# Multi-Party Secret Sharing Protocol

CS/ECE 438 Final Project

github.com/chitianhao/secret_sharing_protocol

Stanley Yang
xyang70@illinois.edu

Tianhao Chi
tchi3@illinois.edu

Tiancheng Wu
twu54@illinois.edu

Dec 2019

## Abstract

**Secret sharing has been a consistently popular concept in the security realm and gained continuous focus as block-chain technology thrives in recent years. In this project, we have a real-world implementation of such protocol that can be useful in many cases; for example, decentralized system secure communication. A secret is distributed among a group of people so that more than a specified number of people can reconstruct the secret back together.**

## 1 Introduction

Think about a case where a financial company A needs to store a passcode for a vault. Of course, we can have a dedicated passcode keeper to maintain the secret information. But what if the passcode keeper left the company and he did not give the passcode back to the company? What if a hacker hacked into the passcode keeper's laptop and stole the passcode? How safe is the vault going to be? The answer is not safe at all. So, is there any way we can let other employees know the passcode, but they can not access the vault by oneself? Hence, we introduce a technique called Multi-Party Secret Sharing (MPSS).

In such a Multi-Party Secret Sharing (MPSS) protocol, a dealer, which is only a peer of the P2P network, comes in and distributes the secret among n parties so that k parties can reconstruct the secret. After the distribution, the peers are able to construct their secret out with Byzantine faults tolerance. In this case, the protocol is implemented in the way that each participant has a encrypted file that is sent by the dealer with no key. But the dealer distributes the secret shares of the key, which can only be constructed back when they gather their shares together, to those peers. Hence, this is a very useful case in the real world.

## 2 Problem Statements

Secret Sharing, also called Shamir's Secret Sharing, is an algorithm in cryptography created by Adi Shamir [2]. It is a form of secret sharing, where a secret is divided into parts, giving each participant its unique part. MPSS is based on Shamir's Secret Sharing and is able to perform the process. We are trying to implement such a protocol that is a TCP based application layer protocol so that a dealer is able to distribute a secret among some peers in the network, and the secret can be reconstructed back.

In our implementation, we are doing a proof of concept approach. We will have a dealer which encrypts a text file with an auto-generated AES key, splits the keys and distributes them to other peers in the network; one of the peers initiates a reconstruct command to other peers, and all the responding peers will get other peers' shares, and decrypts the file.

## 3 Details of Approach

We divide our approach into different parts below. In this section, we will provide a detailed view of our communication protocols, the secret sharing mechanism with example, AES encryption and decryption use cases, and the algorithm describing the whole protocol in pseudo-code.

## 3.1 Protocol

The protocol consists of three phases: CREATE, DISTRIBUTE, and RECONSTRUCT.

**Phase 1: Dealer Creating Secret (CREATE)**

In the first phase, the dealer should use the secret and create Shamir secret sharings for all n nodes so that k nodes can reconstruct the secret back. In our protocol case, k is $\frac{2}{3}$ of n.

With these available shares, the sender is ready to send them to other participants to initiate the process according to our algorithm.

**Phase 2: Dealer Sharing (DISTRIBUTE)**

In this phase, all nodes in the network join and wait for the dealer to come. When the dealer comes, it assigns the desired piece of shares to each node inside the network and sends those shares. There are three rounds of dealer interaction to confirming the secret has successfully assigned:

1. Round 1 - Assign: Dealer sends the assigned piece of share to each trusted node.
2. Round 2 - ACK: Node replies to the dealer with ACK message that the share has been received.
3. Round 3 - Dealer Exit: After all nodes reply with ACK message, dealer broadcast message saying it will exit the channel, so each node knows its share is OK and enter the next phase.

So, we call this process the 3-way DISTRIBUTE as shown in Figure 1.

**Phase 3: Combine Secret (RECONSTRUCT)**

To combine the secret, peers need to know each other's addresses. One node can initiate the RECONSTRUCT process by sending other nodes request message and enter another 4-round message passing and reconstructing:

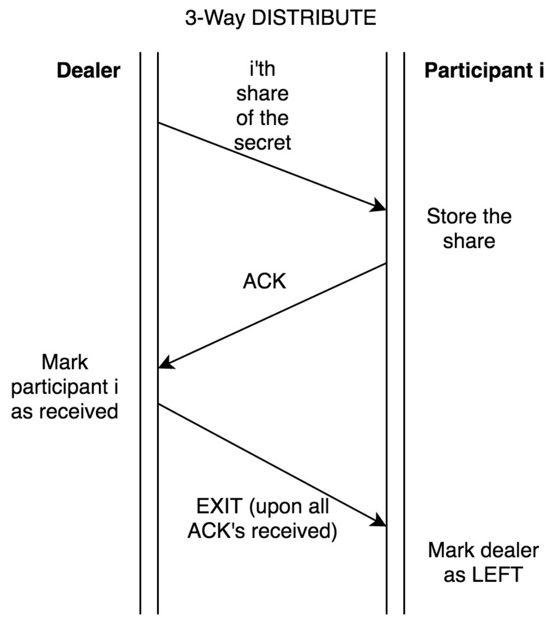

Figure 1: 3-way DISTRIBUTE.

1. Round 1 - Request: One node broadcast request to combine the messages into the final secret to all the nodes.

2. Round 2 - Response: If other nodes are willing to combine the message, they will reply to the original node with Response message.

3. Round 3 - Recovery 1: When the number of nodes who want to combine the messages reaches $\frac{3}{4}$ of the total number of nodes, it enters the recovery phase. Now, the node which initiated the request now broadcasts a Recovery 1 message along with its share to all nodes.

4. Round 4 - Recovery 2: All nodes received the message from Recovery 1 round will start broadcasting its shared part to all the nodes. Behind the scene, when the number of secret received is enough to recover the secret ($\frac{2}{3}$ of total nodes), the secret will be reconstructed successfully and then each node is able to decrypt the file.

So, we call this process the 4-way RECOVERY. The flow is shown below in Figure 2.

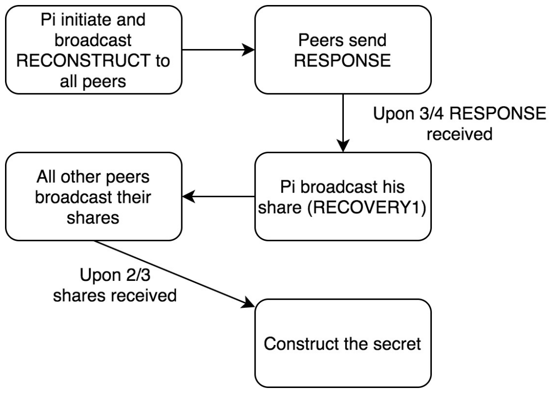
Pi initiate and broadcast RECONSTRUCT to all peers
Peers send RESPONSE
Upon 3/4 RESPONSE received
Pi broadcast his share (RECOVERY1)
All other peers broadcast their shares
Upon 2/3 shares received
Construct the secret

Figure 2: 4-way RECONSTRUCT flow.

## 3.2 Secret Sharing

Our secret sharing algorithm is based on Adi Shamir's secret sharing paper [1]. Shamir's secret sharing scheme accomplishes the functionality mainly using polynomials. For example, if we want to split the secret to n shares so that k different shares can recover the original secret. The polynomial encodes the secret s such that f(0) = s. The polynomial should be degree k, meaning k + 1 points are needed to interpolate it. Upon interpolating the polynomial, a recipient can evaluate it at 0 to find the secret.

For example, I (as the dealer) wanted to share a secret among three friends (participants in our protocol), such that two of them would need to combine their shares to determine my secret. In this case n = 3 and k = 2. If my secret was s = 5, I could create a line (degree-1 polynomial) at random f(x) = 2x + 3 and give Friend 1 f(1) = 5, Friend 2 f(2) = 7 and Friend 3 f(3) = 9. We can uniquely determine my line 2x + 3 and find my secret when two ore more shares are combined. This can be generalized to any polynomial, and not just lines. For example, we can make a cubic polynomial, which is degree 3 and would take 4 points to reconstruct. Figure 3 shows an secret sharing example of degree-2 polynomial.

## 3.3 AES

Although AES does not take a core role in this protocol, we would like the mention it briefly in order to understand our protocol in a real-world scenario. AES has been adopted by the U.S. government and is now used worldwide. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. [7]

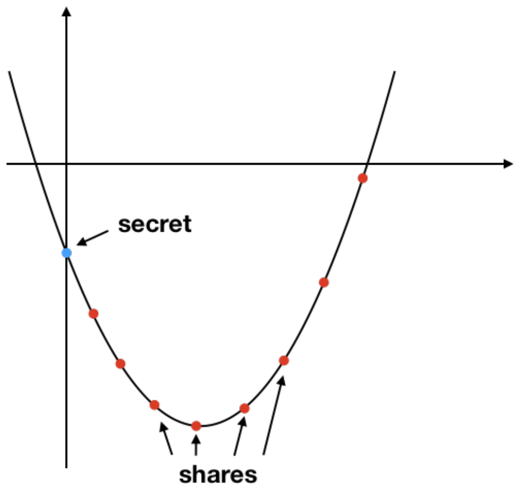

Figure 3: A degree-2 polynomial secret sharing example.

For example, in our use case, a file is encrypted by a dealer who wants all or some of the participants together to get it decrypted. AES is a great encryption and decryption algorithm in this case. The file is like a secret in the safe, which cannot be opened without the consensus of the peers.

## 3.4 Algorithm

```
Setup:

- Each party Pi connect to the P2P network
- All peers in the network are known to all.

=============================================
DISTRIBUTE phase:
---------------------------------------------
As dealer D:

- Connect to the network

- Sample a random 16 byte AES secret key s
    ready for share
- On secret s: sample a random degree-t
    polynomial phi with phi(0) = s so that t
    +1 commitments can reconstruct the
    polynomial
```

```
- Commit to phi and create shares[] <-
    PolyCommit(phi)

- For p in peers:
    - send(p, (ASSIGN, shares[p]))

- if all ACKs received:
    - for p in peers:
        - send(p, EXIT)
---------------------------------------------
As participant Pi:

- Wait for dealer's ASSIGN message

- if ASSIGN received:
    - Store the share
    - send(dealer, ACK)

- if EXIT received:
    - Mark the dealer as LEFT

=============================================
RECONSTRUCT phase:
---------------------------------------------
As participant Pi that initiates RECONSTRUCT:

- broadcast(RECONSTRUCT)

- Wait for message RESPONSE from all other
    peers

- if 3/4 RESPONSE messages received:
    broadcast(RECOVERY1, my_share)

- Wait for (RECOVERY2, share) from all other
    peers and put into shares[]
- if 2/3 RECOVERY2 message received:
    - Combine shares[] and output the secret

---------------------------------------------
As participant Pi that waits for RECONSTRUCT:

- Wait for message RECONSTRUCT
- if RECONSTRUCT received from peer p:
    - send(p, RESPONSE)

- Wait for message (RECOVERY1, share)
- if RECOVERY1 received from peer p:
    - broadcast(RECOVERY2, my_share)

- Wait for (RECOVERY2, share) from all other
    peers and put into shares[]
- if 2/3 RECOVERY2 message received:
    - Combine shares[] and output the secret
```

## 4 Code Implementation

### 4.1 Interactive Mode

MPSS uses interactive input in order to ease the user experience. Our implementation uses the python built-in function input. By executing the python peer python script, the user is able to choose what its role is. "d" is the dealer, who sends AES key to peers; Any other inputs will set the user to be a peer. After entering the role, the user will be prompted to enter the IP address and port numbers (if the user is a peer and wants to use the customized IP address and port, the IP address and port must be added to the network file peer_ip.txt). When the user has both the role and the IP address entered, it can execute according to its role:

**Dealer:**

1. Dealer is created and loads the IPs from the network file.
2. Dealer makes the default constraint for reconstruction(e.g. sending packets to 3 peers, while $\frac{2}{3}$ can reconstruct the secret).
3. A random AES key is generated, and a file is encrypted using this key.
4. The AES key will be packed as a secret and be split into multiple shares.
5. Dealer sends the secret shares to the peers respectively.

**Peer:**

1. Peer is created and loads the IPs from the network file.
2. Peer starts two threads, one is used for receiving shares from other peers, and the other is used for sending its share to other peers.
3. All peers receive AES key from the dealer and cache it in memory.

4. One of the peers initiates a broadcast and send its own share to other peers, and requests shares from other peers.
5. After 2 out of 3 shares are received by the peers, they will reconstruct the shares into AES key.
6. Each of the peers uses the AES key to decrypt the encrypted file that was produced by the dealer, and saves the file to a folder.

In order to execute the program flawlessly, running multiple peers on different terminals is suggested. After these terminals have been up and running, a dealer is needed to be created to send the AES keys to other peers.

### 4.2 Message format

The whole message consists of several segments:

1. Message length: the first four bytes of the message is message length. This is done to handle an arbitrary number of payload without receiving more than it can handle.
2. Message type: We give a distinct identifier to all types of messages. This is done to check the integrity of the message and ensure the correct type of message are send in each phase.
3. Payload: the payload that we attached at the end depends on the type of message. We support the arbitrary type of payload as long as you provide corresponding encode and decode function to process the object from/to message. The two main payloads in our protocols are empty payload and secret sharing piece payload.

To see how the data encoding and decoding work, we have drawn flow charts below in Figure 4 and Figure 5:

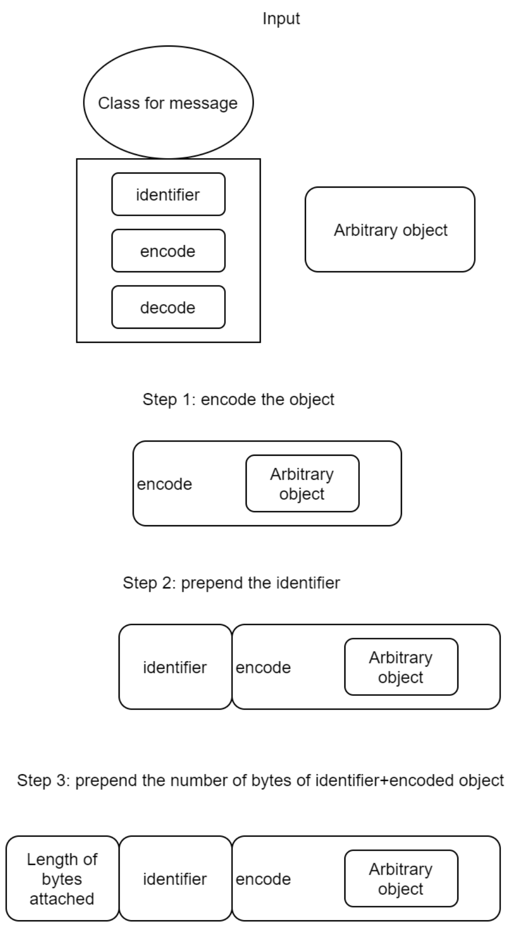

Figure 4: Message encoding.

### 4.3 Node Lifecycle

Each peer will be assigned a role (participant or dealer) through the command line argument or in an interactive way. They will also be assigned the IP and designated port. In order to send and receive data from other nodes at the same time, we need a server to run in the background. So we open a new thread to handle server related tasks, and another thread to wait until enough shares have been received to recover the secret.

The main thread in a participant starts right after the dealer exits. It asks users to input whenever he/she wants to combine the secret shares and starts the RECONSTRUCT phase next.

The server thread binds to a designated port, listens, accepts, and handles each request. It handles and checks the message, and act differently according to the message type.

The reconstruct thread waits until enough shares have been received (In our example $\frac{2}{3}$). Then it will recover secret based on the secret sharing protocol and decrypt the file.

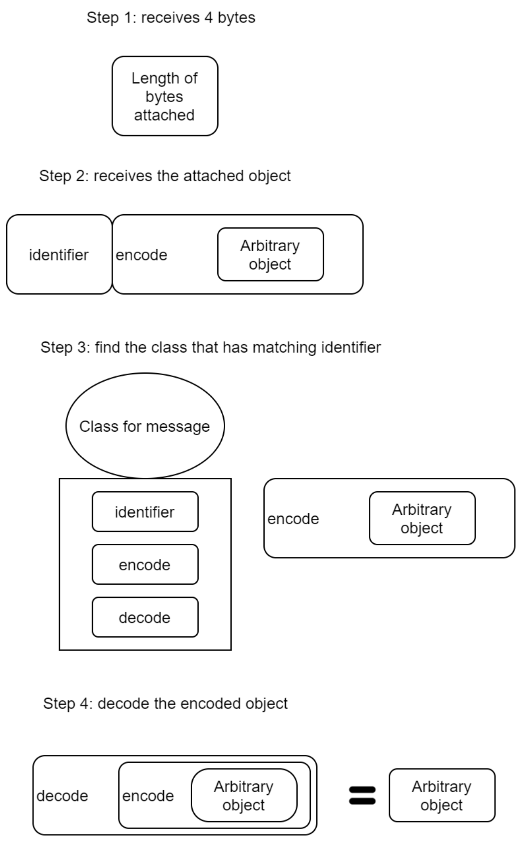

Figure 5: Message decoding.

### 4.4 Secret Sharing Implementation

The implementation is referenced and modified from the pycryptodome secret sharing library [4]. In the main SecretSharing class, the two functions *split* and *combine* are able to support the functionalities we need.

*split(k, n, secret)* is able to split a secret into $n$ shares. All shares are points over a 2-dimensional curve. At least $k$ points (that is, shares) are required to reconstruct the curve, and therefore the secret. Each share must be kept confidential to the person it was assigned to.

*combine(shares)* is able to reconstruct a secret, if enough shares are presented based on polynomial' interpolation: given k points in the 2-dimensional plane $(x_1, y_1).....(x_k, y_k)$. with distinct $x_i's$ , there is one and only one polynomial q(x) of degree k - 1 such that q(x) = $y_i$.

## 5 Result

In order to verify the integrity of our implementation, we split the project into different parts and unit tested them.

1. We tested that the secret sharing algorithm works under many different values of n and k.
2. We checked that AES encryption works for files of different lengths.
3. We modularized our socket to ensure the correct behavior of different types of send/receive. (i.e., send integer, send bytes, send encoded message)
4. We inserted checks under message encoding/decoding to ensure properties such as the correct type of messages is received, and there is no two messages with the same identifier.
5. We printed out useful debug information to details, such as if the correct connections are made.
6. We tested the protocol by changing peer_ip.txt to ensure that it works under different network configurations.

After we piece different parts of this protocol together, we are able to obtain the expected result. That is, the dealer is able to split the AES key into secret pieces and assign the pieces to all nodes, where they can later collaboratively reconstruct the original key and decipher the file.

## 6 Discussion

Multi-Party Secret Secret Sharing Protocol is able to do a lot of tasks in the information security domain, and it should be able to promise that no single user can access the sensitive information alone. But there are more future work we need to consider to improve the MPSS protocol.

**Verifying Integrity** To make sure all packets are reliably transferred with no errors, verification for the packets can be added. In the case where the peer receives a share from the dealer, the peer can validate the packet using some error detections, such as MD5, or other cryptographic validations. If such packet is faulty, the peer should contact the dealer to re-transmit the packet. Doing so will make sure the packet is correct when the dealer is still in the network, and it also prevents man-in-the-middle attacks.

**Terminating Reliable Broadcast** The broadcast from one peer to other peers in the network can be more reliable to achieve termination, validity, integrity, and agreement [3]. Introducing Terminating Reliable Broadcast Protocol to each peer should be able to make sure all peers receive the correct packets while the initiated peer knows other peers have successfully received its broadcast. If the initiated peer fails, it should receive send faulty(SF) or no response at all.

**Various Input Formats** MPSS is designed for the AES key specifically, so the input format is a 16-byte integer. To make MPSS more robust, the future work can be adding string or a list (batch) of string as an acceptable input secret, and eventually, a file can be shared using our MPSS protocol.

**Improve scalability** As the number of nodes grows, scalability may become an issue. We can consider reducing the number of connections between nodes, reducing the size of each message, and use more efficient distributed algorithms such as gossiping to improve some of the communication bottlenecks.

## 7 Conclusion

Shamir's Secret Sharing has been around since 1979 [5], but still not yet widely used regardless of its advanced cryptographic benefits. But with our Multi-Party Secret Sharing Protocol, which includes core features of communication network and distributed system, we can enlighten the future uses of this type of secret sharing protocol.

In today's technology trends, more and more data are being moved from local server racks to clouds, such as Google Cloud Platform, Amazon Web Services, Azure, and etc. To protect the data, these cloud service providers can utilize the MPSS protocol as a type of data security mechanism to prevent data breaches. Also, we have seen the emergence of blockchain in the areas of healthcare, finance, and so on. Hence, we believed that one day, secret sharing protocol would be one of the core technologies, and MPSS will be optimized to be a part of the technology revolution.

## References

[1] Shamir, Adi. 1979. "How to share a secret", Communications of the ACM, 22 (11): 612–613, doi:10.1145/359168.359176.

[2] Anon. 2019. *Shamir's Secret Sharing.* (December 2019). Retrieved December 6, 2019 from https://en.wikipedia.org/wiki/Shamir's_Secret_Sharing

[3] Anon. 2019. *Terminating Reliable Broadcast.* (January 2019). Retrieved December 6, 2019 from https://en.wikipedia.org/wiki/Terminating_Reliable_Broadcast

[4] PyCryptodome. *Secret Sharing Schemes.* Retrieved December 6, 2019 from https://pycryptodome.readthedocs.io/en/latest/src/protocol/ss.html

[5] Anon. 2018. *Shamir's Secret Sharing Scheme.* (January 2018). Retrieved December 7, 2019 from http://point-at-infinity.org/ssss/

[6] Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A. Efros. *Image-to-Image Translation with Conditional Adversarial Networks.* CVPR 2017

[7] Anon. 2019. *Advanced Encryption Standard* (December 2019). Retrieved December 6, 2019 from https://en.wikipedia.org/wiki/Advanced_Encryption_Standard