Achieve Secure Communications for Mobile Crowd Sourcing Applications

by

Shreshth Kumar

Master of Computer Science

University of New Brunswick,

2020

A Report Submitted in Partial Fulfillment of the Requirement

for the Degree of

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor: Rongxing Lu, Ph.D., Computer Science

Examining Board: Arash Habibi Lashkari, Ph.D, Computer Science

Ali Tekeoglu, Ph.D, Computer Science

This report is accepted by the Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2020

Abstract

Nowadays, almost everyone is carrying a mobile device with them wherever they go, and these mobile devices become a potential mine for user's personal data. Actually, mobile and other wearable technologies are equipped with various sensors that can collect different types of data. With all these sensor-rich devices, we can gather precise and accurate data and can use them to develop useful applications like monitoring traffic density, road conditions, recording census data of a location, etc. Obviously, the data must be extracted, transported and processed in bulk. In addition, since data from handheld devices usually has a location-time stamp attached, if the data is not processed carefully, it can reveal information about the user, like locations visited in the past, etc. Hence, a guideline must be set up to secure communications for Mobile Crowd Sourcing (MCS), so that the required data can be securely gathered for MCS applications while preserving users' privacy who shared the data. Aiming at this goal, in this report, we propose a framework to achieve secure communications for MCS applications.

Acknowledgements

My sincere gratitude towards my Mother, Ms. Sunita Verma, who gave everything for me to pursue my master's degree, I thank you so much. I am glad that I am gradually making you proud and seeing you smile at my achievements.

I would also like to appreciate my supervisor Dr. Rongxing Lu for his intellectual guidance, advice and accessibility. Thanks for all your support, your height of knowledge and expertise with your students is simply exceptional.

I also appreciate some special friends that I got for encouraging me when I was down, always being there for me when I needed it. You always believed in me even when you didn't have to and supported me throughout this project. I really appreciate all your support.

Table of Contents

List of Tables

List of Figures

List of Notations

K_{Pu} : Public key

K_{Pr} : Private key

K_{Pu}^{A} : User A's Public key

K_{Pr}^{A} : User A's Private key

K_s : Shared key (or Session key)

K_{Pu}^{W} : Worker's Public key

K_{Pr}^{W} : Worker's Private key

K_{Pu}^{P} : Platform's Public key

K_{Pr}^{P} : Platform's Private key

K_{Pu}^{U} : User's Public key

K_{Pr}^{U} : User's Private key

H(m) : Hash of a message

Z1, Z2, Z3… : Intermediate cipher texts

C1, C2, … : Cipher-text 1, cipher-text 2, ...

query/data = Representation of plaintext query generated by users and data generated by workers

1. INTRODUCTION

1.1. Introduction

With the rapid growth of technology in recent years, we are at a point where we are surrounded by powerful handheld devices. These devices, including smart phones and smart wearable gadgets, are equipped with state-of-the-art rich sensors. These sensors can be used to record different types of data like temperature, location [1][2][3]. The sensors are even able to capture certain events, and anomalies in our environment in a precise and accurate manner. Using the recorded data, we can develop diverse range of applications for the public use to improve the service quality provided by various telecommunication companies and governmental agencies. The idea of a smart city adopted by various countries, like India and China, is aimed towards providing high-quality affordable living, while also satisfying various needs of the city by making smart decisions based on analyzing the urban living environment [4]. This is how the concept of mobile crowdsourcing (MCS) was introduced. In MCS, various applications can be developed using data collected via powerful sensors in smart devices [15].

Data collected via these mobile sensors can be used to derive powerful insights to improve daily living conditions, tracking the environmental changes, provide public safety and various commercial activities. Applications for mobile crowdsourcing include monitoring city noise, city climate, people density, emergency behavior, traffic anomalies and even detecting earthquakes [4][11]. For example, MCS can be used to develop an application that can help monitor road surface condition [1] (Figure 1.1 Road

surface monitoring application). A car can be equipped with a smart phone and used to record anomalies on the surface of the road by using its gyroscope sensor. Combining this data with location data, any potholes or bumps on the road can be identified with pin-point accuracy.

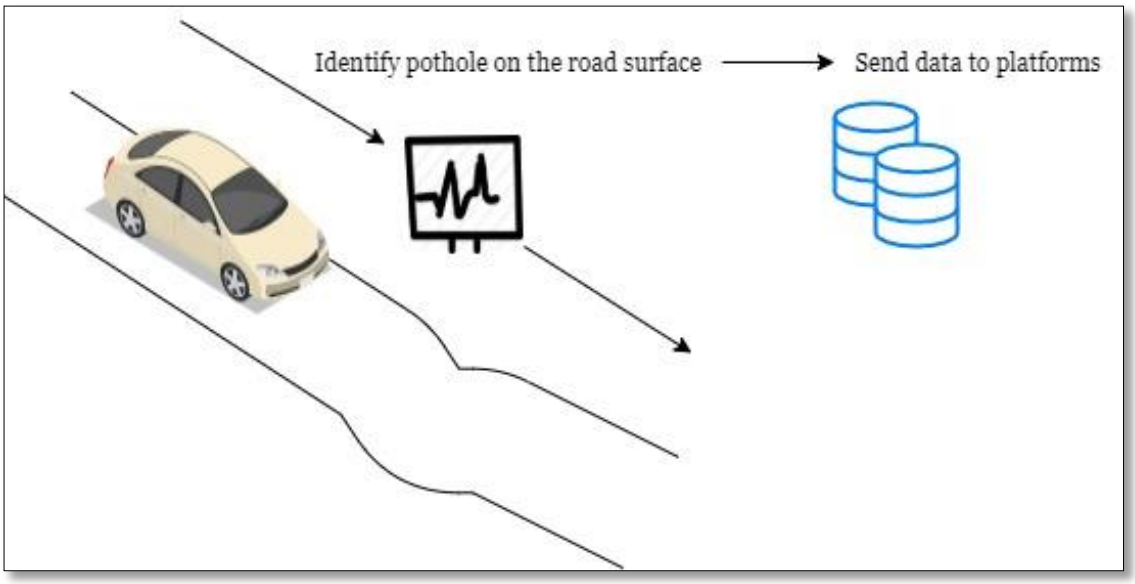


Figure 1.1. Road surface monitoring application. Cars equipped with smart phones helps in identifying road surface.

To develop such applications, there are three entities identified, each of which perform certain roles [5]. First, *Workers* (or participants) gather the data required for an application. This generally includes location and/or time data in addition to solving a task. From the example of road surface monitoring, solving a task would involve detecting anomalies on the surface of the road. Second, *End Users* (or application) process the retrieved data and derive insights from it. From our example, end users would be the authorities that find out the location of pothole and send out that information to take necessary actions. The third consists of, *Platforms* (or task-managing entities) that collect the tasks from users and distribute it to participants. In some MCS

models, the role of platform is performed by group of participants [5] (Figure 1.2 Three entities in MCS).

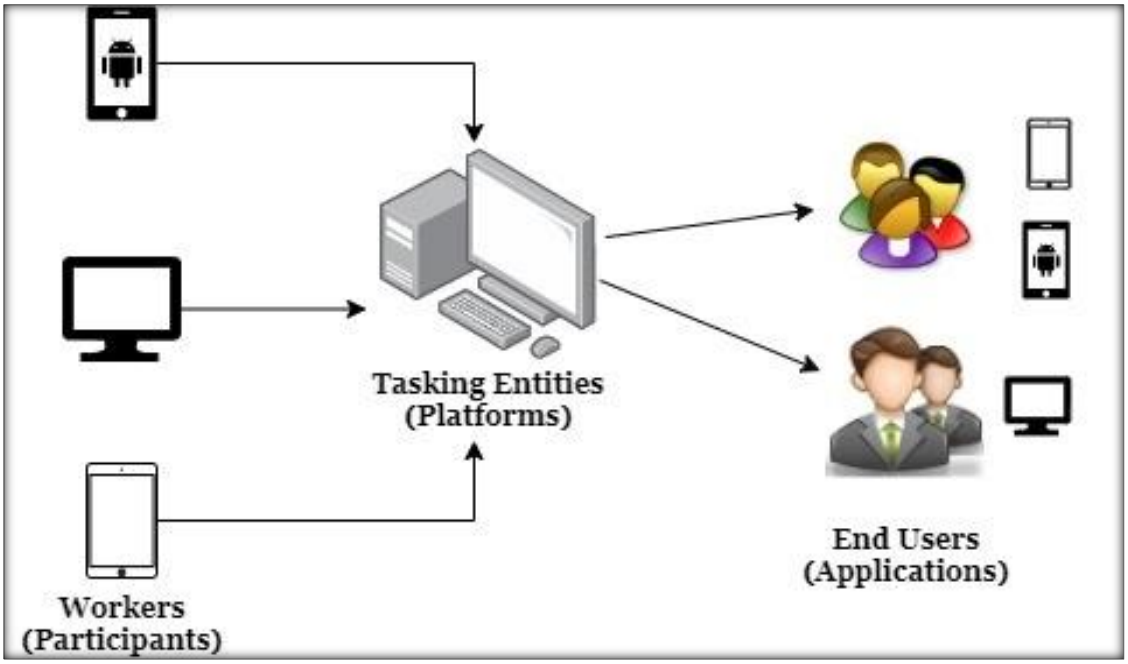


Figure 1.2. Three entities identified in an MCS based application model

1.2. Research problem

The data shared by participants often includes sensitive information related to location and time data and should be managed properly as it can expose participants to various risks related to privacy, trust and security [2][6][12][16]. *First*, privacy of the collected data should be maintained as it can expose participants to various attacks [17]. For example, data collected in a road surface monitoring application will include sensitive location- related data which if compromised, makes it vulnerable to attackers who can use this information to infer evidence about user's physical activity [19]. *Second*, trusting the data is also vital for obtaining precise and reliable insights [8]. Workers may submit unreliable data due to many reasons ranging from gaining incentives for solving a task to intentionally tampering with the application [9][7][13].

Therefore, a worker needs to be evaluated for reliability, loyalty, trustworthiness, honesty and dependability [10]. Additionally, data collected by the workers for a task will also lead to redundant data collection and will further lead to wastage of storage space while increasing incentive cost and network overheads. *Third*, due to the nature of data collected in an MCS application, it is easy for the platform to behave selfishly and launch attacks. It also attracts hackers who have in mind the aim to target and compromise platforms to steal the data. This exposes MCS to some serious security threats such as task tracing attacks, location-based attacks, collusion attacks, eavesdropping and monitoring [5][14][18].

1.3. Research objectives

In this report, we propose an MCS security model focusing on (i) maintaining authenticity and privacy of the queries sent by users; (ii) using pretty-good-privacy encryption scheme to maintain confidentiality, authenticity and nonrepudiation of the data collected by workers; and (iii) preventing single point of failure in platforms by creating and using a platform with multiple servers (P_1, P_2 and, P_3) for managing the private key among multiple servers using secret sharing techniques.

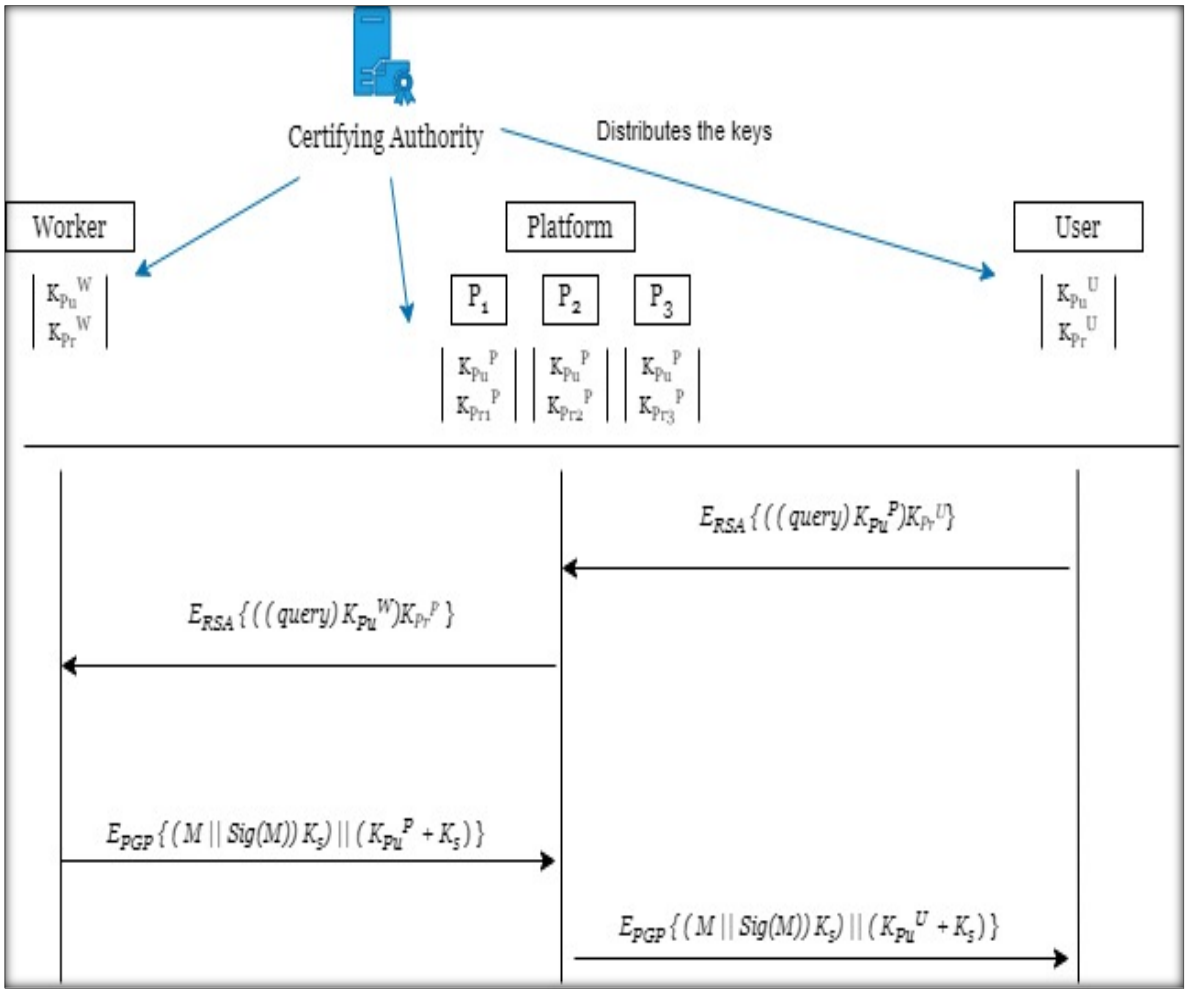


Figure 1.3 Our security model

Specifically, these objectives can be achieved by -

(i) Using RSA, the query from the user is signed by private key of the sender (user) and encrypted by public key of the receiver (platform). This helps in maintaining confidentiality and authenticity of the query as the receiver will first need to decrypt the ciphertext using its own private key, thus maintaining confidentiality; and then verify the message with the sender's public key, thus maintaining authenticity. For example, the users first sign the query with user's private key (K_{Pr}^{U}) and then encrypt the obtained ciphertext with platform's public key (K_{Pu}^{P}). This ciphertext is then sent to

platform for decryption. A platform can decipher the text using its own private key (K_{Pr}^{P}) thus maintaining confidentiality of the query; and then verify its authenticity using user's public key (K_{Pu}^{U}).

(ii) Using PGP (Pretty Good Privacy) encryption scheme, we aim to achieve core principles of information security – confidentiality, authenticity and nonrepudiation – of the data collected by workers. In PGP, a sender should send two ciphertexts to the receiver. Authenticity and nonrepudiation of the data can be achieved by first generating and appending Digital Signature (DS) to the data collected. Then, it is encrypted using a session key (K_s) to generate the first ciphertext, thus maintaining confidentiality. The session key will be further encrypted with the receiver's public key to generate the second ciphertext. Both the two ciphertexts are then sent to the receiver for decryption.

(iii) To overcome the problem of single point of failure in the architecture, we propose to use a platform with multiple servers. We implement our scheme using three servers to manage the platform's public and private key. Public key of all servers remains the same, but each server has a different private key. Using (t,n)-secret sharing scheme, we divide the private key of platform to n-parts and distribute it among servers, such that a minimum number of servers (or *threshold amount t*) are required to recover the private key. This means that t servers of the platform need to work together to recover any data. Also, even if a platform is compromised, it would not lead to any data being compromised. In our research, we use (2,3)-secret sharing technique.

1.4. Purpose of research

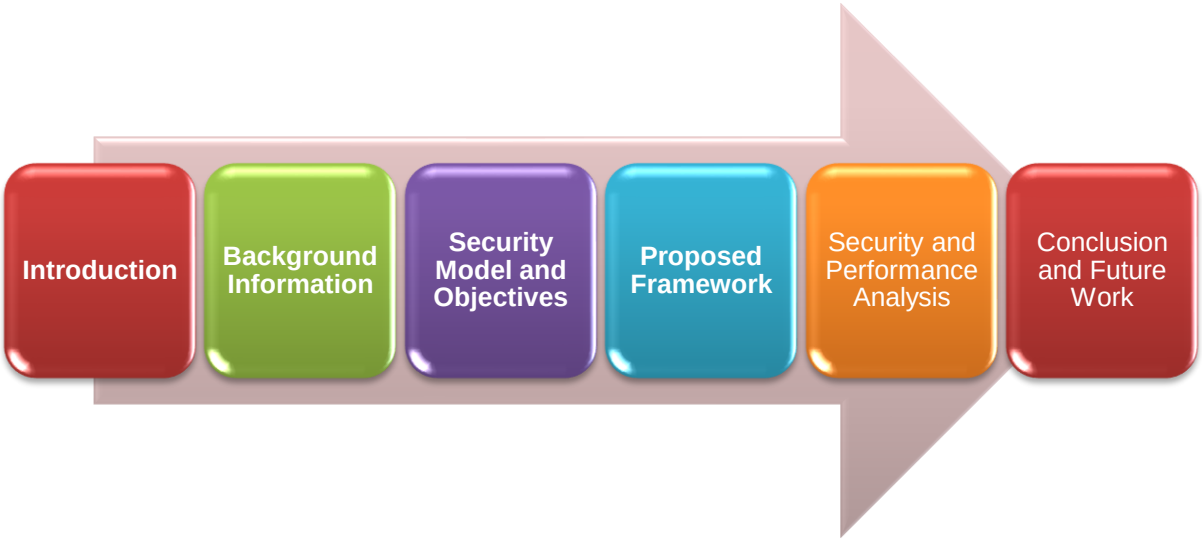
MCS is a vast field with numerous applications for the environment. It can enable us to automate certain tasks (like road surface monitoring, finding parking spots in a parking area, etc.) which helps us to develop useful application for the community. But with the increase in threats to the data shared and privacy of the user, numerous algorithms and privacy techniques are developed to record the data required for the MCS system without compromising user's privacy. Hence, it is important to develop guidelines for MCS applications for securing the user's data. For this study we will be looking into the challenges of a commonly deployed mobile crowdsourcing system and try to build on the technology in order to produce a much secure model for our study.

1.5. Significance of research

MCS applications can be implemented for industries including healthcare, construction and smart transportation which can enable us to monitor city noise, city climate, people density, emergency behaviour, traffic anomalies and even detecting earthquakes. The significance of this research is reflected in a number of benefits. This research is expected to yield what we have outlined earlier as a part of challenges to the MCS system. That is, developing an algorithm that will successfully encrypt/anonymise a user's data while preserving the actual data required for processing for that application.

1.6. Organization of Report

This report is basically divided into five sections:



Chapter 1: Introduction

In this chapter, the purpose is to define mobile crowdsourcing, its significance, challenges mobile-crowd sourcing applications face, why the research is being conducted and what the research is intended to achieve and how it will be achieved.

Chapter 2: Background Information

This chapter is designed to present an analysis of past studies in relation to the research topic. This chapter will also highlight relevant theories and models that will be used to address the whole research as well as aid in the development of variables that will be tested in the primary research.

Chapter 3: Security Model and Objectives

This chapter presents our security model and design objectives. It is designed to present the security objectives covered by our research. Such analysis will include the target platform, frameworks used, and the deployment methods and mechanisms involved.

Chapter 4: Proposed Framework

This chapter presents working of our proposed model. The results gathered will be further analyzed in relation of the objectives stated within the previous chapters.

Chapter 5: Security and Performance Analysis

This chapter gives an evaluation of the research findings collected. It also gives recommendations as well as describing the limitations of the research and providing recommendations for further research.

Chapter 6: Conclusions and Future Work

This chapter provides a conclusion for our findings and proposed model. We also analyse the assumptions and limitations of our work and provide future recommendations to it.

1.7. Definition of terms

- **Mobile Crowdsourcing (MCS):** refers to the idea of crowdsourcing data from powerful sensors available in handheld devices.
- **PGP:** Pretty-Good-Privacy protocol
- **DS:** Digital signature of a file or a message

2. BACKGROUND INFORMATION

2.1. Introduction

This chapter includes the background information required to understand the proposed scheme. We include these topics to help provide an understanding of the basics of privacy and cryptography relevant to our scheme. Cryptography focuses on protecting the data in transmission between users by altering the data to unreadable form before sending [27]. This process is called *encryption*. After receiving the data, the receiver then also uses the same tools to convert the unreadable text back to its original form. This process is called *decryption*. These topics are explained in more detailed in further sections:

2.2. Encryption

Encryption is a process of converting a plaintext to an unreadable *ciphertext* so that any ordinary user cannot read it. This is usually performed by using plaintext as input for an *encryption algorithm* along with a *key* to encrypt it to unreadable text [26]. The only user who can *decrypt* the ciphertext is one with the knowledge of (a) the *algorithm* is used for encryption and (b) *key* used for it. Once a user has both, he can decrypt the ciphertext back to plaintext. Encryption can be categorized into *symmetric* (or private key) and *asymmetric* (or public key) encryption [27].

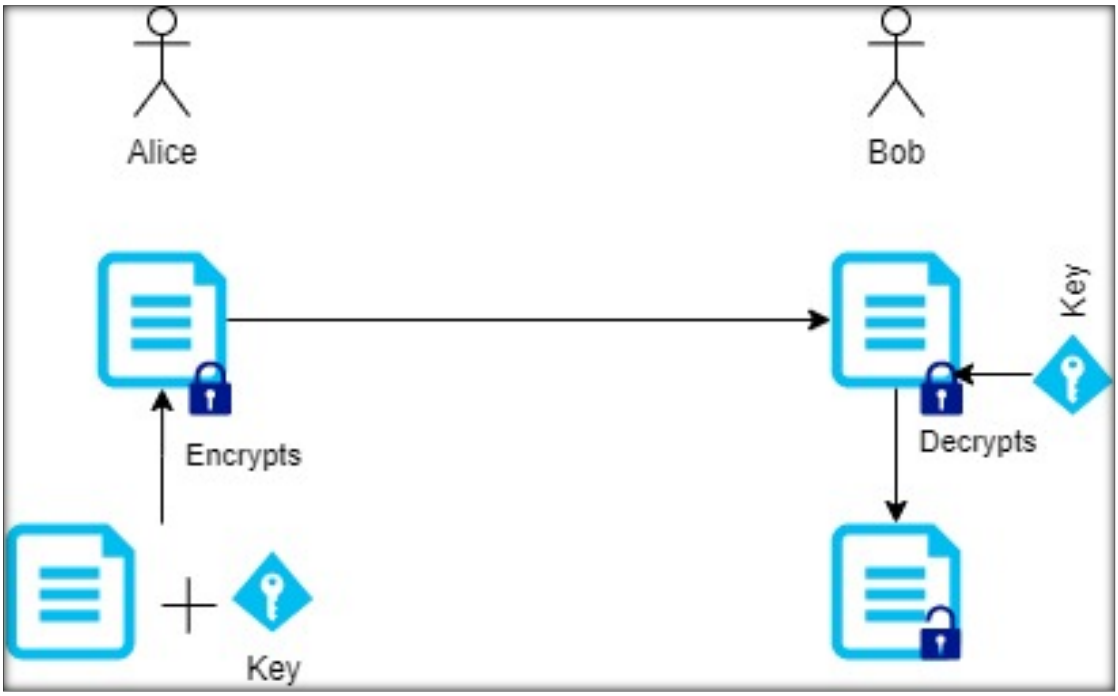


Figure 2.1 Simple Encryption

2.1.1. Symmetric Encryption

In symmetric encryption, only one key is used for encrypting the plaintext. In this algorithm, the sender and the receiver first decide between an algorithm and the key to use for sending their messages. Some examples of symmetric key encryption are AES, DES, 3DES (or triple-DES), Blowfish etc. Once that is decided, they can send and receive encrypted messages to each other without anyone else present in the network reading their conversation. The key used in this process needs to be kept private and is also called as *secret key* or *shared secret* (represented by K_s).

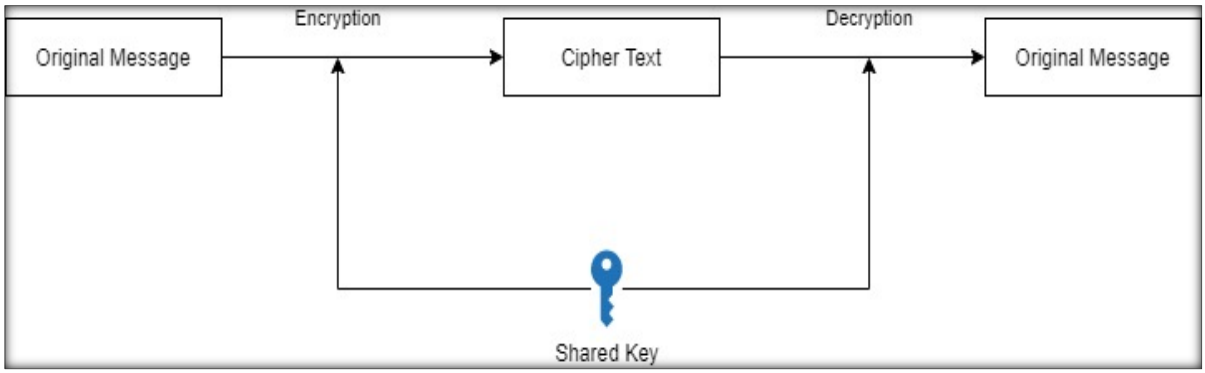


Figure 2.1.1 Symmetric Encryption

The major challenge in this encryption is that the key needs to be transferred to the receiver carefully. If any other user gets hold of the key, then the encryption is no longer useful. Hence to prevent key leakage, we need to transfer the key securely.

2.1.2. Asymmetric Encryption

Asymmetric encryption (also known as public key encryption) algorithm uses two sets of keys: public key (K_{Pu})and a private key (K_{Pr}). If a user encrypts a message using one key (say public key) then that message can only be decrypted by using the other key (private key). Some common examples of asymmetric encryption algorithm and its applications include RSA encryption, Digital Signature Algorithm (DSA) and Diffie-Hellman (key exchange)[23][24][25].

Before the communication begins, each user in the network receives a set of public-private key pair. If a user wants to send a message to another user while maintaining *confidentiality* of the message from other participants, he can encrypt the message using receiver's public key (K_{Pu}) and send it over. Since this message can only be decrypted using receiver's private key (K_{Pr}), only he can read the contents of the message.

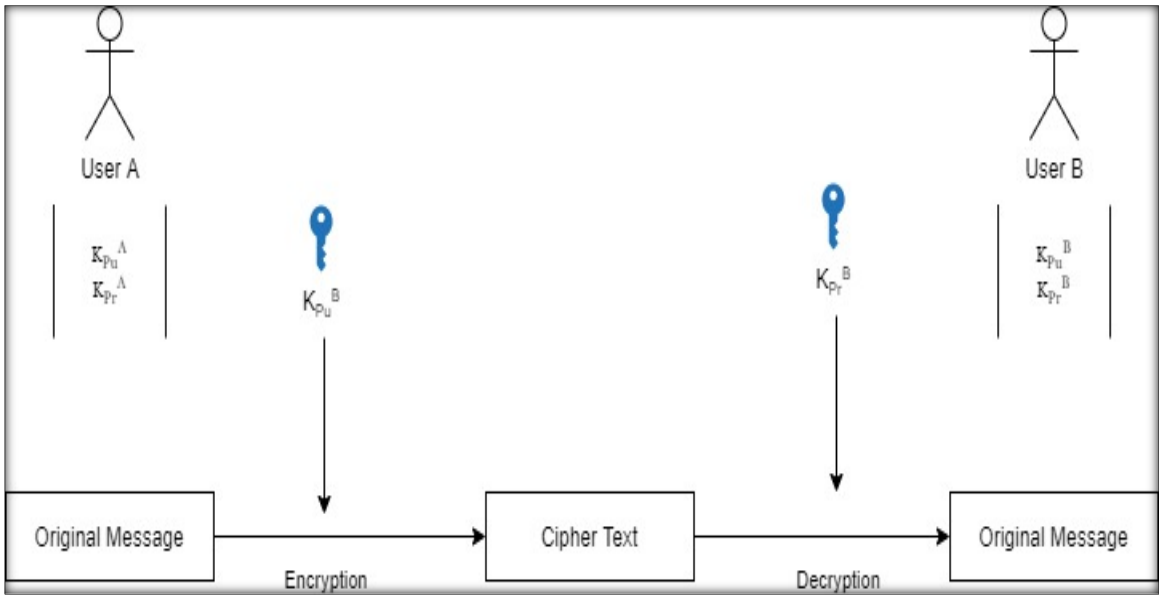


Figure 2.1.2a Asymmetric encryption maintaining *confidentiality* of the message

Now, if a user wants to send a message to another user while making source of the message to be authentic, i.e., maintain *authenticity* of the message, he can sign the message using his own private key (K_{Pr}) and send the signature along with the message. Since this signature can only be generated using sender's private key (K_{Pr}), whoever receives this message and the signature can authenticate its origin. This process is also known as *signature generation.*

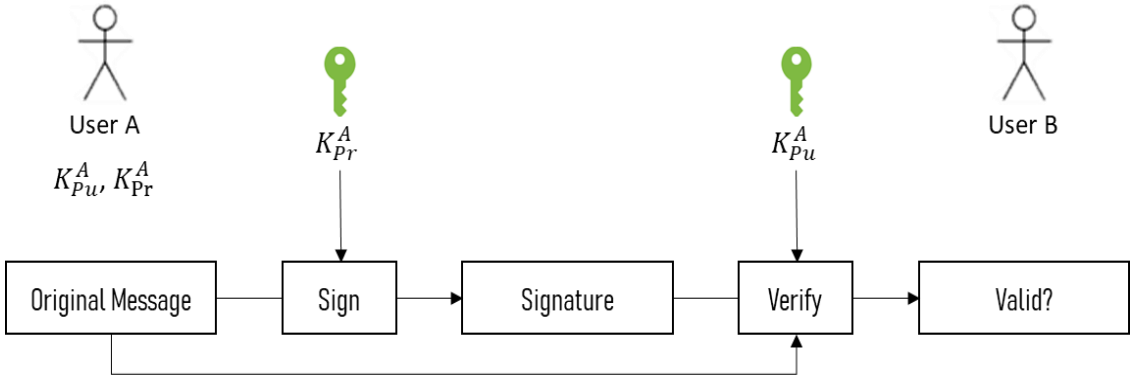


Figure 2.1.2b Asymmetric encryption maintaining *authenticity* of the message

Table 2.2 Difference between Symmetric and Asymmetric Encryption

Symmetric Encryption	Asymmetric Encryption
In symmetric encryption a single key is used to encrypt the message	In asymmetric encryption, sender can encrypt the message using receiver's public key.
The secret key needs to be shared to the receiver	Each user gets a public-private key pair.
It is faster due to its simplistic nature.	It is slower as it needs more time processing public-private key pair for decryption
Symmetric encryption algorithms include RC4, AES, DES, triple-DES, QUAD, etc.	Asymmetric encryption algorithms include RSA, Diffie-Hellman, ECC, ElGamal, DSA, etc.

2.1. Advanced Encryption Standard (AES)

AES is a type of systematic block cipher encryption which uses one key (called round key) from a user to encrypt/decrypt the plaintext [31][20]. This algorithm is fast and efficient for encrypting large data files. AES can use cryptographic keys of length 128, 192 and 256 bits to encrypt/decrypt data by dividing the data in blocks of 128 bits. These can be represented as AES-128, AES-192 or AES-256.

AES compromises of four different byte-oriented transformations – substitute bytes, shift rows, mix columns and adding round key [31]. The plaintext block of 128 bits (16 bytes) is divided into sixteen 8-bit blocks (one-byte per block). This input is then arranged in an array and is passed through various round functions containing all four transformations. We denote the bytes in the array as in0, in1, in2, …, in15, an input state before a transformation as $s = (s_{i,j})$, where $0 \leq i,j \leq 3$ (refer Figure 2.2). The output of a transformation is represented by $s' = (s'_{i,j})$, where $0 \leq i,j \leq 3$.

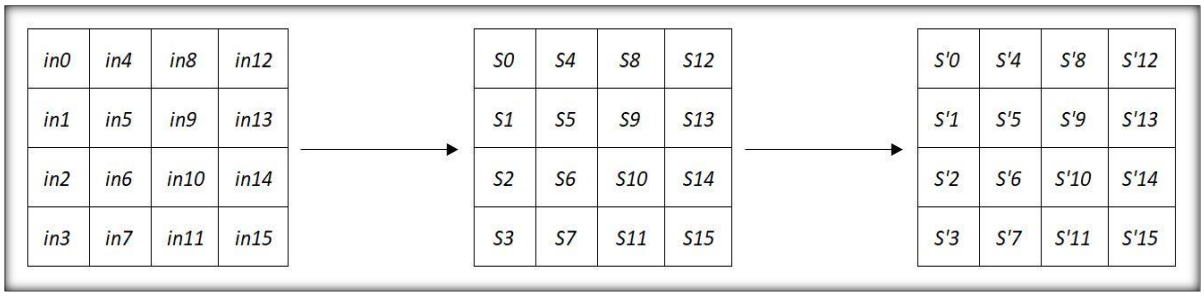


Figure 2.2 AES Algorithm Transformation

These are explained as follows:

1. Substitute Bytes:

This is a non-linear byte-substitution which is performed using a substitution table (S-box). This substitution operates independently on each byte of the input state using S-box *S*.

$$(s'_{i,j}) \leftarrow S\,(s_{i,j})$$

2. Shift Rows:

In this transformation, each row is clinically shifted depending on the indexing of the row, r. So, row zeroth will not be shifted, but first row shifts one row, and the second shifts two rows, and so on.

$$(s'_{i,j}) \leftarrow (s_{i,(j-i) \bmod 4})$$

3. Mix Columns:

The output from shift rows is mixed column-by-column using Mix Columns transformation. This is represented as:

$$\begin{bmatrix} S'(0,c) \\ S'(1,c) \\ S'(2,c) \\ S'(3,c) \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S(0,c) \\ S(1,c) \\ S(2,c) \\ S(3,c) \end{bmatrix}, for\ 0 \le c \le 1$$

4. Add Round Key:

In this process, a round key obtained by dividing the key length is XOR-ed with the corresponding state byte. This is represented as:

$$(s'_{i,j}) \leftarrow (s_{i,j}) \oplus (k_{i,j})$$

This encrypting technique has variable rounds of transformations depending on the key length. It has 10 rounds for key length 128, 12 rounds for key length 192 and 14 rounds for key length 256 bits [27].

2.2. RSA Encryption

RSA encryption algorithm is a public key encryption (PKE), which implies that it uses a pair of keys – public and private keys – for encryption/decryption. In RSA, the public and private key pair is generated based on the hardness of Large Integer Factorization [22] [32]. This method is used to generate a key-pair such that if a message is encrypted using public key (K_{Pu}), it can only be decrypted using the corresponding private key pair (K_{Pr}). Specifically, the RSA algorithm is shown below:

1. Choose two large prime numbers, say p and q, where $p \neq q$.
2. Calculate $n = p * q$ and
3. Compute $\varphi(n) = (p-1) * (q-1)$, where φ is Euler's totient function.
4. Choose a number, say e, which has no common factors with $\varphi(n)$ (other than 1). That is,

$$\gcd(\varphi(n), e) = 1, \text{ and } 1 < e < \varphi(n)$$

5. Find a number, d, such that e * (d-1) is exactly divisible by z (i.e., no remainder). That is,

$$d \equiv e^{-1} \pmod{\varphi(n)}$$

The **public key** (K_{Pu}) is the pair of number (n, e) and **private key** (K_{Pr}) is d.

Encryption Process:

1. Obtain a plaintext M, such that M < n.
2. Compute ciphertext as $C = M^e \bmod n$

Decryption Process:

1. Get the ciphertext, C.
2. Compute plaintext as, $M = C^d \bmod n$.

2.3. Hashing

Hashing is the technique used to transform a variable string of characters into a usually shorter fixed-length value that identifies uniquely to the original string [29].

$$\text{Hash Function} : M \mapsto H(M)$$

The output H(M) obtained by using a hashing algorithm on the string (or message *M*) is a unique value. Moreover, it is not possible to recover M with H(M). A cryptographic hash function helps in easy verification of the input data by mapping it to a hashed value. One-way hash functions (also referred as fingerprinting) inputs a variable length message and generates a fixed size code. Hence, hashes can be used to maintain *integrity* of a message. A Hash function have following properties:

- One-Way: Given M it is easy to compute H(M); but given H(M), it is impossible to obtain M.

- Collision-free: It is highly unlikely to have two same hashes for two different messages. This means given message M_1 and M_2, their hashes won't be equal, i.e., $H(M_1) \neq H(M_2)$.
- Collision-resistance: It is highly unlikely to find M_1 and M_2 such that $H(M_1) = H(M_2)$.

Hashing is used in many encryption algorithms and applications [29] such as Database indexing, Symbol tables, Data dictionaries. Hashing is provably secure as it is hard to *break* a hash due to the complexity involved. Hashes can also used in various communication schemes to generate fingerprints of data and attaching it to the message, maintain its integrity while transfer.

2.4. Digital Signature

Digital signature of a message is helps to maintain both authenticity of the source the message and integrity of the message [21][22][23]. To generate a digital signature of a message, a user as to:

1. Calculate hash of the message.

$$M \rightarrow H(M)$$

2. Encrypt the hash with it's private-key pair, also referred as *certificate* [30], using RSA, or similar, asymmetric encryption algorithm.

$$H(M) \rightarrow E\{(H(M))K_{Pr}\}$$

This output is the signature of the message and is appended to the message before sending it to a receiver. The receiver can use sender's public-key pair to decrypt it,

hence verifying its authenticity. Then, the obtained hash is used to verity integrity of the message.

2.5. PGP Scheme

Pretty-Good-Protocol (or PGP) is a technique designed for email security [28]. Core principles of E-Mail security includes (a) maintaining an email confidential; (b) making sure no unauthorised user can change the contents of the message; and (c) making sure neither the receiver nor the sender can deny in taking part in a communication. PGP protocol meets the following security objectives– confidentiality, integrity and non-repudiation. It uses hashes to maintain integrity of the messages and asymmetric encryption to maintain confidentiality and non-repudiation. Before communication begins, a sender and a receiver, both receives their public-private key pair for encryption involved.

Let suppose a user wants to transfer message *M* to another user using PGP scheme. The encryption process for sender works as follows:

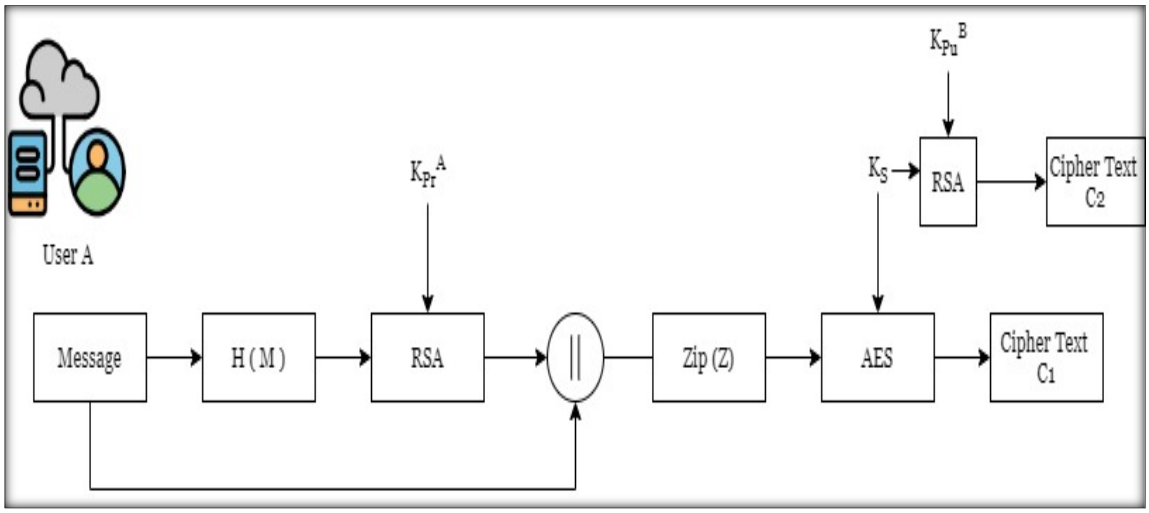


Figure 2.5a PGP Sender's perspective

i. User A generates hash of the message *M* using MD5 hashing algorithm.

ii. The hash is signed with user's private key using RSA (or similar) digital signature algorithm. This is the signature of the message and is appended to it before compression.

iii. User generates a secret key (also known as shared key or K_s).

iv. This shared key (K_s) is used to encrypt the compressed output using AES (or similar) encryption algorithm. The output obtained is referred as the first ciphertext or C_1.

v. The shared key K_s is then encrypted with receiver's public-key using the RSA encryption. The output generated is referred as second ciphertext or C_2.

vi. Both ciphertexts are then compressed and then sent to the receiver.

The decryption process for user *B* goes as follows:

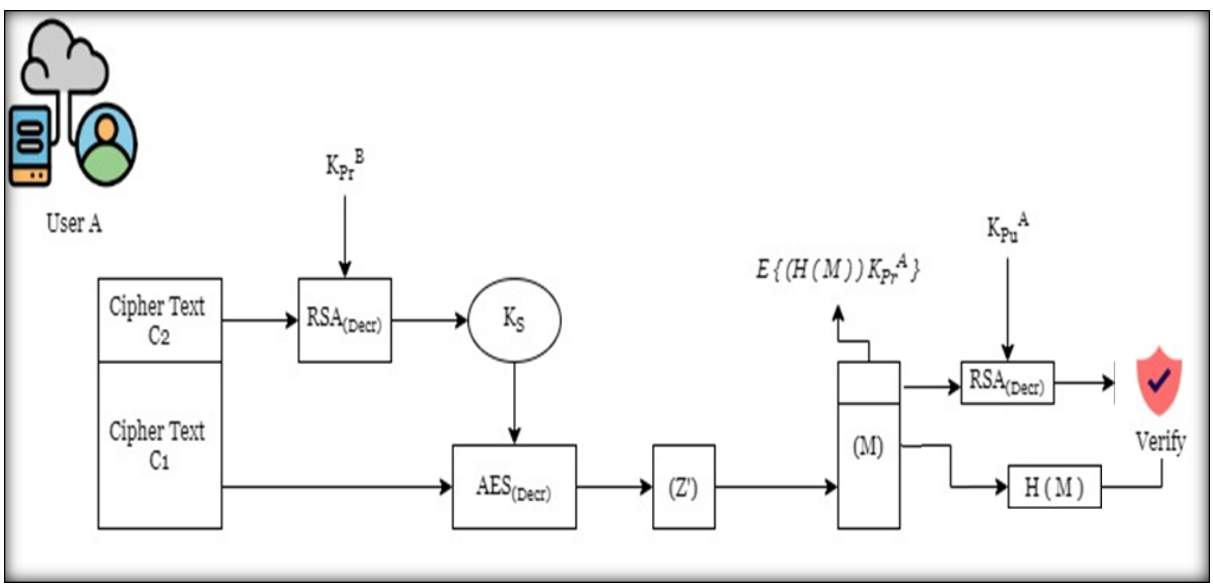


Figure 2.5b PGP Receiver's perspective

i. Decompresses the message to obtain both ciphertexts C_1 and C_2.

ii. Decipher the ciphertext C_2 with its own private key using RSA to obtain the shared key K_s.

iii. Using the shared key, decipher the ciphertext C_1 using AES encryption to obtain the compressed message and its signature.

iv. The output is decompressed, and the signature is verified using sender's public key and the hash of the message.

This scheme ensures confidentiality, authenticity and integrity of the message. This, however, does not protect the users from man-in-the-middle attack. Kuobin, D [28] explains the security issue and introduces an improved PGP scheme.

2.6. Secret Sharing Scheme

This technique was developed to divide and distribute a *secret* among a group of users such that certain number of members of the group can work together and recover the secret. The number of people that needs to work together to recover the key is called the *threshold* number. If a secret is divided among n-members such that any t-members can work together can recover the key, such technique is called as (t,n)-threshold secret sharing. Another common secret sharing technique is Shamir's secret sharing scheme, which is explained further.

2.6.1. (t,n)-Secret Sharing

As explained above, the main idea of (t,n)-secret sharing is to divide a secret, say S, into n-shares (s_1, s_2, s_3,…, s_n) such that: (a) any member having t or more shares s_i can

recover the secret *S*; and (b) any member having less than *t* shares cannot determine the value of *S*. In the following sections we have considered $t = 2, 3$ and explained (2,n)-secret sharing and (3,n)-secret sharing techniques.

2.6.2. (2,2)-Secret Sharing Technique

In order to share a secret *S* among two members (say M_1 and M_2), the *S* is divided into two equal shares s_1 and s_2 such that, $s_1 + s_2 = S$ and, both members can be handed each share. This technique is known as (2,2)-secret sharing. This is achieved by using basic geometry. As we know choosing any two points on a plane would make a line. Now, on a plane, a line intersecting y-axis at *S* is drawn and any two points on the line can be used as shares. These members are then given a point each on the line. If any of the member wants to recover the original message, they can do that by combining the points and deriving the equation of the line. Once they have the equation, the secret message *S* can be calculated by calculating its y-intercept.

Table 2.6.1. (2, 2)-Secret Sharing Technique

	s_1	s_2
M_1	✓	
M_2		✓
$M_1 \cup M_2$	✓	✓

2.6.3. (2,3)-Secret Sharing Technique

In this technique the secret *S* is divided into three shares (s_1, s_2 and s_3) and shared between three members (say M_1, M_2, and M_3) such that any two members can work together can recover the secret. The secret is divided in three shares such that combining

all the shares can reveal the secret message, i.e., $s_1 + s_2 + s_3 = S$. This is achieved by using the same geometrical solution as above, by creating a line on a plane passing through y-axis on *S* and then choosing three points on that line as shares for the members. Now any two members can come together can recreate the equation of the line, and thus can find the secret message *S*.

Table 2.6.2. (2,3)-Secret Sharing Technique

	s_1	s_2	s_3
M_1	✓	✓	
M_2		✓	✓
M_3	✓		✓
$M_1 \cup M_2$	✓	✓	✓
$M_2 \cup M_3$	✓	✓	✓
$M_1 \cup M_3$	✓	✓	✓

2.6.4. (2, *n*)-Secret Sharing Technique

In order to share a secret *S* among *n* members of a group, such that any *two* members can work together can recover the secret, we use (2, *n*)-secret sharing. With the help of basic geometry, we know that two points are enough for drawing a line. Now, by choosing two points on a plane that intersects through y-axis at *S* gives us the equation of that line. Finally, each user is distributed a share (or a point on the line) such that any user having two shares can recreate the equation and get the y-intercept, and hence recover the secret *S*.

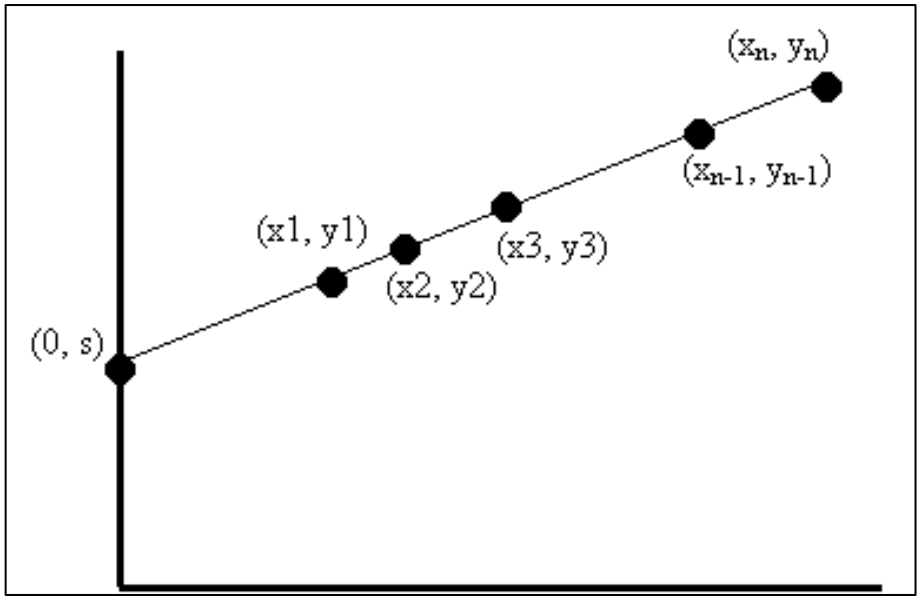


Figure 2.6.4. (2, n)-Secret Sharing

2.6.5. (3, *n*)-Secret Sharing Technique

In order to share a secret *S* among *n* members of a group, such that any *three* members can work together can recover the secret, we use (3, *n*)-secret sharing. Similar to (2, n)-secret sharing, with the help of basic geometry, we know that three points defines a parabola on a plane. Now, by drawing a parabola that intersects y-axis at *S*, any point on the parabola can be used as a share. Finally, each user is distributed a share (or a point on the parabola) such that any user having three shares can recreate the parabola, get the y-intercept, and hence recover the secret S.

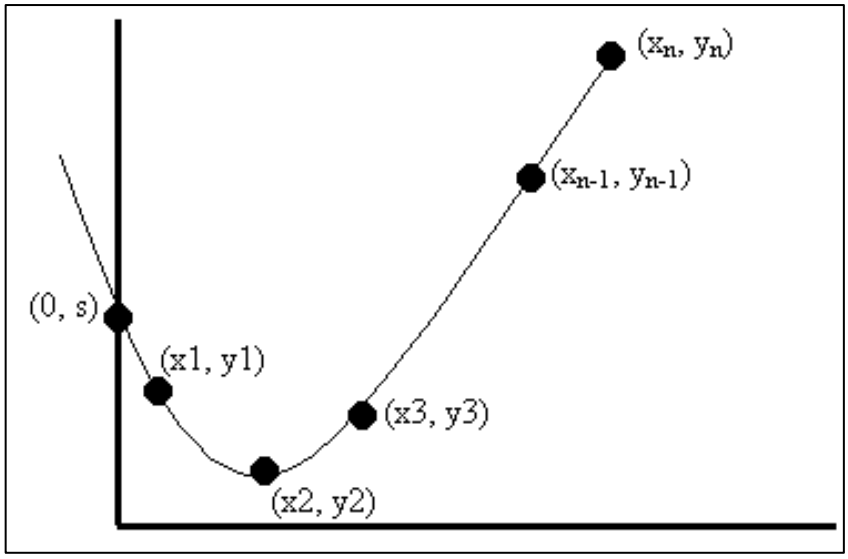


Figure 2.6.5. (3,n)-Secret Sharing

2.6.6. Shamir's Secret Sharing Scheme

This technique was designed to distribute a secret between *d* users such that any *k* users (where, $k < d$) working together can recover the secret. Shamir [33] explains the problem and provides the secret sharing mechanism using polynomial interpolation as a recovery mean. In this technique, each user x_i, is given a pair $(x_i , f(x_i))_{xi \neq 0}$, where $f(x)$ is a polynomial of degree *k* produced over a finite field F_p, *p* is prime, and the secret is given by $f(0)$. With this, we need at least $k + 1$ shares to recover $f(x)$, and then $f(0)$. Hereafter, we recall the sharing and reconstruction algorithms for a value of $k = d - 1$, operating on *n*-bit words. With all these parameters ready, in order to share a secret a_0 into *d* shares, we need to choose $(d - 1)$ random numbers $(a_1, a_2, \ldots, a_{d-1})$ from F_p to construct the polynomial

$$f(x) = (a_{d-1} \cdot x^{d-1} + a_{d-2} \cdot x^{d-2} + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0) \bmod p$$

Every share, i, is then given to each user by (x_i, y_i) where $y_i = f(x_i)$, and the x_i's are all distinct and non-zero. The algorithm is as follows [33]:

Algorithm 1. Shamir's Secret Sharing scheme

Input: A secret a_0, random values $(x_i)_{i=0\ldots d-1}$

Output: Shares $(x_i, y_i)_{i=0\ldots d-1}$

1. $(a_i)_{i = 1..d-1} \leftarrow$ Rand(n)
2. **for** $i = 0$ to d **do**
3. $y_i \leftarrow (a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{d-2} \cdot x^{d-2} + a_{d-1} \cdot x^{d-1}) \bmod p$
4. **return** $(x_i, y_i)_{I = 0\ldots d-1}$

From the above algorithm we get the output – secret pairs (x_i , y_i), where x_i denotes the pair-index and $y_i = f(x_i)$. The reconstruction step is directly derived from polynomial interpolation and is as follows:

$$a_0 = \sum_{i=0}^{d} y_i \cdot \beta_i$$

where each β_i is a precomputed value such that,

$$\beta_i = \prod_{j=0, j\neq i}^{d} \frac{-xj}{xi-xj}$$

Therefore, using at least k shares, participants can recover the coefficients of $f(x)$ using Lagrange's interpolation:

$$f(x) = \sum_{i=0}^{d} \prod_{j=0, j\neq i}^{d} \frac{-xj}{xi-xj} \cdot y_i$$

3. SECURITY MODEL AND OBJECTIVES

In our system model, we consider the three types of entities including *workers* who gather the data for a task, *users* who generate a task, and *platforms* that helps in secure transmission of these tasks and data (as shown in Figure 3. System Overview).

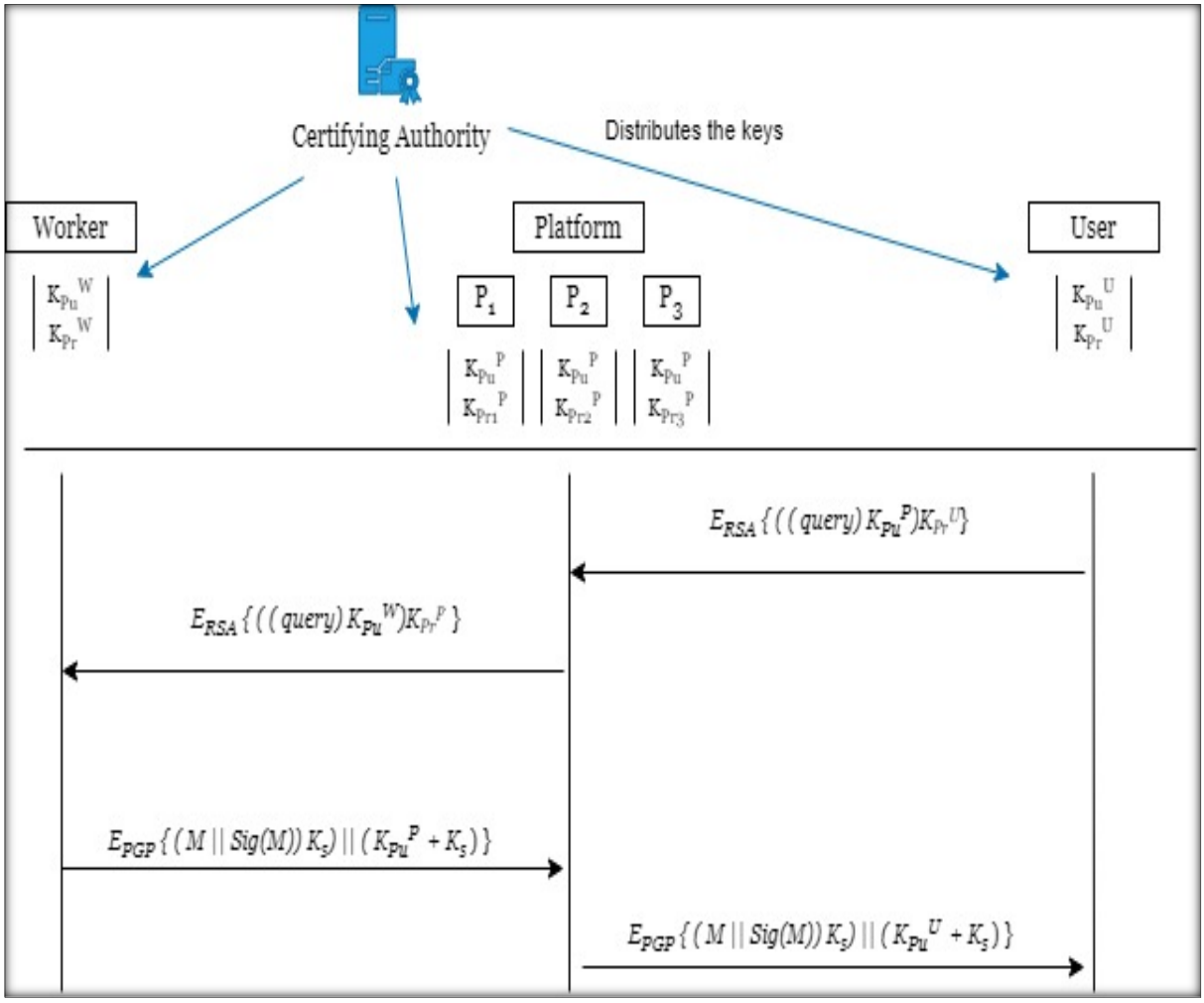


Figure 3. System Overview

3.1. System Model

- Workers: A *worker* in MCS architecture is someone who participates in crowdsensing application by performing a specific assigned task [2]. A worker can be any mobile user that can participate in either a sensory task or computing task. A worker involved in any *sensory task* collects the data required using its sensors and would send the data to platforms for further processing. On the other hand, a worker can also perform *computation tasks* on the data collected, like aggregation or deriving output and submitting it to platforms. The platforms will then aggregate and evaluate the best output and present it to the users.

  In our security model, we focus on workers that perform sensory tasks and records the required data. These workers encrypt the data using PGP encryption scheme and send it to platforms for further processing.

- Users: In a crowd-sending application, users request the data from platforms so that they can derive useful insights from it. These users lack the necessary infrastructure to generate the data they need. So, they request the platforms for the data at a certain cost.

- Platforms: The role of a *platform* in MCS architecture can be played by an organization that provides a platform for a crowdsourcing application. A platform deals with managing all the task requests received from users, identifies proper workers for that tasks and forwards the task to them [2]. Once a platform receives data from the workers, it computes the best output and forwards that to the user who requested it. These platforms are at high security risk as these can be targeted by attackers to compromise the data collection. Also, these platforms themselves can

numerous launch attacks on the data collected [5]. Hence these platforms can become a single point of failure in MCS architecture.

In our security model, we propose to use a platform with multiple servers for the MCS architecture. This avoids the problem of having single point of failure. We propose to use three servers (say P1, P2 and P3) that need to work together to make the application work successfully. A Certificate Authority (CA) distributes the public key (K_{Pu}^{P}) and the private key (K_{Pr}^{P}) among different servers. The public key is same and is shared among all servers, but the private key is shared using a (2,3)-secret sharing scheme. The private key (say X) is divided into three shares x_1, x_2 and x_3 such that by combining all the three shares, you can obtain the private key i.e.,

$$X = x_1 + x_2 + x_3$$

Then, each server is distributed their share such that, P1 gets (x_1, x_2), P2 gets (x_2, x_3) and P3 gets (x_1, x_3), respectively. Any two of the servers can work together to recover the private key as shown in the Table 3.1.

Table 3.1. Our secret sharing scheme

	x_1	x_2	x_3
P1	✓	✓	
P2		✓	✓
P3	✓		✓
P1 ∪ P2	✓	✓	✓
P1 ∪ P2	✓	✓	✓
P1 ∪ P2	✓	✓	✓

3.2. System Model Assumptions

We propose the security model under the following assumptions:

- Sensory workers directly interact with all servers of the platform and can send the recorded data after encryption.
- A worker or a user can send data/query to any of the three servers of the platform for further processing.
- A Certificate Authority (CA) is assumed to be a trusted source and is used for distributing and maintaining keys in the architecture.
- All servers of the platforms are connected to each other and transfer of data between the servers is done securely.
- The communication channel between all entities is secure, i.e., no middleman can intercept any of the messages.
- Both the users and workers generate authentic data.
- Query generated by users is small to be small (up to 50KB)

3.3. Security Objectives

Based on above mentioned security model, we propose a MCS architecture that achieves the following security objectives:

- <u>For users</u>:
- It helps in maintaining confidentiality of the query by encrypting the query with RSA encryption using the platform's public key (K_{Pu}^{P}).
- It also helps in maintaining authenticity of the message sent (or query sent) to the platform by signing the text obtained from above using RSA digital signature algorithm with its own private key (K_{Pr}^{U}).

- For workers:
- To transfer data from workers to platforms, we use PGP encryption scheme. It helps in maintaining authenticity and integrity of the data collected by signing it using Digital Signature (DS).
- To maintain the confidentiality of the data collected, we encrypt the output from above by first using a shared session key (K_s) to get the first ciphertext. Then we encrypt shared key (K_s) using RSA with platform's public key (K_{Pu}^{P}) to get the second ciphertext. These ciphertexts are then sent to platform for further analysis.
- For platforms:
- Avoiding single point of failure in the architecture and preventing it from various security attacks. This is achieved by using a platform with multiple servers (P1, P2, and P3) instead of one and dividing the private into shares among the servers with the help of secret sharing scheme. For example, say X is the private key assigned to platforms by a CA. Now the CA divides the private key using (2,3)-secret sharing technique into three shares (x_1, x_2 and x_3) such that two servers must work together to recover the private key. This not only prevents attacks that can originate from servers themselves but also keeps the data safe in case any of the server is compromised.

4. PROPOSED FRAMEWORK

In this chapter, we provide the details for constructing our proposed scheme. Our scheme focuses on securely transferring the query/task from users to workers via platforms while maintaining confidentiality and authenticity of the query. Once the query is received by a worker, it solves the query and records the required data. Our scheme also proposes a mechanism to transfer the recorded data from workers back to the users while maintaining core principles of information security including confidentiality, authenticity, and integrity of the data.

4.1. System Initialization

The Certificate Authority (CA) plays a crucial role by assisting in system initialization as it is a trusted entity in our model. The CA performs the following role:

- Step 1: The CA first generates the pair of two prime number (*p* and *q*) for the RSA Encryption scheme for all the entities in the model. It then computes Public key (K_{Pu}) and Private key (K_{Pr}) for all entities and securely distributes them.
- Step 2: Each user who joins the MCS network is given a set of public and private keys. A user's public key is represented by K_{Pu}^{U} and the private key is represented by K_{Pr}^{U}.
- Step 3: Each worker who joins the MCS network is also given a set of public and private keys by the CA. A worker's public key is represented by K_{Pu}^{W} and the private key is represented by K_{Pr}^{W}.
- Step 4: The public-private key pair for the platform is divided and distributed among three servers. A platform's public key (represented by K_{Pu}^{P}) remains the same for all

servers of the platform. However, platform's private key (represented by K_{Pr}^{P}) is divided into three equal shares and distributed to among different servers using the (2,3)-secret sharing scheme, such that minimum two servers (the *threshold amount*) are required to recover the private key. For example, say the private key of platform (K_{Pr}^{P}) = X and is divided into three shares including x_1, x_2 and x_3, such that $x_1 + x_2 + x_3 = X$. These shares are then paired and distributed such that P1 gets (x_1, x_2), P2 gets (x_2, x_3), P3 gets (x_1, x_3). Now, any of these servers must first work together to recover the private key (using Lagrange Interpolation), and then decrypt the data received from either users or workers.

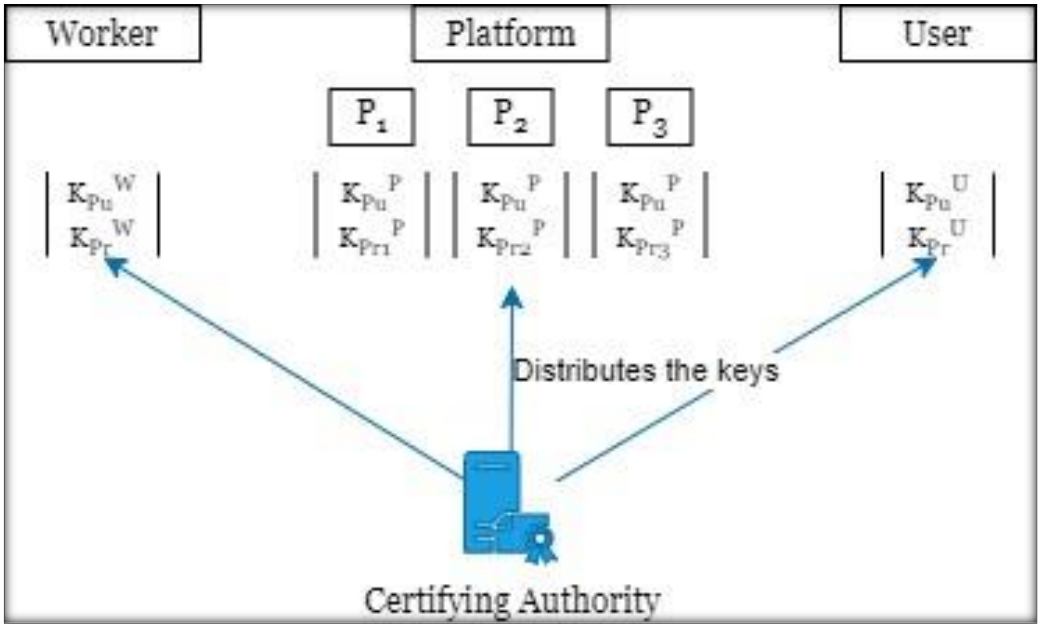


Figure 4.1. CA distributes the key-pair to all entities

The workers preform the following role for initialization:

- Step 1: Since transferring of queries follow's PGP encryption scheme, workers also need to generate a shared session key (represented by K_s) for encrypting and sending the recorded data to platforms.

Once the MCS architecture is initialized, a user initiates the communication by sending a query for a task to any server (P1, P2, or P3). A server, upon receiving the query starts the decryption process by working together with other servers in the architecture. Once the query is extracted, the platform selects the best worker(s) to solve that task and then encrypts the query again before assigning it to specific workers.

4.2. Secure transmission of the Query

In our model, we consider that a user generates a query which consists of a task that needs to be solved. These tasks can either include tasks like recording temperature data, speed, motions etc., which requires minimal to no interaction from workers, or tasks like capturing a photo of public places, recording physical activity etc., which requires maximum interaction from the workers. The generated query is encrypted before the user sends it to the platform to get it transferred to the workers.

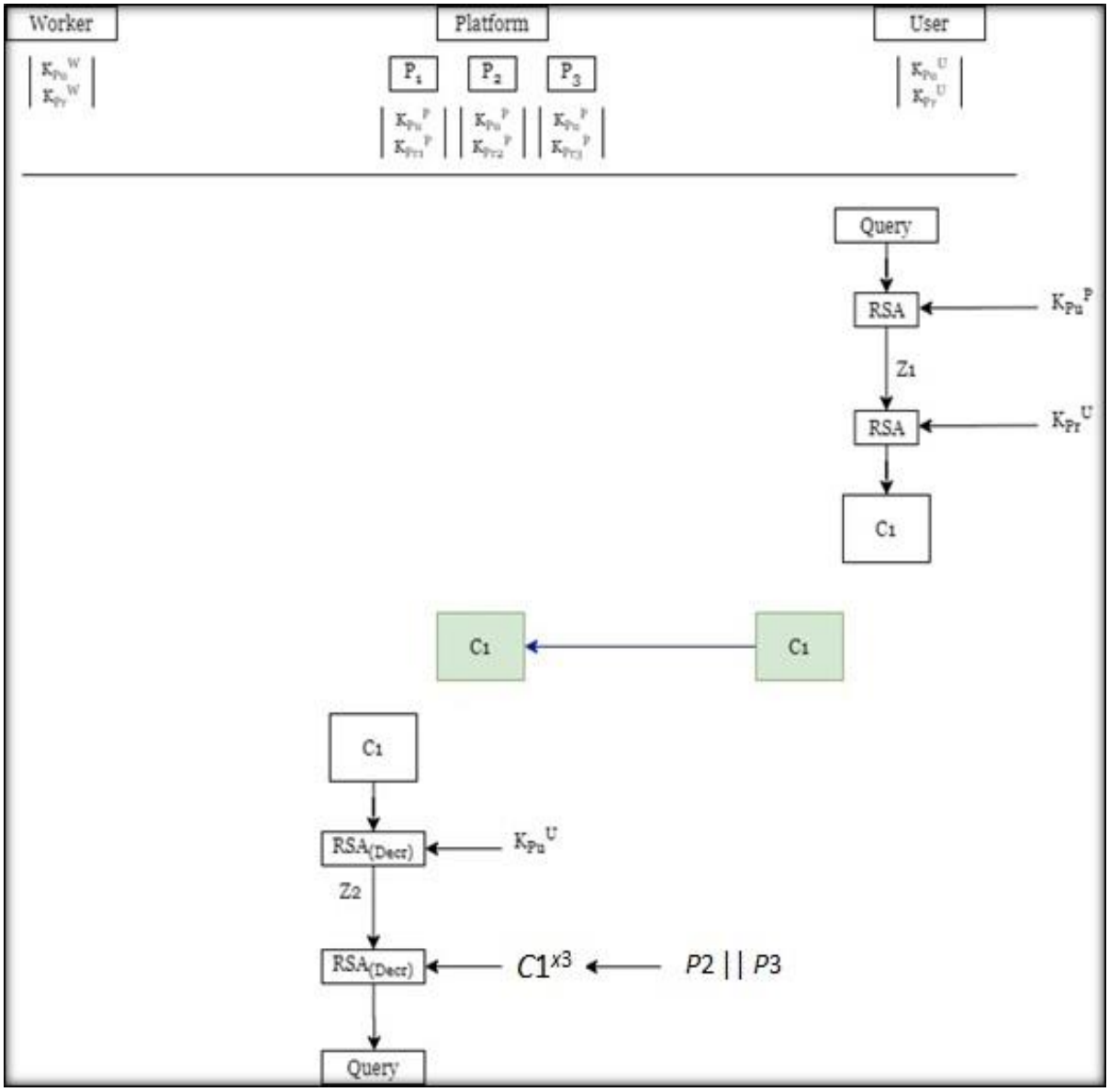


Figure 4.2. Secure transfer of query from user to platform

A user uses the following procedure to encrypt the query as follows:

- Step 1: A user firsts encrypts the query (Q) using RSA encryption with the help of platform's public key (K_{Pu}^{P}) to get an intermediate ciphertext (Z1).
- Step 2: It then signs the output from above (Z1) again using RSA digital signature with user's own private key (K_{Pr}^{U}) to get the final ciphertext (C1). This

is known as *signature generation* as this ciphertext can only be verified using user's public key.

- Step 3: It then sends the final ciphertext to a server of the platform for further processing.

Upon receiving the query, the platform analyzes the requirements of the task and based on that analysis it selects a group of workers to perform the task. The assignment of a task is determined by the requirements of the task as worker selection and task allocation are based on the properties of workers, such as their abilities, locations, interests, etc. Once a server receives a request for a task from users, it first decrypts the message by working together with other servers. The procedure is as follows:

- Step 1: A server (say P1) first verifies the signature of ciphertext (C1) using RSA with the user's public key (K_{Pu}^{U}).
- Step 2: It then sends C1 to one of the other servers in the network and the server will return $C1^{x_3}$ to it.
- Step 3: As P1 holds the shares (x_1, x_2) of the secret key, it then deciphers the query in plaintext by computing $C1^{x_3} \cdot C1^{x_1+x_2}$.

Once a query is successfully recovered by a platform, it is forwarded to a worker by encrypting the query again using the same procedure. A platform (or a server of the platform) forwards the query to worker by using following procedure:

- Step 1: Encrypting the query (Q) using RSA encryption with the help of worker's public key (K_{Pu}^{w}) to get a ciphertext (Z2).
- Step 2: P_1 then must sign the output from above (Z2) again using RSA encryption with platform's own private key (K_{Pr}^{P}) to get the final ciphertext

(C2). This is known as *signature generation* as this ciphertext can only be decrypted using platform's public key. A server again needs to work with other servers to recover $(Z2)^{x_3}$. It then encrypts the output with its shares to get $(Z2^{x_3} \cdot Z2^{x_1+x_2})$ = ((Z2)K_{Pr}^{P}) the final cipher text (C2).

- Step 3: It then selects the best suitable workers for the task and then sends the final ciphertext (C2) to the workers for data collection and further processing.

Once a worker receives the encrypted query from platforms, it needs to decrypt the data using the same procedure, to read the query and perform the specified task. The decryption process of query on worker's side is as follows:

- Step 1: A worker first verifies the ciphertext (C2) using RSA with the platform's public key (K_{Pu}^{P}) to get the intermediate ciphertext (Z2). This step is called *signature verification*.
- Step 2: It then uses his own private key (K_{Pr}^{W}) and RSA decryption function, on the intermediate ciphertext (Z2) obtained from the above step, to get the query in plaintext.

Once the query is received by a worker, it then performs the task and records the specified data. This data is then sent back to the user, via platform, using PGP encryption scheme.

4.3. Secure transmission of the Data

Once the data required for the task is generated (represented as *message*, or *m*) by workers, they start encrypting the data using PGP encryption scheme and send it over to platforms. For PGP, workers must generate a random shared key (also known as session key, or K_S). This is used to encrypt the data along with its digital signature (DS) using AES. The data generated is represented as M.

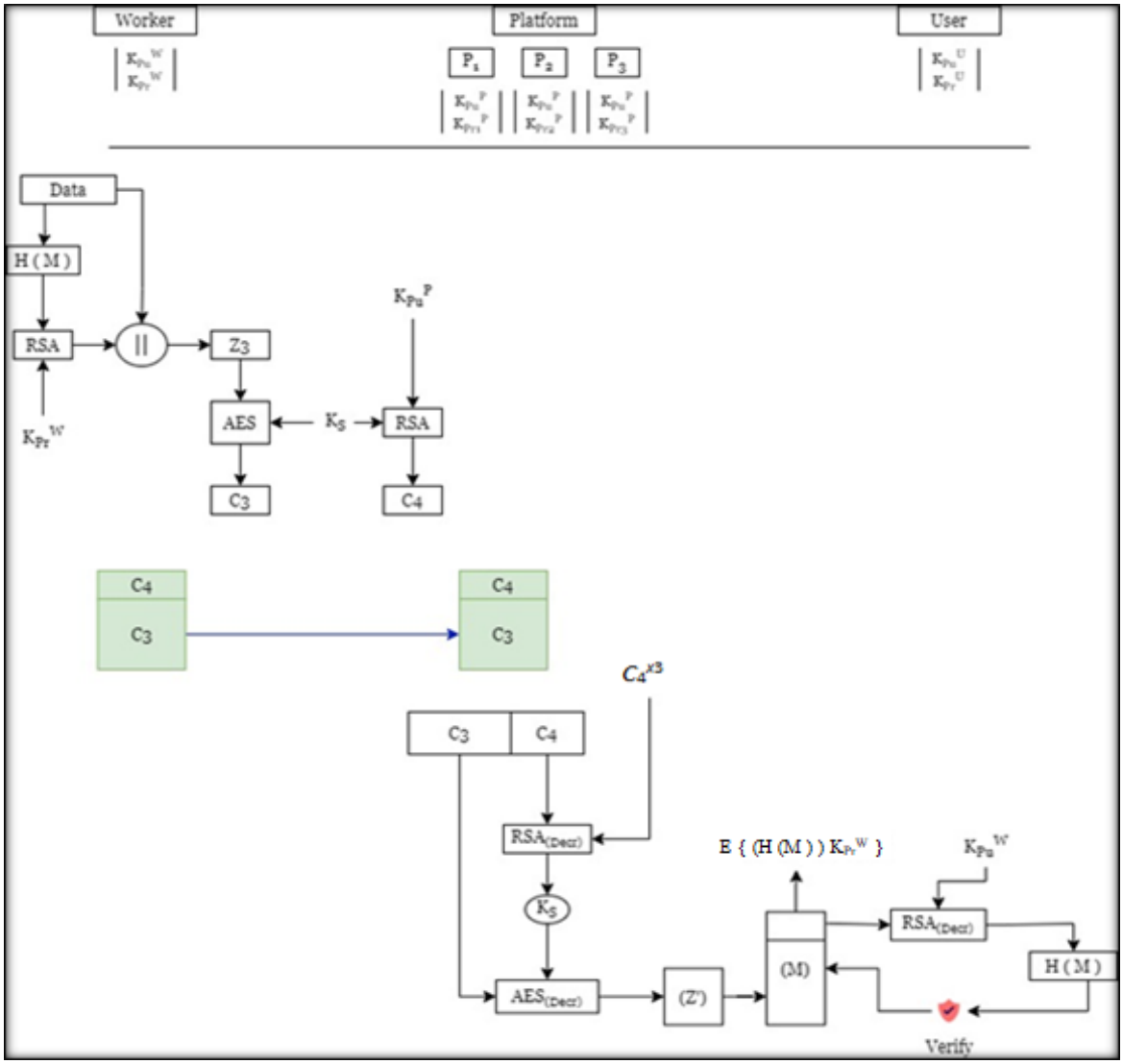


Figure 4.3. Secure transfer of messages from Workers to Platforms

The encryption procedure for workers is as follows:

- Step 1: Worker calculates the hash of the message H(m).
- Step 2: The hash H(m) is then encrypted (signed) with worker's private key (K_{Pr}^{W}) using RSA encryption to create a *digital signature* of the message. Hash maintains the integrity of the message while RSA maintains confidentiality.
- Step 3: This digital signature created is then appended to the message. This output is referred as Z3.
- Step 4: The output Z3 is then encrypted with the shared key generated (K_s) using AES encryption. This output is the first ciphertext or, C3 which is sent to a platform.
- Step 5: The shared secret (K_s) is also encrypted with platform's public key (K_{Pu}^{P}) using RSA encryption to generate the second ciphertext, or C4.
- Step 6: Both ciphertexts C3 and C4 are then sent to a platform that shared the query (say P1) for further processing.

Once a platform (or a server of the platform) receives both ciphertexts, it can start the decryption process to get the data back in plaintext. The decryption process is as follows:

- Step 1: The ciphertext C4 is first decrypted. As discussed above, since platform P1 has a partial-private key (x_1, x_2), it can work together with either platforms P2 (x2, x3) or P3 (x_1, x_3) to decrypt the ciphertext without revealing the secret key. This helps in decrypting the shared key (K_s) generated by worker and used for AES.

- Step 2: The ciphertext C3 is then decrypted using the shared key (K_s) by AES decryption. This helps in recovering Z3 which is data (or m) appended with its digital signature (DS).
- Step 3*:* Then the signature is used to verify the integrity of the message, or the data generated.

Once a platform (or a server of the platform) successfully decrypts the message (m), it then encrypts the data using similar implementation of PGP encryption scheme. The encryption procedure for a platform is as follows:

- Step 1: A server can calculate the hash of the message H(m) or use the hash from previous steps. The it works together with other servers to recover (H(m) ^{x3}) and uses its shares to sign it to obtain ($H(m)^{x_3} \cdot H(m)^{x_1+x_2}$) = (H(m)) K_{Pr}^{P}). This step is known as generating *digital signature* of the message using RSA and platforms private key K_{Pr}^{P}.
- Step 2: This digital signature created is then appended to the message. This output is referred as Z4.
- Step 3: The output Z4 is then encrypted with the shared key generated (K_s) using AES encryption. This output is the first ciphertext or, C5.
- Step 4: The shared secret (K_s) is also encrypted with user's public key (K_{Pu}^{U}) using RSA encryption to generate the second ciphertext, or C6.
- Step 6: Both ciphertexts C5 and C6 are then sent to the user that shared the query for decryption.

Once the user who requested the data receives the ciphertext, he needs to decrypt that text to get the data in plaintext. The decryption process is as follows:

- Step 1: The ciphertext C6 is first decrypted with RSA decryption using user's private key (K_{Pr}^{U}) to obtain the shared key (K_s) generated by the worker and shared by the platform.
- Step 2: The ciphertext C5 is then decrypted using the shared key (K_s) by AES. This helps in recovering Z4 which is data (or m) appended with its digital signature (DS).
- Step 3: Then the signature is used to verify the integrity of the message, or the data generated.

5. SECURITY ANALYSIS AND PERFORMANCE EVALUATION

In this chapter, we analyze the security of our proposed secure communication scheme between entities in mobile crowd sourcing application. We focus on analyzing the proposed secure mechanism for transferring data generated by *workers* using PGP scheme along with AES encryption. We also analyze our mechanism to preserve privacy of the query generated by a worker while maintaining authenticity and confidentiality of the query. We also analyze the security of dividing the private key among platforms. A critical role is played by Certificate Authority (CA) in our scheme as it is responsible for secure distribution of keys to all the entities involved in communication.

In the later section, we discuss the results of performance evaluation of the proposed scheme. Our evaluation focuses on three aspects – analysing the cost for encrypting large data sets (data generated by *workers*) by using AES encryption; analysing the cost for encrypting small data sets (*queries* generated by *users*) using RSA encryption; and, finally cost of generating digital signatures (DS) of large data sets.

5.1. Security Analysis

In this section, we analyze the security of our proposed scheme. Our scheme focuses on – maintaining authenticity and confidentiality of the generated *query* while transfer; and maintaining the same security principles, including integrity, of the generated *data* while transfer. We also analyze the strength of secret sharing and benefits of having a platform with multiple servers.

- <u>Queries (q)</u>:

For query, we presume a user generates a query (q) and send it to a server of the platform using RSA encryption and digital signature. This works by encrypting the message – first with platform's public key pair (K_{Pu}^{P}), and then with user's private key pair (K_{Pr}^{U}).

$$(q) \rightarrow E\{(q)K_{Pu}^{P}\} \rightarrow E\{((q)K_{Pu}^{P})K_{Pr}^{U})\}$$

This message is then sent over to a server for decryption. An adversary gets hold of the transmitted data, but they cannot recover the query as they won't have platform's private key (K_{Pr}^{P}).

- <u>Data (M)</u>:

After receiving the query, workers start to gather the required data (M) for that task. Once data is collected, its digital signature is generated and attached to it. This is done by calculating the hash of the message H(m), and then signing this hash with platform's private key (K_{Pr}^{P}) using RSA. This signature is appended with the data and the output is then processed for encryption.

$$M \rightarrow H(M) \rightarrow M \,||\, E\{(H(M))K_{Pr}^{P}\}$$

The output of the previous step (represented as Z) is then encrypted using AES along with a random generated shared key (K_s) to generate the first ciphertext (C_1).

$$(Z) \rightarrow E\{(Z)K_s\} \rightarrow C_1$$

Second ciphertext (C_2) is generated by encrypting the shared key (Ks) using RSA along with platform's public key (K_{pu}^P).

$$(K_s) \rightarrow E \{ (K_s) K_{Pu}^P \} \rightarrow C_2$$

Both ciphertexts are then sent to platforms for decryption. If an adversary intercepts both ciphertexts, they would need platform's private key (K_{Pr}^P) to start the decryption process.

- <u>Platforms</u>:

According to our scheme, a platform with three servers are set up as P1, P2 and P3. The Certificate Authority (CA) distributes platform's private key shared using the (2,3)-secret sharing scheme. The private key (say X) is divided into x_1, x_2 and x_3 such that,

$$X = x_1 + x_2 + x_3$$

Then, each server is distributed their shares such that P1 gets (x_1, x_2), P2 gets (x_2, x_3) and P3 gets (x_1, x_3) respectively. Without revealing the private key, any two servers can work together to decrypt the messages using the above-mentioned scheme.

The privacy of the data transferred to a platform's server, either by worker or by user, is protected because neither of the servers can decrypt the data on their own, unless there is a collusion attack [5] between two servers. This also prevents in having a single-point-of-failure in the architecture in-case one server goes offline or in-case of cyberattack, the other two servers take over. This provides *high availability* to our system.

5.2. Performance Analysis

In this section, we evaluate the performance of our proposed scheme. The next sections describe our system configuration and then present our results for encryption.

5.2.1. Data Sets

In our implementation, we use total six different sized data files divided in two sets – larger set and smaller set. The larger set consists of three large sized files – 10 Megabytes, 100 Megabytes and 500 Megabytes. The smaller set consists of three small sized files – 1 Kilobytes, 10 Kilobytes and 50 Kilobytes. We use online random file generator to create these files (www.pinetools.com/random-file-generator).

5.2.2. System Configuration

We implement and run our system on a laptop with following specs as shown in Table 5.2.2:

Table 5.2.2. System configuration details

Manufacturer	Acer
Model	Swift SF314-54
Processor	Intel® Core™ i5-8250U with 1.80 GHz
RAM	8 GB
OS	Windows 10

We use python programming language to code and implement our scheme. Python is an object-oriented scripting language which supports various inbuilt packages, like Pycryptodome, which is a package of low-level cryptographic primitives. We implement and compare encryption cost of various encryption algorithms including – AES-128, AES-192 and AES-256. We also compare the encryption/decryption

cost using RSA with 1024-bit key. Moreover, we also analyze the cost of generating and verifying digital signatures of large data sets.

5.2.3. Evaluation Results

We present our implementation and analysis in three parts:

- Digital Signature Cost:

Digital signature of a file is generated by first calculating a hash of a file. Then, this hash is encrypted using an asymmetric encryption algorithm (like RSA) along with user's private key. We generate digital signatures of three files with different sizes – 10MB, 100MB and 500MB – using RSA python's NaCl library (www.pynacl.readthedocs.io/en/stable/signing). Table 5.2.3.a denotes the time taken by our system to generate the signatures.

Table 5.2.3.a Time taken (sec) to generate digital signatures

File Size	Generate Signature	Verify Signature
10 MB	0.309	0.151
100 MB	3.484	1.37
500 MB	15.214	5.617

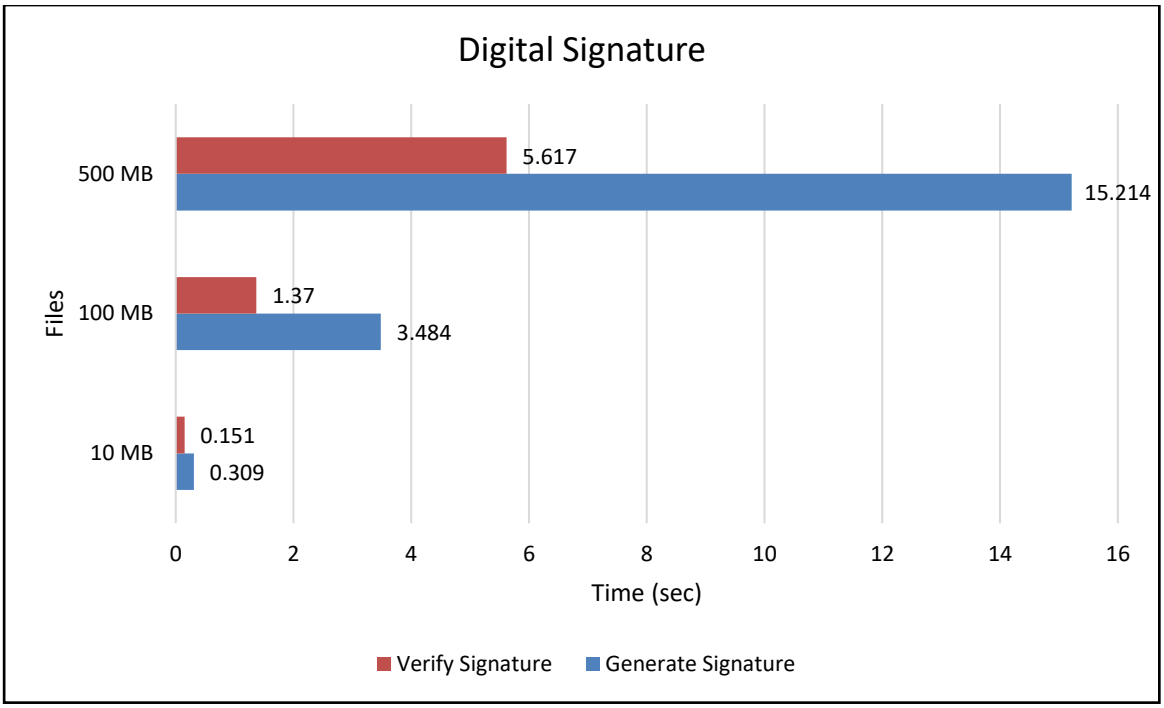


Figure 5.2.3.a Time taken (sec) to generate digital signatures

From the Figure 5.2.3.a we observe that generating signatures takes almost thrice as much than verifying them. We also note that verification of signatures is much faster than generation, when we move to larger files.

- <u>AES Encryption Cost</u>:

In our implementation, we encrypt the three sets of large files using AES-128, AES-196 and AES-256. These variations use different size of keys. AES-128 uses 128-bit key (16 bytes), AES-192 uses 192-bit key (24 bytes) and AES-256 uses 256-bit key (32 bytes). Table 5.2.3.b represents the time taken to encrypt each file using all three variations of AES individually.

Table 5.2.3.b Time taken (sec) by AES-(128, 192 and 256) to encrypt all three files

AES Type	Files					
	10 MB		100 MB		500 MB	
	Encryption	Decryption	Encryption	Decryption	Encryption	Decryption
AES-128	1.002	0.983	10.01	9.536	48.673	46.595
AES-192	0.977	0.945	9.541	9.423	49.574	49.088
AES-256	0.973	0.946	9.566	9.674	49.726	47.86

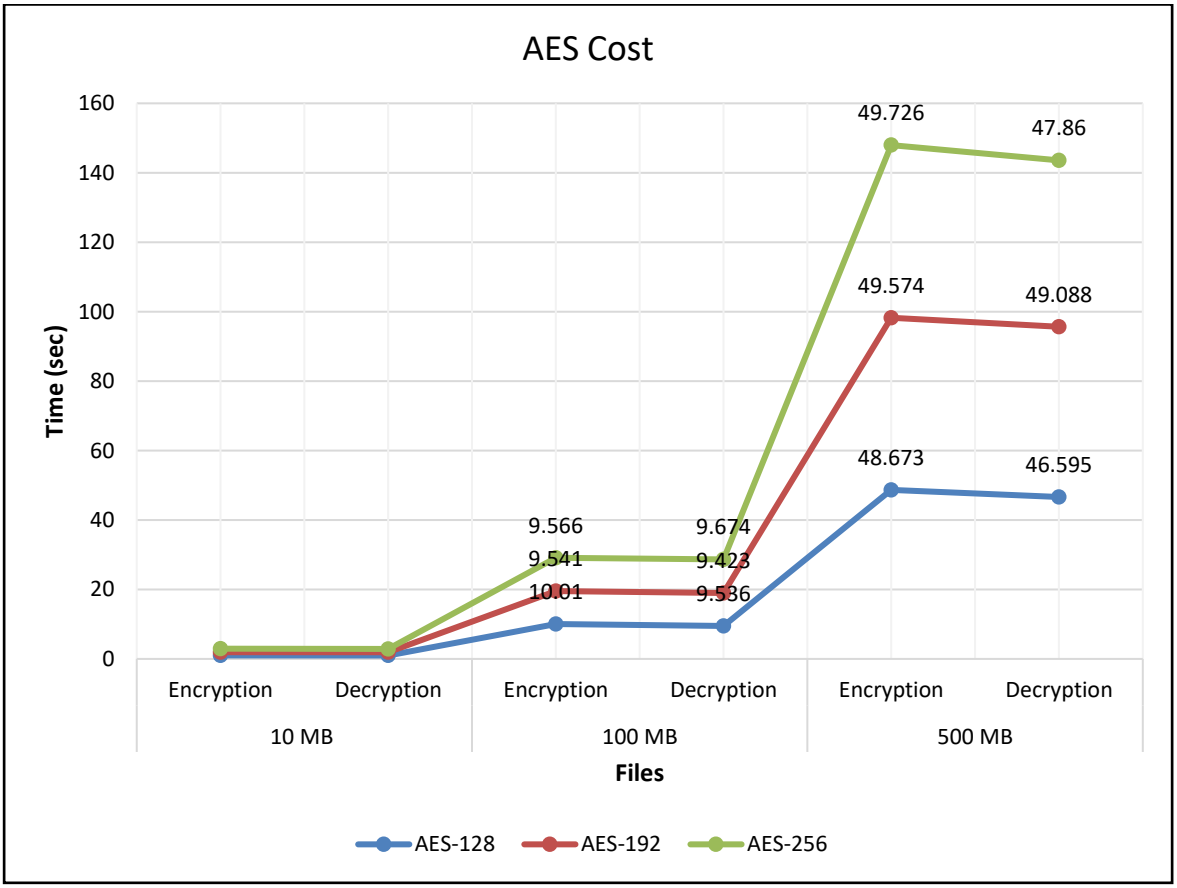


Figure 5.2.3.b Time taken (sec) by AES-(128, 192 and 256) to encrypt all three files

From the Figure 5.2.3.b, we observe that encryption/decryption cost are almost equal

for the same file size when using either AES-128, AES-192 or AES-256. It is recommended to use the strongest encryption using the key of size 32-bytes. We also note that the encryption cost increases when we encrypt larger files.

- RSA Encryption Cost:

As the user queries are encrypted using RSA algorithm, we assume that the queries have small size. Hence, we encrypt the set of three small files – 1KB, 10KB and 50KB – using RSA with 1024-bit key. Table 5.2.3.c denotes the encryption and decryption cost using RSA.

Table 5.2.3.c Time taken (sec) by RSA to encrypt all three size

File Size	RSA (key = 1024 bits)	
	Encryption Cost	Decryption Cost
1 KB	0.662	0.683
10 KB	0.907	1.414
50 KB	3.561	8.736

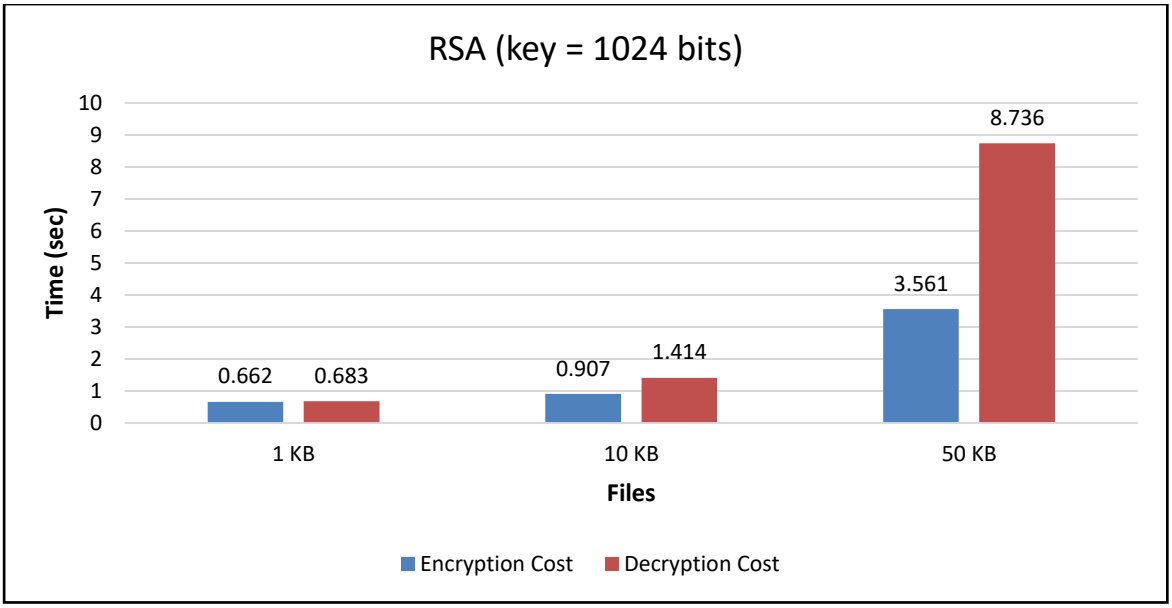


Figure 5.2.3.c Time taken (sec) by RSA to encrypt all three size

From the Figure 5.2.3.c, we observe that decryption cost is higher than encryption cost using RSA. We also note that decryption cost increases exponentially when we move to larger files. We recommend using 2048 or higher bit key for more security. Note that queries are considered of small sizes. If query size gets bigger in future, we recommend using a different approach as RSA is not fast for encrypting large files.

6. CONCLUSION AND FUTURE WORK

In this chapter, we summarize our contribution in this research. In the later section, we discuss some directions for future work and give our final remarks.

6.1. Conclusion

In this research we proposed a secure communication scheme between entities for mobile crowd sourcing applications. Our scheme focuses on maintaining confidentiality, authenticity and availability of the data generated by the workers. This is achieved by using PGP encryption scheme to securely transfer the data. Our scheme also ensures secure transfer of queries from users to workers whilst preserving authenticity and confidentiality of the query. Furthermore, we achieve high availability for platforms by using multiple servers in the architecture and dividing platform's private key-pair among servers. The private key-pair is divided using (2,3)-secret sharing, which allows us to divide and share a secret among group of three users such that two users can work together and recover the shared secret. Finally, we conduct a performance evaluation of our scheme. Our experiments demonstrate efficiency, privacy properties and the encryption/decryption cost on different data sets.

6.2. Future Work

In this section we discuss future work related to our work. In our security model we considered both users and workers to submit genuine data, and the communication channel between all entities to be secure. However, it is not the case in real time. Both workers and users can generate faulty data. Users can generate malicious queries which

can help in narrowing down number of users for that task. This can enable a user to reveal a worker's identity. On the other hand, users can submit unreliable data just to get incentives for any particular task. Even though our work provides a secure communication scheme to transfer data, MCS application is still vulnerable due to attacks including attacks from intercepting the data like man-in-the-middle, or attacks due to faulty data. In the future, we will consider addressing these challenges and present a better and more secure MCS architecture. Also, we consider users to generate query of small sizes, however, with ever-increasing size of data, using RSA is not an efficient way to transfer larger data files. We will work on presenting a secure way to transfer user queries with minimal cost in the future.

REFERENCES

[1] Basudan, S., Lin, X., & Sankaranarayanan, K. (2017). A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing. *IEEE Internet of Things Journal*, *4*(3), 772–782. https://doi.org/10.1109/JIOT.2017.2666783

[2] Feng, W., Yan, Z., Zhang, H., Zeng, K., Xiao, Y., & Hou, Y. T. (2018). A Survey on Security, Privacy, and Trust in Mobile Crowdsourcing. *IEEE Internet of Things Journal*, *5*(4), 2971–2992. https://doi.org/10.1109/JIOT.2017.2765699

[3] Li, Y., Xiao, H., Qin, Z., Miao, C., Su, L., Gao, J., Ren, K., & Ding, B. (2018). Towards Differentially Private Truth Discovery for Crowd Sensing Systems. *ArXiv:1810.04760 [Cs]*. http://arxiv.org/abs/1810.04760

[4] Shin, M., Cornelius, C., Kapadia, A., Triandopoulos, N., & Kotz, D. (2015). Location Privacy for Mobile Crowd Sensing through Population Mapping †. *Sensors (Basel, Switzerland)*, *15*(7), 15285–15310. https://doi.org/10.3390/s150715285

[5] Pournajaf, L., Garcia-Ulloa, D. A., Xiong, L., & Sunderam, V. (2016). *Participant Privacy in Mobile Crowd Sensing Task Management: A Survey of Methods and Challenges*. Association for Computing Machinery. https://doi.org/10.1145/2935694.2935700Fa

[6] Christin, D. (2016). Privacy in mobile participatory sensing: Current trends and future challenges. *Journal of Systems and Software*, *116*, 57–68. https://doi.org/10.1016/j.jss.2015.03.067

[7] Li, T., Jung, T., Qiu, Z., Li, H., Cao, L., & Wang, Y. (2018). Scalable Privacy-Preserving Participant Selection for Mobile Crowdsensing Systems: Participant Grouping and Secure Group Bidding. *IEEE Transactions on Network Science and Engineering*, 1–1. https://doi.org/10.1109/TNSE.2018.2791948

[8] Cheng, L., Niu, J., Kong, L., Luo, C., Gu, Y., He, W., & Das, S. (2017). Compressive Sensing based Data Quality Improvement for Crowd-Sensing Applications. *Journal of Network and Computer Applications*, *77*, 123–134.

https://doi.org/10.1016/j.jnca.2016.10.004

[9] Truong, N. B., Lee, G. M., Um, T.-W., & Mackay, M. (2019). Trust Evaluation Mechanism for User Recruitment in Mobile Crowd-Sensing in the Internet of Things. *IEEE Transactions on Information Forensics and Security*, *14*(10), 2705–2719. https://doi.org/10.1109/TIFS.2019.2903659

[10] Liu, Y.-N., Wang, Y.-P., Wang, X.-F., Xia, Z., & Xu, J. (2018). Privacy-Preserving Data Collection for Mobile Phone Sensing Tasks. *ISPEC*. https://doi.org/10.1007/978-3-319-99807-7_32

[11] Agadakos, I., Polakis, J., & Portokalidis, G. (2017). Techu: Open and Privacy-Preserving Crowdsourced GPS for the Masses. *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '17*, 475–487. https://doi.org/10.1145/3081333.3081345

[12] Yang, L., Zhang, M., He, S., Li, M., & Zhang, J. (2018). Crowd-Empowered Privacy-Preserving Data Aggregation for Mobile Crowdsensing. *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 151–160. https://doi.org/10.1145/3209582.3209598

[13] Jin, H., Su, L., Ding, B., Nahrstedt, K., & Borisov, N. (2016). Enabling Privacy-Preserving Incentives for Mobile Crowd Sensing Systems. *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 344–353. https://doi.org/10.1109/ICDCS.2016.50

[14] Sommerard, R., & Rouvoy, R. (2016). *Towards Privacy-Preserving Data Dissemination in Crowd-Sensing Middleware Platform*. 6. https://hal.inria.fr/hal-01332588

[15] Guo, B., Wang, Z., Yu, Z., Wang, Y., Yen, N. Y., Huang, R., & Zhou, X. (2015). *Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm*. Association for Computing Machinery. https://doi.org/10.1145/2794400

[16] Zhang, C., Zhu, L., Xu, C., Sharif, K., Liu, X., Du, X., & Guizani, M. (2019). Achieving Trust-Based and Privacy-Preserving Customer Selection in Ubiquitous Computing. *ArXiv*.

[17] Restuccia, F., Ghosh, N., Bhattacharjee, S., Das, S. K., & Melodia, T. (2017). Quality of Information in Mobile Crowdsensing: Survey and Research Challenges. *TOSN*. https://doi.org/10.1145/3139256

[18] Luo, G., Yan, K., Zheng, X., Tian, L., & Cai, Z. (2018). Preserving adjustable path privacy for task acquisition in Mobile Crowdsensing Systems. *Information Sciences*. https://doi.org/10.1016/j.ins.2018.12.013

[19] Wang, G., Wang, B., Wang, T., Nika, A., Zheng, H., & Zhao, B. Y. (2016). Defending against Sybil Devices in Crowdsourced Mapping Services. *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 179–191. https://doi.org/10.1145/2906388.2906420

[20] Goubin, L., & Martinelli, A. (2011). Protecting AES with Shamir's Secret Sharing Scheme. In B. Preneel & T. Takagi (Eds.), *Cryptographic Hardware and Embedded Systems – CHES 2011* (pp. 79–94). Springer. https://doi.org/10.1007/978-3-642-23951-9_6

[21] Bellare, M., & Miner, S. K. (1999). A Forward-Secure Digital Signature Scheme. In M. Wiener (Ed.), *Advances in Cryptology—CRYPTO' 99* (pp. 431–448). Springer. https://doi.org/10.1007/3-540-48405-1_28 **{{N07}}**

[22] Khalique, A., Singh, K., & Sood, S. (2010). Implementation of Elliptic Curve Digital Signature Algorithm. *International Journal of Computer Applications*, *2*(2), 21–27. https://doi.org/10.5120/631-876

[23] Harn, L., Mehta, M., & Wen-Jung Hsin. (2004). Integrating Diffie-Hellman key exchange into the digital signature algorithm (DSA). *IEEE Communications Letters*, *8*(3), 198–200. https://doi.org/10.1109/LCOMM.2004.825705

[24] Fujisaki, E., & Okamoto, T. (2013). Secure Integration of Asymmetric and Symmetric Encryption Schemes. *Journal of Cryptology*, *26*(1), 80–101. https://doi.org/10.1007/s00145-011-9114-1

[25] Bellare, M., & Rogaway, P. (n.d.). *Optimal Asymmetric Encryption How to Encrypt with RSA*. 19.

[26] Simmons, G. J. (1979). *Symmetric and Asymmetric Encryption*. Association for Computing Machinery. https://doi.org/10.1145/356789.356793

[27] Yassein, M. B., Aljawarneh, S., Qawasmeh, E., Mardini, W., & Khamayseh, Y. (2017). Comprehensive study of symmetric key and asymmetric key encryption algorithms. *2017 International Conference on Engineering and Technology (ICET)*, 1–7. https://doi.org/10.1109/ICEngTechnol.2017.8308215

[28] Kuobin, D. (2011). PGP E-Mail Protocol Security Analysis and Improvement Program. *2011 International Conference on Intelligence Science and Information Engineering*, 45–48. https://doi.org/10.1109/ISIE.2011.144

[29] Singh, M., & Garg, D. (2009). Choosing Best Hashing Strategies and Hash Functions. *2009 IEEE International Advance Computing Conference*, 50–55. https://doi.org/10.1109/IADCC.2009.4808979

[30] Priya, B. A., & Sheshasaayee, A. (2016). Effective design of a parametrical security model for digital signatures using cryptography. *2016 International Conference on Communication and Electronics Systems (ICCES)*, 1–3. https://doi.org/10.1109/CESYS.2016.7889959

[31] Arom-oon, U. (2017). An AES cryptosystem for small scale network. *2017 Third Asian Conference on Defence Technology (ACDT)*, 49–53. https://doi.org/10.1109/ACDT.2017.7886156

[32] Bonde, S. Y., & Bhadade, U. S. (2017). Analysis of Encryption Algorithms (RSA,

SRNN and 2 Key Pair) for Information Security. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 1–5. https://doi.org/10.1109/ICCUBEA.2017.8463720

[33] Shamir, A. (1979). *How to share a secret*. Association for Computing Machinery. https://doi.org/10.1145/359168.359176

[34] https://github.com/shrayyy/final-year-project

APPENDIX A

AES-thegreenplace-decrypt-timed.py

```
import timeit


setup_code = """

import os, random, struct
from Crypto.Cipher import AES
"""

statement = """

def decrypt_file(key, in_filename, out_filename=None, chunksize=24*1024):

    if not out_filename:
        out_filename = os.path.splitext(in_filename)[0]

    with open(in_filename, 'rb') as infile:
        origsize = struct.unpack('<Q', infile.read(struct.calcsize('Q')))[0]
        iv = infile.read(16)

       decryptor = AES.new(key, AES.MODE_CBC, iv)

        with open(out_filename, 'wb') as outfile:
            while True:
                chunk = infile.read(chunksize)

                if len(chunk) == 0:
                    break

                outfile.write(decryptor.decrypt(chunk))

            outfile.truncate(origsize)


#key2 = b'0123456789abcdef'      # 16 byte

#key2 = b'0123456789abcdef01234567'     # 24 byte

key2 = b'0123456789abcdef0123456789abcdef'     # 32 byte
```

```
decrypt_file(key2, "..\\data\\MB500-E-AES-256.encr", "..\\data\\MB500-D-AES-256.recovered.file", 64)

"""


print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number = 5)/5}")
```

APPENDIX B

AES-thegreenplace-encrypt-timed.py

```
import timeit

setup_code = """
import os, random, struct
from Crypto.Cipher import AES
"""

statement = """

def encrypt_file(key, in_filename, out_filename=None, chunksize=64*1024):

    if not out_filename:
        out_filename = in_filename + '.enc'

    iv = (16 * '\x01')

    x = bytes(iv, 'utf8')

    print(x,len(x))

    encryptor = AES.new(key, AES.MODE_CBC, x)
    filesize = os.path.getsize(in_filename)

    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:

            outfile.write(struct.pack('<Q', filesize))
            outfile.write(x)

            while True:
                chunk = infile.read(chunksize)

                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b' ' * (16 - len(chunk) % 16)

                outfile.write(encryptor.encrypt(chunk))


key2 = b'0123456789abcdef0123456789abcdef'  # 16 byte key, can use bigger - more
bigger means more stronger
```

```
encrypt_file(key2,"..\\data\\MB500.file", "..\\data\\MB500-E-AES-256.encr", 64)

"""

print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number =
5)/5}")
```

APPENDIX C

DS-pynacl-generate-timed.py

```
import timeit

setup_code = """

import nacl.encoding
import nacl.signing
"""

statement = """

# Generate a new random signing key
signing_key = nacl.signing.SigningKey.generate()

# Sign a message with the signing key

#signed = signing_key.sign(b"Attack at Dawn")

with open("..\\data\\MB500.file",'rb') as infile:    # change filename
    msg = infile.read()
    signed = signing_key.sign(msg)


# Obtain the verify key for a given signing key
verify_key = signing_key.verify_key


# Serialize the verify key to send it to a third party
verify_key_hex = verify_key.encode(encoder=nacl.encoding.HexEncoder)


#print(verify_key_hex)
print('signed')


sig_file = open("..\\data\\MB500-Gen-DS.sig",'wb')    # change filename
sig_file.write(verify_key_hex)
sig_file.close()
```

```
signed_file = open("..\\data\\MB500-SignedData-DS.data",'wb')    # change filename
signed_file.write(signed)
signed_file.close()

"""

print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number =
5)/5}")
```

APPENDIX D

DS-pynacl-verify-timed.py

```
import timeit

setup_code = """

import nacl.signing
"""

statement = """

sig_file = open("..\\data\\MB100-Gen-DS.sig",'rb')    # change filename

verify_key_hex = sig_file.read()
sig_file.close()


# Create a VerifyKey object from a hex serialized public key

verify_key = nacl.signing.VerifyKey(verify_key_hex,
encoder=nacl.encoding.HexEncoder)


# Check the validity of a message's signature
# The message and the signature can either be passed separately or
# concatenated together.  These are equivalent:

signed_file = open("..\\data\\MB100-SignedData-DS.data",'rb')    # change filename
signed = signed_file.read()
signed_file.close()


try:
    if verify_key.verify(signed):
        print('Signature Verified')
except:
    print('Signature Verification Failed!')
#verify_key.verify(signed.message, signed.signature)

"""
```

```
print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number =
5)/5}")
```

APPENDIX E

RSA-cryptobook-decrypt-timed.py

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii
import time

start_time = time.time()


# Import Private key

privateKeyFile = open("..\\data\\KB50-RSA-privKeyPair.pem",'r')      # change name
here
privKey = RSA.importKey(privateKeyFile.read())
privateKeyFile.close()

# Decryption

decryptor = PKCS1_OAEP.new(privKey)


in_filename = "..\\data\\KB50-E-RSA.encr"    # path to encrypted file
out_fileRvd = "..\\data\\KB50-D-RSA.recovered.file"      # Path to decrypted file


infile = open(in_filename, 'rb')
outfile = open(out_fileRvd,'wb')        #Recovered file


while True:
    chunk = infile.readline()
    chunk = chunk.strip(b'\n')

    if not chunk:
        break

    outfile.write(decryptor.decrypt(chunk))
```

```
print('Decrypted!')

infile.close()
outfile.close()

#print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number
= 5)/5}")

end_time = time.time()

print(f"Runtime of the program is {end_time - start_time}")
```

APPENDIX F

RSA-cryptobook-encrypt-timed.py

```
import timeit

setup_code = """

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii
import time
"""

statement = """

#Key generation

keyPair = RSA.generate(1024)


pubKey = keyPair.publickey()
#print(f"Public key: (n={hex(pubKey.n)}, e={hex(pubKey.e)})")
pubKeyPEM = pubKey.exportKey()
#print(pubKeyPEM.decode('ascii'))

privkey = keyPair.blind
#print(f"Private key: (n={hex(pubKey.n)}, d={hex(keyPair.d)})")
privKeyPEM = keyPair.exportKey()
#print(privKeyPEM.decode('ascii'))


# Export private key

privateKeyFile = open("..\\data\\KB50-RSA-privKeyPair.pem",'w')      # write private
key
privateKeyFile.write(privKeyPEM.decode('ascii'))
privateKeyFile.close()


# Input files

in_filename = "..\\data\\KB50.file"     # path to file
```

```
out_filename = "..\\data\\KB50-E-RSA.encr"    # path to output file


#Encryption

encryptor = PKCS1_OAEP.new(pubKey)


#Writing bytes into the file

first = 0
with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            while True:
                chunk = infile.read(16)

                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b' ' * (16 - len(chunk) % 16)      # Adding padding

                encr = encryptor.encrypt(chunk)

                outfile.write(encr)
                outfile.write(b'\\n')

print('Encryption complete.')

"""

print(f"Execution time is: {timeit.timeit(setup = setup_code, stmt = statement, number =
1)/1
```

VITA

Candidate's full name: Shreshth Kumar

Universities attended (with dates and degrees obtained): Bachelor of Technology in Information Technology, SRM University, Chennai, India, 2012-2016

Master of Science in Computer Science, University of New Brunswick, Fredericton, Canada, 2018-2020

Publications: In progress

Conference Presentations: None