Privacy-Preserving E-Wallet in Cloud-Era

Final Individual Report

Cheung Long Sang (3035692797)
Lee Hiu Long (3035687077)
Liao Jiayang (3035637280)

Supervisor: Dr KP Chow

Department of Computer Science,
The University of Hong Kong

April 18, 2023
Abstract

Privacy concerns have hitherto been a major barrier to the wide adoption of e-wallets in the market. Transactions on e-wallets may not be confidential. This undermines users’ financial privacy. As a result, there is an increasing interest in privacy coins. Nevertheless, while privacy coins can provide privacy through cryptographic techniques, they are not scalable and have an ambiguous legal status. They are still far from wide adoption. This project aims to develop a e-wallet for fiat money that is both privacy-preserving and scalable using cryptography. Cloud technology is also integrated for better security, resilience and synchronism. A mobile app prototype has been implemented. The underlying app development considerations and confidential transaction protocol are discussed. Limitations and directions for future work are identified. The e-wallet has the potential to be the first privacy-preserving and the first cloud-based e-wallet in the market.

Keywords: E-wallet, Confidential transaction, Cryptography, Cloud technology, E-payment
We would like to express our utmost gratitude to our project supervisor and program coordinator, Dr Chow Kam Pui, for his invaluable patience, unwavering support, and inspiring feedback throughout the whole project.

We are also indebted to our classmates and cohort members, particularly our team members, for their assistance with editing, late-night feedback sessions, and unwavering moral support. We would also like to extend our appreciation to the research assistants and study participants from the university, whose contribution of critical data and insights inspired our work.

Finally, we would be remiss if we did not acknowledge the unwavering support of our families. Their unwavering belief in us has kept our spirits high and our motivation strong throughout the project.
Contents

List of Figures .......................... 1
List of Tables .......................... 2
Abbreviations .......................... 3

1 Introduction .......................... 4
  1.1 Background and Motivation ........... 4
  1.2 Market Survey ...................... 6
  1.3 Objectives and Scope ............... 9
  1.4 Contribution ...................... 9
  1.5 Outline of This Report ............. 10

2 App Development Methodology .......... 11
  2.1 System Design ..................... 11
  2.2 Frontend Methodology .............. 12
    2.2.1 Frontend Tools ................. 12
    2.2.2 Frontend Overview .............. 12
  2.3 Backend Methodology ............... 13
    2.3.1 Backend Tools ................. 13
    2.3.2 Backend Overview .............. 13
  2.4 Cloud Technology .................. 15
  2.5 Novel Functions .................... 16
    2.5.1 Promise ....................... 16
    2.5.2 Collaborative Wallets .......... 16
    2.5.3 Automatic Bill Splitting ....... 17
  2.6 Implementation Considerations ...... 18
    2.6.1 QR Code ....................... 18
    2.6.2 Inconsistency ................. 18
    2.6.3 Failure of Central Server ...... 19
    2.6.4 Wallet Recovery ............... 19

3 Confidential Transaction Methodology .. 21
  3.1 Literature Review .................. 21
  3.2 Auditable Confidential Payment System .. 23
  3.3 ISHE Instantiation ................. 25
  3.4 NIZK Instantiation ................. 25
    3.4.1 NIZK for Transaction Amount Equality 25
3.4.2 NIZK for Non-Negative Transaction Amount and Sufficient Sender Wallet Balance ........................................... 26

3.5 Confidential Transaction Protocol ......................................... 27
  3.5.1 System Setup Protocol .................................................. 27
  3.5.2 Register Protocol ....................................................... 28
  3.5.3 Login Protocol .......................................................... 28
  3.5.4 Send Protocol .......................................................... 28
  3.5.5 Receive Protocol ........................................................ 29
  3.5.6 In-store Payment Protocol ........................................... 30
  3.5.7 Online Payment Protocol ........................................... 31
  3.5.8 Top-Up Protocol ....................................................... 31
  3.5.9 Withdrawal Protocol .................................................. 31
  3.5.10 Bill Payment Protocol ............................................... 31
  3.5.11 Bill Splitting Protocol ............................................... 31
  3.5.12 Auditing Protocol .................................................... 31
  3.5.13 Global Supervision Protocol ..................................... 32

3.6 Implementation Considerations ........................................... 32
  3.6.1 Elliptic Curve Cryptography ........................................ 32
  3.6.2 Curve ................................................................. 32
  3.6.3 Precision ............................................................... 33
  3.6.4 Fast Decryption ....................................................... 33
  3.6.5 Secure Hash Function ............................................... 34
  3.6.6 Implementation ....................................................... 34
  3.6.7 Documentation ....................................................... 34

4 Results and Discussions .................................................. 35
  4.1 User Interface Showcase ................................................ 35
    4.1.1 Basic Functions ................................................... 35
    4.1.2 Novel Functions ................................................... 37
  4.2 Prototype Demo ......................................................... 39
  4.3 Attacks and Defense .................................................... 40
    4.3.1 Threat Models ...................................................... 40
    4.3.2 Experimental Setup ............................................... 41
    4.3.3 Scenario 1 - Different transaction amounts in P2P Transfer .. 43
    4.3.4 Scenario 2 - Different transaction amounts in In-store Payment .. 46
    4.3.5 Scenario 3 - Negative transaction amounts in P2P Transfer .... 50
    4.3.6 Scenario 4 - Insufficient Sender Balance in P2P Transfer ....... 51
  4.4 Efficiency and Cost ..................................................... 53
  4.5 Result Discussion ....................................................... 54
5 Conclusion

5.1 Project Schedule ................................................. 55

5.2 Limitations and Challenges ..................................... 56
   5.2.1 Post-Quantum Vulnerabilities ............................... 56
   5.2.2 Compliance with Regulatory Requirements .............. 56

5.3 Future Work ...................................................... 56
   5.3.1 Post-Quantum Resilience ................................... 56
   5.3.2 Implementations of Collaborative Wallets ............... 56
   5.3.3 Implementations of NIZK Proofs for Various Policies .. 57

5.4 Conclusion ....................................................... 57

References .......................................................... 58

Appendices .......................................................... 61

A Market Research Survey ........................................ 61
   A.1 Original Survey and Raw Data ............................... 61
   A.2 Result Visualization ......................................... 61

B API Documentation ............................................... 61
List of Figures

1. Expected improvement of e-wallets ................................................. 6
2. Demand for multi-devices (Market Survey) ................................. 7
3. Demand for bill splitting (Market Survey) .................................. 7
4. Demand for shared wallets (Market Survey) ................................. 8
5. System Design ............................................................................ 11
6. Backend Database ER Diagram .................................................... 14
7. Backend Database File Structure ................................................... 15
8. ACPS Construction Sketch ............................................................. 24
9. Sample Page of Cryptographic Algorithm Documentation ............. 35
10. Login UI .................................................................................... 36
11. Register UI ................................................................................ 36
12. Terms of Service UI ................................................................... 36
13. Home UI .................................................................................. 36
14. Sidebar UI ............................................................................... 36
15. User Profile UI .......................................................................... 36
16. Tx History UI ........................................................................... 37
17. Scan UI .................................................................................... 37
18. Payment Code UI ...................................................................... 37
19. Wallets UI ............................................................................... 37
20. Send UI ................................................................................... 37
21. Receive UI ............................................................................... 37
22. Top-Up UI ............................................................................... 38
23. Withdrawal UI .......................................................................... 38
24. Bill Payment UI ......................................................................... 38
25. Promise UI ............................................................................... 38
26. Co-wallets UI ........................................................................... 38
27. Co-wallet Creation UI ................................................................. 39
28. Co-wallet Joining UI ................................................................. 39
29. Scenario 1 Simulation Result ....................................................... 44
30. Modified Scenario 1 Simulation Result ...................................... 46
31. Scenario 2 Simulation Result ....................................................... 49
32. Scenario 3 Simulation Result ....................................................... 51
33. Scenario 4 Simulation Result ....................................................... 52
34. Project Schedule ....................................................................... 55
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Contribution</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Frontend Development Tools</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Backend Development Tools</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Typical scenarios for QR code usecases in SuperCloudPay</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Notations</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>Experimental Result for Fast Decryption</td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>Threat Models</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>Information of Scenario 1</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>Information of Scenario 2</td>
<td>46</td>
</tr>
<tr>
<td>10</td>
<td>Information of Scenario 3</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>Information of Scenario 4</td>
<td>51</td>
</tr>
</tbody>
</table>
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACP</td>
<td>Auditable Confidential Payment</td>
</tr>
<tr>
<td>ADCP</td>
<td>Auditable Decentralized Confidential Payment</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>co-wallet</td>
<td>collaborative wallet</td>
</tr>
<tr>
<td>ECC</td>
<td>elliptic curve cryptography</td>
</tr>
<tr>
<td>DCR</td>
<td>decisional composite residuosity</td>
</tr>
<tr>
<td>DDH</td>
<td>decisional Diffie–Hellman</td>
</tr>
<tr>
<td>DLIN</td>
<td>Decision Linear</td>
</tr>
<tr>
<td>e-payment</td>
<td>electronic payment</td>
</tr>
<tr>
<td>e-wallet</td>
<td>electronic wallet</td>
</tr>
<tr>
<td>ISHE</td>
<td>integrated signature and homomorphic encryption</td>
</tr>
<tr>
<td>OCR</td>
<td>Optical Character Recognition</td>
</tr>
<tr>
<td>NIZK</td>
<td>non-interactive zero-knowledge proof</td>
</tr>
<tr>
<td>P2P</td>
<td>peer-to-peer</td>
</tr>
<tr>
<td>PPT</td>
<td>probabilistic polynomial-time</td>
</tr>
<tr>
<td>q-SDH</td>
<td>q-Strong Bilinear Diffie-Hellman</td>
</tr>
<tr>
<td>UTXO</td>
<td>unspent transaction output</td>
</tr>
<tr>
<td>zk-SNARK</td>
<td>zero-knowledge succinct non-interactive argument of knowledge</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background and Motivation

Electronic wallets (e-wallets), also known as digital wallets or mobile wallets, are mobile applications that allow users to make both online and in-store payments [Yu18], using fiat money deposited therein beforehand. Some e-wallets additionally support peer-to-peer (P2P) payment, in which money can be transferred from one wallet to another. They enable users to make instant payments anytime anywhere without transaction fees. In light of such convenience, the portion of Hong Kong people owning an e-wallet surged from 65% in 2017 to 91% in 2020 according to a survey conducted in 2020 [Slo22]. Examples of local e-wallets include Alipay HK, WeChat Pay HK, PayMe, BoC Pay and Tap & Go. With the tremendous growth of the local e-wallet market in recent years, it is anticipated that e-wallets will become the most common e-payment method in Hong Kong by 2025 [Ser22].

Having said that, [Yu18] argues that there are still barriers to wide adoption. Of particular interest is privacy concerns. Transaction data, such as transaction history and the value of each transaction, is visible to e-wallet providers. They can see every transaction that is made through their e-wallets. Although personal data usage is regulated by the Personal Data (Privacy) Ordinance in Hong Kong, such protection is merely legal but not inherent. That is to say, the ordinance can only deter but not prevent e-wallet providers from abusing user financial data. Users have to trade off their financial privacy to centralized parties for electronic payment (e-payment) services.

On the other side of the e-payment ecosystem, privacy coins, or privacy-preserving cryptocurrencies, such as Zcash and Monero, provide transaction confidentiality and anonymity through cryptographic techniques, such as zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) and ring signatures [SCGGMTV14; Van13]. Nevertheless, due to their token-based nature, they are not scalable, i.e. having much higher transaction fees and much lower transaction speeds than their centralized counterparts [BCDF+22]. Moreover, the difficulty in auditing them for regulatory purposes has caused

---

1The term “e-wallets” are mostly used to refer to stored-value ones that are of interest of this project. For non-stored-value ones, unlike their store-valued counterparts in which deposits work similar as bank deposits, only payment credentials and passwords are stored [Kag22].


3Cryptocurrencies, in general, do not offer actual privacy by nature. Most cryptocurrencies, such as Bitcoin and Ether, merely provide pseudonymity.
their ambiguous legal status [DCFSS19]. Consequently, while most consumers still prefer fiat money, privacy coins are far from wide adoption.

A natural question is whether we can enjoy privacy without forgoing scalability and auditability. This motivates us to apply several cryptographic techniques, including integrated signature and homomorphic encryption (ISHE) and non-interactive zero-knowledge (NIZK) proof, used in privacy coins to the centralized architecture of e-wallets to enhance its privacy-preserving capability whilst maintaining transaction auditability for regulatory requirements. Moreover, cloud technology is leveraged to strengthen the security and resilience of the e-wallet as well as provide seamless synchronization across multiple devices. To the best of our knowledge, heretofore no existing e-wallet in the market is based on cloud technology.
1.2 Market Survey

A market survey was conducted in August 2022 to gain a deeper understanding of user needs in e-payment services for designing our e-wallet. The survey result also shows the motivation and significance of the project from a market perspective.

The market survey was launched online via Google Forms (English version) and Tencent Questionnaire (Chinese version). A total of 274 responses were collected from the Greater Bay Area.

As shown in Figure 1, the main improvement expected by the respondents is the e-wallet privacy and security. This highlights the importance of privacy and security protection in the future development of e-wallets. This has motivated us to develop a confidential transaction protocol for financial privacy (see Chapter 3) and deploy the e-wallet onto the cloud for better security (see Section 2.4).

![Expected improvements](image)

Figure 1: Expected improvement of e-wallets

Additionally, over half of the surveyed users expect to use e-wallets across multiple devices, as illustrated in Figure 2. The use of cloud technology can also satisfy such need.
Moreover, the functionalities of bill splitting and shared wallets are deemed useful by the majority of the respondents, as illustrated in Figure 3 and Figure 4.
Figure 4: Demand for shared wallets (Market Survey)

The survey data and result visualizations can be found in Appendix A.
1.3 Objectives and Scope

This project aims to develop a fully-functional mobile app prototype of a privacy-preserving cloud-based e-wallet, named SuperCloudPay, to enhance consumer e-payment experience in terms of privacy, security and convenience while ensuring validity and allowing auditability. The prototype will include basic functions of an e-wallet, including

1. Online payment
2. In-store payment
3. P2P transfer
4. Top-up
5. Withdrawal
6. Bill payment

as well as novel functions, including:

1. Promise: automatic payment by conditions (see Section 2.5.1)
2. Collaborative Wallets: for groups using decentralised consensus (see Section 2.5.2)
3. Automatic Bill Splitting: automatic bill splitting by OCR (see Section 2.5.3)

and unique features, including confidential transaction, and 2) seamless synchronization.

1. Confidential transaction: end-to-end encryption for wallet and transaction information (see Chapter 3)
2. Seamless synchronisation across multiple devices: users can access SuperCloudPay from any number of devices (such as mobile phones, tablets, computers, smartwatches) simultaneously (see Section 2.4)

1.4 Contribution

<table>
<thead>
<tr>
<th>Groupmates</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend Development</td>
<td>Liao Jiayang, Cheung Long Sang Lee Hiu Long</td>
</tr>
<tr>
<td>Backend Development</td>
<td>Cheung Long Sang</td>
</tr>
<tr>
<td>Cryptography</td>
<td>Lee Hiu Long</td>
</tr>
<tr>
<td>Interim Report</td>
<td>Lee Hiu Long</td>
</tr>
<tr>
<td>Website</td>
<td>Lee Hiu Long</td>
</tr>
<tr>
<td>Market Analysis</td>
<td>Liao Jiayang</td>
</tr>
</tbody>
</table>

Table 1: Contribution
1.5 Outline of This Report

The remainder of this report is structured as follows. Chapter 2 presents the methodology used for app development, include the frontend and backend tools and designs and cloud technology. It also includes some important implementation notes for the prototype. Chapter 3 expounds the methodology used for developing the confidential transaction protocol used in SuperCloudPay. It provides a literature review on confidential transaction, presents the confidential transaction protocol, as well as its construction, instantiation and implementation. Chapter 4 demonstrates and experiments on the protocol by a prototype demo and attack simulations. Chapter 5 concludes the report by discussing the limitations of the projects, suggesting directions for future work and summarizing the project achievements.
2 App Development Methodology

This chapter presents the methodology for our prototype development. Section 2.1 provides an overview of the system design. Sections 2.2 and 2.3 present the technology stacks and overviews for frontend and backend developments respectively. Section 2.4 discusses the use of cloud technology to enhance the e-wallet capabilities and provide better user experience. Section 2.5 introduces novel functions of SuperCloudPay. Finally, Section 2.6 add some details to the implementation of the prototype.

2.1 System Design

Figure 5 provides a high-level overview of our system architecture. The system consists of two main components: the frontend and the backend. The frontend is responsible for user interface, as a bridge between users and the cloud. The backend is responsible for managing the database on cloud servers. The frontend communicates with the backend by sending HTTP requests to retrieve or modify data in the database. The backend processes these requests, performs the necessary actions on the database, and sends back the results. We have implemented standard security measures to ensure secure communication between the frontend and backend.

In addition to the traditional system design, SuperCloudPay also implements a confidential transaction protocol for provide privacy-preserving e-payment (see Chapter 3). This means that the transaction amount is end-to-end-encrypted and the cloud does not have access to it.

Figure 5: System Design
2.2 Frontend Methodology

2.2.1 Frontend Tools

Frontend development encompasses the creation of the client-side of an application, which facilitates an interactive user interface, thereby allowing users to handle data without direct communication with the backend.

After a thorough evaluation of various native development tools, such as Swift and Java, as well as cross-platform development tools like Flutter and React Native (refers to Table 2 for a brief comparison among them), we have opted for React Native due to its cross-platform feature and community maturity.

In our project, Expo CLI is utilized to enable prototype testing before launching, thereby allowing for more efficient development. Although React Native CLI offers more iOS and Android APIs, we prefer Expo as our application does not require any APIs that are not provided by Expo.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Java</th>
<th>Swift</th>
<th>Flutter</th>
<th>React Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td></td>
<td></td>
<td>Cross-platform</td>
<td>Cross-platform</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
<td>Swift</td>
<td>Dart</td>
<td>JavaScript</td>
</tr>
<tr>
<td>Community</td>
<td>Mature</td>
<td>Mature</td>
<td>Less Mature</td>
<td>Mature</td>
</tr>
</tbody>
</table>

Table 2: Frontend Development Tools

2.2.2 Frontend Overview

A typical user’s activities on our platform can be divided into two phases: authentication and operation. During the registration process, the frontend of our application interacts with local cryptography protocols to create a new wallet. Some wallet information (not including the secret key, refers to Chapter 3 for a detailed discussion) is then sent to the backend through a registration request. The backend responds by providing an authentication token that the user can use later to fetch data from the frontend.

Once the user has successfully authenticated, they can access the homepage to perform various functions on the platform. These functions include displaying information and wallets, as well as performing transactions. Additionally, the user can access other functions such as the help center and settings.
2.3 Backend Methodology

2.3.1 Backend Tools

The backend of an application comprises a database and a server responsible for managing data and providing secure and seamless client access.

To ensure referential integrity through key constraints, PostgreSQL, a popular relational database management system, is utilized.

Restful API endpoints are designed to facilitate user-server communication, providing standardized routes for various functionalities, including registration, login, and transaction management.

For backend implementation, Django is the preferred option as it offers mature libraries that allow for rapid development. Additionally, Django provides enhanced security features, such as Cross-site scripting (XSS) protection, to prevent client-side code injections. Authentication and atomic requests can also be easily implemented with Django’s built-in libraries.

Token authentication is employed, wherein a token with a time limit is randomly generated upon user login, which the server uses for authentication, ensuring that only authorized data is accessible to users. Another authentication methods can be combined with this for security enhancement.

Table 3 provides a brief summary of tools used in backend development. Please refer to Cheung’s reports for detailed discussion in backend development tools.

<table>
<thead>
<tr>
<th>Components</th>
<th>Tools / Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>Django</td>
</tr>
<tr>
<td>Database Management System</td>
<td>PostgreSQL</td>
</tr>
<tr>
<td>Authentication</td>
<td>Token authentication</td>
</tr>
</tbody>
</table>

Table 3: Backend Development Tools

2.3.2 Backend Overview

Figure 6 and 7 shows the Entity Relationship Diagram and backend database file structure for SuperCloudPay respectively. For a detailed backend description, please refer to Cheung’s report.
Figure 6: Backend Database ER Diagram
2.4 Cloud Technology

Our e-wallet adopts a cloud-based system for the backend. AWS Cloud will be used for the cloud server for its reliability, scalability and cost-effectiveness [Ree09]. Harnessing cloud technology, our e-wallet app can be seamlessly synchronized with and accessed on multiple devices (such as smartphones, tablets, laptops, desktops and smartwatches) simultaneously. With data stored in the cloud, there is no fear of losing data due to the loss of devices or the destruction of physical servers. This strengthens the security and resilience of the e-wallet [APUST+13].
2.5 Novel Functions

2.5.1 Promise

A wallet can make a 'promise' to another wallet by specifying the payment amount and payment conditions in the 'promise'. The result is a logical AND of payment conditions. An example of a promise would be "Alice to Bob, HKD 500, after 2023-02-08 and when Alice has received HKD 1000 from Cathy". Upon 'Make Promise', the payment logic is checked for satisfiability such that the promise is successfully made if and only the formula is Boolean satisfiable.

2.5.2 Collaborative Wallets

Besides holding one personal wallet, SuperCloudPay users can hold multiple collaborative wallets (co-wallets). Collaborative wallets utilise a decentralised consensus logic called 'M out of N authorisation' for transaction authorisation. Cloud technology can be used for such a collaboration.

*M out of N authorisation.* In a collaborative wallet, whilst deposit does not require authorisation, usage and withdrawal of money requires the consent from a predefined number of users (hereinafter referred to as ‘M out of N consensus’). Once the consent condition is satisfied, the money can then be spent or withdrawn. The condition is a logical OR of a set of logical statements specifying the number of consents required from a set of users. For instance, A creates a shared wallet for himself as well as B, C, D, E, F, G, H and I, with consensus logic "2 out of A,B,C,E,G or 1 out of D,E or 3 out of F,G,H,I". All of them can deposit money into the shared wallet. Only D and E can use the money inside the shared wallet without asking for consents from other users. For B, it has to seek consents from A,C,E,G. For H, it has to seek consents from any 2 users from F, G and I. Besides "M out of a set of users", for user convenience, there are some predefined logical statements that can be used by a simple click, such as "math.ceil(N/2) out of N", meaning that consents from more than half users are required to reach the consensus. For example, "1 out of A or math.ceil(N/2) out of N" is a valid consensus logic. Only A can use the money without seeking for consents. For other users, besides themselves, they have to seek consents from 4 other users out of the 9 users including A to use the money. The consent condition can be modified by the wallet creator (i.e. the user who creates the wallet) to be approved by a M out of N consensus.

There are two types of collaborative wallets – shared wallets and organisation wallets.

**Shared Wallet.** Shared wallets are typically for family, friends and classmates. Unverified wallets have a user limit of 50, a balance limit of $5,000 and an annual transaction
limit of $25,000. Verified wallets enjoy a balance limit of $100,000 and unlimited annual transaction limit. To verify a wallet, specify a verified user as the representative. The KYC information of the representative is then binded with the wallet. The representative is obliged to bear legal responsibility for any money laundering or illicit activities associated with the wallet.

**Organisation Wallet.** Organisation wallets are typically for clubs, SMEs and community groups. The wallet balance limit is $100,000. While there is no user limit and annual transaction limit, they require identity verification. KYC documents required include the registration documents. The organisation is obliged to bear legal responsibility for any money laundering or illicit activities associated with the wallet. There are three roles in an organisation wallet – creator, administrator and participants. Administrators are appointed by the creator. They enjoy the same privileges as the creator except the right to appoint administrators. There are two types of organisation wallet – private wallet and public wallet. In private wallets, transactions and wallet balance are encrypted as in personal wallets and shared wallets. In public wallets, transaction records and wallet balances are revealed to the public. SuperCloudPay users can join a public wallet by invitation or via an invitation link. When joining the wallet, a user can choose to reveal its identity (username or phone number) or hide. The consensus process is private, i.e. the voting is anonymous.

### 2.5.3 Automatic Bill Splitting

Upon payment, a wallet can initiate a bill-splitting request with its contacts or other wallets by selecting them from its contact list, or specifying their wallet addresses, or scanning (one-time) QR Codes (see Section 2.6.1). The payment is made and confirmed if and only if every participant of the splitting has paid. A transaction is regarded as failed if it is not confirmed within 15 minutes. The amount paid is refunded afterwards. The bill-splitting request can be made both manually and automatically. The former is done by inputting the amount of each bill-splitting participant manually. The latter is done by first scanning the receipt of the meal so that each dish and its associated price is automatically recognized by Optical Character Recognition (OCR), then selecting the corresponding wallets manually. Service fee can also be considered in the bill-splitting request by one click.
2.6 Implementation Considerations

2.6.1 QR Code

QR code technology is employed by SuperCloudPay to streamline in-store payments and P2P transfers within the e-wallet. Each wallet in SuperCloudPay is associated with a unique wallet address that can be linked to one or multiple users, including collaborative wallet scenarios as discussed in Section 2.5.2. To ensure a generalized solution that does not rely on user-related information, a QR code is generated based on the wallet address.

For in-store payment scenarios, a dynamic QR code is generated that expires after a short period of time for security reasons. This QR code is created based on the wallet address and a random number related to the time of code generation and wallet balances. In P2P transfer scenarios, the payer scans the QR code, while the payee displays it. Since the QR code is used to receive payments, it can be static and directly generated from the wallet address.

To implement this solution, the wallet address is assigned to each wallet during creation, and information for the dynamic QR code is computed before each request for security purposes. A mature react native library called react-native-qrcode-svg is utilized to generate the QR code locally. Typical scenarios for QR code use cases are illustrated in Table 4.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Payer</th>
<th>Payee</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-store Payment</td>
<td>QR Code / Bar Code (Dynamic)</td>
<td>Scanning Terminal</td>
</tr>
<tr>
<td></td>
<td>Device Camera</td>
<td>QR Code (Static)</td>
</tr>
<tr>
<td>Online Payment</td>
<td>Device Camera</td>
<td>QR Code (Dynamic)</td>
</tr>
<tr>
<td>P2P Transfer</td>
<td>QR Code (Dynamic)</td>
<td>Device Camera</td>
</tr>
<tr>
<td></td>
<td>QR Code (Dynamic)</td>
<td>Device Camera</td>
</tr>
</tbody>
</table>

Table 4: Typical scenarios for QR code use cases in SuperCloudPay

2.6.2 Inconsistency

Discrepancies between the user’s local data and the data stored on the cloud server may arise due to network disruptions or local system crashes. SuperCloudPay adopts the principle of ”cloud version is the law”. In other words, the cloud version serves as the only authoritative source of information. To this end, we have implemented a mechanism whereby the cloud server responds to each user request with the current wallet state.

If the wallet state received from the cloud response matches the local wallet state, the
local wallet balance is used. However, if the wallet state from the cloud response does not match the local wallet state, the local app requests the encrypted wallet balance from the cloud. The app then proceeds to locally invoke the RevealBalance function in the cryptographic component, and updates the wallet states and balance accordingly.

2.6.3 Failure of Central Server

Central server failure could arise due to a central system crash or a cloud failure. In such scenarios, it is imperative to ensure the recovery of the database and the provision of critical services without any interruption. Ergo, we propose:

- setting up a load balancer to distribute traffic across multiple servers, thereby preventing any single server from being overwhelmed;
- implementing redundant servers and backup systems to guarantee the continuous operation of critical services;
- implementing a failover system that automatically switches to a backup server in the event of a failure.

2.6.4 Wallet Recovery

The recovery of a wallet may become necessary in case of device loss, hardware failure, local application crashes, change of devices and use of multiple devices (see Section 2.4). One must recover a wallet via an account that is associated with this wallet. For a personal wallet, the account should own the wallet. For a collaborative wallet, the account should be a participant of the wallet. In both cases, the user should provide the wallet public key, which can be generate from the wallet secret key, to the cloud to request for the information of the wallet (this is the reason for the necessity of the aforementioned ownership or participation check). After requesting the information of the wallet from the cloud, such as the wallet name, wallet state and transaction history, the user will use its wallet secret key to reveal the wallet balance and transaction amounts in the transaction history.

In view of the above recovery process, users are encouraged to export their secret key upon wallet creation to cope with the device loss and hardware failure scenarios. There are three forms of the secret key for export, including the raw secret key, the associated mnemonic phrase and the associated QR code (this shows another application of QR code technology in our project).

In the rare event that all self-recovery methods fail, users may seek assistance from regulatory authorities by providing proof of their identity (for example, every phone number
is registered in a real-name manner in Hong Kong and a phone number is required for account registration). Upon authentication by the regulatory authority, SuperCloudPay will first terminate the lost wallet, then utilize the global supervision algorithm to reveal the target wallet balance and transaction history stored in the cloud. Finally, the revealed data will be migrated to a newly created wallet.
3 Confidential Transaction Methodology

3.1 Literature Review

This chapter provides a brief literature review on existing methods for the design of a confidential transaction protocol for e-payment.

The term “confidential transaction” was coined by Maxwell in 2016 [Max16]. A confidential transaction is a transaction that shields the transaction amount from parties other than the sender and receiver(s) in a way that its validity is still verifiable.

Existing literature tends to focus on the protocols for blockchains [YGW16]. In other words, the underlying architecture is decentralized. As a result, adjustments are required to adapt the results to a centralized architecture.

Meanwhile, most foundational works such as [SCGGMVT14; Van13; Max16] adopt an unspent-transaction-output-based (UTXO-based) model. A new line of research has used an account-based model. Since all the existing e-wallets (for fiat money) adopt an account-based model (due to its accountability, auditability and traceability for regulatory, supervisory and monitoring purposes [BIMRWY20]), we chose to study the ones that adopt an account-based model. Here are some examples of such works.

[BAZB20] proposes a confidential transaction protocol on Ethereum via Ethereum smart contract. It harnesses sigma-bulletproofs (proposed in the same paper) to achieve confidentiality. For instantiation, it uses El Gamal encryption scheme with the decisional Diffie–Hellman (DDH) assumption as the underlying harness assumption.

[MDHZX20] proposes an efficient NIZK scheme for confidential transaction using additive-homomorphic encryption. Its security is guaranteed under the Decision Linear (DLIN) and q-Strong Bilinear Diffie-Hellman (q-SDH) assumptions.

[MDBHZX20] improves the practicality of [BAZB20] and [MDHZX20] by proposing an NIZK scheme for confidential transaction using Paillier Encryption secure under the decisional composite residuosity (DCR) and q-SDH assumptions.

[CMTA20] proposes an account-based auditable confidential transaction protocol that supports after-the-fact auditing (including regulation compliance and global supervision) which builds on blockchain. This work won the first prize in 2020 Financial Cryptography contest in China.
The main idea underlying [BAZB20; MDHZX20; MDBHZX20; CMTA20] is as follows:

1. User account balances and transaction amount are encrypted using additive homomorphic encryption (with Encryption algorithm denoted as $Enc$ and decryption algorithm denoted as $Dec$) under the same key. For instance, let AliceBalance and alicepk be the account balance and public key of Alice respectively. Similarly we have BobBalance and bobpk. Then, AliceBalance and BobBalance are encrypted as $A := Enc_{alicepk}(AliceBalance)$ and $B := Enc_{bobpk}(BobBalance)$ respectively. Meanwhile, suppose TxAmount denotes the transaction amount. Then Alice computes $a = Enc_{alicepk}(TxAmount)$ while Bob computes $b = Enc_{bobpk}(TxAmount)$.

2. Generate the following zero-knowledge proofs:

   (a) The encrypted transaction amounts indeed encrypt the same value by Sigma / Schnorr’s protocol. For instance, $Dec_{alicek}(a) = Dec_{bobsk}(b)$.

   (b) The encrypted transaction amount $t$ is non-negative by zero-knowledge range proof.

   (c) The sender balance is non-negative after the transaction. For instance,

   $$AliceNewBalance := Dec_{alicek}(A')$$

   is non-negative where

   $$A' = A - a = Enc_{alicepk}(AliceBalance - TxAmount)$$

   (equality by correctness of additive homomorphic encryption) by zero-knowledge range proof.

After discreet consideration, this project adopts a variant of the Auditable Decentralized Confidential Payment (ADCP) system proposed in [CMTA20] with necessary conversions (from a decentralized architecture to a centralized architecture) for its conceptual simplicity and practical functionality. It is an account-based auditable confidential transaction system that supports after-the-fact auditing for regulatory compliance and global supervision. This echoes the project objective of allowing the transactions to be auditable.
3.2 Auditable Confidential Payment System

Our project adopts an Auditable Confidential Payment (ACP) System, a variant of the ADCP system defined, constructed and instantiated in [CMTA20], for our confidential transaction protocol.

An ACP system is formally defined as follows:

**Definition 3.1 (Auditable Confidential Payment System).** An Auditable Confidential Payment (ACP) system is a 9-tuple \((\text{Setup}, \text{CreateWallet}, \text{RevealBalance}, \text{CreateCTx}, \text{VerifyCTx}, \text{UpdateCTx}, \text{JustifyCTx}, \text{AuditCTx}, \text{OpenCTx})\) of probabilistic polynomial-time (PPT) algorithms such that

- **pp** ← **Setup**\((1^\lambda, 1^\kappa)\): The setup algorithm **Setup** ran by the server takes as input a security parameter \(1^\lambda\) and the maximum balance supported \(1^\kappa\), outputs a public parameter pp which includes an auditor public key \(pk_a\).

- \((pk, sk)\) ← **CreateWallet()**: The wallet creation algorithm **CreateWallet** ran by a user outputs a public key pair \((pk, sk)\).

- \(b := \text{RevealBalance}(sk, B)\): The balance reveal algorithm **RevealBalance** ran by a user takes as input a secret key \(sk\) and an encrypted balance \(B\), outputs the plaintext balance \(b\).

- \(ctx := \text{CreateCTx}(sk_s, pk_s, pk_r, v)\): The confidential transaction creation algorithm **CreateCTx** ran by a sender takes as input a sender secret key \(sk_s\), a sender public key \(pk_s\), a receiver public key \(pk_r\) and a transaction amount \(v\), outputs a confidential transaction \(ctx = (pk_s, pk_r, C_s, C_r, C_a, \sigma, \pi)\) where \(\sigma\) and \(\pi\) denote the signature and validity proof of the transaction respectively.

- \(\text{bit} := \text{VerifyCTx}(ctx)\): The confidential transaction verification algorithm **VerifyCTx** ran by the server takes as input a confidential transaction \(ctx\), outputs 0 if it is valid, 1 otherwise.

- \((B'_s, B'_r) := \text{UpdateCTx}(B_s, B_r, ctx)\): The balance update algorithm **UpdateCTx** ran by the server takes as input a sender wallet balance \(B_s\), a receiver wallet balance \(B_r\) and a confidential transaction \(ctx\), outputs the updated sender wallet balance \(B'_s\) and updated receiver wallet balance \(B'_r\).

- \(\pi := \text{JustifyCTx}(pk, sk, \{ctx\}, f)\): The confidential transaction justification algorithm ran by a user takes as input a public key \(pk\), a secret key \(sk\), and a set of confidential transactions \(\{ctx\}\) that it participated in and a policy \(f\), outputs a proof \(\pi\) for \(f(pk, \{ctx\}) = 1\) for auditing.
• bit ← AuditCTx(pk, {ctx}, f, π): The confidential transaction audit algorithm ran by an auditor takes as input a public key pk, a set of confidential transactions {ctx}, a policy f and a proof π, outputs 0 denoting accept or 1 denoting reject.

• v := OpenCTx(sk_a, ctx): The confidential transaction open algorithm ran by an auditor takes as input an auditor secret key sk_a and a confidential transaction ctx, outputs the plaintext transaction amount v.

The correctness and security model of an ACP system follow from those of an ADCP system by altering the equivalence of input-output relations in the two systems and removing requirements associated with decentralized functionalities.

An ISHE scheme and NIZK proof systems are used for construction as depicted in Figure 8. Informally, an ISHE scheme is a scheme that combines a digital signature scheme and a homomorphic (public-key) encryption scheme such that a single key pair is used in both signature (for authenticity) and encryption (for confidentiality). Meanwhile, an NIZK system can be informally described as a system allowing a prover to prove a statement to a verifier without leaking any information and interacting with the verifier. Exploiting these two cryptographic objects, the validity of a transaction can be verified without revealing the transaction amount and user wallet balances. Please refer to [CMTA20] for formal definitions of an ISHE scheme and a NIZK system and their security definitions.

Figure 8: ACPS Construction Sketch

The instantiation in [CMTA20], which utilizes Twisted El-Gamal Encryption, a variant of El Gamal Encryption proposed in the same paper, is also used in our project.
3.3 ISHE Instantiation

ISHE is instantiated by Twisted El-Gamal Encryption and Schnorr Signature in [CMTA20] as follows:

- \( pp \leftarrow \text{Setup}(1^\lambda) \): run \((G, g, p) \leftarrow \text{GroupGen}(1^\lambda), h \overset{\$}{\leftarrow} G^*\), output \( pp = (G, g, h, p) \) as public parameters. The randomness and message spaces are \( \mathbb{Z}_p \).

- \((pk, sk) \leftarrow \text{KeyGen}(pp)\): on input \( pp \), choose \( sk \overset{\$}{\leftarrow} \mathbb{Z}_p \), set \( pk = g^{sk} \).

- \( C \leftarrow \text{Enc}(pk, m; r) \): compute \( X = pk^r, Y = g^rh^m \), output \( C = (X, Y) \).

- \( m := \text{Dec}(sk, C) \): parse \( C = (X, Y) \), compute \( h^m = Y/X^{sk^{-1}} \), recover \( m \) from \( h^m \).

- \( \sigma \leftarrow \text{Sign}(sk, m) \): pick \( r \overset{\$}{\leftarrow} \mathbb{Z}_p \), set \( A = g^r \), compute \( e = H(m, A), z = r + sk \cdot e \mod p \) where \( H \) is a predefined cryptographic hash function, output \( \sigma = (A, z) \).

This instantiation is CPA-secure as proven in [CMTA20].

3.4 NIZK Instantiation

3.4.1 NIZK for Transaction Amount Equality

The following shows the instantiation of NIZK Proof of Knowledge for equality proof on instance \((pk_s, pk_r, pk_a, X_s, X_r, X_a, Y)\) by applying Fiat-Shamir transform to the Sigma protocol proposed in [CMTA20]:

1. Prover
   
   (a) pick \( a, b \overset{\$}{\leftarrow} \mathbb{Z}_p \)
   
   (b) compute \( A_s = pk_s^a, A_r = pk_r^a, A_a = pk_a^a \) and \( B = g^ah^b \)
   
   (c) compute \( e \) by hashing the transcript of interaction in the Sigma protocol (i.e. the instances and \( A_s, A_r, A_a, B \))
   
   (d) compute \( z = a + e \cdot r \mod q \) and \( t = a + e \cdot v \mod q \) where \( q \) is the order of the group
   
   (e) output \( \pi = (A_s, A_r, A_a, B, z, t) \)

2. Verifier accepts if and only if the following four equations hold simultaneously:
   
   \( pk_s^z = A_sX_s^z, pk_r^z = A_rX_r^z, pk_a^z = A_aX_a^z \) and \( g^zh^t = BY^e \)

This instantiation is secure as proven in [CMTA20].

Note that for a valid confidential transaction, \((X_s, X_r, X_a, Y) = (C_s.X, C_r.X, C_a.X, C_s.Y)\).
3.4.2 NIZK for Non-Negative Transaction Amount and Sufficient Sender Wallet Balance

The instantiation is provided in [BBBPWM18]. For implementation, we use the python-bulletproofs library.
3.5 Confidential Transaction Protocol

Our confidential transaction protocol adopts an Auditable Confidential Payment (ACP) system (Setup, CreateWallet, RevealBalance, CreateCTx, VerifyCTx, UpdateCTx, JustifyCTx, AuditCTx, OpenCTx). For the definitions, constructions and instantiations of each algorithm, please refer to Section 3.2. There are three main roles in the protocol, including Sender, Receiver and Cloud. Sender refers to any sender who would like to initiate a transaction. Receiver refers to the receiver(s) of the transaction. Cloud refers to the back-end server built on cloud. The protocol consists of seven subprotocols that support the core functionalities of the e-wallet app, namely, System Setup, Register, Login, Send, Receive, Online Payment, In-store Payment, Top-Up, Withdrawal, Bill Payment, Bill Splitting, Auditing and Global Supervision. The subprotocols are described in the following subsections. Note that the protocol can be applied to encompass multiple currency by simply including multiple balances for a wallet.

For brevity, we use the following notations for discussions hereinafter.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enc</td>
<td>ISHE Encryption Algorithm</td>
</tr>
<tr>
<td>Dec</td>
<td>ISHE Decryption Algorithm</td>
</tr>
<tr>
<td>pk_s</td>
<td>Sender public key</td>
</tr>
<tr>
<td>pk_r</td>
<td>Receiver public key</td>
</tr>
<tr>
<td>pk_a</td>
<td>Auditor public key</td>
</tr>
<tr>
<td>sk_s</td>
<td>Sender secret key</td>
</tr>
<tr>
<td>sk_r</td>
<td>Receiver secret key</td>
</tr>
<tr>
<td>sk_a</td>
<td>Auditor secret key</td>
</tr>
<tr>
<td>v</td>
<td>transaction amount in plaintext</td>
</tr>
<tr>
<td>C_s</td>
<td>transaction amount ciphertext, encrypted under sender public key</td>
</tr>
<tr>
<td>C_r</td>
<td>transaction amount ciphertext, encrypted under receiver public key</td>
</tr>
<tr>
<td>C_a</td>
<td>transaction amount ciphertext, encrypted under auditor public key</td>
</tr>
<tr>
<td>B_s</td>
<td>sender wallet balance ciphertext, encrypted under sender public key</td>
</tr>
<tr>
<td>B_r</td>
<td>receiver wallet balance ciphertext, encrypted under receiver public key</td>
</tr>
</tbody>
</table>

Table 5: Notations

3.5.1 System Setup Protocol

The System Setup Protocol is used once only when we initiate our e-wallet system. It consists of two steps. First, the setup algorithm Setup is ran by the server to obtain a
set of public parameters for the whole system. Then, an initialization for fast decryption (see 3.6.4) is done to optimize the efficiency of the system.

3.5.2 Register Protocol

It is only used upon account creation. It consists of four steps.

1. The user enters its username, phone number and password to the app.

2. The wallet creation algorithm CreateWallet is triggered by frontend to create a personal wallet associated with the username, a public key, a secret key and a balance (initially 0).

3. The username, phone number, password and public key are sent to Cloud.

4. Cloud stores them as well as an encrypted wallet balance to Cloud. The encrypted wallet balance should be an encryption of 0, which can be obtained by \( \text{ISHE.Enc}(pk, 0; r) \) where \( r \xleftarrow{s} \mathbb{Z}_p \).

One similarly defines the wallet creation protocol for collaborative wallets.

3.5.3 Login Protocol

1. Upon login, check the wallet state is same with the cloud version. If not replace the wallet balance by the decryption of the cloud balance ciphertext \( B \) by running RevealBalance(\( sk, B \)).

3.5.4 Send Protocol

1. Sender gets the receiver public key \( pk_r \)

   (a) by manually input the wallet address, or

   (b) by scanning a QR code, or

   (c) from the contact book, or

   (d) etc

2. Sender creates a confidential transaction \( ctx \) of transaction amount \( v \) and sends it to Cloud by running \( ctx \xleftarrow{} \text{CreateCTx}(sk_s, pk_s, pk_r, v) \) where \( ctx = (pk_s, pk_r, C_s, C_r, C_a, \sigma, \pi) \) with \( \sigma \) and \( \pi \) denoting the signature and validity proof of the transaction respectively.
3. **Cloud** checks if the transaction is valid, i.e. satisfying the following conditions: 1) \( C_s, C_r \) and \( C_a \) correspond to the same plaintext transaction amount \( v \); 2) \( v \) is non-negative; and 3) the wallet balance of **Sender** is sufficient, by \( \text{VerifyCTx} \). If not, it sends an error message to **Sender**, and aborts the phase.

4. Finally, **Cloud** runs \( \text{UpdateCTx} \) to decrease the encrypted sender wallet balance and increase the encrypted receiver wallet balance. The state of each wallet balance is increased by 1. After that, a message denoting successful transaction is sent to both **Sender** and **Receiver**.

5. **Sender** decreases the wallet balance by \( v \) and increases the wallet state by 1.

For **Receiver**, the wallet balance and wallet state will get updated when **Receiver** logs in to the app. See Section 3.5.3.

Similarly, one can define the protocol for the Promise novel function.

### 3.5.5 Receive Protocol

1. **Receiver** requests a confidential transaction by sending a transaction amount ciphertext \( C = \text{Enc}(pk_s, v) \) that encrypts the transaction amount \( v \) under the sender public key \( pk_s \) and a memo \( \text{memo} \) to **Cloud**.

2. **Cloud** forwards \((C, \text{memo})\) to **Sender**.

3. **Sender** creates a confidential transaction \( ctx \) of transaction amount \( v \) and memo \( \text{memo} \) and sends it to **Cloud** by running \( ctx \leftarrow \text{CreateCTx}(sk_s, pk_s, pk_r, v) \) where \( ctx = (pk_s, pk_r, C_s, C_r, C_a, \sigma, \pi) \) with \( \sigma \) and \( \pi \) denoting the signature and validity proof of the transaction respectively.

4. **Cloud** forwards \( C_r \) to **Receiver** as a transaction request.

5. **Receiver** decrypts \( C_r \) by its secret key \( sk_r \) to obtain a plaintext value \( \hat{v} = \text{Dec}(sk_r, C_r) \), and checks if it is the transaction amount agreed upon. Then, it tells **Cloud** whether it accepts the transaction request. In case it does not, the phase is aborted.

6. **Cloud** checks if the transaction is valid, i.e. satisfying the following conditions: 1) \( C_s, C_r \) and \( C_a \) correspond to the same plaintext transaction amount \( v \); 2) \( v \) is non-negative; and 3) the wallet balance of **Sender** is sufficient, by \( \text{VerifyCTx} \). If not, it sends an error message to both **Sender** and **Receiver**, and aborts the phase.

7. Finally, **Cloud** runs \( \text{UpdateCTx} \) to decrease the encrypted sender wallet balance and increase the encrypted receiver wallet balance. The state of each wallet balance
is increased by 1. After that, a message denoting successful transaction is sent to both Sender and Receiver.

8. Sender decreases the wallet balance by $v$ and increases the wallet state by 1. Receiver increases the wallet balance by $v$ and increases the wallet state by 1.

### 3.5.6 In-store Payment Protocol

It has an additional step of the merchant (Receiver) obtaining the consumer (Sender) public key compared to the Receive protocol (see Section 3.5.5).

1. Sender shows its wallet payment code, which is a static QR code that embeds its public key $pk_s$, to the Point-Of-Sales terminal.

2. Receiver requests a confidential transaction by sending a transaction amount ciphertext $C = \text{Enc}(pk_s, v)$ that encrypts the transaction amount $v$ under the sender public key $pk_s$ and a memo $memo$ to Cloud.

3. Cloud forwards ($C, memo$) to Sender.

4. Sender creates a confidential transaction $ctx$ of transaction amount $v$ and memo $memo$ and sends it to Cloud by running $ctx \leftarrow \text{CreateCTx}(sk_s, pk_s, pk_r, v)$ where $ctx = (pk_s, pk_r, C_s, C_r, C_a, \sigma, \pi)$ with $\sigma$ and $\pi$ denoting the signature and validity proof of the transaction respectively.

5. Cloud forwards $C_r$ to Receiver as a transaction request.

6. Receiver decrypts $C_r$ by its secret key $sk_r$ to obtain a plaintext value $\tilde{v} = \text{Dec}(sk_r, C_r)$, and checks if it is the transaction amount agreed upon. Then, it tells Cloud whether it accepts the transaction request. In case it does not, the phase is aborted.

7. Cloud checks if the transaction is valid, i.e. satisfying the following conditions: 1) $C_s, C_r$ and $C_a$ correspond to the same plaintext transaction amount $v$; 2) $v$ is non-negative; and 3) the wallet balance of Sender is sufficient, by VerifyCTx. If not, it sends an error message to both Sender and Receiver, and aborts the phase.

8. Finally, Cloud runs UpdateCTx to decrease the encrypted sender wallet balance and increase the encrypted receiver wallet balance. The state of each wallet balance is increased by 1. After that, a message denoting successful transaction is sent to both Sender and Receiver.

9. Sender decreases the wallet balance by $v$ and increases the wallet state by 1. Receiver increases the wallet balance by $v$ and increases the wallet state by 1.
3.5.7 Online Payment Protocol

It is just a special case of the Send Protocol (see Section 3.5.4) in which Sender gets the merchant (Receiver) public key by redirecting from an online store.

3.5.8 Top-Up Protocol

It is just a special case of the Receive Protocol (see Section 3.5.5) in which a convenient store or a bank (owning a SuperCloudPay wallet) acts as Sender.

3.5.9 Withdrawal Protocol

It is just a special case of the Send Protocol (see Section 3.5.4) in which a bank (owning a SuperCloudPay wallet) acts as Receiver.

3.5.10 Bill Payment Protocol

It is just a special case of the Send Protocol (see Section 3.5.4) in which a merchant (owning a SuperCloudPay wallet) acts as Receiver.

3.5.11 Bill Splitting Protocol

It is just a special case of the In-store Payment Protocol (see Section 3.5.6) in which there are multiple Sender specified by the user requesting the bill splitting.

3.5.12 Auditing Protocol

The Auditing protocol is used when an auditor requests to check the satisfiability of certain confidential transactions $\{ctx\}$ of a user towards a regulation policy $f$. A typical usecase is the user is suspected (but not yet convicted) for criminal transactions. It consists of five steps.

1. Cloud requests the user to generate a proof for $\{ctx\}$ with respect to $f$.

2. The user runs the confidential transaction justification algorithm JustifyCTx for $\{ctx\}$ with respect to $f$ to obtain a corresponding proof $\pi$.

3. The user sends $\pi$ to Cloud.

4. Cloud forwards $\pi$ to the regulator.

5. The auditor runs the confidential transaction audit algorithm AuditCTx for $\{ctx\}$ with respect to $f$ to check whether $\{ctx\}$ complies with $f$. If not, subsequent regulatory actions may be performed by the auditor.
3.5.13 Global Supervision Protocol

The Global Supervision Protocol is used when a supervisor requests to inspect a confidential transaction $ctx$. A typical usecase is the user is convicted for a criminal transaction. It consists of only one step of the supervisor running the confidential transaction open algorithm $OpenCTx$ to get the plaintext transaction amount. Any implication of the amount may suggest a subsequent legal action.

3.6 Implementation Considerations

3.6.1 Elliptic Curve Cryptography

The underlying security assumption of the instantiation (i.e. Twisted El Gamal) is the DDH assumption, which involves discrete logarithms in finite cyclic groups [CMTA20]. Instead of working over the multiplicative group of integers modulo $n$ (i.e. $(\mathbb{Z}_n)^\times$) as in ”textbook RSA”, we choose work over a prime-order elliptic curve over a finite field with a prime order, i.e. $\mathbb{F}_p = \{0, \ldots, p-1\}$ with $p$ prime $^4$. This is because elliptic curve cryptography (ECC) can achieve the same level of security while using a smaller key size. For example, a 224-bit ECC provides comparable security to a 2048-bit RSA [GPWES04]. Hence, ECC enjoys smaller keys, ciphertexts and signatures, resulting in higher efficiency [KMV00]. This is especially useful since we require the frontend to run cryptographic operations.

3.6.2 Curve

According to Version 9.1 of “IT Security Guidelines by Office of the Government Chief Information Officer” published in August 2022, ”for information classified as CONFIDENTIAL or above, the symmetric encryption key length shall be at least 128-bit for the AES encryption or equivalent, whereas the asymmetric encryption key length shall be at least 2048-bit for the RSA encryption. Alternatively, the requirement can be met by Elliptic Curve Cryptography (ECC) encryption with key length of at least 224-bit or equivalent”$^5$. Since Curve448 offers 224-bit security, it is chosen to be the elliptic curve of interest.

$^4$Every field of order $p$ is isomorphic to $\mathbb{F}_p$.

3.6.3 Precision

As an e-wallet in Hong Kong, SuperCloudPay should support float of 2 decimal places. However, as mentioned in Section 3.6.1, our cryptographic operations are working on a finite group over $\mathbb{F}_p$. To support such precision to Hong Kong cent, in $\text{CreateCTx}$, the message value to be encrypted should be set as 100 times the transaction amount, while in $\text{RevealBalance}$ the plaintext message obtained should be divided by 100 to reveal the “real value” of wallet balance. Note that as long as $v < (p - 1)/100$, we have $100 \cdot v < p$, which means multiplying $v$ will not resulting in collision. Meanwhile, the order of the field underlying Curve448 is $726838724295606890549323807888004534353641360687318060281490199\cdots$ and a typical transaction amount is much smaller than this number. As a result, the correctness of our protocol is not compromised by supporting precision to Hong Kong cent.

3.6.4 Fast Decryption

As shown in 3.3, in the last step of decryption, we have to compute the discrete log of some values. Note that a brute-force algorithm requires exponential time. The most efficient classical algorithm, namely Pollard’s $\rho$ method, stills requires $O(\sqrt{|G|})$ where $|G|$ denotes the order of the group [HM11]. In our case, the order of the group associated to Curve448 is $18170968107390172263733095197200113358841034017182951507037254979514600396153\cdots$ However, recall that our message space $\mathcal{M} = \{x : 0 \leq \text{MAX\_BAL} \cdot 100\}$ is comparatively small since a typical MAX\_BAL is 100,000. Hence, a hash table can be used for look-up, in which case it’s just a $O(1)$ operation. Undoubtedly, there are trade-offs, including the time for creating the hash table, saving it to json and loading the json in frontend. For a MAX\_BAL 100,000, on a MacBook Pro with 2.6 GHz 6-Core Intel Core i7 processor, the time for creating a ciphertext dictionary (in fact this name is inaccurate as the dictionary is used for the last step of decryption only, but we will keep this name for better readability) then saving it to json is 110.4853 minutes, which is still manageable in terms of the setup for the whole system, as it is run by us only. For loading the json file, it requires 15.3869 seconds. You may find it impractical as this seems imply decryption takes at least 15 seconds, but the json file can be loaded when the user open the app, even before the user logs in to the app. By using a python dictionary, the decryption time is 0.007 seconds, which is independent of MAX\_BAL, and much faster than Pollard’s $\rho$ method. The following table summarizes the experimental result:
<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating ciphertext dictionary</td>
<td>110.2153 minutes</td>
</tr>
<tr>
<td>Creating ciphertext json</td>
<td>16.1961 seconds</td>
</tr>
<tr>
<td>Loading ciphertext json</td>
<td>15.3869 seconds</td>
</tr>
<tr>
<td>Decryption</td>
<td>0.0070 seconds</td>
</tr>
</tbody>
</table>

Table 6: Experimental Result for Fast Decryption

The result shown is a simple average of the results obtained by running the first operation for 10 times for and the remaining operations for 100 times).

### 3.6.5 Secure Hash Function

Note that the python native hash algorithm is not cryptographically-secure. Hence, we adopts the `hashlib` library for cryptographic hash function, and **SHA-256** is chosen to be the hash function that is used for hashing in our protocol, such as the signature for a pre-confidential transaction and the Fiat–Shamir challenge in a NIZK proof.

### 3.6.6 Implementation

The protocol has been implemented in Python with the `secrets` library for cryptographic strong psuedonumber generation, the `hashlib` library for cryptographic hash function, the `ECPy` library for elliptic curve functionalities and `python-bulletproofs` for zero-knowledge range proof.

### 3.6.7 Documentation

To facilitate app development, an internal documentation for cryptographic algorithms APIs are compiled. An example page is shown in Figure 9.
4 Results and Discussions

In this chapter, the confidential protocol is visualized through our prototype by showcasing how honest users use our app. Then we will simulate how malicious users attack our protocol and verify the security of our protocol.

For the experiments, we choose to not to deploy on the cloud as it suffices to simulate Cloud by local backend server for demonstration purpose.

4.1 User Interface Showcase

4.1.1 Basic Functions

Upon entering SuperCloudPay, the login page (Figure 10) is shown. If the user has not registered for a SuperCloudPay account, it will have to browse to the register page (Figure 11) prior to login to the app. A successful registration must involve a valid username, password, phone number (with verification) and an assent to the terms of service (Figure 12).

After successfully logging in to the app, the homepage (Figure 13) will be display. The homepage provides an entrance to many functions. First, the icon on the top left corner can be clicked to show the sidebar (Figure 14) in which some wallet information and a navigation are shown. For instance, when the user clicks "My Profile", it will navigate to
the user profile page (Figure 15) where user account information and a wallet dashboard are shown.

The user can return to the homepage via the bottom navigation bar. In fact, the bottom navigation bar provides navigation to the transaction history (Figure 16), scan (Figure 17), payment code (Figure 18) and wallet dashboard pages (Figure 19).
From the small buttons in the middle of the homepage (Figure 13), users can navigate to the pages of other basic functions, namely Send (Figure 20), Receive (Figure 21), Top-Up (Figure 22), Withdrawal (Figure 23) and Bill Payment (Figure 24).

4.1.2 Novel Functions

The last two functions to demonstrate are two of our novel functions – Promise and Collaborative Wallet, whose descriptions are detailed in Section 2.5.1 and Section 2.5.2 respectively.
On the collaborative wallet page, by clicking the first and second bottom buttons, one can navigate to Co-wallet Creation (Figure 27) and Co-wallet Joining (Figure 28) pages respectively.
4.2 Prototype Demo

The links for demo videos are as follows:

- System Setup: https://connecthkuhk-my.sharepoint.com/:v:/g/personal/u3568707_connect_hku_hk/Ed1wwRm3rtxL1mRQBbWzSD0B5IK9rbi-KbgY8SfTdzMBGQ?e=SSXZqA
- Core Functions: https://connecthkuhk-my.sharepoint.com/:v:/g/personal/u3568707_connect_hku_hk/EQxujsP9d39HsJ58DuTLH3UBamRb5JKE_0ml225VgDo8hA?e=7ljadV
- Global Supervision: https://connecthkuhk-my.sharepoint.com/:v:/g/personal/u3568707_connect_hku_hk/EaNzCRt0g59Jv508Fz928I8BELqTV_pgpsNigmQ0jYqVRA?e=avZDkF
4.3 Attacks and Defense

4.3.1 Threat Models

Naturally, we have the following threat models:

<table>
<thead>
<tr>
<th>Threat Models</th>
<th>Security Goals</th>
<th>Relevant Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious Cloud</td>
<td>Confidentiality, Validity, Auditability</td>
<td>N/A</td>
</tr>
<tr>
<td>Honest but Curious Cloud</td>
<td>Confidentiality</td>
<td>N/A</td>
</tr>
<tr>
<td>Malicious Sender / Customer</td>
<td>Validity</td>
<td>Conditions 1, 2, 3</td>
</tr>
<tr>
<td>Malicious Receiver / Merchant</td>
<td>Validity</td>
<td>Condition 2</td>
</tr>
</tbody>
</table>

Table 7: Threat Models

where conditions 1, 2 and 3 respectively denote the three statements for which zero-knowledge proofs are generated, as stated in Section 3.1, i.e.

Condition 1 The encrypted transaction amounts encrypt the same value, i.e. \( v = \text{Dec}_{sk_s}(C_s) = \text{Dec}_{sk_r}(C_r) = \text{Dec}_{sk_a}(C_a) \);

Condition 2 The encrypted transaction amount is non-negative in plaintext, i.e. \( v \geq 0 \);

Condition 3 The sender balance is non-negative after the transaction, i.e. \( \text{SenderNewBalance} = \text{Dec}_{sk_s}(B'_s) \) is non-negative where \( B'_s = B_s - C_s = \text{Enc}_{pk_s}(\text{SenderOriginalBalance} - v) \) (equality by correctness of additive homomorphic encryption).

An implicit assumption behind our centralized architecture is the honesty of Cloud. For "Honest but Curious Cloud", confidentiality follows from the semantic security of ISHE. More specifically, throughout the whole protocol, the transaction amount is end-to-end encrypted (homomorphically) by ISHE. In other words, it is only known to Sender and Receiver, and the medium of transfer (i.e. Cloud) does not have access to it. This can ensure the confidentiality of the transaction, and can thus achieving the security goal.

By Table 7, possible attacks to our confidential transaction protocol are:

- Malicious Sender violating condition 1;
- Malicious Sender violating condition 2;
- Malicious Sender violating condition 3;
- Malicious Sender violating conditions 1 and 2;
- Malicious Sender violating conditions 1 and 3;
- Malicious Sender violating conditions 2 and 3;
- Malicious **Sender** violating conditions 1, 2 and 3;
- Malicious **Receiver** violating condition 2;
- Malicious **Sender** and **Receiver** violating a combination of conditions 1, 2 and 3.

Note that a malicious receiver can only violate condition 2. Receiver’s inability to violate condition 3 is obvious as condition 3 is related to the solvability of sender. For condition 1, recall that in all payment contexts, **Sender** is responsible for producing the ciphertexts $C_s, C_r, C_a$. Meanwhile, under all subprotocols in which **Receiver** initiates the confidential transaction, **Sender** is still the one who creates the confidential transaction (object) $ctx$. Hence, a checking for the parity of the transaction amount $v = \text{Dec}_{sk_s}(C)$ where $C := \text{Enc}_{pk_s}(v)$ can be done in frontend. One may wonder the case when **Sender** is also malicious and hence not creating a $ctx$ through our app. In fact, it reduces to the case that malicious **Sender** violating condition 2 because if $C_s$ and $C_r$ are not encrypting the same (negative) value, then condition 1 is violated. Thus, we will not check this case.

Since the validity checks are performed in an “AND” manner, it suffices to verify that any confidential transaction violating conditions 1, 2 and 3 separately cannot pass the validity check. As presented in Section 3.5, there are 7 payment contexts. To avoid repetition, instead of exhausting all scenarios checking all 28 cases, we choose to perform the experiments under the following four scenarios, which already encompass the fundamental protocol logic:

1. Different transaction amounts in P2P Transfer
2. Different transaction amounts in In-store Payment
3. Negative transaction amounts in P2P Transfer
4. Insufficient Sender Balance in P2P Transfer

Note that the above attacks cannot be conducted through the app. They are also not related to the cloud backend. Hence, we can simulate the attacks in Python without connecting to the frontend and backend. Moreover, in the simulations, we can play all four roles, **Cloud**, **Sender**, **Receiver** and **Auditor** without logging in and out.

### 4.3.2 Experimental Setup

Listing 1 shows the codes for the setup for the simulations.

```python
# import utilities
from utils import *
```
# import PublicParams
from params import *

# import crypto algorithms
from acps import *
import ishe
import nizk

pp: PublicParams = Setup (MAX_BAL=100)
# The global supervising authority generates a public key on Curve448.
(pka, ska) = ishe.KeyGen(pp)
pp.pka = pka
pp.fastDecryption()
cipher_dict = load_json ('ciphertext_space.json')

sender_wallet: Wallet = CreateWallet(name="Sender", info="", pp=pp)
pks = sender_wallet.pk
sks = sender_wallet.sk
Bs: Ciphertext = ishe.Enc(pk=pks, m=50*100, pp=pp,
r=secrets.randbelow(pp.cv.field))
sender_balance = RevealBalance(sender_wallet, Bs, cipher_dict, pp)
# print("Original sender wallet cloud balance:", sender_balance)

receiver_wallet: Wallet = CreateWallet(name="Sender", info="", pp=pp)
pkr = receiver_wallet.pk
skr = receiver_wallet.sk
Br: Ciphertext = ishe.Enc(pk=pkr, m=10*100, pp=pp,
r=secrets.randbelow(pp.cv.field))
receiver_balance = RevealBalance(receiver_wallet, Br, cipher_dict, pp)
# print("Original receiver wallet cloud balance:", receiver_balance)
print()

Listing 1: Experimental Setup

Explanation. Recall the *100 operation is for precision. Note that since our experiments are independent of the efficiency of the protocol, we have chosen '100' to be the maximum balance MAX_BAL that our e-wallet supports (in this simulations) for better efficiency in the simulations. Moreover, the original sender and receiver wallet balances are set merely arbitrarily as they are independent of the simulations (as long as the transaction value is larger than 50 in Scenario 4).
4.3.3 Scenario 1 - Different transaction amounts in P2P Transfer

<table>
<thead>
<tr>
<th>Scenario Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious Player</td>
</tr>
<tr>
<td>Payment Context</td>
</tr>
<tr>
<td>Condition Violated</td>
</tr>
</tbody>
</table>

Table 8: Information of Scenario 1

Procedure.

1. **Sender** gets the receiver public key (a) by scanning a QR code, (b) by redirecting from an online store, (c) from the contact book, or (d) etc.

2. **Sender** creates a confidential transaction.

3. **Sender** forges the transaction amount encrypted under receiver public key and sends the updated confidential transaction to Cloud.

4. **Cloud** checks the validity of the transaction.

Implementation. Listing 2 shows the codes for the simulation.

```python
# Step 1 - Sender gets the receiver public key.
# nothing to do in this simulation

# Step 2a - Sender initiates a confidential transaction.
# transaction amount
v = 5
memo = "Scenario 1 - Simple Forge"

tx: CTx = CreateCTx(sender_wallet, pkr, v, memo, pp)

# Step 2b - Sender forges Cr and send the confidential transaction to Cloud.
vee = 10  # fake tx amount encrypted under receiver pk; *100 for precision
ctx.Cr = ishe.Enc(pkr, vee*100, pp, secrets.randbelow(pp.cv.field))

# Step 3 - Cloud checks the validity of the transaction.
validity: bool = VerifyCTx(ctx, pp)
if validity:
    print("The transaction is valid.")
else:
    print("The transaction is invalid.")
print()
```

Listing 2: Scenario 1
**Result and Discussion.** Figure 29 shows the result of this simulation. The transaction is invalid. In fact, not even the signature is valid. This is because the ciphertext is changed but the signature is only valid for the original ciphertext (by collision resistance of SHA-256 hash function). A modification to this attack is "forge from beginning", that is to say, simulating the whole \texttt{CreateCTx} algorithm instead of invoking it and mutate a component \((C_r)\) of its output \((ctx)\).

![Figure 29: Scenario 1 Simulation Result](image)

**Procedure - Modified.**

1. **Sender** gets the receiver public key \((a)\) by scanning a QR code, \((b)\) by redirecting from an online store, \((c)\) from the contact book, or \((d)\) etc.

2. **Sender** creates a malicious confidential transaction by simulating \texttt{CreateCTx} maliciously (fake \(C_r\)).

3. **Cloud** checks the validity of the transaction.

**Implementation - Modified.** Listing 3 shows the codes for the modified simulation.

```python
# Step 1 - Sender gets the receiver public key.
# nothing to do in this simulation
```

44
Step 2 - Sender initiates a malicious confidential transaction by simulating CreatCTx.

```python
v = 5
vee = 10  # fake tx amount encrypted under receiver pk
r = secrets.randbelow(pp.field)

Cs = ishe.Enc(pks, int(v*100), pp, r)  # tx amount / sender pk
Cr = ishe.Enc(pkr, int(vee*100), pp, r)  # fake amount / receiver pk
Ca = ishe.Enc(pp.pka, int(v*100), pp, r)  # tx amount / supervisor pk

# generate NIZK proofs for the three conditions
proof = nizk.Prove(pks, pkr, pp.pka, Cs, Cr, Ca, r, v, pp)

# sign the pre-ctx
message = str(sender_wallet.state)+str(sender_wallet.pk)+str(pkr)+str(pka)+str(Cs)+str(Cr)+str(Ca)+str(proof)
sigma = ishe.Sign(sender_wallet.sk, secure_hash(message), pp)

memo = "Scenario 1 - Complex Forge"
ctx = CTx(sender_wallet.pk, pkr, Cs, Cr, Ca,
          sender_wallet.state, memo, proof, sigma)
```

# Step 3 - Cloud checks the validity of the transaction.
validity: bool = VerifyCTx(ctx, pp)
if validity:
    print("The transaction is valid.
")
else:
    print("The transaction is invalid.
")
```

Listing 3: Scenario 1 - Modified

**Result and Discussion.** Figure 30 shows the result of this simulation. The transaction is invalid. This is because Condition 1 is violated and hence $pkr^z \neq A_rX_r^z$. Consequently, condition$_r$ is False as shown in Listing 4 and

```python
def Verify(instance: Instance, proof: Proof, pp: PublicParams) -> bool:
    # some codes
    LEFT_r: Point = pp.cv.mul_point(proof.z, instance.pkr.W)
    RIGHT_r: Point = pp.cv.add_point(proof.Ar, pp.cv.mul_point(e, instance.Xr))
    condition_r: bool = compare_points(LEFT_r, RIGHT_r)
    # some codes
```

45
Figure 30: Modified Scenario 1 Simulation Result

```python
Validity: bool = condition_s and condition_r and condition_a and condition_sra
return Validity
```

Listing 4: Python Implementation of Verify Equal Algorithm, Extracted

4.3.4 Scenario 2 - Different transaction amounts in In-store Payment

<table>
<thead>
<tr>
<th>Scenario Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious Player</td>
</tr>
<tr>
<td>Payment Context</td>
</tr>
<tr>
<td>Condition Violated</td>
</tr>
</tbody>
</table>

Table 9: Information of Scenario 2

Procedure.

1. **Receiver** (Merchant) gets **Sender** (Consumer) pk.

2. **Receiver** requests a confidential transaction by sending a transaction amount cipher-text $C_r$ that encrypts the transaction amount $v$ under the sender public key pks and a memo to Cloud.
3. Cloud forwards \((pkr, C, memo)\) to Sender.

4. Sender decrypts \(C\) and creates a malicious confidential transaction by simulating CreatCTx maliciously (fake \(C_s\)).

5. Cloud forwards \(pks\) and \(C_r\) to Receiver as a transaction request.

6. Receiver decrypts \(C_r\) by its secret key to obtain a plaintext value, and checks if it is the transaction amount agreed upon. Then, it tells Cloud whether it accepts the transaction request. In case it does not, the phase is aborted.

7. Cloud checks the validity of the transaction.

Implementation. Listing 5 shows the codes for the simulation.

```python
# Step 1 - Receiver (Merchant) gets Sender (Consumer) pk
# nothing to do in this simulation

# Step 2 - Receiver requests a confidential transaction by sending a
transaction amount ciphertext \(C_r\)
# that encrypts the transaction amount \(v\) under the sender
public key \(pks\) and a memo to Cloud:

### V = 10.2 # transaction amount

### r = secrets.randbelow(pp.field)

### C = ishe.Enc(pks, int(v*100), pp, r) # tx amount encrypted under
# sender pk

### memo = "Scenario 2"

# Step 3 - Cloud forwards \((pkr, C, memo)\) to Sender.
# nothing to do in this simulation

# Step 4 - Sender decrypts \(C\) and creates a malicious confidential
transaction by simulating CreatCTx.

### v = ishe.Dec(sks, C, cipher_dict, pp) / 100 # /100 for precision

### vee = 0.2 # fake tx amount to be encrypted under sender pk

### r = secrets.randbelow(pp.field)

### Cs = ishe.Enc(pks, int(vee*100), pp, r) # fake amount / sender pk

### Cr = ishe.Enc(pkr, int(v*100), pp, r) # tx amount / receiver pk

### Ca = ishe.Enc(pka, int(v*100), pp, r) # tx amount / supervisor pk

# generate NIZK proofs for the three conditions

### proof = nizk.Prove(pks, pkr, pka, Cs, Cr, Ca, r, v, pp)

# sign the pre-ctx
```
message = str(sender_wallet.state)+str(pks) + \
    str(pkr) + str(pka)+str(Cs)+str(Cr)+str(Ca)+str(proof)
sigma = ishe.Sign(sender_wallet.sk, secure_hash(message), pp)

ctx = CTx(sender_wallet.pk, pkr, Cs, Cr, Ca, 
        sender_wallet.state, memo, proof, sigma)

# Step 5 - Cloud forwards pks and Cr to Receiver as a transaction request.
# nothing to do in this simulation

# Step 6 - Receiver decrypts Cr by its secret key to obtain a plaintext value vv, 
# and checks if it is the transaction amount agreed upon. Then it tells Cloud whether it accepts the transaction request. In case it does not, the phase is aborted.
vv = ishe.Dec(skr, Cr, cipher_dict, pp) / 100  # /100 for precision
print("\nReceiver confirmed?", vv == v, "\n")

# Step 7 - Cloud checks the validity of the transaction.
validity: bool = VerifyCTx(ctx, pp)
if validity:
    print("\nThe transaction is valid.\n")
else:
    print("\nThe transaction is invalid.\n")
print()

Listing 5: Python Implementation of Scenario 1, Modified

**Result and Discussion.** Figure 31 shows the result of this simulation. Although in this scenario, Malicious Sender chooses to forge $C_s$ instead of $C_r$ so that Merchant would not reject the transaction request, the transaction is still invalid due to different transaction amounts encrypted. Evidently, NIZK Proof checks equality as explained above.
Figure 31: Scenario 2 Simulation Result
4.3.5 Scenario 3 - Negative transaction amounts in P2P Transfer

<table>
<thead>
<tr>
<th>Scenario Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious Player</td>
</tr>
<tr>
<td>Payment Context</td>
</tr>
<tr>
<td>Condition Violated</td>
</tr>
</tbody>
</table>

Table 10: Information of Scenario 3

Procedure.

1. **Sender** gets the receiver public key (a) by scanning a QR code, (b) by redirecting from an online store, (c) from the contact book, or (d) etc.

2. **Sender** creates a confidential transaction with a negative transaction amount.

3. **Cloud** checks the validity of the transaction.

Implementation. Listing 6 shows the codes for the simulation.

```python
# Step 1 - Sender gets the receiver public key.
# nothing to do in this simulation

# Step 2 - Sender initiates a confidential transaction.
v = -10.2  # negative transaction amount
memo = "Scenario 3"

cxt: CTx = CreateCTx(sender_wallet, pkr, v, memo, pp)

# Step 3 - Cloud checks the validity of the transaction.
validity: bool = VerifyCTx(cxt, pp)

if validity:
    print("\nThe transaction is valid."")
else:
    print("\nThe transaction is invalid."")

print()
```

Listing 6: Python Implementation of Scenario 3

Result and Discussion. Figure 32 shows the result of this simulation. The transaction is invalid due to negative transaction amount encrypted. This is because `VerifyCTx` invokes NIZK range proof with range \( \{ v : 0 \leq v \leq \text{MAX}_\text{BAL} \} \) to check if the transaction amount is non-negative.
4.3.6 Scenario 4 - Insufficient Sender Balance in P2P Transfer

<table>
<thead>
<tr>
<th>Scenario Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious Player</td>
</tr>
<tr>
<td>Payment Context</td>
</tr>
<tr>
<td>Condition Violated</td>
</tr>
<tr>
<td>Sender</td>
</tr>
<tr>
<td>P2P Transfer</td>
</tr>
<tr>
<td>Condition 3</td>
</tr>
</tbody>
</table>

Table 11: Information of Scenario 4

Procedure.

1. **Sender** gets the receiver public key (a) by scanning a QR code, (b) by redirecting from an online store, (c) from the contact book, or (d) etc.

2. **Sender** creates a confidential transaction with a transaction amount larger than its wallet balance.

3. **Cloud** checks the validity of the transaction.

Implementation. Listing 7 shows the codes for the simulation.

```python
# Step 1 - Sender gets the receiver public key.
# nothing to do in this simulation
```

Listing 7:
# Step 2 - Sender initiates a confidential transaction.

\[ v = 50.1 \quad \# \text{transaction amount} > \text{sender wallet balance} \]

\[ \text{memo} = "\text{Scenario 4}" \]

cxt: CTx = CreateCTx(sender_wallet, pkr, v, memo, pp)

# Step 3 - Cloud checks the validity of the transaction.

validty: bool = VerifyCTx(cxt, pp)

if validty:
    print("\nThe transaction is valid."")
else:
    print("\nThe transaction is invalid."")
print()
return

Listing 7: Scenario 4

**Result and Discussion.** Figure 33 shows the result of this simulation. The transaction is invalid due to insufficient sender wallet balance. Recall as set in Section 4.3.2, the original sender wallet balance is 50. Hence, a confidential transaction with transaction amount \( 50.1 > 50 \) cannot pass the validity check by the correctness of zero-knowledge range proof (set is \( \{ v : 0 \leq v \leq 50 \} \) in this case).

![Scenario 4 Simulation Result](image-url)
4.4 Efficiency and Cost

The efficiency of the protocol follows from the efficiency of the ISHE and NIZK algorithms proposed by [CMTA20]. Since the efficiency evaluation of [CMTA20]’s ADCP system was done by the authors in a different experimental setup and security parameter (they chose a 32-bit message space and a 128-bit security level), we choose not to compare its performance with our ACP system as any comparison is just a comparison between environmental setups and security parameters. The reason that the efficiency scale-up of applying a centralized architecture cannot be qualitatively analysed is because the authors did not evaluate the efficiency of the system on a blockchain. In other words, the efficiency evaluation was merely done for cryptographic operations. Hence, for the efficiency of the ACP system, please refer to Section 8.3 of [CMTA20]. A caveat is that our system setup protocol will require more time than their setup algorithm since we have an initialization for fast decryption step (see Section 3.6.4).

Having said that, we still deem our adaptation of the ADCP system to be much more efficient than the original adoption of ADCP [CMTA20]. This is because [CMTA20] deploys on the Ethereum blockchain, which currently takes 15 seconds or above per transaction. The transaction time may be slower in case of network congestion or even remain unconfirmed for days if the gas fees paid by the sender is too slow.

In fact, the original adoption of ADCP [CMTA20] is expensive in terms of gas fee. Each transaction costs 15000000 gas in total, which takes about 16500 HKD\(^6\), obviously impractical for real-life usecase. On the contrary, our adaptation uses a centralized database, which is not exposed to this problem.

\(^6\)This amount is estimated by https://www.cryptoneur.xyz/en/gas-fees-calculator
4.5 Result Discussion

The confidential transaction protocol proposed in Section 3.5 offers transaction confidentiality by design (i.e. end-to-end encryption of transaction amount), as demonstrated in Section 4.2 (the balances are encrypted so that the transaction amount cannot be revealed by adding or subtracting the original balance by the new balance) and explained in Section 4.3.

Moreover, as verified in the simulation of malicious attacks, the protocol can ensure the validity of each transaction whilst preserving confidentiality. Utilizing NIZK proof by running \texttt{VerifyCTx}, the three aforementioned conditions can be checked without knowing the actual wallet balance of Sender and the actual transaction amount. This can protect user privacy. Meanwhile, checking those three conditions can prevent malicious users from creating money for free by collusion, which is critical in a payment system.

Meanwhile, auditability is supported by global supervision as demonstrated in 4.2. Although global supervision violates privacy by design, privacy-preserving auditing for specific policies can be conducted by triggering \texttt{JustifyCTx} and \texttt{AuditCTx}. Despite the exclusion of these two algorithms in our project scope (see Section 5.2 for explanation), these two algorithms do not require the change in the \textit{ctx} structure, and hence can be included in a future work without affecting the current work. In fact, this means any NIZK proof for auditing policy needs not to be predetermined upon system setup and can be included when needed.

Furthermore, the protocol does not consist of any decentralized elements. The centralized architecture underlying the protocol may allow it to be much more scalable than privacy coins, which adopts a decentralized architecture. Compared to the original adoption of ADCP in the PGC paper, this adaptation can theoretically allow confidential transactions to be conducted at a much higher speed and at a much lower cost. Removing the decentralized elements can also allow the use of cloud database, for which the advantages are discussed previously. In addition, it provides a more feasible solution to upgrade existing e-wallet products in privacy protection, with minor changes in the current data structure, compared to the decentralised UTXO-based model.
5 Conclusion

This chapter concludes the report by reviewing the project schedule (Section 5.1), highlighting some limitations of this project and challenges for putting the project into practice (Section 5.2), as well as suggesting directions for future research.

5.1 Project Schedule

Figure 34 shows our project schedule. We have completed all three phases and a fully-functional mobile app prototype has been developed.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Month</th>
<th>Tasks</th>
<th>Deliverables</th>
</tr>
</thead>
</table>
| 1 Inception | Aug 2022 | • Brainstorm ideas for FYP and confirm the FYP topic  
• Conduct background research on e-wallets  
• Conduct market survey  
• Enhance knowledge and skills in cryptography and app development | 2 Oct 2022  
• Detailed project plan  
• Project web page |
| | Sep 2022 | • Devise main features  
• Design user interface  
• Literature on cryptographic techniques and protocols for confidential transactions  
• Research on cloud platform to use  
• Design backend database | |
| 2 Elaboration | Oct 2022 | • Implement Login, Register, Homepage, Sidebar, User Profile, Wallet Dashboard (basic)  
• Design confidential transaction protocol | 22 Jan 2023  
• Preliminary implementation  
• Detailed interim report |
| | Nov 2022 | • Implement Bottom Navigation Bar, Transaction History, Scan, Payment Code (basic)  
• Design confidential transaction protocol | |
| | Dec 2022 | • Implement Send, Receive, Top-up, Withdrawal, Bill Payment (basic)  
• Implement confidential transaction protocol | |
| | Jan 2022 | • Implement Promise (novel)  
• Fine tune the user interface  
• Test and debug the entire prototype  
• Write interim report | |
| 3 Construction | Feb 2022 | • Implement Collaborative Wallet (novel) | 18 Apr 2023  
• Finalized tested implementation  
• Final report |
| | Mar 2022 | • Implement Automatic Bill Splitting (novel)  
• Integrate cryptographic protocols into the backend | |
| | Apr 2022 | • Connect backend to the cloud server  
• Finish implementation of whole app  
• Test and debug the entire app  
• Write final report | |

Figure 34: Project Schedule
5.2 Limitations and Challenges

5.2.1 Post-Quantum Vulnerabilities

As mentioned in Section 3.2, the ACP System is instantiated with Twisted ElGamal of which security is based on the DDH assumption. However, this number-theoretic assumption can be broken by quantum computing using Shor’s algorithm [Sho99]. Hence, the assumption will no longer hold in the post-quantum era, resulting in post-quantum vulnerabilities of our protocol and hence our e-wallet.

5.2.2 Compliance with Regulatory Requirements

Regulated by the Hong Kong Legislative Council and Monetary Authority, the marketization of e-wallets is subject to increasingly stringent regulations and compliance requirements. These requirements can be broadly classified into two parts:

**Part 1: Capital Requirements.** E-wallet providers are required to ensure a minimum capital of 25 million HKD before they can enter the market. This is mandated by the Payment Systems and Stored Value Facilities Ordinance (Cap. 584) by the HKMA, which stipulates that ”a store value facility (SVF) licensee must have a minimum paid-up share capital of HK$25 million and evidence of sufficient working capital for protecting the float.” Meeting this capital requirement has proven to be a challenge for the introduction of SuperCloudPay into the market, which may limit the impact of our project.

**Part 2: Licensing Requirements.** Since the inception of the SVF licensing regime in 2016, only a total of 16 SVF licensees\textsuperscript{7} have been licensed in Hong Kong, indicating that the licensing procedures for SuperCloudPay are likely to be complex.

5.3 Future Work

5.3.1 Post-Quantum Resilience

As highlighted in Section 5.2, our current confidential transaction protocol is not-quantum safe. To achieve post-quantum resilience, quantum-resistant instantiations, such as lattice-based [Pei16] and supersingular-isogeny-based [GPST16] schemes.

5.3.2 Implementations of Collaborative Wallets

The decentralized logic of collaborative wallets are not implemented in our prototype. For implementation, a potential approach is to use threshold cryptography, in which the secret

\textsuperscript{7}SVF licensees in Hong Kong include all onboarding e-wallet and prepaid card payment services providers with value storage features.
key is shared among multiple parties and a threshold number of parties must collaborate in the decryption or signature protocol [KY02]. In our case, the wallet secret key should be shared among multiple participants of the wallet. When a wallet participant would like conduct a confidential transaction, if the number of participants who consent to this transaction meets the pre-defined threshold number, the transaction can be signed, creating a valid transaction request (in terms of valid signature).

5.3.3 Implementations of NIZK Proofs for Various Policies

As mentioned in Section 4.5, the prototype only supports the global supervision auditing function, but not the privacy-preserving auditing function. This is because further research is required for deciding what policies to include so that they useful under Hong Kong’s legal environment. Hence, a direction for future work is to conduct a legal research to gain a better understanding on the legal demand of auditing and to implement the NIZK proofs for certain policies based on the research result.

5.4 Conclusion

This project develops a cloud-based privacy-preserving e-wallet by leveraging cryptography and cloud technology. The project has completed market analysis, including a background research and a market survey on e-wallet adoption in the Greater Bay Area, the design and implementation of frontend, backend and confidential transaction protocol. The protocol is based on an ACP system and various cryptographic techniques, including ISHE and NIZK proof. Harnessing cryptography, transaction confidentiality, validity and auditability can be guaranteed simultaneously, as shown in the experiments.

At a theoretical level, SuperCloudPay may outperform traditional e-wallets in terms of privacy and security, and prevail over privacy coin transaction protocols in terms of scalability and auditability. At a practical level, SuperCloudPay can enhance the e-payment experience of consumers by improving convenience, privacy and security while preserving auditability for regulators. It has the potential to be the first privacy-preserving and the first cloud-based e-wallet in the market.

Nevertheless, the potential impact of SuperCloudPay may be limited by its post-quantum vulnerabilities and the real-world regulatory requirements. Future research directions are suggested to enhance its post-quantum resilience and functionalities.
References

[APUST+13] Anca Apostu et al. “Study on advantages and disadvantages of Cloud Computing—the advantages of Telemetry Applications in the Cloud”. In: *Recent advances in applied computer science and digital services* 2103 (2013).


Appendices

A Market Research Survey

A.1 Original Survey and Raw Data

- Google Form (English version): https://forms.gle/yinvsjr64Ey6HK2n8
- Combined raw data of survey (English Chinese version): https://docs.google.com/spreadsheets/d/1RgB_YkYmp47-WGftuuqDDfLhNrr89V6zB/edit?usp=sharing&ouid=106265686999015997461&rtpof=true&sd=true

A.2 Result Visualization

https://drive.google.com/file/d/1RP3VmBU07RFKLKvapMSsfSQx9fFZzitj/view?usp=sharing

B API Documentation

https://connecthkuhk-my.sharepoint.com/:b:/g/personal/u3568707_connect_hku-hk/EaMc4AoYeshJhC8WUr7ygpAB00GTkvX-6tpIXVnrz2OI3A?e=VUIl6N