**Summary of the Crypto platform with MPC technology project**

**1 Background**

In digital assets, to make a transaction, we need two keys, a public key, and a private key. Public key is visible to everyone and private key must only be known to the owner. Private key management is extremely important because if your private key is accessible then your account’s data can be manipulated.

Multi-party computation (MPC) is a subfield of cryptography that enables multiple parties to jointly compute a function without revealing the inputs. The primitive that powers MPC is the ability to split a piece of data into multiple encoded parts known as secret shares. On their own, the shares reveal nothing about the original data. However, if two parties perform the same operation on a set of shares and then recombine them, it is as if that operation was performed on the original data.

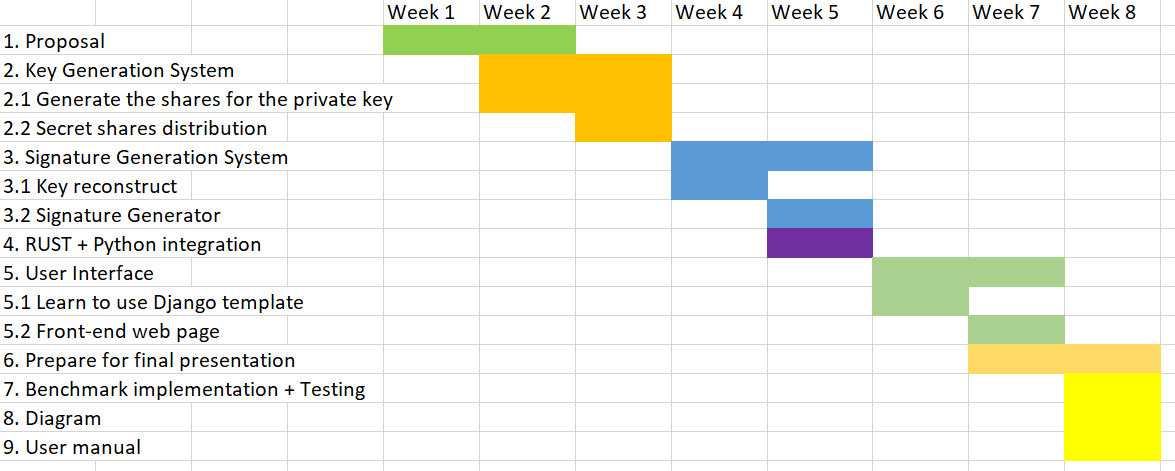
Shamir’s secret sharing (SSS) is one of the classic threshold secret sharing (TSS), which is used to secure a secret in a distributed way. In SSS, the private key is distributed into n parties based on polynomial interpolation, and a subset of size t (t < n) are involved in the reconstruction of the private key for the transaction. This makes digital assets safer, because never ever any server can get access to the entire private key, even if the attacker gets access to one or two servers cannot get the entire private key.

**2 Goal**

The goal of the project is mainly applying MPC (multi-party computation) to the current QAS crypto platform to secure the private key generation and management. The Shamir’s secret sharing is firstly implemented in RUST to ensure the speed and safety of the computation of private keys. The RUST crate should be wrapped into a Python package which is compatible to the current crypto platform of QAS. The final step is to implement an application for the private key management. The user could set the parameters for the secret sharing algorithm in the front-end (website) and the system will automatically compute the secret shares in the backend and send to all the parties by email. Every time the admin account makes a transaction, the system will broadcast a link for entering the partial private key to all the parties by email. The system will reconstruct the original private key in the backend. Only enough shares are collected from parties can the transaction be approved.

2.1 Timeline

The timeline is updated compared to that in the summary for midterm review. The main update is on the front-end implementation. As the implementation of secret sharing algorithm is finished ahead of schedule, the front-end implementation is also included during this internship.



**3 Private Key management system**

3.1 Logic of the key management system

The key generation is based on Shamir’s (K, N) threshold scheme. S is the secret that we wish to encode. It is divided into N parts: S1, S2, S3, …., Sn. After dividing it, a number K is chosen by the user in order to decrypt the parts and find the original secret. If we know less than K parts, then we will not be able to find the secret S (i.e.) the secret S cannot be reconstructed with (K – 1) parts or fewer.

3.2 Mathematical principle

Interpolation theorem: There exists a unique polynomial of degree at most k - 1 that interpolates the k data points. The idea is to build a polynomial with the degree (K – 1) such that the constant term is the secret number and the remaining (K – 1) coefficients are random. This constant term can be found by using any K points out of N points generated from this polynomial.

Example: Let the secret code S = 65, N = 4, K = 2.

1. Initially, in order to encrypt the secret code, we build a polynomial of degree (K – 1).
2. Therefore, let the polynomial be y = a + bx. Here, the constant part ‘a’ is our secret code.
3. Let b be any random number, say b = 15.
4. Therefore, for this polynomial y = 65 + 15x, we generate N = 4 points from it.
5. Let those 4 points be (1, 80), (2, 95), (3, 110), (4, 125). Clearly, we can generate the initial polynomial from any two of these 4 points and in the resulting polynomial, the constant term a is the required secret code.

3.3 Private key distribution

Firstly, we generate a k – 1 degree polynomial, with the value of the private key as the constant coefficient and randomly generated (k – 1) coefficients. Next, we randomly select n points from the polynomial and send to n parties.

3.4 Private key reconstruction

Given k points {(x0, y0), (x1, y1)…(xk-1, xk-1)}, we can write a polynomial of k – 1 based on Lagrange polynomial Interpolation. The constant coefficient of this polynomial is the original private key.

Firstly, we produce a set of Lagrange polynomials *Li(X)* each associated with a particular *xi*.

Lagrange polynomial for *xi*,

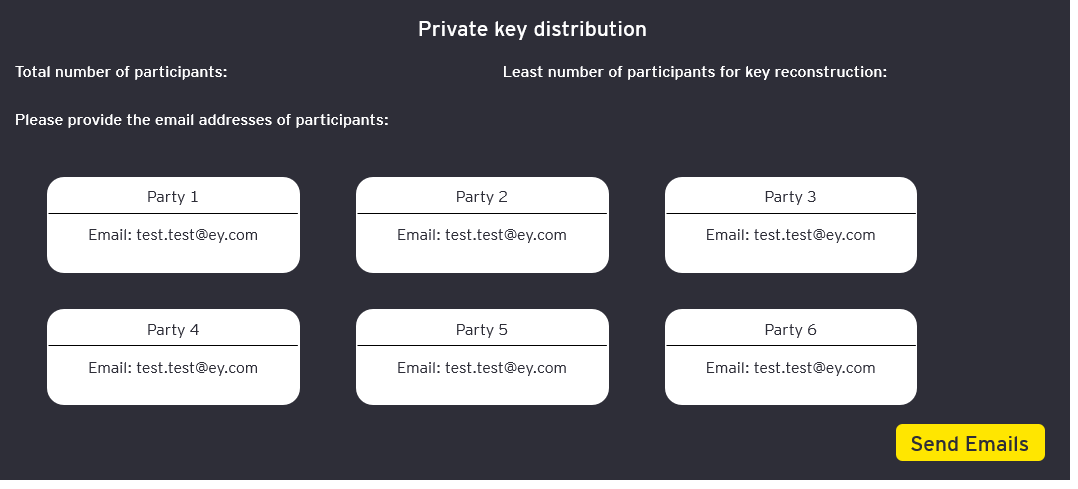
Secondly, multiply each by *yi*, so that each polynomial goes through the point *(xi, yi)*.

Finally, sum up these polynomials,

Finally, recovering the private key can be done by computing,

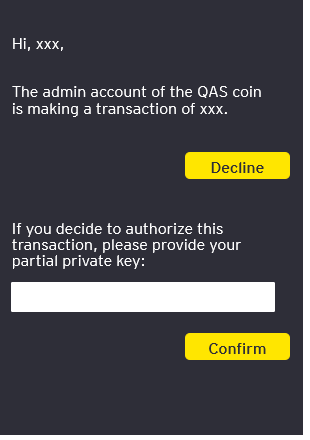
**4. Front-end Design**

4.1 Key distribution



In the private key distribution page, the user can set the parameters of the secret sharing algorithm and enter the information of the participants. Once the “send emails” button is clicked, the system will compute the secret shares automatically and send these shares into the emails of each party. The original private will then be deleted from the admin account, so that no transaction can be made without enough partial keys from the parties.

4.2 Key reconstruction



When the admin account is making a transaction, the original private key need to be reconstructed. The server will broadcast a link to the emails of all the parties. Each party can either decline this transaction or provide their partial private key in the link. The partial private key will be inserted into the database of the system. Once enough shares are received, the system will compute the original private key in the backend and make a transaction.