold cryptography. This is the first approach of using Shamir's secret sharing algorithm for dividing the session key of a VoIP session. The session key was divided into M chunks and each of the chunks was escrowed to the M escrow agents. An N-out-of-M key retrieval mechanism was implemented. This allowed to retrieve the key by retrieving at least N chunks out of M chunks where the value of N is always less than or equal to the value of M. Practically, the system was implemented with 5 escrow agents with a threshold value of 3. This increased the security as if an LEA officer wants to retrieve the key chunks for a session, he or she has to convince at least 3 escrow agents. In case of fraudulent request, it is difficult to convince all those 3 escrow agents with a fraudulent notice. On the other hand, the reliability of the key being retrieved properly increased as if any of the 2 escrow agents are unavailable for some reason, the remaining 3 escrow agents can provide the key chunks.

Acknowledgments. This work was a part of a Master thesis project in the Department of Communication Systems (CoS) in the Royal Institute of Technology (KTH), Sweden. We would like to thank Professor Gerald Q. Maguire Jr. for his continuous support and guidance throughout the project.

References

1. Abelson, H., Anderson, R., Bellovin, S.M., Benaloh, J., Blaze, M., Diffie, W., Gilmore, J., Neumann, P.G., Rivest, R.L., Schiller, J.I., Schneier, B.: The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption (1997), http://www.schneier.com/paper-key-escrow.pdf
2. Zhou, H., Mutka, M.W., Ni, L.M.: Multiple-key cryptography-based distributed certificate authority in mobile ad-hoc networks. In: Global Telecommunications Conference, GLOBECOM 2005, vol. 3. IEEE, St. Louis (2005)
3. Evripidis, R.: Lawful Interception and Countermeasures: In the era of Internet Telephony, School of Information and Communication Technology (COS/CCS), 2008-20, Royal Institute of Technology (KTH), Stockholm (2008), http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080922-Romanidis_Evripidis-with-cover.pdf
4. Hossen, M.S.: A Session Initiation Protocol User Agent with Key Escrow: Providing authenticity for recordings of secure sessions, Department of Communication Systems (CoS), Royal Institute of Technology (KTH), TRITA-ICT-EX-2010:1, Stockholm (2010), http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/100118-Md._Sakhawat_Hossen-with-cover.pdf
5. Morshed, M.S.J.: VoIP Lawful Intercept: Good Cop/Bad Cop, Department of Communication Systems (CoS), Royal Institute of Technology (KTH), TRITA-ICT-EX-2010:28, Stockholm (2010), http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/100221-Muhammad_Sarwar_Jahan_Morshed-with-cover.pdf
6. Shamir, A.: How to share a secret. Communications of the ACM 22(11) (1979)
7. RSA Laboratories, http://www.rsa.com/RSALABS/node.asp?id=2259
8. MiniSIP homepage, http://www.minisip.org
9. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol, IETF RFC 3261, IETF Network Working Group (2002)
10. Arkko, J., Carrara, E., Lindholm, F., Naslund, M., Norrman, K.: MIKEY: Multimedia Internet KEYing, IETF RFC 3830, IETF Network Working Group (2004)
11. Libcurl-the Multiprotocol File Transfer Library, http://curl.haxx.se/libcurl/
12. Shamir's Secret Sharing Scheme, http://point-at-infinity.org/ssss/
13. GNU General Public License, http://www.gnu.org/licenses/gpl.html