International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 4 Issue 5, May-2015, pp: (256-263), Impact Factor: 1.252, Available online at: www.erpublications.com

Secure Multiparty Communication with Verifiable Outsourced Decryption for Mobile Cloud Computing

Kalyani Verma[1], Sangeeta[2]
[1]M.Tech. Student, R. N. College of Engineering and Technology, Panipat, Haryana
[2]Assistant Professor, R. N. College of Engineering and Technology, Panipat, Haryana

Abstract: Due to recent advances in hardware and communication technology, the paradigm of computing has been shifted from fixed to mobile. But the mobile devices possess limited data storage capacity besides inadequate battery power. We can augment mobile device with cloud services but which in turn imposes serious security implications. Confidentiality and integrity are the two major challenges in mobile cloud computing. To tackle these aforementioned securities issues we need to design a lightweight security framework for mobile cloud computing. The framework should either use lightweight algorithm or provide some mechanism in which noncritical computational expensive tasks can be securely outsourced to other device. Our design focuses on the following schemes: Securely Outsourced Ciphertext Attribute Based Encryption (SO-CP-ABE) scheme and Provable Data Possession (PDP) scheme. SO-CP-ABE scheme provides outsourced secure multiparty communication mechanism while PDP scheme provides a mechanism for verifying the integrity of file stored at remote cloud server. Using SO-CP-ABE scheme, lightweight mobile devices can securely outsource computational expensive encryption and decryption to cloud service provider. During this process no information is leaked to cloud service provider. In PDP scheme, the file is preprocessed only once and the integrity of file can be verified without possessing the file with a constant amount of network overhead. The implementation results show the efficiency of the proposed framework in terms of computation, storage and communication.

Keywords: Encryption, Decryption, Mobile Cloud Computing, Cipher-text.

1. Introduction

Mobile devices provide anywhere form of computing. With the evolution in chip making technology, computing devices becomes smaller year by year. They can be carried in pocket and one can use them while moving. Most of the today's mobile devices run at a speed higher than 1 GHz. With the introduction of 3G, 4G and Wi-Fi one can connect to other devices anytime and anywhere. In nutshell a mobile device can do processing and communicating while it is in motion. But as the volume of data is growing exponentially, demand of more storage as well as processing of these data is also growing. But still mobile devices do lack in computation capability besides limited storage and limited battery power than the fixed devices. Hence these constraints in terms of resources put a limit on use of mobile devices over long period of time. Mobile Cloud Computing (MCC) amalgamates mobile computing to the cloud computing. In Mobile Cloud Computing, mobile devices use cloud infrastructure for storing and processing of the data where cloud performs the resource intensive computational task besides providing an illusion of infinite storage to the mobile user. In this computing paradigm processing and storage of data is done outside the mobile device. Mobile Cloud Computing [1] at its simplest refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just Smartphone users but a much broader range of mobile subscribers.

Mobile Cloud Computing [2] integrates the cloud computing into the mobile environment and overcomes obstacles related to the performance (e.g., battery life, storage, and bandwidth), environment (e.g., heterogeneity, scalability, and availability), and security (e.g., reliability and privacy) discussed in mobile computing. The cloud provides the infrastructure and servers which provide the IT resources or information services, like Infrastructure-as-a-Service (IaaS) which includes databases, all types of server, parallel and distributed computing systems, storage devices, Platform-as-a-Service (PaaS) which includes support platform, operation platform and development platform, Software-as-a-Service (SaaS) which includes all kinds of software, data and information. The mobile network provides a reliable information transmission between mobile terminal and the cloud. They may include cellular networks which uses 3G and 4G technology or some other mean infrastructure less means of communication like Wi-Fi.

2. Related Work

In [3], a framework for secure data service was proposed for mobile cloud computing. In this framework, data can be securely outsourced to the cloud storage. It can be securely and efficiently shared with other mobile user. It uses the proxy re-encryption [4] and identity based encryption [5, 6] schemes to achieve confidentiality. In [7], a framework for secure storage service for mobile cloud computing was proposed. The proposed framework provides security and integrity of the file uploaded to untrusted server. It also provides a mechanism for user authentication. The file is encrypted using the session key and hashed message authentication code of the file is computed using this session key. The encrypted file along with encrypted session key is uploaded to cloud. In this framework, a telecommunication module is used which stores and generates the user's password, keys and other information. It uses asymmetric encryption scheme for sharing of the keys and password; and uses symmetric encryption scheme for encrypting the file. As these keys, passwords and other information are stored over cloud, a malicious user can use this information to pretend the user. This framework ignores the computation and storage limitation of mobile device.

In [8], three different schemes are proposed to ensure the confidentiality and integrity of the users' files stored on cloud. The first scheme is **Encryption based scheme (EnS)**. This scheme uses symmetric key encryption. When users wants to upload the file, he generates two keys; Encryption Key (EK) and Integrity Key (IK). User file is encrypted using EK and ciphertext file is generated. Then message authentication code (MAC) of the encrypted file is generated. File is then uploaded to cloud storage. When user wants to retrieve the file, it is downloaded to mobile device and integrity of the file is checked by recalculating the MAC of the file. It is then decrypted. This scheme uses standard cryptographic functions. The next scheme is **Coding based scheme (CoS).** It reduces the computation overhead incurred by the encryption and decryption in **EnS**. This scheme uses lightweight encoding vectors. The file is divided into chunks which are encrypted using these coding vectors. Coding vectors are obtained by applying recursive hash function on the string formed by concatenating password, filename and file size. This scheme consumes less computation power than the **EnS** scheme.

Third scheme is **Sharing based scheme (ShS).** This scheme is highly efficient scheme in terms of energy consumption. It uses simple Exclusive-OR (XOR) operator for encrypting the file. It is based on secret sharing scheme. File is divided into d chunks. Mobile user creates ($d-1$) random shares. The d share is generated by applying XOR operation over all the $d-1$ random shares. The file is then encrypted by XORing of these random share blocks with the file. The file can be decrypted by XORing all the d share. All these shares are randomly distributed over cloud server. If they collude, they can easily decrypt the file. Hence, this scheme is not collusion proof.

In [8], ciphertext policy attribute based encryption was introduced. This scheme provides role based access control mechanism. This scheme is based on Shamir secret sharing scheme. Data is encrypted using some access policy. This access policy is embedded to the ciphertext. The encrypted file can be decrypted if the user key satisfies access policy using which, this file was encrypted. This scheme was resilient to collusion attack. It uses bilinear pairing which in turn is computationally expensive. So, as such it was not fit for mobile devices.

3. Proposed Security Framework

3.1 Securely Outsourced Ciphertext Policy Attribute Based Encryption

In the proposed system, a mobile device is a device which gathers the data and can communicate to cloud server for storing and retrieving the stored data. The mobile device must be equipped with a wireless communication system. The data is encrypted using attribute based encryption before uploading to cloud storage. The proposed system consists of four major/core components which includes Mobile Devices (MD) or more specifically Data Owner (DO) and Data Receiver (DR), Encryption Service Provider (ESP), Storage Service Provider (SSP) and Decryption Service Provider (DSP). Figure 1 shows the architecture of the proposed framework.

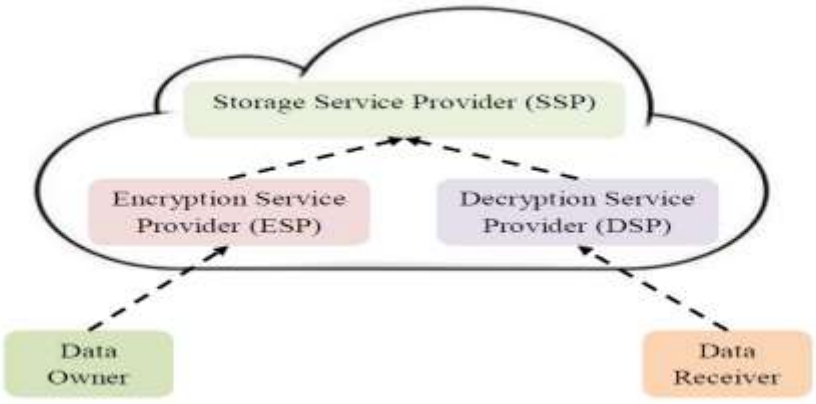

Figure 1: System Architecture of the Confidentiality Framework

Proposed security framework provides secure multiparty communication and remote integrity verification operations. It uses SO-CP-ABE scheme for sharing a secret and PDP scheme for remote integrity verification. Mobile device has limited computation power along with small storage. It is operated by the rapidly deplorable battery. Also the transmission of data over wireless channel consumes a lot of energy and cause battery to be rapidly depleted. So the proposed framework should handle the following issues:

- It should consume a little computation power that is it should not get overwhelmed by the computation expensive cryptographic operations
- It should consume a little a battery power by minimizing the network traffic
- It should not cause extra overhead in term computation and storage.

So proposed framework addresses the following issues using SO-CP-ABE and PDP as follows:

- As the CP-ABE is computational expensive so we outsource the encryption and decryptions to some cloud server which computes the bulk of the computation task at its own end without exposing the any secret/critical data.
- It uses the PDP scheme which can provide the proof of integrity remotely. The MU need not to possess the file after uploading to storage server. It can remotely check whether the file is altered over cloud or not. It requires a little pre-processing before the file is uploaded to cloud storage.
- It uses a little storage for the keys and tags which is of constant size and is independent of the size of the file.

The overall system consists of following entities:

- Trusted Authority (TA) which setups the system and generates keys
- Mobile User (MU) which can be data owner (DO) or data receiver (DR)
- ESP which acts as a computation entity for executing partial encryption
- DSP which acts as a computation entity for executing partial decryption
- InSP which acts as a computation entity for executing the PDP

Fig. 2 shows the overall architecture of proposed security framework involving various entities.

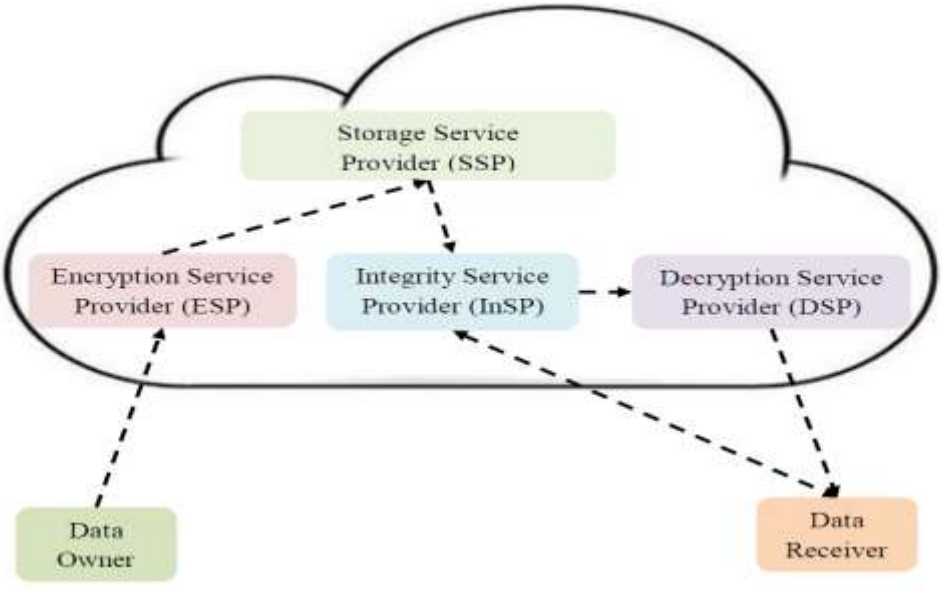

Figure 2: Complete Architecture of Proposed Security Framework

The proposed model consists of six major phases which are given as follows:

1. **System Setup and Key Generation:** In this phase the TA setups the bilinear pairing for the SO-CP-ABE scheme. It generates the public keys and private key for both the SO-CPABE scheme and the PDP scheme. The secret keys which are used in the SO-CP-ABE scheme are also generated after authentication to TA.

2. **Encryption:** In this phase a file is encrypted using the SO-CP-ABE scheme according to a specified access policy. Both MU/DO as well the ESP are involved in this phase. The MU breaks the whole access policy tree in two sub trees. One is known only to the MU/DO and other is sent to the ESP. The MU/DO encrypts the file and process the sub tree

which he owns. Similarly the ESP processes it own access policy sub tree. And finally combine the processed sub trees to create the complete processed access policy sub tree.

3. **Tag Generation:** In this phase a tag file is generated corresponding to the encrypted file. This tag file along with the encrypted file is then sent to SSP which stores and indices the files.

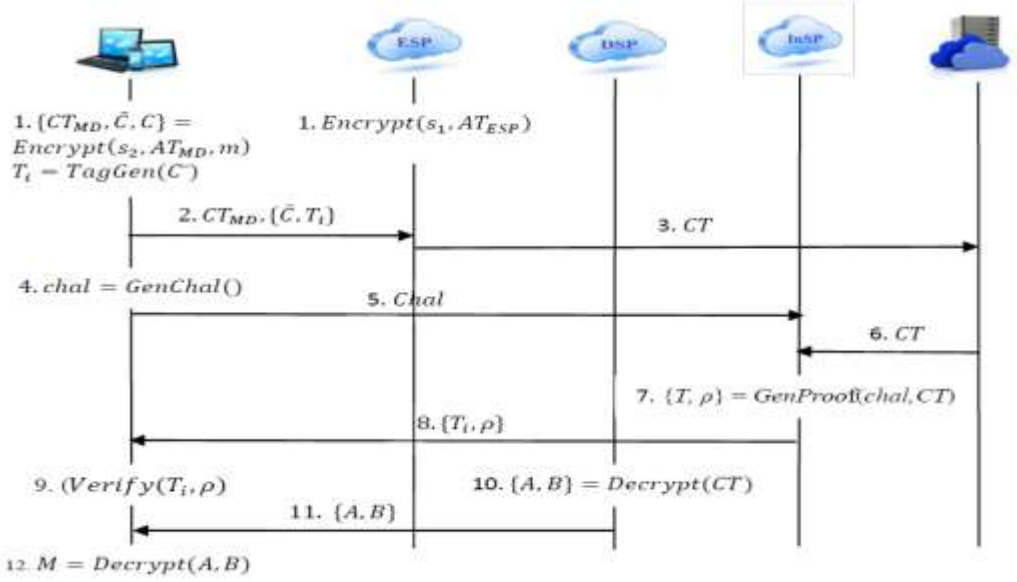

Figure 3: Sequence Diagram showing Processing and Message

4. **Proof Generation:** In this phase, InSP generates the proof of data possession against a challenge. The challenge is generated by the MU and sent to the InSP for generating the proof of data possession. The InSP produces the proof which is then sent to the MU.

5. **Proof Verifications:** The proof sent by InSP is then used to determine the possession of unaltered data.

6. **Decryption:** When the MU/DR ascertains that the stored file is an intact file or unaltered file. The MU/DR asks the DSP to decrypt the file which in turn retrieves the file from the SSP. It partially decrypts the file without knowing any secret. In fact most of the decryption task is done at the DSP. Then partially decrypted file is then sent to the MU/DR. The MU/DR finally decrypts the file to get the plaintext file.

4. Implementation and Results Analysis

4.1 Implementing SO-CP-ABE

We have used the native cpabe toolkit to implement SO-CP-ABE. But it requires thorough understanding of implementation of the CP-ABE toolkit. The CP-ABE toolkit consists of two modules:

• **libswabe**: It implements the core of cryptographic operations
• **cpabe**: It uses functions define in the libswabe and provides user interface

The organization of code in libswabe is provided as follows:

- **bswabe_prv_t* bswabe_keygen(bswabe_pub_t* pub, bswabe_msk_t* msk, char** attributes)** is used for creating the private or secret key corresponding to attributes.
- **bswabe_cph_t* bswabe_enc(bswabe_pub_t* pub, element_t m, char* policy)**: This function takes public key, message m and policy as input. Any text can be converted to an element using pbc built-in library function. Initially policy string is converted into a policy tree where random polynomials are associated with every node using void
- fill_policy(bswabe_policy_t* p, bswabe_pub_t* pub, element_t e) where e represents secret s. Finally ciphertext is computed corresponding to the message m and store along with policy tree in a file.
- **int bswabe_dec(bswabe_pub_t* pub, bswabe_prv_t* prv, bswabe_cph_t* cph)**: This function takes a ciphertext cph, public key and private key. It decrypts cph to generate plaintext m.

The organization of code in cpabe is provided as follows:

- **setup.c:** This file is used a front end for initializing the setup and uses bswabe_setup() for this purpose. It stores the public and master key in files.
- **enc.c**: It is used as a front end for encrypting the file. It uses bswabe_enc() for encrypting a single element. The complete file is encrypted and stores in <filename>.cpabe file.
- **dec.c**: It is used to finally decrypt the .cpabe file. It uses bswabe_dec() for decrypting the single element.

The modified organization of libswabe module is as follows:

- bswabe_keygen() is modified to bswabe_prv_t* bswabe_keygen(bswabe_pub_t* pub, bswabe_msk_t* msk, prv_random_t **pr, char** attributes) which now creates a transformed private key.
- void create_root_poly(bswabe_pub_t* pub, mu_poly** mpoly, esp_poly** esppoly) is added for generating the s, s and s . It creates the root polynomial which is associated with the root node of AT.
- bswabe_enc() is modified, now it looks like bswabe_cph_t* bswabe_enc(bswabe_pub_t* pub, element_t m, char* policy, mu_poly* mpoly). It creates the access policy tree corresponding to AT only which is represented using mu_poly.
- void bswabe_esp_enc(bswabe_pub_t* pub, char* policy, esp_poly* esppoly, bswabe_cph_t* cph) is added for creating the access policy tree of AT . It combines the AT and AT for creating the whole access policy tree. It then stores access policy tree and ciphertext in a <filename>.cpabe file.
- bswabe_dec() is modified and now it looks like int bswabe_dec(bswabe_pub_t* pub, bswabe_prv_t* prv, bswabe_cph_t* cph, partial_cph_t** p_cph). It only computes partial ciphertext which is then stored in a file.
- int bswabe_dec_mu(prv_random_t* pr, partial_cph_t* p_cph, element_t m) is added for completely decrypting partial ciphertxet.

The modified organization of cpabe is as follows:

• **setup.c:** Besides creating and storing public and master key it now creates and stores:
o random_exponent file for storing the random exponent
o esp_poly for storing access policy tree of the ESP
o mu_poly for storing access policy tree of the MU.
• **enc.c** takes access policy of the MU only.
• **enc_esp.c** is newly created and is used by the ESP for generating the complete ciphertext.
• **dec.c** now only creates partial ciphertext.
• **dec_mu.c** is newly created frontend file and is used to create executable which must be executed by the MU. It is produce the plaintext file corresponds to a partially decrypted file.

The modified source code can be rebuilt using make utility. Instruction for building the whole project is as follows:

- In the root directory of libswabe runs make command.
- Then run make install. It will install the libswabe library.
- Before executing make in the root directory of cpabe, we need to create the man files corresponding to the newly created files. Also the man files of the previous executable are needed to be modified as now they take different parameter set and performs different function. Then runs make in root directory of cpabe.
- Finally runs make install which will install the complete SO-CP-ABE library and manual files

4.2 Implementing PDP

PDP application is developed using pdp source code which is available at []. The source is thoroughly tailored to our custom requirement. The following changes have been made:

- A new function void write_pdp_challenge(PDP_challenge *challenge) is written in order to store the challenge in a challenge file.
- A new function PDP_challenge * read_pdp_challenge(const char *filename)is created to map the challenge stored in a file to PDP_challenge struct.
- A new function PDP_proof *read_pdp_proof()is created to map the proof stored in a file to PDP_proof struct.
- A new function void write_pdp_proof(PDP_proof *prf) is created for storing a PDP_proof struct to a file.

Finally make utility is used to completely build the application as follows:

• In the root directory of pdp runs make command

4.3 Results Analysis:

The SO-CP-ABE scheme uses bilinear pairing and hashing. We can theoretically analyze the SO-CP-ABE scheme in terms of number of computationally expensive cryptographic operations performed at various the MD, the ESP and the DSP.

A. Setup Phase

Execution time is for the setup phases SO-CP-ABE is provided below:

Table 1 Performance Evaluation of Setup Phase

	Execution Time (Clock Ticks)
SetUp	52588

Execution time includes time for generating public and master key in the SO-CP-ABE scheme. As the setup phase is executed by the TA, so it does not incur any load to the MD. Setup phase takes constant time for execution. Its time complexity is $O(1)$.

B. KeyGen Phase

KeyGen Phase of the SO-CP-ABE scheme is used to produce secret key corresponding to the set of attributes. Figure 4 shows the computation time as a function of size of attribute set and grows linearly. Computation overhead is calculated in terms of $log10$ $(Clocks\ Ticks(K))$.

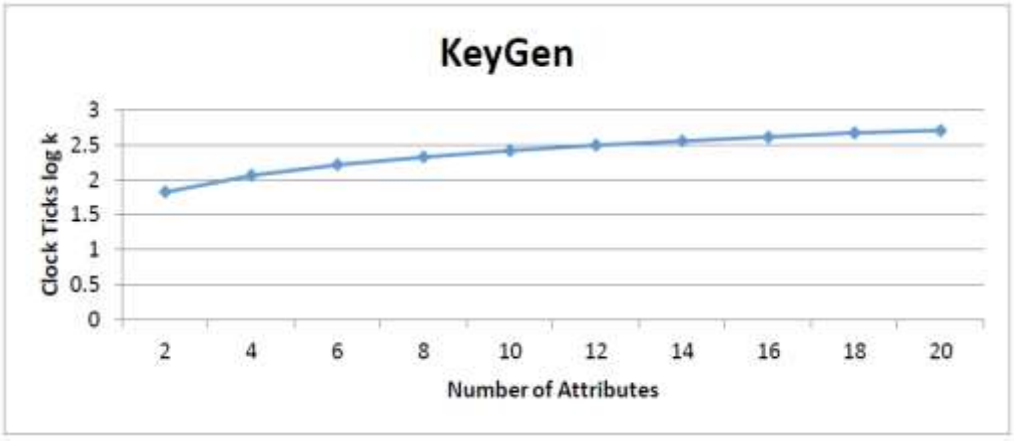

Figure 4 : Performance Evaluation of the Key Generation

C. Encryption Phase

In this phase encryption is performed using the specified policy tree consists of the set of attributes. In the Figure 5, computation overhead incurred on the ESP and the MU is compared. The computation overhead is calculated in terms of $log10$ $(Clocks\ Ticks(K))$.

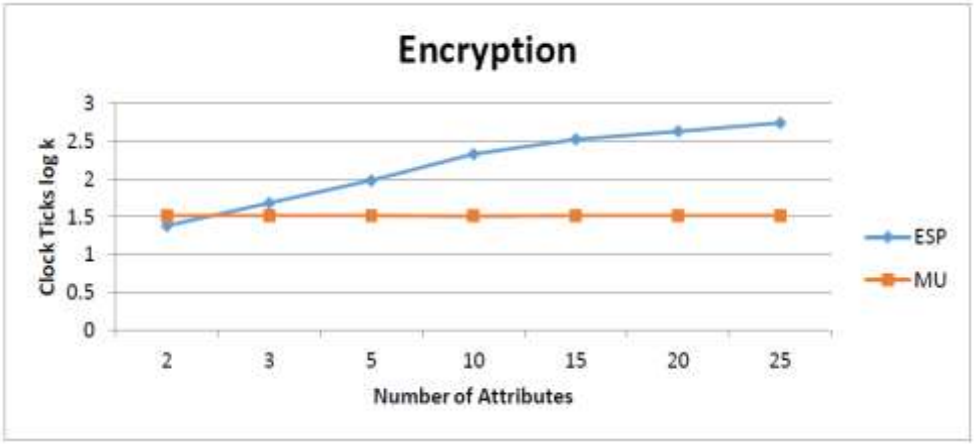

Figure 5: Performance Evaluation of the Outsourced Encryption

From the Figure 5, we can conclude that 55% of the encryption computation is performed by the ESP.

D. TagGen Phase

In this phase tags are generated for corresponding to individual blocks of the encrypted file. Figure 6 shows tagging time as a function of file size. And tagging time grows linearly with file size.

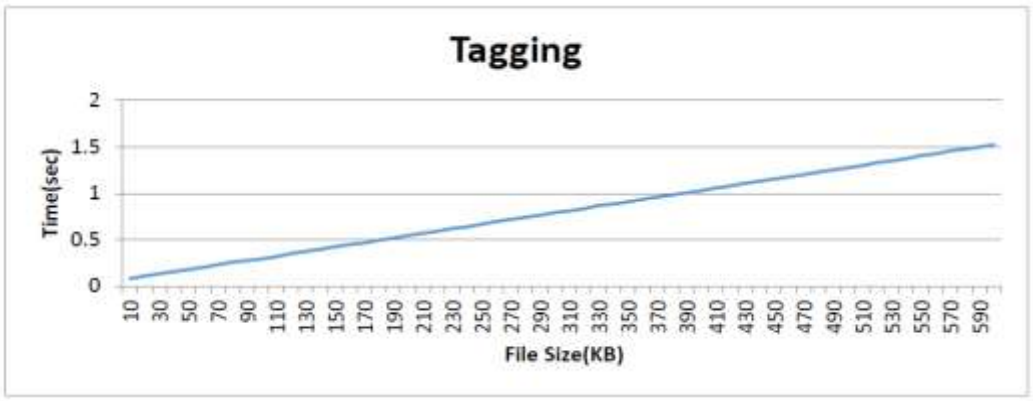

Figure 6: Performance Evaluation of the Tagging

E. ChalGen Phase

As the size of challenge is fixed so it will consume constant execution time. So complexity of this phase is $O(1)$. Table 2 shows the time taken for the generation of challenge.

Table 2 Performance Evaluation of ChalGen, ProofGen and ProofCheck

	Execution Time(sec)
ChalGen	0.055513
ProofGen	2.90406
ProofCheck	0.066884

F. Proof Gen phase

This phase also takes constant time. So the complexity of this phase is O(1). Table 23 shows the time taken for the generation of proof.

G. Proof Check

This phase also take constant time. So its complexity is also O(1). Table 2 shows the time taken for the verification of proof.

H. Decryption Phase

In the Figure7, computation overhead incurred on the DSP and the MU is compared. Computation overhead is calculated in terms of $log10$ ($Clocks\ Ticks(K)$).

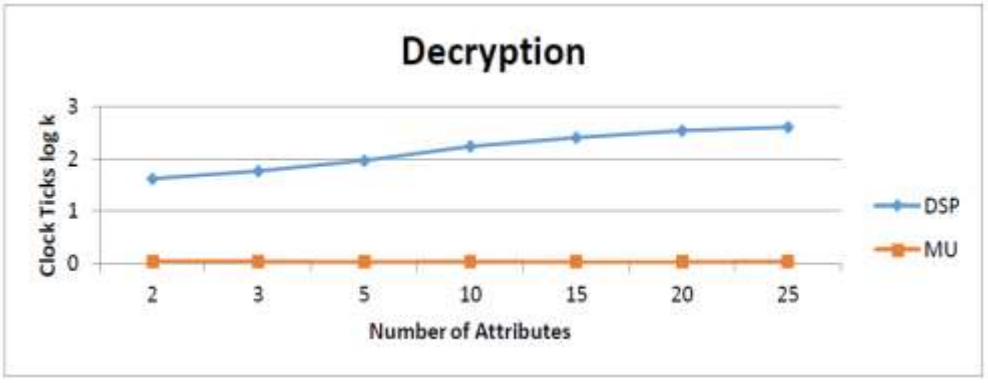

Figure 7 : Performance Evaluation of the Outsourced Decryption

It is quite clear from the figure that most of decryption task is performed by the DSP only. The MU on the other hand is relieved from the computation overhead

5. CONCLUSION

The proposed framework provides data integrity and remote integrity verification for multiparty communication. In our solution data is securely outsourced to cloud storage with minimal cost. We have implemented the Securely Outsourced Attribute Based Encryption (SO-CP-ABE) which securely outsourced the basic and computational expensive Ciphertext Policy Attribute Encryption (CP-ABE). Hence using SO-CP-ABE resource constraint mobile device can securely outsourced the noncritical computational expensive part of encryption and decryption operations. Also the scheme is collusion resistant that is the private keys of multiple users cannot collude to decrypt the data. From the results it quite clear that a lot of computational overhead is saved while outsourcing CP-ABE to cloud service provider. In case of encryption 55% and in case of decryption 90% of the total computation is done by cloud service provider.
To ensure the confidentiality of remotely stored data, provable data possession was implemented. It verifies integrity of data stored at untrusted store. This scheme access minimum number of file blocks. This scheme incur a low overhead a cloud server and also have constant communication complexity. It uses probabilistic sampling instead of verifying the whole file thus making it a practical approach for verifying large files.

6. REFERENCES

[1]. "The Intersection of Cloud and Mobility" [Online]. Available: http://www.nist.gov.
[2]. H. Suo, Z. Liu, J. Wan and K. Zhou, "Security and Privacy in Mobile Cloud Computing", IEEE International Wireless Communications and Mobile Computing Conference, Sardinia, Italy, pp. 655-659, July 2013.
[3]. W. Jia, H. Zhu, Z. Cao, L. Wei and X. Lin, "SDSM: A Secure Data Service Mechanism in Mobile Cloud Computing", IEEE Conference on Computer Communications Workshops, Shanghai, China, pp. 1060-1065, Apr. 2011.
[4]. J. Shao and Z. Cao, "CCA-Secure Proxy Re-Encryption without Pairings in Public Key Cryptography", "International Conference on Practice and Theory in Public Key Cryptography, Springer, Irvine, CA, USA, pp. 357-376, Mar. 2009.
[5]. S. Yu, C. Wang, K. Ren and W. Lou, "Achieving Secure Scalable and Fine-Grained Data Access Control in Cloud Computing", IEEE INFOCOM, San Diego, CA,USA, pp. 1-9, Mar. 2010.
[6]. M. Green and G. Ateniese, "Identity-Based Proxy Re-Encryption", ACM International Conference on Applied Cryptography and Network Security, Zhuhai, China, pp. 288-306, June 2007.
[7]. S.C. Hsueh, J.Y. Lin, M.Y. Lin, "Secure Cloud Storage for Convenient Data Archive of Smart Phones", IEEE International Symposium on Consumer Electronics, Singapore, pp. 156-161, June 2011.
[8]. J. Berthencourt, A. Sahai and B. Warters, "Ciphertext-policy Attribute Based Encryption", IEEE symposium on Security and Privacy, Washington, DC, USA, pp. 321–334, 2007.