**QAS crypto platform Private Key Management System v0.0.0**

**August 2022**

Table of contents

Chapter 1 Introduction of MPC implementation project ---------------------------------------------------3

* 1. Background
  2. Goal

Chapter 2 Private Key Management System--------------------------------------------------------------------4

2.1 Logic of the key management system

2.2 Mathematical principle

2.3 Private key distribution

2.4 Private key reconstruction

Chapter 3 User Guide-------------------------------------------------------------------------------------------------6

Chapter 4 Guidance for developers-----------------------------------------------------------------------------11

4.1 Set up

4.2 Integrate RUST with Python

Chapter 1 Introduction of MPC implementation 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.

* 1. 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.

Chapter 2 Private key management System

2.1 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.

2.2 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.

2.3 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.

2.4 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*,

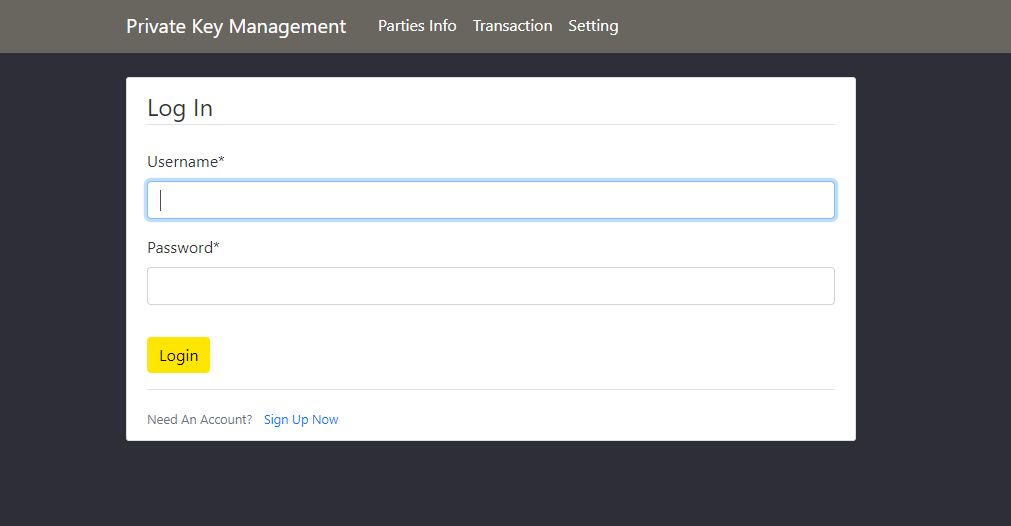
Secondly, multiply each by *yi*, so that each polynomial goes through the point *(xi, yi)*.

Finally, sum up these polynomials,

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

Chapter 3 User Guide

* Login of admin account



Current Admin Account:

Username: Vivian

Password: Qwer1234!

(You are also allowed to register for a new admin account!)

* Settings for the key distribution

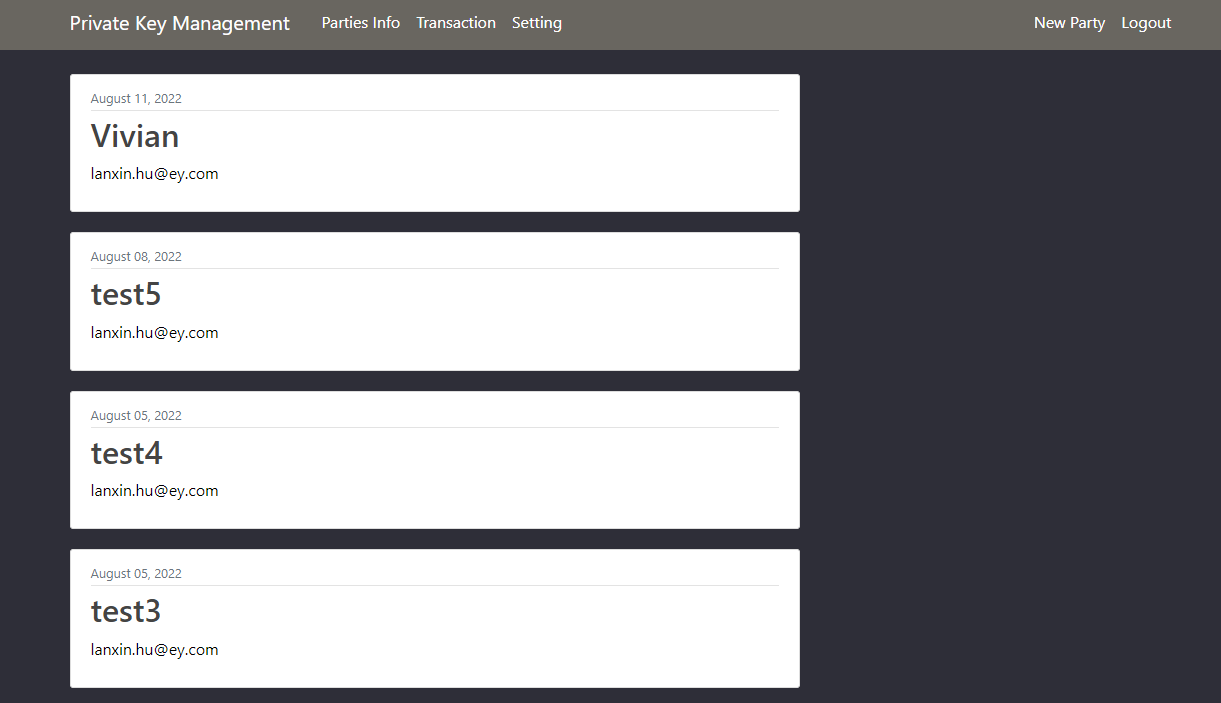
After login, you can set number of participants and threshold for key reconstruction.

By clicking “Setting” in the menu bar, you will be redirected to this webpage, and you may enter the parameters here.



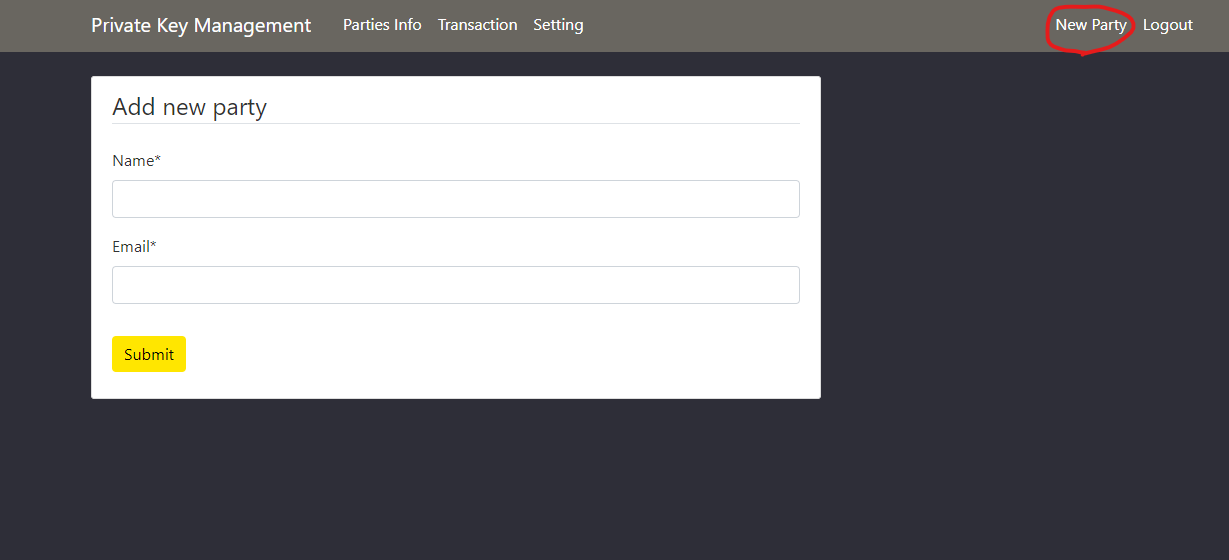
* The information of all parties

Then, you can see the information including names and email addresses in the “Parties Info” page. You may go to this page by clicking “Parties Info” in the menu bar. You can see the information for all participants in this page.

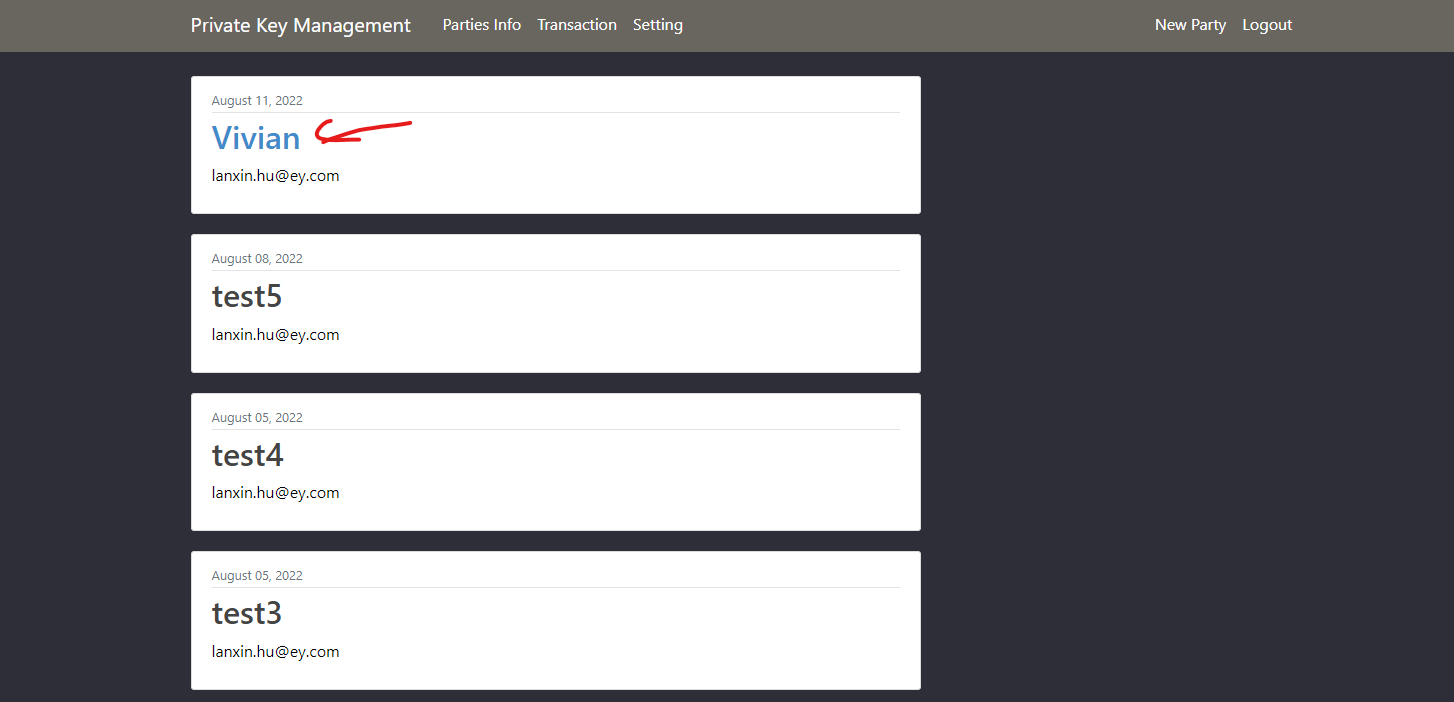


* Add new parties

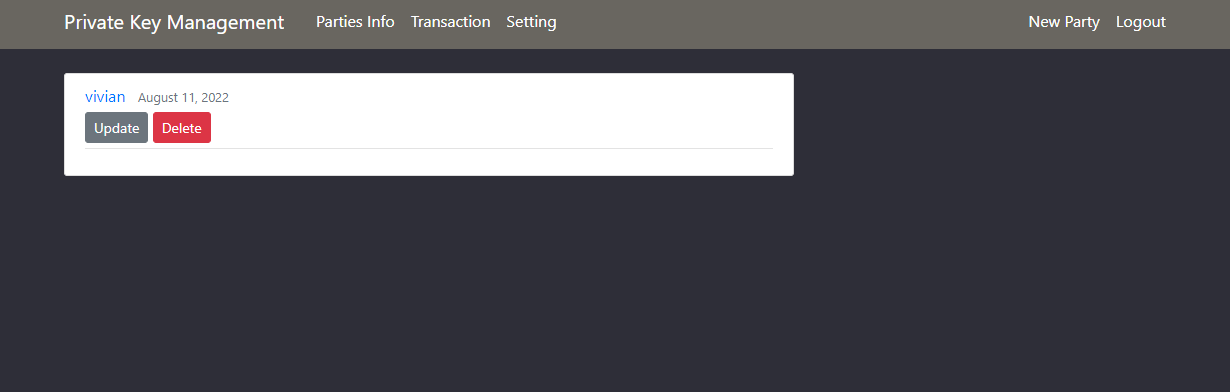
By clicking “New Party” on the menu bar, you can enter the information for a new party including name and email address, and then click “Submit”.

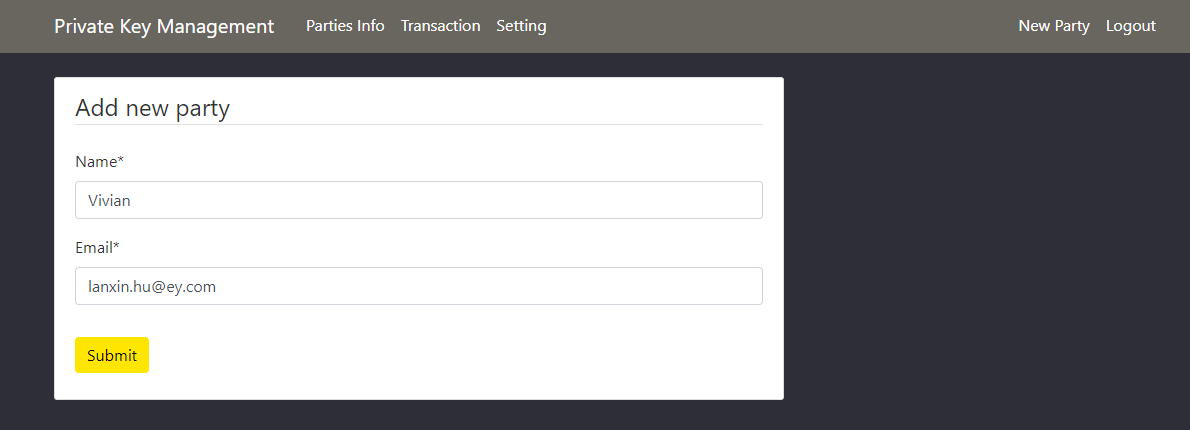


* Edit or delete the existing party

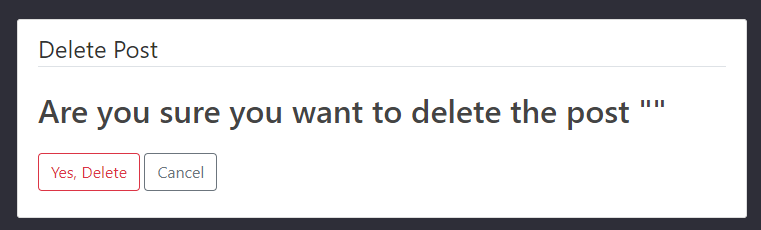


By clicking the name of a party, you will be redirected to the page for editing or deleting this party.

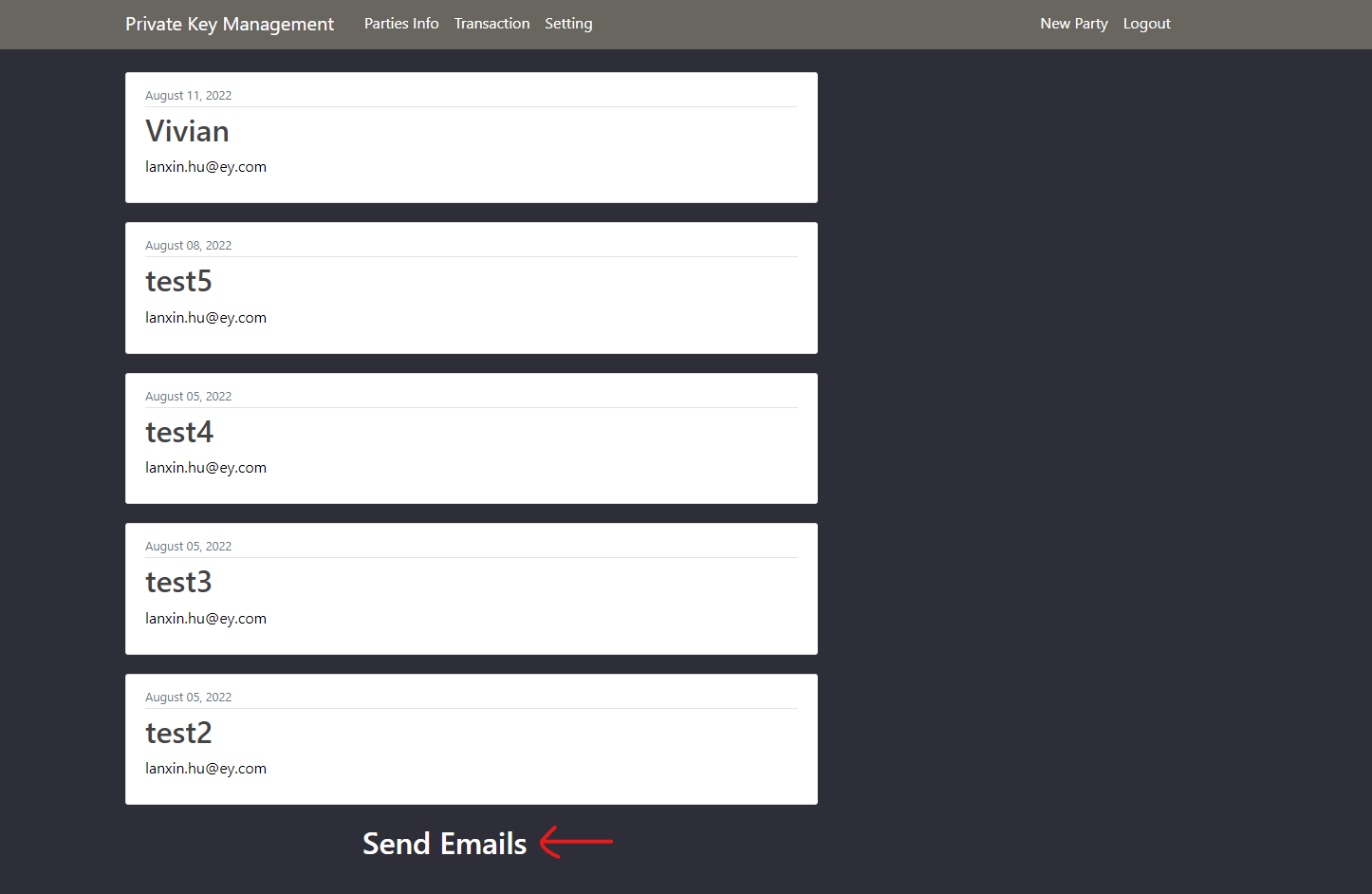


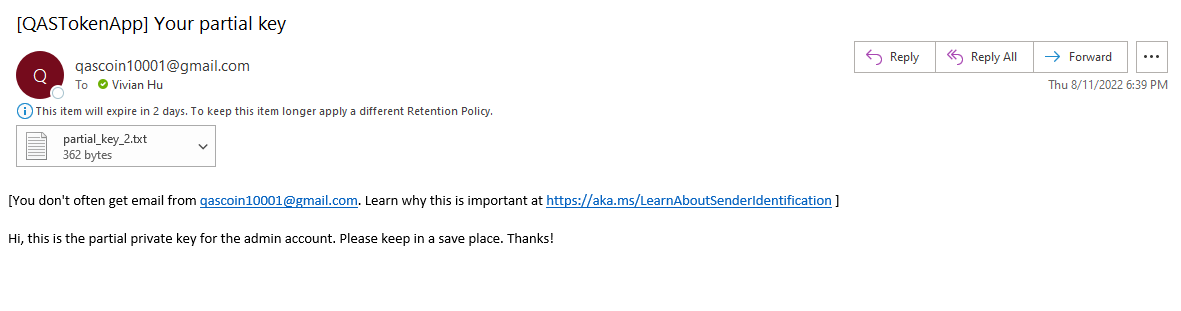
Edit the party info. After clicking “Update”, you will be redirected to a webpage for modifying the information for the selected party, and you may click “Submit” for updating this information in the database of the backend.

Delete the selected party.

* Distributing partial private keys

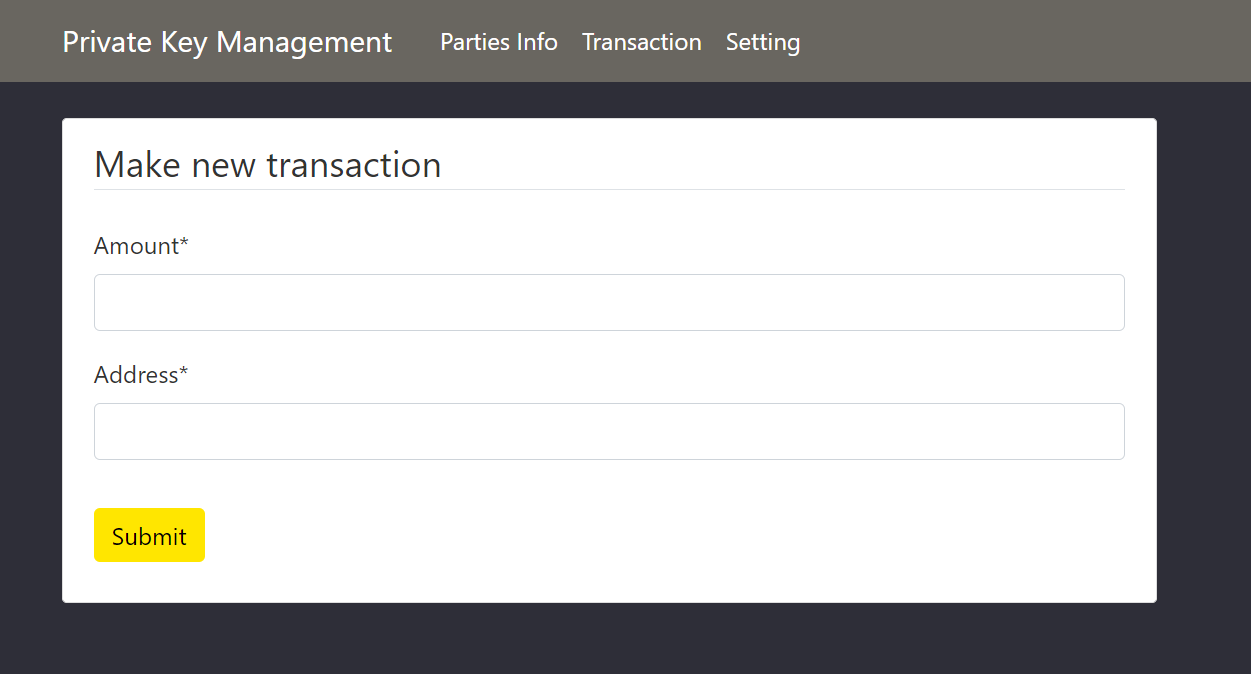
After entering all the required information for the participants, the user can click the “Send Emails” button. Then, the backend will automatically generate secret shares and send to all the participants by email.





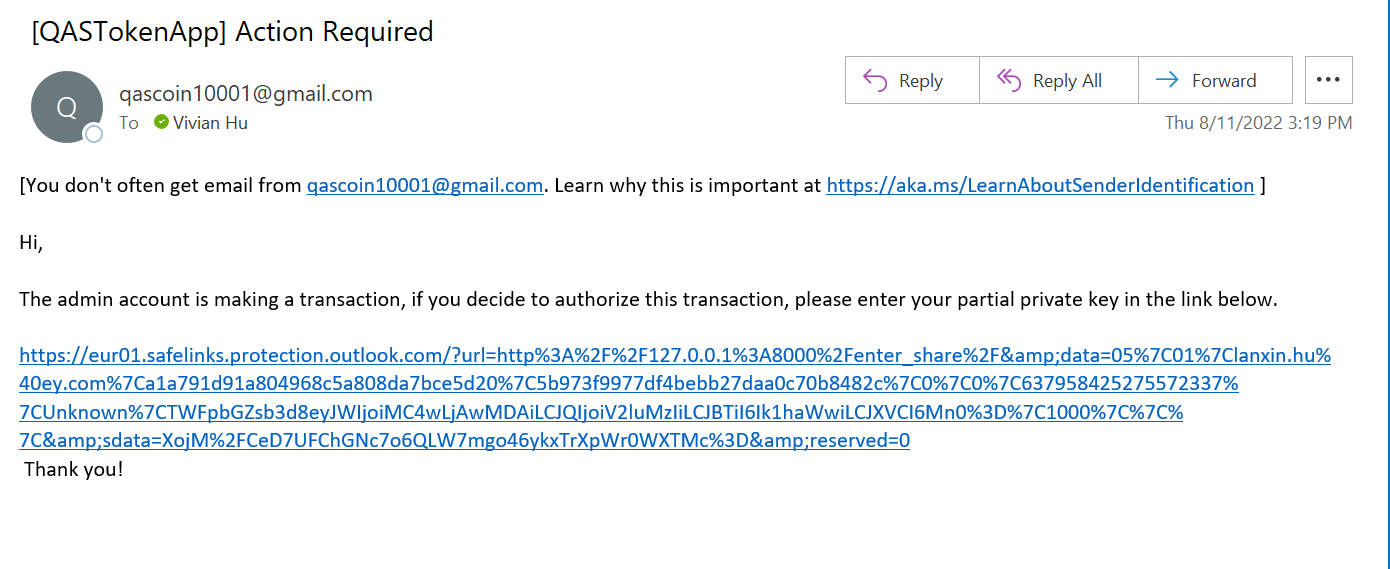
* Make a transaction

The admin account can make a transaction in this webpage.

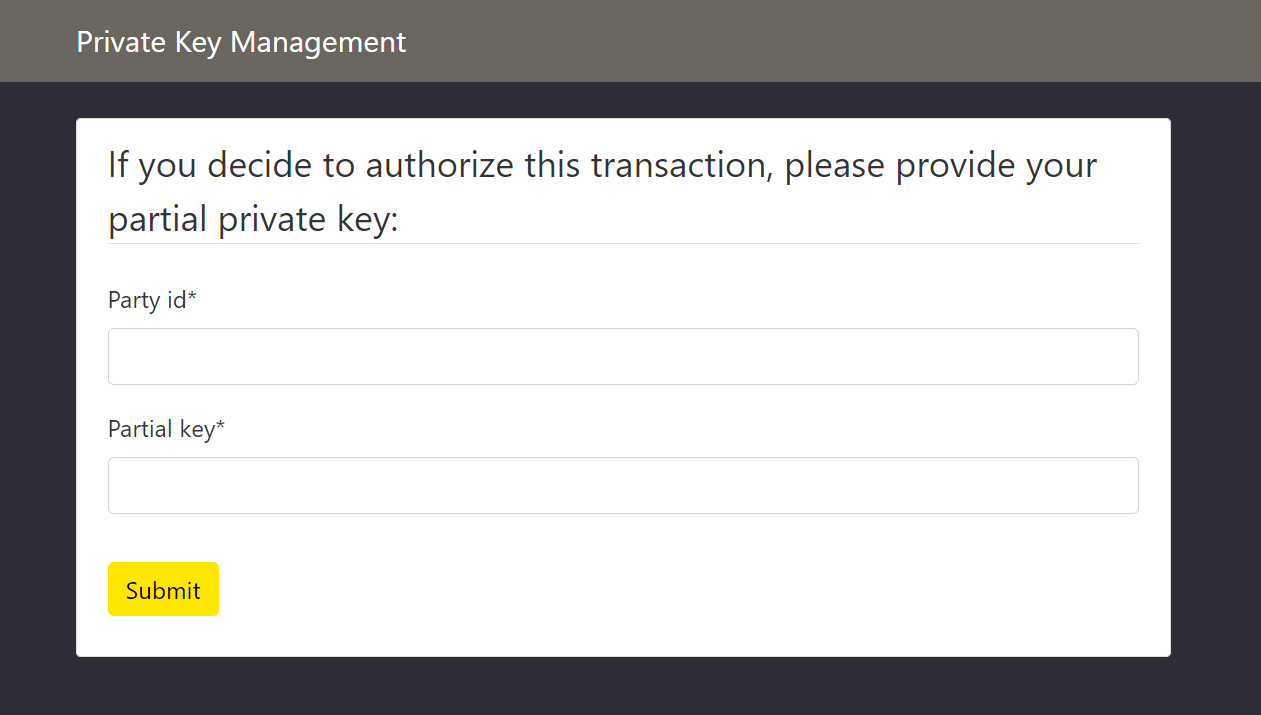


* Private key reconstruction

When the admin account submits a new transaction, the system will broadcast a link to all parties to request for their partial keys by email.



Each party can enter their partial private key in the link, ‘http://127.0.0.1:8000/enter\_share/’. When sufficient parties provide their partial keys, the system will automatically reconstruct the original private key and save into the file ‘./keys/new\_private\_key.txt’ .

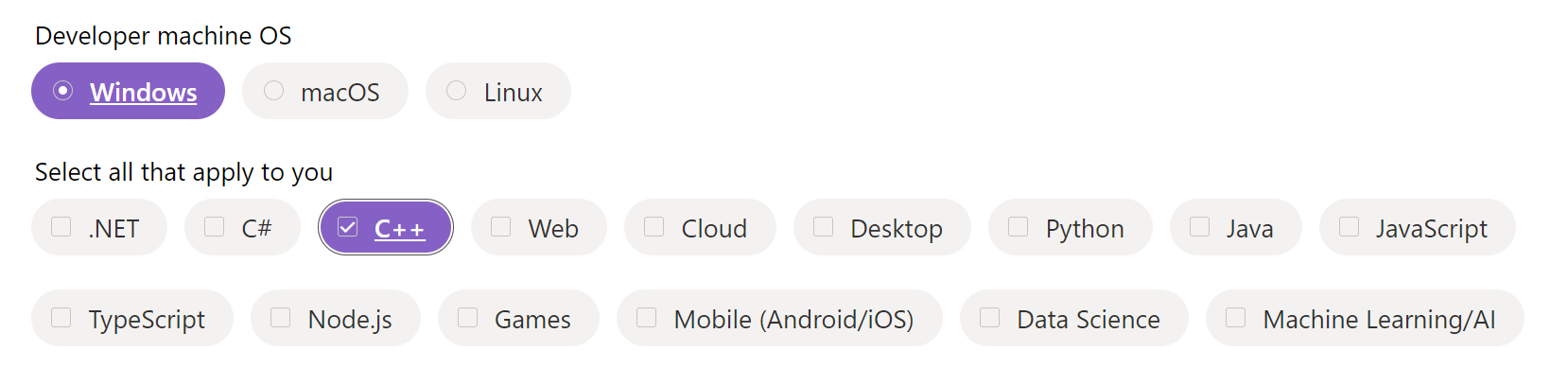


Chapter 4 Guidance for developers

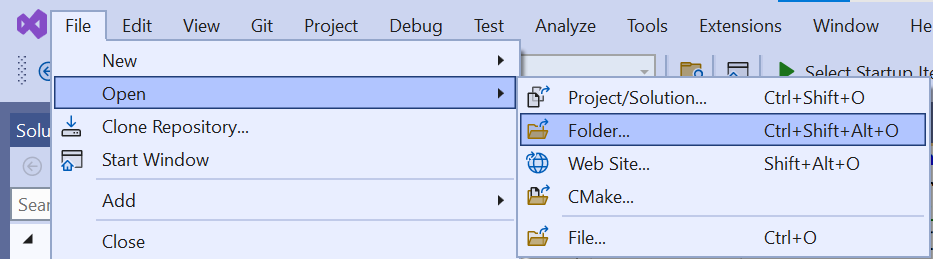
4.1 Set up

Firstly, download Microsoft Visual Studio C++ for the dependencies for the RUST crate.

https://visualstudio.microsoft.com/downloads/



Inside the VS, open the folder of ‘QAS\_key\_management\_UI’.



In the path of ‘./QAS\_key\_management\_UI’, create a new virtual environment.

QAS\_key\_management\_UI> virtualenv -p python3 venv

Then, activate this virtual environment.

QAS\_key\_management\_UI> venv\Scripts\activate

It will look like this (venv) QAS\_key\_management\_UI>

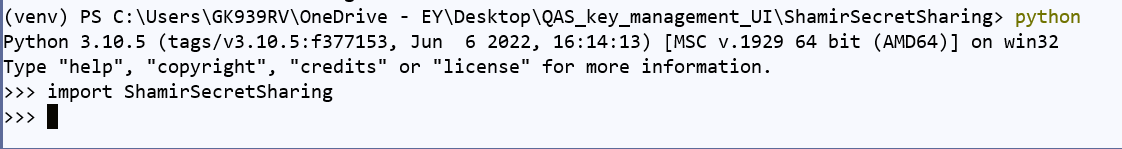
Then run pip install -r requirements.txt to install the dependencies.

Next, we cd into the ShamirSecretSharing directory to activate the ShamirSecretSharing into our virtual environment.

(venv) QAS\_key\_management\_UI> cd .\ShamirSecretSharing\

(venv) QAS\_key\_management\_UI\ShamirSecretSharing> maturin develop

To test if the ShamirSecretSharing Python package is installed successfully, we can test by trying to import this package into the python environment.



Finally, we can go back to the QAS\_key\_management\_UI> directory and run the website by running

(venv) QAS\_key\_management\_UI> python .\manage.py runserver

4.2 Integrate RUST with Python

Initialize a new RUST package with PyO3 bindings.

$ maturin init

Specify metadata about the new package in the [manifest](https://doc.rust-lang.org/cargo/appendix/glossary.html#manifest) file “secret\_sharing/ Cargo.toml”. Set the crate-type as “cdylib”, and add pyo3 to the dependencies.

[lib]

name = "secret\_sharing"

crate-type = ["cdylib"]

[dependencies]

pyo3 = { version = "0.16.5", features = ["extension-module"] }

Annotate the RUST function with #[pyfunction] to turn it into a PyCFunction and wrap the result in a PyResult.

#[pyfunction]

fn reconstruct\_private\_key(filename: &str) -> PyResult<String> {

let shares = 5;

let threshold = 3;

let private\_key = reconstruct::<GF256, CompactShamir>(filename, threshold);

Ok(private\_key)

}

Add the function to the #[pymodule]

#[pymodule]

fn secret\_sharing(\_py: Python<'\_>, m: &PyModule) -> PyResult<()> {

m.add\_function(wrap\_pyfunction!(reconstruct\_private\_key, m)?)?;

Ok(())

}

The above will expose the reconstruct\_private\_key function in the Python module called secret\_sharing.

Build the package and install it into the Python virtualenv previously created and activated.

$ maturin develop

The secret\_sharing package is then ready to be used from Python.

$ python

>>> import secret\_sharing

>>> secret\_sharing.split\_private\_key("privateKey.txt")

Shares:

1|6ca345ec3b5f5432cf66e1e7dc8d1ff4ca7bad5ff254dcede49d31f60ffc1c7b

2|24c0c1fe5a27a8a0a8cf9fc4fba05e58b7521ea430bb44d2514850c6f535f675

3|86097316f7d1086c927700c7d1fe9afc52fafdba00abe3357edee7afca4dfe34

4|a7dbd65f52393310ee7ec03d8b0ce0475193b4ec6cc3c4e2ea36c80256e01b94

5|051264b7ffcf93dcd4c65f3ea15224e3b43b57f25cd36305c5a07f6b699813d5

>>> secret\_sharing.reconstruct\_private\_key("shares.txt")

Private Key: ce6af70496a9f4fef5de7ee4f6d3db502fd34e41c2447b0acb0b869f3084143a