Pisces: Private and Compliable Cryptocurrency Exchange

Ya-Nan Li
The University of Sydney
yanan.li@sydney.edu.au

Tian Qiu
The University of Sydney
tqiu4893@uni.sydney.edu.au

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

***Abstract*—Cryptocurrency exchange platforms such as Coinbase, enable users to purchase and sell cryptocurrencies conveniently just like trading stocks/commodities. However, because of the nature of blockchain, when a user withdraws coins (i.e., transfers coins to an external on-chain account), all future transactions can be learned by the platform. This is in sharp contrast to conventional stock exchange where all *external* activities of users are always hidden from the platform. Since the platform knows highly sensitive user private information such as passport number, bank information etc, linking all (on-chain) transactions raises a serious privacy concern about the potential disastrous data breach in those cryptocurrency exchange platforms.**

In this paper, we propose a cryptocurrency exchange that restores user anonymity for the first time. To our surprise, the seemingly well-studied privacy/anonymity problem has several new challenges in this setting. Since the public blockchain and internal transaction activities naturally provide many non-trivial leakages to the platform, internal privacy is not only useful in the usual sense but also becomes necessary for regaining the basic anonymity of user transactions. We also ensure that the user cannot double spend, and the user has to properly report *accumulated* profit for tax purposes, even in the private setting. We give a careful modeling and efficient construction of the system that achieves constant computation and communication overhead (with only simple cryptographic tools and rigorous security analysis); we also implement our system and evaluate its practical performance.

I. Introduction

Just like stocks and other commodities, people buy or sell cryptocurrencies on exchange platforms, mostly, on centralized platforms such as Coinbase, which are essentially marketplaces for cryptocurrencies. There, customers can pay fiat money like U.S dollars to get some coin, e.g, Bitcoin, or transfer their coin in the platform to an external account (*withdrawal*) [1]. Despite the promise of decentralized exchange, those centralized trading platforms still play a major role for usability and even regulatory reasons. For example, the annual trading volume of Binance was up to 9580 billion USD in 2021 [2], and Coinbase also had 1640 billion USD in 2021 [3].

[1]Or transfer coins into accounts in the platform from an external account (*deposit*), then sell for fiat money; and *exchange* one coin, e.g., BTC, to get some other coin, e.g., ETH. See Fig.1, and Sec.III-A for details.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23220
www.ndss-symposium.org

Like conventional stock exchanges, these exchange platforms must comply with regulations including Know Your Customer (KYC). They require businesses to verify the identity of their clients. Essentially, when a client/user registers an account at the exchange platform, he is normally required to provide a real-world identification document, such as passport, or a stamped envelope with address, for the platform to verify. Also for trading purposes, bank information is also given.

Serious privacy threats. Despite provided convenience, those centralized cryptocurrency exchange platforms cause much more serious concern about potential privacy breaches.

As we have witnessed, many data breach instances exist [4]. A more worrisome issue in the exchange setting is that exchange can be seen as a bridge between the real world and the cryptocurrency world, which amplify the impacts of potential privacy breach (in exchange, of user records including identities and accounts). Users may *deposit* coins into the platform from, or *withdraw* coins (transfer out of the platform) to, their personal accounts on a blockchain.

Since most of the blockchains are transparent (except very few chains such as Monero [32], Zcash [8]) and publicly accessible (e.g., Bitcoin, Ethereum), the platform can essentially extract all transaction history knowing the real identity of a user. In the former case, the platform immediately links the real identity to his incoming addresses, and traces back all previous transactions on-chain; while even worse in the latter case, the platform, knowing the real identity of a user, and knows exactly which account/address of the cryptocurrency the user requested to withdraw (transfer to), and all future transactions. For instance, the platform could easily deduce that a user Alice bought a Tesla car with Bitcoin, as she withdrew them from the platform and then transferred them to Tesla's Bitcoin account (which could be public knowledge).

It follows that existing centralized cryptocurrency exchange immediately "destroys" the pseudonym protection of blockchains, and the platform could obtain a large amount of information that is not supposed to be learned, e.g., the purchase/transaction histories of clients *outside of* the platform. This is even worse than conventional stock/commodity exchange where privacy may be breached within the system, but user information outside of the system is not revealed.

We would like to design a cryptocurrency exchange system that could at least restore the user's anonymity/privacy so that it is unlinkable between external transaction records and the user's real identity.

Insufficiency of external anonymity mechanisms. The first potential method is keeping the existing exchange unchanged and cutting the link between the exchange and the external blockchain by making on-chain payments/transfers (for coin deposits and withdrawals) on every blockchain anonymous, so that nobody can link the payer and the payee. Unfortunately, all those external anonymity solutions are insufficient.

First, fully anonymous on-chain payments such as Zcash only support their own native coins, while in most exchange platforms, Bitcoin, Ethereum, and many other crypto tokens are the main objects of exchange, and cannot be supported.

More exotic solutions like anonymous and private layer-2 payment solutions [26], [25], [35], [31], [24] and smart contract based private payment solutions [15], [21] also exist. One may wonder whether we can let the platform be the payer in those solutions during coin withdrawal. However, existing solutions mainly consider k-anonymity (where k is the number of active users in an epoch) against the hub in [28], [25], [35] or the leader in [31] or only outsiders in [24], *not against the payer himself*. In our case, the platform is the payer and knows exactly the payee's address during a coin withdrawal.

Recent works of [36], [25] even considered an anonymous (k-anonymity) payment hub against payers, assuming a fixed denomination. Besides that k is usually small, withdraw transactions in the exchange platform can hardly be of a fixed amount. When two users withdraw different amounts of coins, the platform can trivially tell them apart.

Unexplored anonymity within the exchange platform. The above analysis hints that relying on an external anonymity mechanism alone is insufficient, we need to further strengthen the anonymity protection *within* the platform. Anonymity issues are classical topics that have been extensively studied in different settings, including in cryptocurrencies; yet, we will demonstrate that a large body of those works are not applicable to our setting of exchange system.

First, not only anonymous payment hub solutions cannot be directly applicable, but even the techniques (e.g., viewing the exchange platform as the hub instead, while each user can be both a payer and payee) are not sufficient either for the *"internal"* anonymity. The key difference, again, lies in the functionality difference between payment hubs (and other payment-related solutions in general) and exchange platforms.

Usually, in an anonymous payment hub, payer-payee exchange some information first, and then each runs some form of (blockchain facilitated) fair exchange protocol with the hub. For anonymity, they would require a bunch of payers and payees to have some on-chain *setup* first with the hub, and k active payments, so that the link between each pair of payer-payee can be hidden among those k transactions; otherwise, each individual incoming transaction can be recognized by the hub. But in an exchange platform, there is no other entity for such *setup*, each individual request would be independent of the view of the platform: when users A and B purchase some BTCs from the exchange platform, these purchase requests can trivially be distinguished by the platform (i.e., $k = 1$).

Another issue (not covered in the payment solutions) in anonymous exchange is that every exchange transaction between a user and the platform contains two highly correlated parts: the transaction from user to platform and that from platform to user. The amounts are based on the exchange rate, e.g., A pays 1 BTC, for 15 ETHs. While in (anonymous) payment solutions, any two transactions can be completely independent, e.g., A pays 10 BTCs to B (e.g. platform here), while B pays 1 ETH to A.

There are also some works on private Decentralized EXchange (DEX for short) [14], [10], [17] where users exchange cryptocurrencies with each other. The privacy model in DEX is different from that of our centralized setting. It keeps the transaction information secret *except* for the trading parties. Again, in our setting, the platform is one of the trading parties who can learn the information of the other trivially.

Atomic swap across different ledgers supports the exchange between different cryptocurrencies. While atomic swap pays much effort into ensuring fairness, the only privacy-preserving atomic swap work [20] reduces the confidentiality and anonymity properties to the underlying blockchains. If the swap protocol involves cryptocurrencies on transparent blockchains, like Bitcoin and Ethereum, these two transactions can be linked easily via their amounts. There is only one private fiat-to-Bitcoin exchange [39]. During withdrawals, the client chooses one UTXO and mixes it among k transactions. To prevent linkability by the transaction amount, it requires each withdrawal to be fixed for 1 BTC. And if two clients choose the same BTC, only one of them would get paid.

It follows that the natural question of anonymous cryptocurrency exchange is still open.

Further challenges. Besides the issues mentioned above not covered in existing studies, the anonymous cryptocurrency exchange setting has several other features that bring about more challenges: since the exchange system is always connected with external blockchain (e.g., via the *deposit* and *withdrawal* of coins), it automatically leaks highly non-trivial information (e.g., 3 BTCs has been deposited, and 2.9 BTCs has been withdrawn/transferred out 2 minutes later) such that how to best deal with them requires care.

The right anonymity/privacy goal. From a first look, we may just handle the withdrawal operation and define a basic, direct anonymity notion, that breaks the link between the receiving account and user identity and leaves other operations unchanged for efficiency. A bit more formally, given two different users and a specified withdraw transaction, we can require that it is infeasible to distinguish which one conducts the withdrawal if *both* of them are eligible. However, if we examine the *anonymity set* of the withdrawal, it only consists of users who have enough amount of the specified coin, which could be few. For example, for some unpopular assets, maybe only a very small number of users own such kinds of coins; or one user may hold a significantly larger amount of the coin than others. When a large-volume withdrawal of such a token takes place, it is easy for the platform to identify the user.

We then turn to consider stronger anonymity. One may suggest gradually strengthening anonymity by allowing fewer unnecessary leakages (keeping some internal transaction data *private* such as amount) and leaving seemingly safe information such as coin names as now (to avoid potentially complex solutions for protecting such info). Unfortunately, many of the remaining transaction metadata, together with the inherent

leakages such as 3 BTCs withdrawn by someone to an external address, can still reduce the anonymity set. It is hard to have a reasonably stronger anonymity without full internal transaction privacy (excluding the inherent leakage during withdraw/deposit), as it is unclear what is the actual consequence of each specific leakage. For these reasons, we choose a definition that insists the system does not leak anything more than necessary to the exchange platform (essentially requiring privacy). We will explain more in Sec. IV-A.

Preserving major compliance functionalities. We also need to preserve all the critical functionalities that are currently provided by centralized exchanges, including compliance such as verifying users' identities (KYC), generating tax reports for users and checking sufficient reserve for the platform[2].

There are many types of assets/coins in an exchange system, and their prices fluctuate over time. Users gain a profit by capitalizing on the price difference between buying and selling. It is often mandatory for users to pay taxes on their *accumulated* profits over time. At each year's end, users obtain a tax report from the platform so that they can report their annual profit, e.g., to the Internal Revenue Service for tax filing. For example, based on the suggested tax policy of Coinbase [7], transactions that result in a tax are called taxable events. Taxable events as capital gains include selling cryptocurrency for cash, converting one cryptocurrency to another, and spending cryptocurrency on goods and services (e.g., withdrawing cryptocurrency).

In the current transparent exchange system, the platform records the whole transaction history for each account and can easily extract their taxable events. The platform can also check the reserve easily as it knows the asset details of each account. This ensures that the platform possesses sufficient assets to meet the withdrawal requests of users.

However, in the anonymous setting (which now also requires privacy), the platform has no idea about the asset details of each account. It cannot prove solvency in the same way as before. Furthermore, the platform knows neither the actual profit nor the relationship between these transactions with any user. Without careful designs to calculate accumulated profit (without violating privacy/anonymity), some users could always claim they made no profit.

Striving for practical performance. Privacy-preserving constructions normally use zero-knowledge proofs. Although the deposit and withdrawal assets are public, the exchange details (e.g., 1 BTC for 15 ETHs) should be hidden and proven in zero knowledge that the transaction is valid and the prices are recorded correctly. In theory, zkSNARK [12] may enable succinct proof size and verification time. But the proof generation incurs heavy computation for users. Σ-protocols may also be useful, but hiding the exchanged asset types in all n types of assets usually requires communication/computation cost growing at least *linear* in n. For a practical design, we need to reduce the communication and computation overhead to be as small as possible (e.g., ideally *constant* cost).

[2]There are some related works in accountable privacy (e.g., PGC [16], UTT [37], Platypus [38], [9], etc), but they only focus on the payment with a single type of asset and enforce limits on one transaction amount or account balance or sum of all sent or received values. Note that their compliance requirements cannot cover reporting tax that needs profit computation using the specific buying price without linking to that transaction.

A. Our contributions

Modeling. We for the first time formally define the private and compliable cryptocurrency exchange. We give a basic version of anonymity first as a warm-up, which only cares about the withdrawal operation. As we briefly discussed above, hiding only part of transaction data may not give a reasonably strong anonymity. In the end, we define the security model insisting that the exchange leaks essentially no information to the platform. In this way, we obtain the best possible anonymity (given that public withdrawal is always there). We also carefully define *soundness* properties such as *overdraft prevention*, and *compliance*. For details, we refer to Sec. IV.

Constructions. We first give a very simple construction satisfying the basic withdraw anonymity and showcase its limitations. We then design the first private and compliable exchange system which is provably secure in the full private model. Users are hidden in a large anonymity set, and they cannot withdraw more assets than they own, or report false compliance information. To obtain full anonymity, the user's information is concealed as much as possible in each transaction, including user identities and the exchanged assets details. Soundness properties are ensured via efficient NIZK proofs specially designed for our purposes. Note that proving the correctness of an exchange request usually leads to a proof whose size is linear to the total number of asset types; instead, we propose an efficient construction with *constant* cost in both communication and computation which is independent with the number of asset types and users in the system.

Performance evaluations. We implement and evaluate our Pisces system to test the cost breakdown in each operation and compare it with those in plain exchange (without anonymity). Considering the presence of TLS communication, our overhead is minimal. We also compare with other relevant systems[3] for further evidence. See Sec. VI for details.

II. Technical overview

We first provide a high-level overview of the technique. Typically, there are two main parties involved: the platform and the user. However, in certain cases such as tax filing, there may also be an external authority involved.

Workflow of exchange system. First, the user provides the real identity to the platform during registration. Then the user interacts with the platform to deposit, exchange (e.g., 1 BTC for 15 ETHs), and withdraw assets. For compliance, the user generates his compliance report, gets it certified by the platform, and reports to an authority. The platform also generates information to check its own solvency.

We use Fig. 1 to visualize these (simplified) procedures. Each interactive protocol can be expressed by ① ② ③ steps. The user sends a compliance report to the authority who verifies it in step ④. The platform checks in step ⑤ whether its internal state satisfies the platform compliance rule.

[3]There is a concept of updatable anonymous credential [13], that share similar theoretical structure of proving properties of attributes in anonymous credential; however, their main application to incentive systems supports only limited functionalities and the achieved anonymity is weak. We have to design more complicated compliance functions.

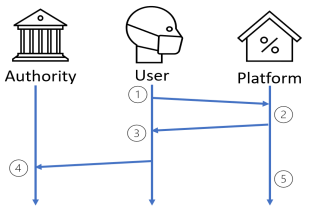

Fig. 1: Overview of exchange system: ① *Transaction request*; ② *Transaction processing*: platform verifies the transaction and process it; ③ *Transaction completion*: user completes the transaction; ④ *Compliance verification*: authority verifies the compliance report of the user; ⑤ *Compliance check*: platform checks internal state with the platform compliance rule

Constructions. Based on the workflow above, we illustrate the design idea of our efficient system Pisces step by step.

Basic anonymity. As a warm-up, to just break the link between an outgoing transaction (withdraw) and the history within the platform, we can introduce a preparation step. There, users ask for *one-time* anonymous credentials (as tokens) with amounts hidden to the platform (simply via a "partially" blind signature) that later can be used to do withdrawal. Users also submit a committed record containing asset details of the token for compliance purposes. Now, the user balance becomes hidden, and users should prove that the sum of all hidden amounts is valid depending on his previous balance via zero-knowledge range proof. When withdrawing, users could directly reveal such a one-time credential (a valid signature on a random identifier/nonce and transaction, etc.), which easily prevents double-spending. Since now user balance is also hidden, all future operations including exchange, withdraw preparation will involve (efficient) zero-knowledge proofs of validity.

Full anonymity. Additional protection on the exchanges and deposits is needed. Especially, the exchanges should not only be anonymous but also keep *asset type* and *amount* private. Each exchange transaction requires the value of exchange-in and exchange-out to be equal. To calculate the value, users need to show the used prices (now committed) correspond to two of the prices among all available assets. Further zero-knowledge proofs on membership and equality will be leveraged. But doing them efficiently requires care and will be explained soon in *practical considerations*.

Supporting compliance. The above full anonymity construction is oversimplified, as we have not considered compliance issues. For example, since each user can have multiple credentials, he could give one credential with, say 10 Bitcoins, as a gift to any person, who may not even registered with the platform. Then the gift receiver could use the credential to do the anonymous trading with the platform without revealing his real-world identity, which is not compliant even with the basic KYC regulation. Moreover, we would support common compliance goals without hurting anonymity/privacy. In particular, we use tax filing as an example of client-compliance.

First, all transactions from the same user should be bound together (without revealing the content) to derive accumulated profit; we thus let each user maintain one long-term *registration credential* that contains two attributes "cost" and "gain" to record the buying cost, and selling gain for exchange-out and withdraw transactions (the taxable transactions in e.g., Coinbase). Such a credential is issued when the user registers to the system and will be updated properly after each transaction. To conveniently update it, transaction metadata would also be recorded in a secure way, e.g., each coin type, buying/selling price and amount, etc. (as in current exchange platforms such as Coinbase). Each transaction corresponds to two one-time *asset credentials* w.r.t the exchanged assets which contain the corresponding trading prices and amounts.

When users request to exchange or withdraw, they need to provide proof of ownership for a valid asset credential with sufficient amounts, a valid registration credential, evidence of fair exchange, and accurate records of updated costs and gains. However, when generating the compliance report, revealing this information directly to the platform would compromise user privacy. For instance, if a user has a substantial profit, it increases the likelihood of being linked to previous large-scale withdrawals. To address it, we employ a workaround by having the platform blindly sign (thus providing validity proof) to generate the report without revealing sensitive information.

Practical considerations. With above considerations, the users and platform have to engage in multiple non-interactive zero-knowledge proofs, some of which may be heavy if not done properly. Particularly, for each exchange transaction (e.g., user wants to buy 1 BTC using 15 ETHs), the user takes his ETH asset certificate, proves in zero-knowledge that the asset type belongs to $[n]$ via a membership proof (as asset type needs to be kept private); and proves the used prices (committed in the new asset certificates) are exactly the current prices of the exchange-in and exchange-out assets, which also need membership proofs. To facilitate such a proof, one idea is to let the user to commit to a vector $\vec{v}$ with n dimension, and prove that $\vec{v}$ is a vector of bits and contains only one entry (corresponding to his asset certificate) as 1. Then homomorphically evaluating the linear combinations may get commitments of price$\times$amount of BTC, and ETH respectively, the user can further prove the resulting committed values are equal. The proof size is already at least $O(n)$, and computation cost even more. Recent work of one-out-of-many proof or many-of-many proof may reduce the proof size to logarithmic in n but the computation cost of proof generation and verification is still (super)linear in n [27], [21].

Instead, we let the platform generate signatures on each asset name and the price and make them public, called *price credential*. To capture the price fluctuation and avoid users using out-of-date price credentials, each price credential contains the timestamp of the latest price update. In the exchange transaction, the user proves that the new exchanged asset record contains the name and price and she knows the valid signature on them and the latest timestamp. It can be verified by the platform's *single* public key, and we can bring down the communication and computation cost to be constant. For details, please refer to Sec. V.

Extensions and open questions. For client compliance, there are many other regulation rules such as limiting the transaction frequency, transaction amounts, all sending or receiving amounts, and scrutinizing the receiver addresses in case of financing terrorism. Our techniques can be extended very easily to also support those. See Sec. V-A for more discussions.

Solvency issues. There are also many platform-compliance requirements. One notable one is the solvency problem that the platform should be able to check it has sufficient reserve. For example, Provisions [18] presents a privacy-centric approach to validating solvency within a financial exchange without the need to disclose sensitive information such as its Bitcoin addresses, total holdings, etc. But in our setting, transaction details are hidden in privacy-preserving exchanges, which may increase the risk of solvency issues, and users may exchange/withdraw a large amount of certain cryptocurrency privately that exceeds the platform's reserve, thus causing a potential "bank run". According to the Basel Accords [1] for the banking industry (we also use it as the platform-compliance rule in Sec. V), it usually requires the banks to (i) provide sufficient liquidity (e.g., keep enough asset to cover the total withdraws of last month), and (ii) keep a sufficient minimal reserve (e.g., 10% of the total assets held by all users in the platform, in the form of a major currency such as USD [5]; in our setting, Bitcoin). We show that our Pisces system with full anonymity also satisfies the first platform-compliance requirement. As the platform is still aware of the total amount of incoming/outgoing Bitcoins (and any other cryptocurrency tokens), the needed information could still be derived.

To maintain a minimal reserve, there might be some practical mitigation, e.g., actively monitoring the total withdrawal amount/pattern for each coin, limiting the exchange and withdrawal amount/frequency, etc, or involving a third-party auditor (similar to the tax authority to keep the aggregated information to manage risks). Those can be supported by extending our design. But a more rigorous solution remains open. Also, there could be even more strict and complicated rules that may require the platform to keep sufficient reserve and liquidity for every single type of coins [11]; or require the platform to generate publicly verifiable proofs of solvency.

We remark that with our basic anonymity requirement, the platform knows all the holdings of each account (except the link between the inside and external on-chain accounts), and thus can still derive all needed information for both requirements and the stricter rules.

However, a more systematic investigation of solvency issues in a fully anonymous setting (e.g., allowing the platform to gain extra side information for solvency purposes) may again have a further impact on the anonymity.

As a first step in studying privacy in exchange systems with efficient compliance support, there are many interesting questions and challenges to explore (e.g., supporting broader compliance rules). For a more systematic investigation, we leave them as interesting open problems.

III. Preliminary

Notations. Throughout this paper, we denote with $\lambda \in \mathbb{N}$ the security parameter, and by poly(λ) any function which bounded by a polynomial in λ. An algorithm $\mathcal{A}$ is said to be PPT if it is modeled as a probabilistic Turing machine that runs in time polynomial in λ. Informally, we say that a function is negligible if it vanishes faster than the inverse of any polynomial. A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive integer c, there exists an integer x_0 such that $|f(x)| < 1/x^c$ for all $x > x_0$. It is denoted by *negl*. For a finite set S, $x \leftarrow\!\!\$\ S$ means that x is chosen uniformly from S. If n is an integer, $[n]$ denotes the set of positive integers $1, 2, \ldots, n$. We use $\vec{v}$ to denote a vector. We write $\langle \mathsf{A}, \mathsf{B} \rangle$ to denote interactive algorithms A, B engage in an interactive protocol, take their respective inputs, and share some transcripts.

We briefly introduce some cryptographic primitives here and defer details to the full version [30] due to the page limits.

Commitments. A commitment scheme allows one to commit to a chosen value secretly, with the ability to only open to the same committed value later. A commitment scheme Π_{cmt} consists of the following PPT algorithms:

$\mathsf{Setup}(1^\lambda) \to pp$: generates the public parameter *pp*.
$\mathsf{Com}(pp, m; r) \to com$: generates the commitment for the message m using the randomness *r*. For ease of notation, we omit pp in the input.

We require a commitment scheme to be *hiding* and *binding*. A commitment is additively homomorphic if it satisfies that for any messages m_1, m_2 and randomnesses r_1, r_2: $\mathsf{Com}(m_1; r_1) + \mathsf{Com}(m_2; r_2) = \mathsf{Com}(m_1 + m_2; r_1 + r_2)$.

Blind signatures. A blind signature scheme Π_{bs} for signing committed n messages has the following algorithms:

$\mathsf{KeyGen}(pp) \to (pk, sk)$: takes public parameter pp as input, outputs a key pair (pk, sk). pp, pk are implicit inputs of others.$\mathsf{Com}(\vec{m}, r) \to c$: given messages $\vec{m} \in \mathcal{M}^n$ and randomness r, computes a commitment c.
$\langle \mathsf{BlindSign}, \mathsf{BlindRcv} \rangle$: it is an interactive protocol between the signer and user, with inputs (sk, c) and $(\vec{m}, r)$ respectively. User outputs a signature σ.
$\mathsf{Vrfy}(\vec{m}, \sigma) \to b$: it checks $(\vec{m}, \sigma)$ pair and outputs 0/1.

We require a blind signature scheme to be correct and have the properties of *unforgeability* and *blindness*.

Zero-knowledge argument of knowledge (ZKAoK). The prover proves knowledge of w such that (x, w) is in some NP relation R. Here x is the statement and w is the witness. The zero-knowledge argument of knowledge [29] can be simulated perfectly and there exists an expected polynomial-time extractor $\mathcal{E}$ that, given black-box access to a successful prover, computes a witness w with probability 1. It is denoted by $\mathsf{ZKAoK}[(w); (x, w) \in R]$.

A. Syntax

An exchange system involves three entities: the platform P, the user U and an authority A. The system consists of the following PPT algorithms: Setup, PKeyGen, Verify, Check as well as interactive protocols: $\langle \mathsf{Join}, \mathsf{Issue} \rangle$, $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$, $\langle \mathsf{Exchange}, \mathsf{Update} \rangle$, $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$, $\langle \mathsf{File}, \mathsf{Sign} \rangle$. To present the syntax, we prepare some data structures.

Transaction requests reqs. For each transaction, U's input includes a transaction request to specify details. We denote it as a data structure and consider five kinds of requests as follows. To keep the syntax general and simple, we add an optional attribute aux to each request. aux could contain several sub-attributes required by the operation but not included in the listed attributes, be different for different operations, and be specified by the detail construction.

- $req_{joi} := (info, aux)$ denotes join request, where $req_{joi}.info$ is user's information for joining the system.
- $req_{dep} := (name, amt, aux)$ denotes deposit request, where $req_{dep}.name$ is asset name, $req_{dep}.amt$ is asset amount.
- $req_{exc} := (name_{in}, amt_{in}, name_{out}, amt_{out}), aux$ denotes exchange request, where $req_{exc}.name_{in}$ is exchange-in asset name, $req_{exc}.amt_{in}$ is exchange-in asset amount, $req_{exc}.name_{out}$ is exchange-out asset name, $req_{exc}.amt_{out}$ is exchange-out asset amount.
- $req_{wit} := (name, amt, aux)$ is withdraw request, where $req_{wit}.name$ is asset name, $req_{wit}.amt$ is asset amount.
- $req_{fil} := (uid, cp, aux)$ denotes file request, where $req_{fil}.uid$ is user identifier, $req_{fil}.cp$ is compliance information.

Transaction records $\mathbf{Rd_{reg}}$ and $\mathbf{Rd_{ast}}$. Each record is an information credential pair, where the information contains several attributes. It is generated or updated during transactions and kept privately by users. We denote record Rd as a data structure and consider two kinds of records: the registration record Rd_{reg} and the asset record Rd_{ast}.

- $Rd_{reg} := (non, uid, cp, cred)$ denotes a registration record, including three attributes and the credential $Rd_{reg}.cred$ on the three attributes. $Rd_{reg}.non$ is the random nonce to uniquely identify it. $Rd_{reg}.uid$ is the owner's unique identifier. $Rd_{reg}.cp$ is the compliance information. Each user holds only one valid Rd_{reg}, which is initialized in join transaction, updated in exchange and withdraw transaction via revoking the old one and generating a new one.
- $Rd_{ast} := (non, uid, name, amt, acp, cred)$ denotes an asset record, including five attributes and a credential $Rd_{ast}.cred$ on the five attributes. Similar to Rd_{reg}, $Rd_{ast}.non$ is the random nonce and $Rd_{ast}.uid$ is the owner's identifier. $Rd_{ast}.name$ is the asset name, $Rd_{ast}.amt$ is the amount of asset, and $Rd_{ast}.acp$ is asset-related compliance information. Notably, in a private yet compliable setting, assets cannot be accumulated trivially in terms of quantity since they are tied to different asset kinds and compliance-related information such as selling prices. Thus each user could hold multiple asset records.

Concrete algorithms.

- Setup: The public parameters *epp* for the exchange system is set. *epp* includes the public parameters for cryptographic primitives. For simplicity of syntax, we let epp also include some publicly available information, such as the external blockchain, all coin prices in the exchange system, some metadata like the time, etc.
- PKeyGen: P runs the key generation algorithm to generate a key pair (pk, sk) and makes pk public to users. It initializes its internal state st as $\emptyset$.
- $\langle$Join, Issue$\rangle$: It is a register protocol. U runs the interactive algorithm Join(epp, pk, req_{joi}), and P runs the interactive algorithm Issue(epp, pk, sk, st), where st denotes the internal state of P. After the interaction, P outputs a signal bit b indicating whether the operation succeeds or not, and updates its internal state to st'. If $b = 1$, the user outputs the unique user identifier uid and the registration record Rd_{reg}.
- $\langle$Deposit, Credit$\rangle$: It is a deposit transaction for users to deposit assets to P. P runs Credit(epp, pk, sk, st) and U runs Deposit$(epp, pk, uid, Rd_{reg}, req_{dep})$. The asset name and amount are specified in req_{dep}. After the interaction, P outputs a signal bit b indicating whether the operation succeeds or not, and updates its internal state to st'. If $b = 1$, U gets a new asset record Rd_{ast}^{out} for the deposited asset.
- $\langle$Exchange, Update$\rangle$: It is an exchange transaction for users to exchange assets with P. The names and amounts of exchange-in and exchange-out assets are specified by req_{exc}. U runs Exchange$(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{exc})$, and P runs Update$(epp, pk, sk, st)$. After the interaction, P outputs a signal bit b, indicating whether the operation succeeds or not, and updates its internal state to st'. If $b = 1$, U outputs three records: the updated ones Rd'_{reg} and Rd'_{ast}, and a newly generated asset record Rd_{ast}^{out} for exchange-out asset with name $req_{exc}.name_{out}$.
- $\langle$Withdraw, Deduct$\rangle$: It is a withdraw transaction for users to withdraw a kind of asset from P to the blockchain. The name and amount of withdrawn asset are specified in req_{wit}. U runs Withdraw$(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{wit})$, and P runs Deduct$(epp, pk, sk, st)$. After the interaction, P outputs a signal bit b, indicating whether the operation succeeds or not, and updates its internal state to st'. If $b = 1$, U outputs two updated records Rd'_{reg}, Rd'_{ast}.
- $\langle$File, Sign$\rangle$: It is a two-party protocol in which U files compliance information periodically and requests P to sign it. U runs File$(epp, pk,\ uid, Rd_{reg}, req_{fil})$ and P runs Sign(epp, pk, sk, st). After the interaction, P outputs a single bit b, indicating whether the operation succeeds or not, and updates its internal state to st'. If $b = 1$, U outputs an updated record Rd'_{reg} and a compliance document *doc* certified by P. Similar to transaction record Rd, $doc := (uid, cp, mt, sig)$ is also a data structure including three attributes and a signature *doc.sig* on them, where *doc.uid* is the user identifier, *doc.cp* is the reported compliance information, and *doc.mt* is the metadata such as time.
- Verify: The authority runs Verify(epp, pk, doc) to check the validity of the submitted compliance document, including the consistency of metadata in *epp* and *doc.mt*, and the valid signature. It outputs a bit b with $b = 1$ indicating a passing check, and vice versa.
- Check: P runs Check(epp, st) for self-checking the internal state's compliance with platform rules specified in epp. The output is a single bit b, with $b = 1$ indicating a passing check and vice versa.

IV. Security models

In this section, we formally define security models to capture the desired security properties of a private yet compliable exchange system. Along the way, we show the motivation, importance, and ideas of defining such properties.

To the best of our knowledge, this is the first security modeling of the centralized exchange system. Security modeling of this work is involved in four aspects. (1) we put less trust in the platform than in existing plain exchange systems where the platform is always assumed to be honest. While our model gives the platform more power in some security definitions, especially for anonymity, the platform can be completely malicious; (2) When modeling privacy/anonymity, naive attempts for a "direct" anonymity (without privacy on other parts of transactions, or trying to strive a best balance between efficiency and hiding only part of the transactions) may not work well because of potential consequences of each seemingly benign leakage (within the exchange system).

We will elaborate on it in Sec. IV-A; (3) Besides desired anonymity, we also define *soundness* properties of overdraft prevention and client compliance security that require care too; (4) For platform compliance, we require that the honest platform can always self-check whether its internal state satisfies the platform compliance rule.

We first give a high-level description of the security requirements of the system.

Correctness. The honest user gets the correct balance amount in his account from deposit, exchange, and withdrawal, also gets the correct number of real assets from withdrawal, and gets a valid signature on his compliance information that can be verified by the authority.

Anonymity. Given a withdraw/deposit transaction, the malicious platform should not link it to any specific user, except that the user has to expose the identity, such as depositing /withdrawing fiat money from/to the bank. We start discussing it from the basic anonymity that only focuses on the withdraw or deposit transactions. Although the basic anonymity scheme could be simple and does not bring extra challenges to compliance (especially platform compliance), we show that the basic anonymity, especially for withdraw, may not be sufficient, since the platform could narrow down the anonymity set based on other transactions. Thus we further explore the best possible (full) anonymity and model it.

Overdraft prevention. It ensures users cannot possess or spend more assets than they actually own in the system. It prevents malicious users from conducting fraudulent deposits, exchanges, or withdrawals.

Compliance. It requires that both users and the platform to comply with the regulations expressed as functions, and we call the corresponding compliance F-client-compliance and G-platform-compliance. All entities are required to provide compliance information according to respective compliance rules, and none of them can deceive the authority with incorrect information as long as the user does not collude with the platform. For example, F could be a tax report function on accumulated profit, and G could be a solvency-related function on the coin reserve and liquidity. Tax-report-client-compliance ensures that the user cannot cheat with a value less than his latest accumulated profit this year. Solvency-platform-compliance ensures that the platform maintains appropriate liquidity based on the monthly assets inflow and outflow.

A. Preparations for the models

Note that the bank accounts leak the user's identity when depositing or withdrawing fiat money which is unavoidable. So we consider privacy only during cryptocurrency trading. Besides, the deanonymization attack in the network layer is out of the scope of our work. The attacker links multiple transactions by IP address, but users can protect themselves using an anonymous network like Tor [22], [6].

We provide oracles to capture the adversary's capability. To model the capabilities of the malicious platform, we provide oracles: $\mathcal{O}^1_{\mathsf{Join}}$, $\mathcal{O}^1_{\mathsf{Deposit}}$, $\mathcal{O}^1_{\mathsf{Exchange}}$, $\mathcal{O}^1_{\mathsf{Withdraw}}$, $\mathcal{O}^1_{\mathsf{File}}$. To model the capabilities of malicious users, we define the oracles: $\mathcal{O}^2_{\mathsf{PKeyGen}}$, $\mathcal{O}^2_{\mathsf{Issue}}$, $\mathcal{O}^2_{\mathsf{Credit}}$, $\mathcal{O}^2_{\mathsf{Update}}$, $\mathcal{O}^2_{\mathsf{Deduct}}$, $\mathcal{O}^2_{\mathsf{Sign}}$. We also provide $\mathcal{O}_{\mathsf{Public}}$ for every party to model access to some public ongoing information, such as a secure blockchain system, the prices of all assets, currencies, stocks, cryptocurrencies, and a global clock, etc.

Reference-record map $\textsc{Map} : (uid, ref) \rightarrow Rd$*:* When $\mathcal{A}$ acts as a malicious platform, it is allowed to induce honest users to conduct transactions by querying oracles. However, some oracles require specifying records as input, which are private to honest users and unavailable to $\mathcal{A}$. To enable $\mathcal{A}$ to identify different records without knowing what they are, we let $\mathcal{A}$ specify the reference string ref[4] for each record and Oracles keep the map $\textsc{Map}$ from key tuple (uid, ref) to value Rd for $\mathcal{A}$'s later queries. For notational convenience, we let $\textsc{Map}(uid, ref)$ denote the record Rd. In the queries, ref_{reg} is the reference for the registration record, $ref^{\mathsf{in}}_{\mathsf{ast}}$ is the reference for the spending asset record, and $ref^{\mathsf{out}}_{\mathsf{ast}}$ is the reference for the buying asset record.

- $\mathcal{O}_{\mathsf{Public}}$: when queried, it returns the public information *pub*, such as the registration information, bank account, asset prices and related wallet addresses, etc. For all queries to other oracles, they inherently invoke $\mathcal{O}_{\mathsf{Public}}$ at first. We do not repeat these moves in the oracle descriptions.
- $\mathcal{O}^1_{\mathsf{Join}}(req_{joi}, ref_{\mathsf{reg}})$: it interacts with $\mathcal{A}$ by running the protocol $\langle \mathsf{Join}, \mathsf{Issue} \rangle$, where oracle runs $\mathsf{Join}(epp, pk, req_{joi}) \rightarrow (uid, Rd_{reg})$. If Join algorithm outputs $\perp$, then oracle outputs $\perp$. Otherwise, oracle adds $(uid, ref_{\mathsf{reg}}, Rd_{reg})$ to $\textsc{Map}$ and outputs uid to $\mathcal{A}$.
- $\mathcal{O}^1_{\mathsf{Deposit}}(uid, req_{dep}, ref_{\mathsf{reg}}, ref^{\mathsf{out}}_{\mathsf{ast}})$: oracle first gets record $Rd_{reg} = \textsc{Map}(uid, ref_{\mathsf{reg}})$ from $\textsc{Map}$ per references. Then it interacts with $\mathcal{A}$ by running $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$ protocol, where oracle runs $\mathsf{Deposit}(epp, pk, uid, Rd_{reg}, req_{dep}) \rightarrow (Rd'_{reg}, Rd^{out}_{ast})$. If $\mathsf{Deposit}$ algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\textsc{Map}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$ and adds a new tuple $(uid, ref^{\mathsf{out}}_{\mathsf{ast}}, Rd^{out}_{ast})$ to $\textsc{Map}$. $\mathcal{A}$ gets interaction transcripts but no more output from oracle.
- $\mathcal{O}^1_{\mathsf{Exchange}}(uid, req_{exc}, ref_{\mathsf{reg}}, ref^{\mathsf{in}}_{\mathsf{ast}}, ref^{\mathsf{out}}_{\mathsf{ast}})$: oracle first gets records $Rd_{reg} = \textsc{Map}(uid, ref_{\mathsf{reg}})$, $Rd_{ast} = \textsc{Map}(uid, ref^{\mathsf{in}}_{\mathsf{ast}})$ from $\textsc{Map}$ per references. Then it interacts with $\mathcal{A}$ by running $\langle \mathsf{Exchange}, \mathsf{Update} \rangle$ protocol, where oracle runs $\mathsf{Exchange}(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{exc}) \rightarrow (Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast})$. If $\mathsf{Exchange}$ algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\textsc{Map}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$ and $\textsc{Map}(uid, ref^{\mathsf{in}}_{\mathsf{ast}}) \leftarrow Rd'_{ast}$, and adds a new tuple $(uid, ref^{\mathsf{out}}_{\mathsf{ast}}, Rd^{out}_{ast})$ to $\textsc{Map}$. $\mathcal{A}$ gets interaction transcripts but no more output from oracle.
- $\mathcal{O}^1_{\mathsf{Withdraw}}(uid, req_{wit}, ref_{\mathsf{reg}}, ref^{\mathsf{in}}_{\mathsf{ast}})$: oracle first gets records $Rd_{reg} = \textsc{Map}(uid, ref_{\mathsf{reg}})$, $Rd_{ast} = \textsc{Map}(uid, ref^{\mathsf{in}}_{\mathsf{ast}})$ from $\textsc{Map}$ per references. Then it interacts with $\mathcal{A}$ by running $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$ protocol, where oracle runs $\mathsf{Withdraw}(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{wit}) \rightarrow (Rd'_{reg}, Rd'_{ast})$. If $\mathsf{Exchange}$ algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\textsc{Map}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$, $\textsc{Map}(uid, ref^{\mathsf{in}}_{\mathsf{ast}}) \leftarrow Rd'_{ast}$. $\mathcal{A}$ gets interaction transcripts but no more output from oracle.
- $\mathcal{O}^1_{\mathsf{File}}(uid, req_{fil}, ref_{\mathsf{reg}})$: oracle first get records $Rd_{reg} = \textsc{Map}(uid, ref_{\mathsf{reg}})$ from $\textsc{Map}$ per references. Then

[4]Note that the reference string *ref* used by $\mathcal{A}$ is different from the identifier(nonce) of the record which is privately chosen by the honest user or oracle randomly.

it interacts with $\mathcal{A}$ by running $\langle \mathsf{File}, \mathsf{Sign} \rangle$ protocol, where oracle runs $\mathsf{File}(epp, pk, uid, Rd_{reg}, req_{fil}) \rightarrow (Rd'_{reg}, doc)$. If File algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\textsc{Map}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$. $\mathcal{A}$ gets interaction transcripts but no more outputs from oracle.

- $\mathcal{O}^2_{\mathsf{PKeyGen}}$: It can only be invoked once. When triggered, run $(pk, sk) \leftarrow \mathsf{PKeyGen}(epp)$. It initializes the internal state as $st \leftarrow \emptyset$. It outputs pk.
- $\mathcal{O}^2_{\mathsf{Issue}}$: $\mathcal{A}$ runs Join algorithm and interacts with the $\mathcal{O}^2_{\mathsf{Issue}}$ oracle. $\mathcal{O}^2_{\mathsf{Issue}}$ runs Issue algorithm, takes (epp, pk, sk) as input, and receives user's transcript ts as external input. It outputs a signal bit b indicating whether the operation succeeds or not. If $b = 0$, it outputs $\perp$.
- $\mathcal{O}^2_{\mathsf{Update}}$: $\mathcal{A}$ runs Exchange algorithm and interacts with the $\mathcal{O}^2_{\mathsf{Update}}$ oracle. $\mathcal{O}^2_{\mathsf{Update}}$ runs Update algorithm, takes (epp, pk, sk) as input, and receives user's transcript ts as external input. It outputs a signal bit b indicating whether the operation succeeds or not. If $b = 0$, it outputs $\perp$.
- $\mathcal{O}^2_{\mathsf{Credit}}$: it is similar to $\mathcal{O}^2_{\mathsf{Update}}$ except that here they run the $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$ protocol and $\mathcal{A}$ gets $\{Rd_{ast_i}\}$.
- $\mathcal{O}^2_{\mathsf{Deduct}}$: it is similar to $\mathcal{O}^2_{\mathsf{Update}}$ except that here they run $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$ and $\mathcal{A}$ gets $\{Rd'_{reg}, Rd'_{ast_i}\}$.
- $\mathcal{O}^2_{\mathsf{Sign}}$: it is similar to $\mathcal{O}^2_{\mathsf{Update}}$ except that here they run the $\langle \mathsf{File}, \mathsf{Sign} \rangle$ protocol and $\mathcal{A}$ gets doc.

B. Basic anonymity

Basic anonymity guarantees that even a malicious platform cannot link the wallet address with any honest user. It consists of basic withdraw anonymity and deposit anonymity.

Basic withdraw anonymity. We define the model in Fig. 2, the adversary $\mathcal{A}$ interacts with any honest user by querying the anonymity oracle set: $\mathcal{O}_{\mathsf{anony}} = \{\mathcal{O}^1_{\mathsf{Join}}, \mathcal{O}^1_{\mathsf{Deposit}}, \mathcal{O}^1_{\mathsf{Exchange}}, \mathcal{O}^1_{\mathsf{Withdraw}}, \mathcal{O}^1_{\mathsf{File}}, \mathcal{O}_{\mathsf{Public}}\}$ oracles. The adversary submits $(uid_0, uid_1, ref^0_{\mathsf{ast}}, ref^1_{\mathsf{ast}}, req_{wit})$ as the challenge. It also outputs some internal state information st.

$\mathrm{Exp}^{\mathrm{ano-wit}}(\mathcal{A}, \lambda)$

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$

$(pk, st) \leftarrow \mathcal{A}(epp)$

$(uid_0, uid_1, ref^0_{\mathsf{ast}}, ref^1_{\mathsf{ast}}, req_{wit}, st) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{anony}}}(st)$

if $\textsc{Map}(uid_0, ref^0_{\mathsf{ast}}) = \perp$ *or* $\textsc{Map}(uid_1, ref^1_{\mathsf{ast}}) = \perp$

  return 0 //no record mapped by references $ref^0_{\mathsf{ast}}, ref^1_{\mathsf{ast}}$

if $req_{wit}.amt \neq \textsc{Map}(uid_0, ref^0_{\mathsf{ast}}).amt$ or

  $req_{wit}.amt \neq \textsc{Map}(uid_1, ref^1_{\mathsf{ast}}).amt$ or

  $req_{wit}.name \neq \textsc{Map}(uid_0, ref^0_{\mathsf{ast}}).name$ or

  $req_{wit}.name \neq \textsc{Map}(uid_1, ref^1_{\mathsf{ast}}).name$ **return** 0

else $b \leftarrow\$ \{0, 1\}$

Interacts with $\mathcal{A}$ by running

  $\mathsf{Withdraw}(epp, pk, uid_b, \textsc{Map}(uid_b, ref^b_{\mathsf{ast}}), req_{wit})$

$\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}^*_{\mathsf{anony}}}(st)$

  // * requires no query with references $ref^0_{\mathsf{ast}}, ref^1_{\mathsf{ast}}$

return $(\hat{b} == b)$

Fig. 2: Basic withdraw anonymity experiment

Definition 1 (Basic withdraw anonymity)**.** *We say that an exchange system provides basic withdraw anonymity if for all PPT $\mathcal{A}$ and λ, in the experiment shown in Fig. 2, it holds that*

$$|\Pr[\mathrm{Exp}^{\mathrm{ano-wit}}(\mathcal{A}, \lambda) = 1] - 1/2| \leq negl(\lambda)$$

Warm-up construction. To achieve the basic withdraw anonymity, an intuitive idea is cutting the link to the user's real identity within the withdraw operation, and leaving all other operations plain. Our warm-up construction follows this simple idea by partitioning the withdraw operation into two separate steps: first, users log in their plain account and request for a one-time anonymous credential (just use blind signatures) on the coin they plan to withdraw; second, they could show an anonymous credential without login to get the asset withdrawn on-chain. If the withdrawn amount is arbitrary and different users withdraw different amounts of assets, the special amount helps the platform identify a specific withdrawal. To handle this problem, our method is hiding the withdrawn amount in the first step where the user request the anonymous credential with a committed amount. We introduce a brief idea here and defer the detailed description to the full version [30]

For example, Alice has 50 BTCs in her account and Bob has 100 BTCs. Alice wants to withdraw 5 BTCs and Bob wants to withdraw 2 BTCs. Firstly, they commit on these values and prove they have enough balance and request the platform to issue credentials. Once the proof gets verified, the platform signs blindly and stores these commitments in their accounts. Alice unblinds the signature and shows it to the platform later for withdrawing 5 BTCs. The platform cannot distinguish it is Alice or Bob since both of them have enough BTCs and have requested. Afterwards, if they want to withdraw or exchange, they should prove that they have enough balance after deducting all committed amounts from the plain balance.

Basic deposit anonymity. This property prevents the platform from linking the user's past on-chain transactions to his identity via deposit operations. Modeling it can be regarded as a symmetric work of basic withdraw anonymity except that all deposit requests are achievable naturally for any user.

To add deposit anonymity, one more one-time anonymous credential can be employed. Its workflow is an inverse version of anonymous withdraw. When depositing assets, the user, as an anonymous guest, initializes the process by requesting the platform to issue a one-time use anonymous credential, which contains the asset name and amount. Then he requires the platform to credit the asset balance to his account using that credential without showing the asset details.

Limitations of basic withdraw anonymity. The above constructions are efficient and satisfy the basic anonymity model, but we observe that the anonymity is limited.

It is well-known that the anonymity strength depends on the size of the anonymity set. The greater the anonymity set is, the higher the level of anonymity a user can achieve. When considering the anonymity set for a withdrawal transaction, it comprises users who can withdraw from the view of the platform. In the above scheme, anonymous credentials are requested from real-name accounts. The anonymity set consists of users who have requested credentials for the same asset and their account balance exceeds the withdrawn amount. The following example narrows down the set size to one.

Example 1: Alice deposits 50 BTCs, Bob deposits 100 XRPs and 100 BTCs, and Clare deposits 1000 BTCs. Only Alice and Bob request anonymous credentials for BTCs. Later, a withdrawal of 51 BTCs occurs, it can thus be linked to Bob as he is in possession of a sufficient amount of BTCs.

C. Full anonymity

As we mentioned above, the anonymity set of basic anonymity could be quite small. So we explore the stronger anonymity of the withdrawal. We first attempt to get perfect anonymity with all users in the anonymity set. Unfortunately, it is impossible due to some *unavoidable leakage*. We will elaborate it later. For other kinds of leakage, we check that whether it is even worth to prevent, as any protective measure comes with cost. After some attempts, it turns out that any other leakage could be used to reduce the anonymity set. We explain that with some examples. Finally, we define the full anonymity via interactive indistinguishability. It achieves best possible anonymity by constraining the information leakage to the minimum.

Stronger anonymity is needed. Basic withdraw anonymity only ensures the anonymity set includes those eligible users who own enough of the withdrawn asset and are capable to withdraw. It is acceptable for some popular assets that a lot of people own and the withdrawn amount is small such that the anonymity set is large enough. But it rules out many interesting scenarios involving rare assets owned by a few or substantial quantities held by a minority. Thus we aim to explore a stronger model that provides a larger anonymity set.

Perfect anonymity is impossible. In the ideal case, the anonymity set of each withdraw transaction consists of all registered users in the system which is called perfect anonymity. Unfortunately, this cannot be real since the platform can always exclude some users using public information. For example, given a withdrawal of 100 Bitcoins and a newly enrolled user Alice, she is in no way the user of this withdrawal if there is no such big amount deposit in the system after her registration.

Due to the special setting of exchange platform, some information is unavoidably public to the platform, which we call *unavoidable leakage*, like transaction types (deposit or exchange or withdraw), users' registration information (due to KYC requirement), deposited and withdrawn asset details (the asset name and amount, bank accounts, and wallet addresses), and even some out-of-band information like the users' behavioral preference.

To achieve the best possible anonymity, where the users can only be excluded from the anonymity set given the unavoidable leakage, it seems that only the unavoidable leakage is acceptable. But a series of natural questions are: why do we need to hide so many? Can we leak a little bit more, like the privacy (identity, coin name, and amount) of the exchange? Does it hurt the best possible anonymity?

Necessity of privacy and towards best possible anonymity. To answer the above questions, we identify the avoidable leakage information which can be concealed using some cryptographic tools. Concretely, we assort the avoidable leakage into five classes according to the transaction: the identity in depositing coins, the identity in exchange, the contents in exchange including the coin name and amount, and the identity in withdrawing coins, and the compliance information in filing operation. We hide each of them and leak the other part to test whether the anonymity set is affected.

It is easy to see the identity in withdrawal cannot be leaked. For the three leakages occurring in the deposit and exchange, we show an example of the exchange system with a series of transactions and check the anonymity set if any avoidable leakage is allowed.

Example 2: In a cryptocurrency exchange system, there are a bunch of users registered and doing transactions, then David and Ella joined. After that somebody (David) deposits 10 BTCs. Then there is an exchange transaction: somebody (Alice, a registered user) exchanges some coins (2 BTCs to 20 ETHs). Then a withdrawal happens: somebody withdraws 5 XRPs. Check its anonymity set:

1. If the identity of deposit is leaked, then the platform knows that David deposits 10 BTCs, and Ella is excluded from the anonymity set and David is included.
2. If the identity of exchange is leaked, the platform knows that David and Ella cannot withdraw 5 XRPs, then both David and Ella are excluded from the anonymity set.
3. If the content of exchange is disclosed, the platform knows it is a BTC-to-ETH exchange and excludes David and Ella.

As for the compliance information in the filing operation, someone may consider just leaking the summary of compliance information is fine. But in this case, many users might have not generated any transactions in a given tax year, which can be inferred from their zero profit. Excluding these sleeping users reduces the anonymity set. Therefore, the privacy of compliance information should also be protected. The identity of the filing operation could be leaked by the regulatory authority to the platform which is an unavoidable leakage.

In a nutshell, protecting privacy is necessary. We need to go toward the best possible anonymity.

When modeling the best possible anonymity, we want to prevent any avoidable leakage. However, since public information could have various and complicated relationships with the events in the exchange system, it is tricky to exactly quantify the potential influence, which may be leveraged by the adversary to win trivially. Instead, we define the full anonymity via interaction indistinguishability.

Interaction indistinguishability. This property requires that the interaction between the user and platform leak nothing except the public information. To include the interaction of all kinds of transactions, we design the experiment as follows. In a high level, the adversary $\mathcal{A}$ acts as the malicious platform and the experiment simulates two worlds with the same initialization. $\mathcal{A}$ can add honest users to both worlds and interact with them by submitting different query pairs. The queries are sent to the worlds via a challenger $\mathcal{C}$ who forwards query pair to two worlds depending on a random bit b. To model the unavoidable leakage, the queries should contain the same public information. After a series of interactions, $\mathcal{A}$ still cannot distinguish which world is based on which one of the query pair. It means that for any kind of transaction the interaction does not leak more than the unavoidable leakage. Otherwise, $\mathcal{A}$ can send different queries for the transaction and distinguish the two worlds successfully.

The two worlds are simulated via two sets of oracles: $\mathcal{O}^a_{\mathsf{IND}} = \{\mathcal{O}^{1,a}_{\mathsf{Join}}, \mathcal{O}^{1,a}_{\mathsf{Deposit}}, \mathcal{O}^{1,a}_{\mathsf{Exchange}}, \mathcal{O}^{1,a}_{\mathsf{Withdraw}}, \mathcal{O}^{1,a}_{\mathsf{File}}, \mathcal{O}^{a}_{\mathsf{Public}}\}$ with separated internal map $\textsc{Map}^a$ for $a \in \{0,1\}$. $\mathcal{C}$ chooses one bit b randomly at the beginning. $\mathcal{A}$ sends queries to the challenger $\mathcal{C}$ which are in pair (Q^0, Q^1) to interact with oracles. For each query pair, $\mathcal{C}$ checks that they could be different but must contain the same public information which represents the unavoidable leakage as we discussed before (see Def. 2). Then $\mathcal{C}$ forwards Q^b to the oracle in $\mathcal{O}^0_{\mathsf{IND}}$ and forwards Q^{1-b} to the oracle in $\mathcal{O}^1_{\mathsf{IND}}$.

With these queries as input, these oracles interact with $\mathcal{A}$ with different states, and we denote it in terms of $\langle \mathcal{A}(st^0), \mathcal{O}^0_{\mathsf{IND}}(Q^b)\rangle$ and $\langle \mathcal{A}(st^1), \mathcal{O}^1_{\mathsf{IND}}(Q^{1-b})\rangle$. But it cannot distinguish which are induced by which queries. Therefore, the interaction leaks nothing but public information. We formally define the interactive indistinguishability in Fig. 3.

Definition 2 (Publicly consistent queries)**.** *$\mathcal{A}$ submits a publicly consistent query pair (Q^0, Q^1), which satisfy all the following conditions:*

- *First of all, both queries would succeed, and are for the same type of oracle.*
- *For queries to $\mathcal{O}^1_{\mathsf{Join}}$, with the same the request info req_{joi} and they get the same user identifier uid as output.*
- *For queries to $\mathcal{O}^1_{\mathsf{File}}$, both with the same user identifier uid.*
- *For queries to $\mathcal{O}^1_{\mathsf{Deposit}}$ and $\mathcal{O}^1_{\mathsf{Withdraw}}$, the users can be different but the name and amount of the assets and the on-chain addresses are the same in both queries. For fiat money deposit/withdraw, the users and bank accounts are the same.*

$\mathrm{Exp}^{\mathsf{IND}}(\mathcal{A}, \mathcal{C}, \lambda)$

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$
$(pk, st) \leftarrow \mathcal{A}(epp)$
$\mathcal{C}$ randomly chooses $b \leftarrow\$ \{0,1\}$
Run $\mathcal{A}^{\mathcal{C}(\mathcal{O}^0_{\mathsf{IND}}, \mathcal{O}^1_{\mathsf{IND}})}(st)$ for N steps: // N=poly(λ)
In each step:
  $(Q^0, Q^1, st^0, st^1) \leftarrow \mathcal{A}(st)$
  if (Q^0, Q^1) are not *publicly consistent*, **then return** 0;
  else $\mathcal{C}$ forwards Q^b to $\mathcal{O}^0_{\mathsf{IND}}$, Q^{1-b} to $\mathcal{O}^1_{\mathsf{IND}}$,
  Run $\langle \mathcal{A}(st^0), \mathcal{O}^0_{\mathsf{IND}}(Q^b)\rangle$ and $\langle \mathcal{A}(st^1), \mathcal{O}^1_{\mathsf{IND}}(Q^{1-b})\rangle$
  // It simulates $\mathcal{A}$ induces honest users' behaviors
Finally, $\mathcal{A}$ halts, and outputs $\hat{b}$
return $(\hat{b} == b)$

Fig. 3: Interaction indistinguishability experiment

Definition 3 (Interaction indistinguishability)**.** *The interaction indistinguishability is described in Fig. 3. We say that an exchange system provides interaction indistinguishability if for all PPT $\mathcal{A}$ and λ it holds that*

$$\left|\Pr[\mathrm{Exp}^{\mathsf{IND}}(\mathcal{A}, \lambda) = 1] - 1/2\right| \leq negl(\lambda)$$

Remark 1 (Relation with basic anonymity)**.** *The interaction indistinguishability implies the basic anonymity if the exchange identity, and exchange content are also public and identical in both queries. Only the withdraw identity is concealed like in the basic withdraw anonymity experiment.*

Remark 2 (Best possible anonymity)**.** *We claim that the interaction indistinguishability achieves the best of possible anonymity. The interaction indistinguishability covers all kinds of transactions with specific public information. When we specify that the public information exclusively comprises the unavoidable leakage as defined in Def. 2, we can ensure that the platform **learns nothing** about the user from their interactions, except for the unavoidable leakage. Recall the Example 2, we can see any avoidable leakage in these cases excludes some users. If all avoidable leakages are prevented, the anonymity set expands to encompass a broader range of users, now including both David and Ella.*

D. Soundness definitions

Extractor. In overdraft prevention and client-compliance experiments, adversary $\mathcal{A}$ who acts as a malicious user, gets some valid records after querying oracles and keeps them secret. It means that after some successful anonymous transactions, the experiment does not know how many assets the users actually own and their correct compliance information. So *it is hard to decide whether $\mathcal{A}$ breaks the overdraft prevention or compliance properties*. To deal with this dilemma, in those security experiments, we introduce an extractor $\mathcal{E}$ that can output the user identity and detailed information for each transaction. Note that both overdraft prevention and compliance are soundness properties. We mimic the classic proof-of-knowledge style of definition, and the extractor can rewind $\mathcal{A}$ to the former state and $\mathcal{A}$ reuses its randomness $r_{\mathcal{A}}$, similar to the proof of knowledge extractor [29]. Then the experiment is able to check if any overdraft or compliance cheating happens.

1) Overdraft prevention: Overdraft prevention requires that users cannot spend more than they own within the platform. Concretely it ensures no malicious users could exchange or withdraw more assets than they actually own. Using the transaction details extracted by the extractor $\mathcal{E}$, the experiment can check whether an overdraft happens: (1) the user gets credited more assets than his deposit or exchange-in; (2) the user gets deducted less asset than his withdrawal or exchange-out; (3) the remainder amount of asset is negative; (4) the exchange is unfair; (5) the user steals others' asset.

We formally define overdraft prevention via the following experiment. $\mathcal{A}$ acts as malicious users and interacts with extractor $\mathcal{E}$ via querying oracles: $\mathcal{O}_{\mathsf{od}} = \{\mathcal{O}^2_{\mathsf{Issue}}, \mathcal{O}^2_{\mathsf{Credit}}, \mathcal{O}^2_{\mathsf{Deduct}}, \mathcal{O}^2_{\mathsf{Update}}, \mathcal{O}^2_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{public}}\}$. $\mathcal{A}$ can query at most $N = poly(\lambda)$ times, then it halts. $\mathcal{E}$ extracts a set of successful transaction histories $\{h_t\}$ for $t \in [N]$, where each transaction history $h_t = (uid, Rd_{reg}, Rd_{ast}, Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, ts_t, pub_t)$ includes user id uid, the input records (Rd_{reg}, Rd_{ast}), the output records $(Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast})$, the transaction transcript ts_t, and the related public information pub_t, where some records could be empty for some transactions. For example, Rd^{out}_{ast} is empty in withdraw transaction. Especially, transaction transcript $ts_t := (name, amt, \dots)$ is a tuple of attributes including the asset name $ts_t.name$ and amount $ts_t.amt$, etc. Public information $pub_t := (pr_{in}, pr_{out}, \dots)$ is a tuple of attributes including input-asset price $pub_t.pr_{in}$, output-asset price $pub_t.pr_{out}$, etc. Please note, there could be some other

metadata per the implementation need, so we cannot specify all the attributes and some attributes could be empty for different transactions. Finally, the experiment sequentially checks each transaction history to figure out whether any one of the above overdraft cases happens. Especially, in deposit and withdraw transactions, ts_t contains the deposited or withdrawn asset information: $ts_t.name = i$, $ts_t.amt = k_i$ denotes that the user deposits or withdraws the asset i with amount k_i. To facilitate the check, the experiment maintains a list RdSet for tracking asset records that have not been spent till the checkpoint, which is initialized as empty.

$\mathrm{Exp}^{\mathrm{od}}(\mathcal{A}, \mathcal{E}, \lambda)$

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda)), (1^n, st) \leftarrow \mathcal{A}(epp)$, for some $n \in \mathbb{N}$

$(pk, sk) \leftarrow \mathsf{PKeyGen}(epp, 1^n)$

Run $\mathcal{A}^{\mathcal{O}_{\mathsf{od}}}(epp, pk, st)$

if any oracle aborts **then return** 0;

else continue until $\mathcal{A}$ halts

Run $\{h_t\} \leftarrow \mathcal{E}^{\mathcal{A}}(epp)$

// $\mathcal{E}$ could control the randomness of $\mathcal{A}$

Set $\mathsf{RdSet} \leftarrow \emptyset$

For $t = 1$ to N, check h_t :

Parse $h_t = (uid, Rd_{reg}, Rd_{ast}, Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, ts_t, pub_t)$

For Deposit transaction :

if $Rd^{out}_{ast}.name \neq ts_t.name$ or $Rd^{out}_{ast}.amt \neq ts_t.amt$ **then return** 1

// the name or amount of credited asset record is wrong

else let $\mathsf{RdSet} \leftarrow \{Rd'_{reg}, Rd^{out}_{ast}\} \cup \mathsf{RdSet}$

For Exchange transaction :

if any of the followings happens, **then return** 1 :

– $\{Rd_{reg}, Rd_{ast}\} \not\subseteq \mathsf{RdSet}$; // invalid records

– $Rd'_{ast}.amt < 0$ // deducted amount exceeds asset amount

– $(Rd_{ast}.amt - Rd'_{ast}.amt) \cdot pub_t.pr_{in} \neq Rd_{ast}.amt \cdot pub_t.pr_{out}$

// the deducted value is not equal to the credited value

else $\mathsf{RdSet} \leftarrow \mathsf{RdSet} \setminus \{Rd_{reg}, Rd_{ast}\} \cup \{Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}\}$

For Withdraw transaction :

if any of the followings happens, **then return** 1 :

– $\{Rd_{reg}, Rd_{ast}\} \not\subseteq \mathsf{RdSet}$;

– $Rd_{ast}.name \neq ts_t.name$

– $Rd'_{ast}.amt < 0$ or $Rd_{ast}.amt - Rd'_{ast}.amt < ts_t.amt$

// withdraws more asset than the deducted amount;

else let $\mathsf{RdSet} \leftarrow \mathsf{RdSet} \setminus \{Rd_{reg}, Rd_{ast}\} \cup \{Rd'_{reg}, Rd'_{ast}\}$

return 0

Fig. 4: Overdraft prevention experiment.

Definition 4 (Overdraft Prevention)**.** *As shown in Fig. 4, we say that an exchange system can prevent overdraft if for all PPT $\mathcal{A}$ and λ, there exists $\mathcal{E}$ such that it holds that*

$$\Pr[\mathrm{Exp}^{\mathrm{od}}(\mathcal{A}, \mathcal{E}, \lambda) = 1] \leq negl(\lambda)$$

2) Compliance: This property requires both clients and the platform to comply with the regulation rules. Here we represent these rules using compliance functions, and formalize both client compliance and platform compliance.

In the client compliance experiment, $\mathcal{A}$ acts as malicious users and interacts with extractor $\mathcal{E}$ via querying oracles: $\mathcal{O}_{\mathsf{clie-comp}} = \{\mathcal{O}^2_{\mathsf{Issue}}, \mathcal{O}^2_{\mathsf{Credit}}, \mathcal{O}^2_{\mathsf{Deduct}}, \mathcal{O}^2_{\mathsf{Update}}, \mathcal{O}^2_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{public}}\}$. $\mathcal{A}$ outputs a certified document doc^* with four attributes (uid, cp, mt, sig). $\mathcal{E}$ extracts a set of successful transaction histories $\{h_t\}$ as in overdraft prevention experiment. $\mathcal{A}$ wins, if doc^* passes the authority verification, i.e., $\mathsf{Verify}(epp, pk, doc^*) \rightarrow 1$, but there exists extracted transaction history that does not follow basic client-compliance rules (we will specify in the following), or the submitted valid document doc^* is inconsistent with the extracted transaction histories $\{h_t\}$. Concretely, each transaction in $\{h_t\}$ satisfy that: (1) the user has already registered (KYC rule); (2) all records in one transaction belong to the same user (AML rule to avoid secretly transferring assets to other accounts); (3) the compliance-related information in each asset record, such as buying and selling price, is correct (general compliance rule). To facilitate the check, the experiment maintains a list RU of all registered users, which is initialized as empty.

If all transaction histories in $\{h_t\}$ pass the above check, consistency checks between doc^* and $\{h_t\}$ per function F will also be done. Specifically, in one transaction, the compliance information cp_t is collected from $\{h_t\}$ (e.g., the asset prices and amount) and is added to the user's the compliance information set $\{\mathsf{cp}\}_{uid}$. Then F is applied to $\{\mathsf{cp}\}_{doc^*.uid}$ to get the final result $\widetilde{cp}_{doc^*.uid}$. See Fig. 5 for details.

Definition 5 (Client Compliance)**.** *The client compliance experiment is shown in Fig. 5. We say that an exchange system is client-compliant w.r.t. a compliance function F if for all PPT $\mathcal{A}$ and λ, there exists $\mathcal{E}$ such that*

$$\Pr[\mathrm{Exp}^{\mathrm{clie-comp}}(\mathcal{A}, \mathcal{E}, F, \lambda) = 1] \leq negl(\lambda)$$

For *platform compliance*, it is similar with the correctness, and we require that the internal state of the honest platform is always satisfied with the platform compliance rule. For example, the platform can self-check whether it owns sufficient cash and assets to cover fund outflows for the previous 30 days according to the regulation rule [1].

V. Private and Compliable Exchange System

In this section, we present the generic construction of the private and compliable exchange system Π_{Pisces} that achieves full anonymity, overdraft prevention, and compliance, and we provide formal security analysis. Before that, we give concrete compliance rules that our system aims to comply with for both clients and the platform. Following that, we introduce the concept of price credentials and illustrate the high-level idea about the construction.

Concrete compliance rules. For F-client-compliance, we take the tax report as an example of F, called tax-report-client-compliance. It requires clients to report the investment profit yearly (total gain minus total cost). The taxable profit is calculated when clients sell their assets via exchange or withdraw transactions. Concretely, $cost = pr_i \cdot k_i$ and $gain = \overline{pr}_i \cdot k_i$, where k_i is the exchange-out or withdrawn asset amount, pr_i is the buying price and $\overline{pr}_i$ is the selling price of the asset.

$\mathrm{Exp}^{\mathrm{clie-comp}}(\mathcal{A}, \mathcal{E}, F, \lambda)$

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda)), (1^n, st) \leftarrow \mathcal{A}(epp)$, for some $n \in \mathbb{N}$

$(pk, sk) \leftarrow \mathsf{PKeyGen}(epp, 1^n), \mathsf{RU} \leftarrow \emptyset$

Run $doc^* := (uid, cp, mt, sig)/\perp \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{clie-comp}}}(epp, pk, st)$

if oracle aborts or $\mathsf{Verify}(epp, pk, doc^*) = 0$ **then return** 0

Run $\{h_t\} \leftarrow \mathcal{E}^{\mathcal{A}}(epp)$ // $\mathcal{E}$ controls the randomness of $\mathcal{A}$

For $t = 1$ to N, check h_t :

Parse $h_t = (uid, Rd_{reg}, Rd_{ast}, Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, ts_t, pub_t)$

For Join transaction :

let $\mathsf{RU} \leftarrow \{uid\} \cup \mathsf{RU}, \{\mathsf{cp}\}_{uid} = \emptyset$

For Deposit transaction :

if any of the followings happens, **then return** 1 :

– single transaction involves different user identifiers;

– $uid \notin \mathsf{RU}$;

// also check them in exchange and withdraw transactions

– $Rd^{out}_{ast}.acp \neq pub_t.pr_{out}$ // price was wrong

For Exchange transaction :

if $Rd'_{ast}.acp \neq Rd_{ast}.acp$ or $Rd^{out}_{ast}.acp \neq pub_t.pr_{out}$ **then return** 1

else collect cp_t from h_t, add cp_t to $\{\mathsf{cp}\}_{uid}$

For Withdraw transaction :

if $Rd'_{ast}.acp \neq Rd_{ast}.acp$ **then return** 1

else collect cp_t from h_t, add cp_t to $\{\mathsf{cp}\}_{uid}$

if $\{\mathsf{cp}\}_{doc^*.uid} = \emptyset$, **then return** 0

else compute $\widetilde{cp}_{doc^*.uid} \leftarrow F(\{\mathsf{cp}\}_{doc^*.uid})$

// F is a function specified by the compliance rule

if $doc^*.cp \neq \widetilde{cp}_{doc^*.uid}$ **then return** 1

else return 0

Fig. 5: F-Client-Compliance experiment.

Then he reports the accumulated cost $cp_1 = \sum pr_i \cdot k_i$ and gain $cp_2 = \sum \overline{pr}_i \cdot k_i$ to the authority.

For G-platform-compliance, we refer to the liquidity coverage ratio (LCR) requirement of Basel Accords [1], a series of banking regulations established by representatives from major global financial centers. LCR mandates that banks hold sufficient cash and liquid assets to cover fund outflows for 30 days. The platform always knows clearly the inflows/outflows for each coin including fiat money transfers (as the platform receives or transfers them out) by checking its internal state. It can prepare enough amount of coins for all kinds of assets. Thus this LCR-platform-compliance is compatible with our fully anonymous setting.

About price credential and price fluctuation. To achieve efficient private exchange with compliance, we introduce price credentials denoted as $px := (time, name, pr, sig)$, where the signature $px.sig$ is signed by the platform on the current time $px.time$, the coin name $px.name$, and the corresponding current price $px.pr$. To tackle price fluctuation without leaking coin information, the platform keeps signing the latest prices for all coins at the same timestamp. In the exchange transaction, the user proves that the newly exchanged asset record contains the name and price, and they know a valid signature on these values from the latest timestamp's price credential. The user also ensures that the exchange is fair based on these prices and amounts. This approach enables the user to prove with just a single credential. It significantly reduces communication and computation costs to a constant level.

High-level idea. Before giving the formal algorithms, let us illustrate the high-level construction idea with five concrete transaction examples as follows. The platform only knows some public information, like all registered users and their bank accounts, the name and amount of deposited and withdrawn assets, as shown in Fig. 6.

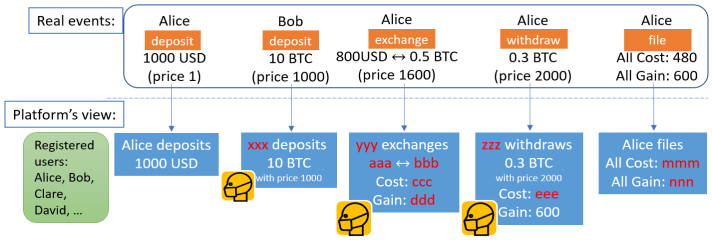

Fig. 6: Platform's view in the exchange system

- When deposits 1000 USD, the bank account reveals Alice's identity. Then Alice gets an asset credential or signature generated by the platform. To prevent double-spending and ensure regulatory compliance, the signed message must contain additional attributes beyond asset details. These attributes include a unique asset identifier, Alice's user identifier. They should be hidden from the platform to keep the user anonymous. Alice is required to prove in zero-knowledge that the blinded user identifier is equal to that in her registration credential. To meet tax-report-client-compliance, the profit of selling assets in exchange and withdraw transactions need to be computed. This necessitates including the exact cost of the assets when they were purchased in deposit and exchange transactions. To this end, we introduce the buying price as an additional attribute in asset credentials.
- When Bob deposits 10 BTC, the only difference from a fiat deposit is that the platform does not know Bob's identity.
- When Alice exchanges 800 USD for 0.5 BTC, she uses a 1000 USD asset credential to request two new credentials for the remaining 200 USD and 0.5 BTC, keeping their details hidden. The platform grants her request under specific conditions, including demonstrating in zero-knowledge that she has enough USD, ensuring the credentials share the same user identifier, confirming the non-negative remaining USD amount, matching the BTC's price with the latest credential, and verifying the total exchange value equivalence.
  We give each asset one separate asset credential for practicality especially with compliance. Intuitively, if all assets are in one credential their attributes would increase linearly with the asset names. Compliance makes it more complicated, because every asset transaction would have to be recorded as an attribute in the credential. This means that the credential attributes would keep growing. It is practical to separate each transaction asset, but it is hard to collect all transactions to calculate the total profit. We solve it by accumulating profit for each transaction involving exchanged-out or withdrawn assets, and recording it on user's exclusive registration cre-

dential as two attributes: accumulated cost and accumulated gain. Concretely, Alice shows her registration credential, and requests the platform to issue a new registration credential on a new index, updated cost and gain which are consistent with real cost (amount times buying price) and gain (amount times selling price).
- When withdraws 0.3 BTC, it is similar to the exchange operation, except for the exchanged-in asset. Alice also verifies the receipt of the withdrawn BTC on the blockchain.
- When files all cost 480 and all gain 600, Alice shows a valid registration credential with her identity, and requests an updated registration credential with a new index, reset cost and gain as zeros, and a file credential used to show to the regulatory authority. The file credential contains Alice's real identity, the correct cost 480 and gain 600, and some regulatory auxiliary information. Since the cost and gain are hidden from the platform to avoid information leakage, Alice should prove the committed cost and gain are equal to the ones in her registration credential, and the platform signs blindly. Alice unblinds it and submits the message signature pair to the authority for tax report.

A. An efficient Pisces construction

In this section, we give an efficient Pisces construction Π_{Pisces} from additive homomorphic commitment, blind signature and zero-knowledge proof[5]. Let Com be an additively homomorphic commitment scheme, $\Pi_{\mathrm{bs}} = (\mathsf{KeyGen}, \mathsf{Com}, \langle \mathsf{BlindSign}, \mathsf{BlindRcv} \rangle, \mathsf{Vrfy})$ be a blind signature scheme using Com to blind messages, and ZKAoK is the underlying proof system. The platform maintains the registered user set USet and the identifier set ID which are initially empty. Formal construction is in Fig. 7. The concrete instantiation is presented in Sec. VI.

Extended compliance support. Our construction also easily supports other regulation policies. We just give sketches here due to the page limitation. For example, AML requires that users cannot exchange or withdraw too many times in a time period. It can be achieved by adding a counter in the registration record. In each exchange or withdraw transaction, the user proves that the counter in his latest registration record is smaller than some value and the counter will be incremented by one in the newly issued record. Other rules are similar, such as transaction amounts, and (total) value of exchanged assets.

We can also enforce tax filing by prohibiting users who have not filed tax last year from exchanging and withdrawing. It can be achieved by adding a year number in the registration record indicating the year when the user filed tax last time. In each exchange or withdraw transaction, the user shows the year number in his latest registration record and this number is credited by one when the user has filed his tax.

B. Security analysis

Theorem 1 (Interaction indistinguishability)**.** *If* Π_{bs} *has blindness, the underlying* ZKAoK *is zero-knowledge, and the commitment is hiding, then the Pisces construction* Π_{Pisces} *has interaction indistinguishability.*

[5]Note that these primitives are also used to construct updatable anonymous credentials and an incentive system in [13] which does not support the exchange functionality and compliance rule required in our setting.

Proof: We prove this theorem by a sequence of hybrid experiments $(\mathsf{G}_{\mathrm{real}}, \mathsf{G}_1, \mathsf{G}_{\mathrm{sim}})$. $\mathsf{G}_{\mathrm{real}}$ is the original IND experiment. G_1 modifies $\mathsf{G}_{\mathrm{real}}$ by simulating the ZKAoK proof. $\mathsf{G}_{\mathrm{sim}}$ modifies G_1 by replacing the original commitments with commitments on random strings. Since the underlying ZKAoK is zero-knowledge, G_1 can be distinguished from $\mathsf{G}_{\mathrm{real}}$ with only negligible probability. Due to that the commitment scheme is hiding and the blind signature has blindness, $\mathsf{G}_{\mathrm{sim}}$ can be distinguished from G_1 with only negligible probability. Thus $\mathsf{G}_{\mathrm{sim}}$ can be distinguished from $\mathsf{G}_{\mathrm{real}}$ with only negligible probability. Furthermore, in $\mathsf{G}_{\mathrm{sim}}$, $\mathcal{A}$'s view is fully simulated, independent of b, $\mathcal{A}$'s advantage in $\mathsf{G}_{\mathrm{sim}}$ is 0. So $\mathcal{A}$ wins in $\mathsf{G}_{\mathrm{real}}$ with at most negligible probability. We describe $\mathsf{G}_1, \mathsf{G}_{\mathrm{sim}}$ as follows.

G_1: This experiment modifies $\mathsf{G}_{\mathrm{real}}$ by simulating the ZKAoK proof. It works as follows: at the beginning, $\mathcal{C}$ chooses $b \leftarrow\$ \{0,1\}$ and generates $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and zero-knowledge trapdoor $td \leftarrow \mathsf{Sim}(1^\lambda)$. $\mathcal{C}$ sends pp to $\mathcal{A}$ and initializes two sets of oracles $\mathcal{O}^0_{\mathsf{IND}}$ and $\mathcal{O}^1_{\mathsf{IND}}$. In the following oracle queries, the proofs are generated by $\mathcal{C}$ using td: $\pi \leftarrow \mathsf{Sim}(td, x)$. G_1 proceeds in steps, and each time $\mathcal{A}$ queries an oracle, it sends $\mathcal{C}$ a pair of queries (Q^0, Q^1). $\mathcal{C}$ first checks that they are *publicly consistent* according to Def. 2, then simulates different oracles.

- For $\mathcal{O}^1_{\mathsf{Join}}$ oracle, $Q^0 = (req^0_{joi}, ref^0_{\mathsf{reg}})$ and $Q^1 = (req^1_{joi}, ref^1_{\mathsf{reg}})$. To answer them, $\mathcal{C}$ behaves as in $\mathsf{G}_{\mathrm{real}}$ except for the following modification. The ZKAoK proofs π^0, π^1 are simulated using td. $\mathcal{C}$ replies $\mathcal{A}$ with (com^b, π^0) and (com^{1-b}, π^1). If $\mathcal{A}$ accepts the proofs, it runs $\hat{\sigma}^0 \leftarrow \mathsf{BlindSign}(pp, pk, com^0)$ and $\hat{\sigma}^1 \leftarrow \mathsf{BlindSign}(pp, pk, com^1)$ and sends them to $\mathcal{C}$ and continues.
- For $\mathcal{O}^1_{\mathsf{Deposit}}$ oracle, $Q^0 = (uid^0, req^0_{dep}, ref^0_{\mathsf{reg}}, ref^{\mathsf{out0}}_{\mathsf{ast}})$ and $Q^1 = (uid^1, req^1_{dep}, ref^1_{\mathsf{reg}}, ref^{\mathsf{out1}}_{\mathsf{ast}})$. To answer them, $\mathcal{C}$ behaves as in $\mathsf{G}_{\mathrm{real}}$ except the following modification: It simulates the ZKAoK proofs π^0, π^1 using td. $\mathcal{C}$ replies $\mathcal{A}$ with $(\{com^b_u\}^2_{u=1}, \pi^0)$ and $(\{com^{1-b}_u\}^2_{u=1}, \pi^1)$. If $\mathcal{A}$ accepts the proofs, it runs the BlindSign algorithm and sends the respective blinded signatures to $\mathcal{C}$ and continues.
- For $\mathcal{O}^1_{\mathsf{Exchange}}$ oracle, $Q^0 = (uid^0, req^0_{exc}, ref^0_{\mathsf{reg}}, ref^{\mathsf{in0}}_{\mathsf{ast}}, ref^{\mathsf{out0}}_{\mathsf{ast}})$ and $Q^1 = (uid^1, req^1_{exc}, ref^1_{\mathsf{reg}}, ref^{\mathsf{in1}}_{\mathsf{ast}}, ref^{\mathsf{out1}}_{\mathsf{ast}})$. To answer them, $\mathcal{C}$ behaves as in $\mathsf{G}_{\mathrm{real}}$ except that it simulates the ZKAoK proofs π^0, π^1 using td. $\mathcal{C}$ replies $\mathcal{A}$ with $(\{com^b_u\}^7_{u=1}, \pi^0)$ and $(\{com^{1-b}_u\}^7_{u=1}, \pi^1)$. If $\mathcal{A}$ accepts the proofs, it runs the BlindSign algorithm and sends the blinded signatures to $\mathcal{C}$ and continues.
- For $\mathcal{O}^1_{\mathsf{Withdraw}}$ oracle, $Q^0 = (uid^0, req^0_{wit}, ref^0_{\mathsf{reg}}, ref^{\mathsf{in0}}_{\mathsf{ast}})$ and $Q^1 = (uid^1, req^1_{wit}, ref^1_{\mathsf{reg}}, ref^{\mathsf{in1}}_{\mathsf{ast}})$. To answer them, $\mathcal{C}$ behaves as in $\mathsf{G}_{\mathrm{real}}$ except that it simulates the ZKAoK proofs π^0, π^1 using td. $\mathcal{C}$ replies $\mathcal{A}$ with $(\{com^b_u\}^4_{u=1}, \pi^0)$ and $(\{com^{1-b}_u\}^4_{u=1}, \pi^1)$. If $\mathcal{A}$ accepts the proofs, it runs the BlindSign algorithm and sends the blinded signatures to $\mathcal{C}$ and continues.
- For $\mathcal{O}^1_{\mathsf{File}}$ oracle, $Q^0 = (uid^0, req^0_{fil}, ref^0_{\mathsf{reg}})$, and $Q^1 = (uid^1, req^1_{fil}, ref^1_{\mathsf{reg}})$. To answer them, $\mathcal{C}$ behaves as in $\mathsf{G}_{\mathrm{real}}$ except that it simulates the ZKAoK proofs π^0, π^1 using td. $\mathcal{C}$ replies $\mathcal{A}$ with $(\{com^b_u\}^3_{u=1}, \pi^0)$ and $(\{com^{1-b}_u\}^3_{u=1}, \pi^1)$. If $\mathcal{A}$ accepts the proofs, it runs the BlindSign algorithm and sends the blinded signatures to $\mathcal{C}$ and continues.

Note that from $\mathsf{G}_{\mathrm{real}}$ to G_1, the only difference is that the ZKAoK proofs are simulated. Due to that the ZKAoK

- Setup$(1^\lambda) \to epp$
  epp is the system public parameter containing both static parameters such as the blind signature public parameter pp, the total assets kinds n, the maximum balance number $v_{max} = p - 1$ for some super-poly p, and dynamic parameters such as current price $\overline{pr}_i$ and the price credential $\mathsf{px}(i)$ of each asset $i \in [n]$.
- PKeyGen$(epp) \to (pk, sk)$
  P: $(pk, sk) \leftarrow$ KeyGen$(1^\lambda, pp)$
- $\langle$Join(epp,pk,req_{joi}), Issue$(epp,pk,sk)\rangle \to (uid, Rd_{reg}, b)$:
  //U outputs Rd_{reg}, P outputs b
  U: $uid, rid, r \leftarrow\$ \mathbb{Z}_p$, sends $(req_{joi}, uid, com, \pi)$ to P
  $req_{joi} = (info)$, $com = \mathsf{Com}(uid, rid, cp_1, cp_2; r)$, π proves:
  $-com$ contains uid, rid, r and $(cp_1, cp_2) = (0, 0)$.
  P: if $uid \in$ USet or the proof π is invalid, outputs $b = 0$. Otherwise, adds uid to USet, replies with $\hat{\sigma}_{reg}$:
  $\hat{\sigma}_{reg} \leftarrow \mathsf{BlindSig}(pp, pk, sk, com)$
  U: outputs $Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$
  $\sigma_{reg} \leftarrow \mathsf{BlindRcv}(pp, pk, \hat{\sigma}_{reg}, r)$
- $\langle$Deposit$(epp,pk,uid, Rd_{reg}, req_{dep})$, Credit$(epp, pk, sk)\rangle \to (Rd^{out}_{ast}, b)$:
  //$Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$
  //U outputs Rd^{out}_{ast}, P outputs b
  U: $aid, r_1, r_2 \leftarrow\$ \mathbb{Z}_p$, sends $(req_{dep}, com_1, com_2, \pi)$ to P, where
  $req_{dep} = (i, k_i)$, $com_1 = \mathsf{Com}(uid, rid, cp_1, cp_2; r)$,
  $com_2 = \mathsf{Com}(uid, aid, i, k_i, pr_i; r_2)$, π proves that:
  – he owns a valid signature σ_{reg} w.r.t. com_1;
  – com_2 contains uid, aid, i, k_i, pr_i, where uid is the same as that in com_1 and i, k_i are the same as those in req_{dep}.
  P: if does not receive asset or the proof π is invalid, outputs $b = 0$. Otherwise, replies with $\hat{\sigma}_{ast}$ and outputs $b = 1$.
  //$\hat{\sigma}_{ast} \leftarrow \mathsf{BlindSig}(pp, pk, sk, com_2)$
  U: outputs $Rd^{out}_{ast} = (uid, aid, i, k_i, pr_i, \sigma_{ast})$
  //$\sigma_{ast} \leftarrow \mathsf{BlindRcv}(pp, pk, \hat{\sigma}_{ast}, r_2)$
- $\langle$Exchange $(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{exc})$, Update$(epp, pk, sk)\rangle \to (Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, b)$:
  //epp contains price credentials: $px(i) = (mt, i, \overline{pr}_i, \sigma_i)$
  //$px(j) = (mt, j, \overline{pr}_j, \sigma_j)$, mt is the timestamp as metadata
  //$req_{exc} = (i, k_i, j, k_j)$
  U: $rid', aid', aid^{out}, \{r_u\}^7_{u=1} \leftarrow\$ \mathbb{Z}_p$, sends $(rid, aid, \{com_u\}^7_{u=1}, \pi)$ to P, where
  $com_1 = \mathsf{Com}(uid, rid, cp_1, cp_2; r)$,
  $com_2 = \mathsf{Com}(uid, aid, i, v_i, pr_i; r_2)$,
  $com_3 = \mathsf{Com}(uid, rid', cp'_1, cp'_2; r_3)$,
  $com_4 = \mathsf{Com}(uid, aid', i, v'_i, pr_i; r_4)$,
  $com_5 = \mathsf{Com}(uid, aid^{out}, j, v_j, \overline{pr}_j; r_5)$,
  $com_6 = \mathsf{Com}(mt, i, \overline{pr}_i; r_6)$,
  $com_7 = \mathsf{Com}(mt, j, \overline{pr}_j; r_7)$, π proves that:
  – he owns valid platform's signatures w.r.t. $com_1, com_2, com_6, com_7$;
  – the revealed rid and aid are identifiers in com_1 and com_2;
  – there are identifiers aid', rid', aid^{out} in com_3, com_4, com_5;
  – the i-th asset in com_2 is enough, i.e., $k_i \geq 0$ and $v_i - k_i = v'_i \geq 0$;
  – com_2, com_4 and com_6 share the same asset kind i;
  – com_2 and com_4 share the same price pr_i;
  – com_5 and com_7 share the same kind j and price $\overline{pr}_j$;
  – com_3 contains $cp'_1 = cp_1 + k_i \cdot pr_i$, $cp'_2 = cp_2 + k_i \cdot \overline{pr}_i$;
  – fairness: com_5 contains number $v_j = k_j > 0$ and price $\overline{pr}_j$ satisfying $\overline{pr}_i \cdot k_i = \overline{pr}_j \cdot k_j$, which also implies $k_i > 0$;
  – $\{com_u\}^5_{u=1}$ share the same uid.
  P: if rid or $aid \in$ ID or π is invalid, outputs $b = 0$. Otherwise, replies with blind signatures $\hat{\sigma}_{reg}, \hat{\sigma}_{ast}, \hat{\sigma}^{out}_{ast}$ on com_3, com_4, com_5 respectively, adds rid, aid to ID outputs $b = 1$.
  U: outputs $Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}$,
  $Rd'_{reg} = (uid, rid', cp'_1, cp'_2, \sigma'_{reg})$,
  $Rd'_{ast} = (uid, aid', i, v'_i, pr_i, \sigma'_{ast})$,
  $Rd^{out}_{ast} = (uid, aid^{out}, j, k_j, \overline{pr}_j, \sigma^{out}_{ast})$.
  //$\sigma'_{reg}, \sigma'_{ast_i}, \sigma^{out}_{ast}$ are unblinded signatures of $\hat{\sigma}_{reg}, \hat{\sigma}_{ast}, \hat{\sigma}^{out}_{ast}$
- $\langle$Withdraw$(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{wit})$, Deduct $(epp, pk, sk)\rangle \to (Rd'_{reg}, Rd'_{ast}, b)$:
  U: $rid', aid', \{r_u\}^4_{u=1} \leftarrow\$ \mathbb{Z}_p$, sends $(req_{wit}, rid, aid, \{com_u\}^4_{u=1}, \pi)$ to P, where $req_{wit} = (i, k_i)$,
  $com_1 = \mathsf{Com}(uid, rid, cp_1, cp_2; r_1)$,
  $com_2 = \mathsf{Com}(uid, aid, i, v_i, pr_i; r_2)$,
  $com_3 = \mathsf{Com}(uid, rid', cp'_1, cp'_2; r_3)$,
  $com_4 = \mathsf{Com}(uid, aid', i, v'_i, pr_i; r_4)$, π proves that:
  – he owns valid platform's signatures w.r.t. com_1, com_2;
  – rid and aid are identifiers in com_1 and com_2;
  – there are new identifiers rid', aid' in com_3, com_4 respectively;
  – all these commitments share the same uid;
  – the i-th asset in com_2 is enough, i.e., $v_i \geq 0$ and $v_i - k_i = v'_i \geq 0$;
  – com_2 and com_4 share the same asset kind i (as that in req_{wit}) and price pr_i;
  – com_3 contains $cp'_1 = cp_1 + k_i \cdot pr_i$, $cp'_2 = cp_2 + k_i \cdot \overline{pr}_i$, $\overline{pr}_i$ is the current price of asset i.
  P: if rid or $aid \in$ ID or π is invalid, outputs $b = 0$. Otherwise, replies with blind signatures $\hat{\sigma}_{reg}, \hat{\sigma}_{ast}$ w.r.t. com_3, com_4, adds rid, aid to ID, outputs $b = 1$.
  U: outputs Rd'_{reg}, Rd'_{ast}, where
  $Rd'_{reg} = (uid, rid', cp'_1, cp'_2, \sigma'_{reg})$, $Rd'_{ast} = (uid, aid', i, v'_i, pr_i, \sigma'_{ast})$.
- $\langle$File$(epp, pk, uid, Rd_{reg}, req_{fil})$, Sign$(epp, pk, sk)\rangle \to (Rd'_{reg}, doc, b)$:
  //$Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$, $req_{fil} = (uid, cp_1, cp_2)$
  U: $rid', \{r_u\}^3_{u=1} \leftarrow\$ \mathbb{Z}_p$, sends $(uid, rid, \{com_u\}^3_{u=1}, \pi)$ to P, where
  $com_1 = \mathsf{Com}(uid, rid, cp_1, cp_2; r_1)$,
  $com_2 = \mathsf{Com}(uid, rid', cp'_1, cp'_2; r_2)$,
  $com_3 = \mathsf{Com}(uid, cp_1, cp_2, mt; r_3)$, π proves that:
  – he owns a valid platform's signature w.r.t. com_1;
  – the revealed rid is the identifier in com_1;
  – com_1, com_2, com_3 share the same uid;
  – com_2 contains an identifier rid^* and $(cp^*_1, cp^*_2) = (0, 0)$;
  – com_3 contains cp_1, cp_2 which are the same as those in com_1;
  – com_3 contains the timestamp metadata mt and the uid in req_{fil}.
  P: if $rid \in$ ID or $uid \notin$ USet or π is invalid, outputs $b = 0$. Otherwise, replies with blind signatures $\hat{\sigma}_{reg}$ on com_2, $\hat{\sigma}_{cp}$ on com_3, adds rid to ID, outputs $b = 1$.
  U: outputs Rd'_{reg}, doc, where
  $Rd'_{reg} = (uid, rid', cp'_1, cp'_2, \sigma'_{reg})$, $doc = (uid, cp_1, cp_2, mt, \sigma_{cp})$.
  //$\sigma'_{reg}, \sigma_{cp}$ are unblinded signatures of $\hat{\sigma}_{reg}, \hat{\sigma}_{cp}$
- Verify$(epp, pk, doc) \to b$: //$doc = (uid, cp_1, cp_2, mt, \sigma)$
  Authortiy: gets the current timestamp mt' from epp. If $mt = mt'$ and Vrfy$(pk, (uid, cp_1, cp_2, mt), \sigma) \to 1$, then it outputs $b = 1$ indicating the verification succeeds. Otherwise, outputs $b = 0$.
- Check$(epp, st) \to b$:
  P: monitors the fund outflows and checks if it owns enough asset for the LCR-platform-compliance.

Fig. 7: Our efficient construction Π_{Pisces} of Pisces

scheme is zero-knowledge, we have that $|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A},\lambda)=1]| \leq negl(\lambda)$.

$\mathsf{G}_{\mathrm{sim}}$: This experiment modifies G_1 by replacing the original commitments with commitments on random strings. $\mathsf{G}_{\mathrm{sim}}$ proceeds in steps, and each time $\mathcal{A}$ invokes an oracle, it sends $\mathcal{C}$ a pair of queries (Q^0, Q^1). $\mathcal{C}$ first checks that they are *publicly consistent*, then simulates different oracles as follows.

- For $\mathcal{O}^1_{\mathsf{Join}}$ oracle, to answer queries Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces commitment com^0, com^1 on random strings r^0, r^1.
- For $\mathcal{O}^1_{\mathsf{Deposit}}$ oracle, to answer Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces commitments $\{com^0_u\}^2_{u=1}$ and $\{com^1_u\}^2_{u=1}$ on random strings.
- For $\mathcal{O}^1_{\mathsf{Exchange}}$ oracle, to answer Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces commitments $\{com^0_u\}^7_{u=1}$ and $\{com^1_u\}^7_{u=1}$ on random strings.
- For $\mathcal{O}^1_{\mathsf{Withdraw}}$ oracle, to answer Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces commitments $\{com^0_u\}^4_{u=1}$ and $\{com^1_u\}^4_{u=1}$ on random strings.
- For $\mathcal{O}^1_{\mathsf{File}}$ oracle, to answer Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces commitments $\{com^0_u\}^3_{u=1}$ and $\{com^1_u\}^3_{u=1}$ on random strings.

In each case of $\mathsf{G}_{\mathrm{sim}}$, $\mathcal{A}$'s view is independent of b. Thus, $\mathcal{A}$ just outputs a random guess $\hat{b}$ in $\mathsf{G}_{\mathrm{sim}}$, so its advantage is 0: $\Pr[\mathsf{G}_{\mathrm{sim}}(\mathcal{A},\lambda)=1] - 1/2 = 0$

Note that from G_1 to $\mathsf{G}_{\mathrm{sim}}$, we change that the commitments are on the random strings. Due to the hiding property of the commitment scheme and the blindness of Π_{bs} (which is also based on the hiding property of the commitment), we have that $|\Pr[\mathsf{G}_1(\mathcal{A},\lambda)=1] - \Pr[\mathsf{G}_{\mathrm{sim}}(\mathcal{A},\lambda)=1]| \leq negl(\lambda)$. In summary, we have that

$$
\begin{aligned}
&|\Pr[\mathrm{Exp}^{\mathsf{IND}}(\mathcal{A},\lambda)=1] - 1/2| = |\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda)=1] - 1/2| \\
\leq &|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda)=1] - \Pr[\mathsf{G}_1(\mathcal{A},\lambda)=1]| \\
&+ |\Pr[\mathsf{G}_1(\mathcal{A},\lambda)=1] - \Pr[\mathsf{G}_{\mathrm{sim}}(\mathcal{A},\lambda)=1]| \\
&+ |\Pr[\mathsf{G}_{\mathrm{sim}}(\mathcal{A},\lambda)=1] - 1/2| \leq negl(\lambda)
\end{aligned}
$$

■

Theorem 2 (Overdraft prevention)**.** *If the underlying* ZKAoK *has argument of knowledge, the commitment is binding and* Π_{bs} *is unforgeable, then* Π_{Pisces} *has overdraft prevention.*

Proof: In the overdraft prevention experiment, the adversary $\mathcal{A}$ wins if it withdraws more asset than it has deposited or exchanged. Given the transaction histories $h_t = (uid, Rd_{reg}, Rd_{ast}, Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, ts_t, pub_t)$ extracted by $\mathcal{E}$ for $t \in [N]$. If $\mathcal{A}$ wins, there exists at least one transaction with problems indicating that either the input or output records within it are flawed such that one of the following events happens:

1. The input record is not generated from previous transactions, i.e., $Rd \notin \mathsf{RdSet}$;
2. The user steals other honest users' assets;
3. The generation of asset record is wrong in one of the following cases:
   Deposit: $Rd^{out}_{ast}.name \neq ts_t.name$ or $Rd^{out}_{ast}.amt \neq ts_t.amt$;
   Exchange: $Rd'_{ast}.name \neq Rd_{ast}.name$ or $Rd'_{ast}.amt < 0$ or $(Rd_{ast}.amt - Rd'_{ast}.amt) \cdot pub_t.pr_{in} \neq Rd^{out}_{ast}.amt \cdot pub_t.pr_{out}$;
   Withdraw: $Rd_{ast}.name \neq ts_t.name$ or $Rd'_{ast}.amt < 0$ or $Rd_{ast}.amt - Rd'_{ast}.amt < ts_t.amt$.

For events 1 and 2, the input records are problematic which are forged or stole by $\mathcal{A}$. $\mathcal{A}$ may forge the asset records or reuse records with a different identifier. In order to use others' asset, $\mathcal{A}$ must guess the *aid* correctly. For event 3, the new generated asset records are problematic, $\mathcal{A}$ gets these records by cheating the issuer. The security of our scheme can be reduced to the underlying cryptographic building blocks. This includes standard primitives like commitment, blind signature, and non-interactive ZKAoK. We elaborate it case by case.

(1) Suppose that Pr[Event 1 happens] is non-negligible. In this case, it leads to at least one of the following contradictions:

- The record Rd is valid but was not generated via querying oracles, which breaks the unforgeability of Π_{bs};
- The asset record Rd is reused with another identifier. In this case, since the used identifier would be detected, the revealed identifiers must be different $aid \neq aid'$. It means one commitment produces two different openings which contradicts the binding property of the commitment.

(2) Suppose that Pr[Event 2 happens] is non-negligible. Here the input records are valid records generated from previous transaction but belong to other honest users. If $\mathcal{A}$ uses this asset record, it must know the respective *aid* which is kept privately by the honest user. It contradicts that $\mathcal{A}$ can only guess it correctly with negligible probability.

(3) Suppose that Pr[Event 3 happens] is non-negligible. In this case, $\mathcal{A}$ gets asset records with wrong attributes from a transaction. It leads to at least one of the following contradictions:

- For deposit transaction, $\mathcal{A}$ gets an asset record which is different from the deposit request. It happens only if one commitment produces two different openings which contradicts the binding property or $\mathcal{A}$ uses the incorrect witness to generate a valid proof, which breaks the argument of knowledge of underlying ZKAoK.
- For exchange transaction, $\mathcal{A}$ gets new exchange-out asset record which is different from the old one or gets exchange-in asset with more amount by breaking the fair exchange rule. It leads to at least one of the following contradictions:
  - for the commitments of records, $\mathcal{A}$ generates a valid proof with incorrect witness, which breaks the argument of knowledge of underlying ZKAoK;
  - $\mathcal{A}$ opens the commitment to different values and generates the proof. It means one commitment produces two different openings which contradicts the binding property of the commitment scheme;
  - the price credentials are forged by $\mathcal{A}$, thus the platform's signatures. It contradicts to the unforgeability of Π_{bs}.
- For withdraw transaction, the user should prove that it owns enough asset for the withdraw request by committing on the old asset and new asset. Then he proves that the opening of the asset name is the same as that in the request and the deducted amount is the same as the withdrawal amount and the new amount is non-negative. Now the new asset record does not meet at least one of these requirements. It happens only if one commitment produces two different openings which contradicts the binding property or $\mathcal{A}$ uses the incorrect witness to generate a valid proof, which breaks the argument of knowledge of underlying ZKAoK.

■

Theorem 3 (Compliance)**.** *If the underlying* ZKAoK *has argument of knowledge, the commitment is binding, and* Π_{bs} *is unforgeable, then* Π_{Pisces} *has tax-report-client-compliance.*

Proof: We prove the tax-report-client-compliance as follows. In this experiment, $\mathcal{A}$ wins if it outputs doc which passes the authority verification but is inconsistent with the transaction histories. Given that $\mathcal{E}$ extracts transaction histories $h_t = (uid, Rd_{reg}, Rd_{ast}, Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, ts_t, pub_t)$ for $t \in [N]$. $\mathcal{A}$ wins if one of the following events happens:

1. doc was not obtained from queries but passed verification;
2. The user uses others' registration record: the user identities of asset and registration records are not the same;
3. The price was inconsistent as follows: In deposit transaction, $Rd^{out}_{ast}.acp \neq pub_t.pr_{out}$; or in exchange transaction, $Rd'_{ast}.acp \neq Rd_{ast}.acp$ or $Rd^{out}_{ast}.acp \neq pub_t.pr_{out}$; Or in withdraw transaction, $Rd'_{ast}.acp \neq Rd_{ast}.acp$.
4. doc was obtained by interacting with $\mathcal{O}_{\mathsf{Sign}}$, but the inconsistency happens since the compliance information was updated incorrectly in some exchange or withdraw transaction;
5. The user identifier has not been registered : $uid \notin \mathsf{RU}$ in any transaction expect for Join;

In a high level, event 1 happens meaning that $\mathcal{A}$ forges a valid signature which is contradicted by the unforgeability of Π_{bs}. Events 2, 3, or 4, happens meaning that $\mathcal{A}$ finds the collisions of commitment that violate the binding property of commitment, or $\mathcal{A}$ proves on a wrong statement that violates the argument of knowledge property of non-interactive ZKAoK. Event 5 happens which contains three possible cases. The first is $\mathcal{A}$ forges a record with new uid, which violates the unforgeability of Π_{bs}. The second is $\mathcal{A}$ finds collisions on uid, which violates the binding property of commitment. The third is that $\mathcal{A}$ proves a wrong statement including the unregistered uid, which violates the argument of knowledge property of non-interactive ZKAoK. So, we reduce the security of our scheme to the unforgeability of Π_{bs}, the binding property of commitment, and the argument of knowledge property of non-interactive ZKAoK. We elaborate it case by case.

(1) Suppose that Pr[$\mathcal{A}$ wins and event 1 happens] is non-negligible. In this case, $\mathcal{A}$ works honestly for each transaction but sends a doc to the authority which contains the incorrect cp_1, cp_2 and a forged signature on them. It breaks the unforgeability of the blind signature.

(2) Suppose that Pr[$\mathcal{A}$ wins and event 2 happens] is non-negligible. In this case, a valid transaction is generated but the records belong to different users. However, the user needs to prove that all records belong to himself by proving they contain the same uid which is a contradiction. So it breaks the argument of knowledge of the underlying ZKAoK.

(3) Suppose that Pr[$\mathcal{A}$ wins and event 3 happens] is non-negligible. In this case, $\mathcal{A}$ generates a commitment for its new asset containing its *uid*, asset identifier *aid*, asset name *i*, amount k_i and price pr_i. Here pr_i is different from the real price w.r.t the output of $\mathcal{O}_{\mathsf{Public}}$. For the deposit transaction, the price is different from the public price. For the exchange transaction, the price is different from that of the old asset record or the price credential. For the withdraw transaction, the price is different from that of the old asset record. The occurrence of the incorrect price leads to at least one of the following contradictions:

- $\mathcal{A}$ uses the incorrect witness to generate a valid proof, which breaks the argument of knowledge of underlying ZKAoK;
- $\mathcal{A}$ opens the commitment to different values and generates the proof. It means one commitment produces two different openings which contradicts the binding property of the commitment scheme;
- In the exchange transaction, $\mathcal{A}$ manipulates the price by using the price credential forged by itself. It breaks the unforgeability of the blind signature scheme Π_{bs}.

(4) Suppose that Pr[$\mathcal{A}$ wins and event 4 happens] is non-negligible. In this case, for at least one exchange or withdraw transaction the new compliance information cp^*_1, cp^*_2 was incorrect but the proof is valid. It leads to at least one of the following contradictions:

- When computing cp^*_1, cp^*_2, $\mathcal{A}$ uses some incorrect selling prices different from the output of $\mathcal{O}_{\mathsf{Public}}$. Its success implies that it breaks the argument of knowledge of underlying ZKAoK, or breaks the binding property of the commitment scheme, or forges a price credential (in the exchange transaction) which breaks the unforgeability of the blind signature.
- When proving the correctness of cp^*_1, cp^*_2, $\mathcal{A}$ just uses the incorrect witness to generate a valid proof, which breaks the argument of knowledge of underlying ZKAoK;
- $\mathcal{A}$ opens the commitment to different compliance information values and generates the proof. It means one commitment produces two different openings which contradicts the binding property of the commitment scheme.

(5) Suppose that Pr[$\mathcal{A}$ wins and event 5 happens] is non-negligible. In this case, uid has not registered but $\mathcal{A}$ generates a valid transaction on it which leads to at least one of the following contradictions:

- $\mathcal{A}$ forges a registration record in which the σ_{reg} should be issued by the platform via a blind signature scheme, so $\mathcal{A}$ breaks the unforgeability of the blind signature;
- $\mathcal{A}$ does not have the σ_{reg} but generates a valid proof in the deposit, exchange or withdraw protocol, so it breaks the argument of knowledge of the underlying ZKAoK.

■

VI. Performance Evaluations

In this section, we describe our instantiation, prototype implementation and the performance evaluation. The evaluation results show that our design is efficient and practical.

Instantiation and implementation. We instantiate the anonymous exchange system using the Pointcheval Sanders blind signatures [34] and Pedersen commitment [33]. The ZKAoKs are instantiated with Σ-protocol on the knowledge of DLog, its equality, and range. We implement this instantiation of the anonymous exchange system with Java. We use the open source Java library upb.crypto[6] and the bilinear group provided by mcl(bn256)[7]. We run experiments on MacBook Air (1.6 GHz Dual-Core Intel Core i5, 16GB memory).

TABLE I: Avg. computation cost in milliseconds.

Party	Join	Deposit	Exchange	Withdraw
Pisces-user	9	11	46	37
Pisces-platform	7	14	88	62

[6]upb.crypto: https://github.com/upbcuk.

[7]mcl: https://github.com/herumi/mcl.

Performance. We test the pure computation time cost and communication cost of each procedure to show the efficiency. Then to show the practicality, we make two comparisons. One is to compare the secure exchange with plain exchange to show the overhead is truly small. The other is to compare with other anonymous credential applications, including Privacy Pass and the privacy-preserving incentive system (PPIS for short).

Computation cost. We test the computation time cost of each party in each procedure of the anonymous exchange system. As shown in table I, we can see that each party's time cost for each procedure is less than $88ms$, which is quite efficient.

Communication cost. We measure the communication cost of each procedure and none of them exceeds 12kb. Concretely, in the Join and deposit procedures, the user adds ˜2.6kb and ˜3.3kb data to the request, respectively. The platform adds a ˜1.8kb data to both responses. In the exchange and withdraw procedure, the user adds ˜12kb and ˜8.7kb data to the request, respectively. The platform adds ˜2.3kb and ˜2.8kb data to the response, respectively.

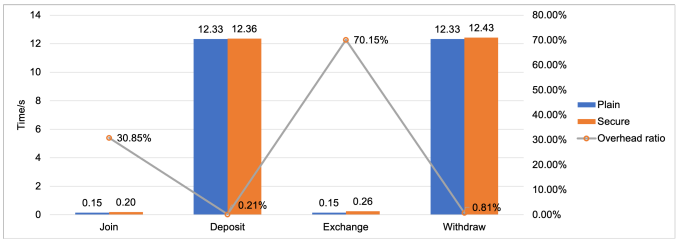

Fig. 8: Comparison between plain exchange system and Pisces

Comparison with plain exchange. To demonstrate its practicality, we have taken into consideration the cost of secure communication and have provided a comparison of the estimated time costs between plain operations and secure operations, as shown in Fig. 8. For plain operations, we have estimated the lower-bound time costs by considering only communication cost and on-chain transaction confirmation time, assuming the computation cost to be 0. We detail the estimation of plain and secure join, deposit, exchange, and withdraw in the following. Consider the optimal network performance, $30 - 40ms$ is the desired round-trip time (RTT)[8]. We pick RTT $= 30ms$.

a) Join operation: For a new user, the plain join includes the sign-up procedure and identity verification for KYC without on-chain confirmation cost. The communication cost includes one TLS handshake with at least 2 round-trip time (RTT for short) cost, 2 RTTs for sign-up setting username and password, and at least 1 RTT for identity verification. Totally the time cost is 5 RTT say $150ms$. The secure join runs all the plain join process and additionally runs the $\langle \mathsf{Join}, \mathsf{Issue} \rangle$ protocol. The time overhead includes 1 RTT for interaction latency, user and platform computation time $16ms$, and data transfer time $\frac{2.6kb}{10MB/s} + \frac{1.8kb}{100MB/s} \approx 0.278ms$. (We assume for a user device the uploading speed is $10MB/s$ and the downloading speed is $100MB/s$) The total time cost is $196.278ms$.

b) Deposit operation: A plain ETH deposit includes one handshake with a platform costing at least 2 RTTs, the log-in procedure to get the receipt address costing 1 RTT, an on-chain payment request costing at least 1 RTT, and an Ethereum transaction confirmation time $12.21s$. The total time cost of the plain deposit is $12.33s$. In an anonymous ETH deposit, users do not log in to the platform saving 1 RTT, but run all other procedures of plain deposit. Then users additionally interact with the platform running $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$, where the total computation cost is $25ms$, the interaction cost is 1 RTT, and the data transfer time is $\frac{3.4kb}{10MB/s} + \frac{1.8kb}{100MB/s} \approx 0.358ms$. The total time cost of the secure deposit is around $12.355s$.

c) Exchange operation: A plain exchange includes server authentication via TLS handshaking at least 2 RTTs, user login costing 1 RTT, price fetching with 1 RTT, and sending exchange request with 1 RTT. The total time cost of a plain exchange is at least 5 RTT, around $150ms$. The secure exchange removes login but additionally runs the $\langle \mathsf{Exchange}, \mathsf{Update} \rangle$ protocol, where the computation cost is $134ms$, interaction equals the exchange request sending, and data transfer costs $\frac{12kb}{10MB/s} + \frac{2.8kb}{100MB/s} \approx 1.228ms$. The total time cost of secure exchange is around $255.228ms$.

d) Withdraw operation: A plain withdraw of ETH includes server authentication via TLS handshaking at least 2 RTTs, user login costing 1 RTT, sending withdraw request with 1 RTT, and waiting for the on-chain confirmation with $12.21s$. The total time cost of a plain withdraw is around $12.33s$. The secure exchange gets rid of the login, saving 1 RTT, but additionally requests the price with 1 RTT, and runs the $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$ protocol, where the computation cost is $99ms$, interaction equals the withdraw request sending, and data transfer costs $\frac{8.7kb}{10MB/s} + \frac{2.3kb}{100MB/s} \approx 0.893ms$. The total time cost of a secure exchange is about $12.43s$.

The results show that the time costs for plain and secure operations are similar, with the overhead of each secure operation being less than $0.11s$. Notably, the overhead ratio of secure deposit and withdrawal is less than 1%.

Comparison with Privacy Pass and PPIS. To provide a better understanding of the practicality of our system, we conduct performance comparisons with widely used anonymous user-authentication mechanism Privacy Pass [19]. Privacy Pass published preliminary tests on consumer hardware, indicating that creating a pass in the extension takes less than 40ms[9]. Although the test environments may not be identical to ours, as both are on consumer hardware, the key takeaway is that each procedure of our system incurs similar time costs as Privacy Pass, showcasing its practicality. It's important to note that our system offers additional functionalities beyond Privacy Pass's anonymous authentication. We also test the time cost of the privacy-preserving incentive system (PPIS) [13]. The results, as shown in table II, demonstrate that our system is more complicated and more private, yet similarly practical to PPIS.

TABLE II: Avg. computation cost of each party per procedure over 100 runs in milliseconds.

Party	Join	Earn	Exchange	Spend
PPIS [13]-user	10	8	N/A	30
PPIS [13]-provider	9	12	N/A	72

[8]Network latency: https://www.ir.com/guides/what-is-network-latency.

[9]Privacy Pass FAQ: https://privacypass.github.io/faq/

VII. Conclusion

In this paper, we give the first study of cryptocurrency exchange that supports user anonymity and compliance requirements simultaneously. The platform cannot get more information from the transactions other than that has to be public. Users cannot get more assets from the platform so double spending is prohibited and they have to correctly report their accumulated profits for tax purposes, even in a private setting. Also, critical compliance functions are to be supported. Our construction is efficient and achieves constant computation and communication overhead with only simple cryptographic tools and rigorous security analysis. Additionally, we implement our system and evaluate its practical performance.

There are many interesting questions that could be further investigated, including more systematic study of privacy preserving solvency solutions as mentioned earlier, as well as more general private yet accountable/compliable (e.g., leveraging more advanced cryptographic tools such as [23]) exchange system, potentially even in the decentralized setting.

Acknowledgment

We would like to thank our shepherd and anonymous reviewers of NDSS24 for valuable feedbacks. This work was supported in part by research awards from Stellar Development Foundation, Ethereum Foundation, Protocol Labs, SOAR Prize, and University of Sydney's Digital Sciences Initiative through the Pilot Research Project Scheme.

References

[1] "Basel accords: Purpose, pillars, history, and member countries," https://www.investopedia.com/terms/b/basel_accord.asp, April 2022.

[2] "Binance revenue and usage statistics (2022)," https://www.businessofapps.com/data/binance-statistics/, September 2022.

[3] "Coinbase revenue and usage statistics (2022)," https://www.businessofapps.com/data/coinbase-statistics/, September 2022.

[4] "Data breaches," https://www.coindesk.com/tag/data-breaches/, October 2022.

[5] "Currency composition of international foreign reserves," https://data.imf.org/?sk=e6a5f467-c14b-4aa8-9f6d-5a09ec4e62a4, April 2023.

[6] "Tor browser," https://www.torproject.org/, 2023.

[7] "Understanding crypto taxes," https://www.coinbase.com/learn/crypto-basics/understanding-crypto-taxes, 2023.

[8] "Zcash," 2023. [Online]. Available: https://z.cash

[9] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," in *AFT*. Association for Computing Machinery, 2020, p. 255–267.

[10] C. Baum, B. David, and T. K. Frederiksen, "P2DEX: privacy-preserving decentralized cryptocurrency exchange," in *ACNS*. Springer, 2021, pp. 163–194.

[11] Binance, "Proof of reserves," https://www.binance.com/en/proof-of-reserves, July 2023.

[12] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *TCC*. Springer, 2013, pp. 315–333.

[13] J. Blömer, J. Bobolz, D. Diemert, and F. Eidens, "Updatable anonymous credentials and applications to incentive systems," in *CCS*. ACM, 2019, pp. 1671–1685.

[14] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *IEEE S&P*. IEEE, 2020, pp. 947–964.

[15] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *FC*. Springer, 2020, pp. 423–443.

[16] Y. Chen, X. Ma, C. Tang, and M. H. Au, "PGC: Decentralized confidential payment system with auditability," in *ESORICS*. Springer, 2020, pp. 591–610.

[17] S. Chu, Q. Xia, and Z. Zhang, "Manta: Privacy preserving decentralized exchange," *Cryptology ePrint Archive*, p. 1607, 2020.

[18] G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh, "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," in *CCS*. ACM, 2015, pp. 720–731.

[19] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy Pass: Bypassing internet challenges anonymously," *PoPETs*, vol. 2018, no. 3, pp. 164–180, 2018.

[20] A. Deshpande and M. Herlihy, "Privacy-preserving cross-chain atomic swaps," in *FC*. Springer, 2020, pp. 540–549.

[21] B. E. Diamond, "Many-out-of-many proofs and applications to anonymous zether," in *IEEE S&P*. IEEE, 2021, pp. 1800–1817.

[22] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation onion router," in *USENIX Security*. USENIX Association, 2004.

[23] H. Feng and Q. Tang, "Witness authenticating NIZK proofs and applications," in *CRYPTO*. Springer, 2021, pp. 3–33.

[24] K. Gjøsteen, M. Raikwar, and S. Wu, "PriBank: Confidential blockchain scaling using short commit-and-proof NIZK argument," in *CT-RSA*. Springer, 2022, pp. 589–619.

[25] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. K. Thyagarajan, "Foundations of coin mixing services," in *CCS*. ACM, 2022, pp. 1259–1273.

[26] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *CCS*. ACM, 2017, pp. 473–489.

[27] J. Groth and M. Kohlweiss, "One-out-of-many proofs: Or how to leak a secret and spend a coin," in *EUROCRYPT*. Springer, 2015, pp. 253–280.

[28] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS*. The Internet Society, 2017.

[29] Lindell, "Parallel coin-tossing and constant-round secure two-party computation," *Journal of Cryptology*, vol. 16, pp. 143–184, 2003.

[30] Y. nan Li, T. Qiu, and Q. Tang, "Pisces: Private and compliable cryptocurrency exchange," https://arxiv.org/abs/2309.01667, 2023.

[31] L. K. L. Ng, S. S. M. Chow, D. P. H. Wong, and A. P. Y. Woo, "LDSP: shopping with cryptocurrency privately and quickly under leadership," in *ICDCS*. IEEE, 2021, pp. 261–271.

[32] S. Noether, "Ring signature confidential transactions for monero," Cryptology ePrint Archive, Paper 2015/1098, 2015.

[33] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*. Springer, 1991, pp. 129–140.

[34] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *CT-RSA*. Springer, 2016, pp. 111–126.

[35] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. Esgin, J. K. Liu, J. Yu, and T. H. Yuen, "Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts," in *IEEE S&P*. IEEE, 2023, pp. 2020–2038.

[36] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A^{2}L: Anonymous atomic locks for scalability in payment channel hubs," in *IEEE S&P*. IEEE, 2021, pp. 1834–1851.

[37] A. Tomescu, A. Bhat, B. Applebaum, I. Abraham, G. Gueta, B. Pinkas, and A. Yanai, "UTT: decentralized ecash with accountable privacy," *Cryptology ePrint Archive*, p. 452, 2022.

[38] K. Wüst, K. Kostiainen, N. Delius, and S. Capkun, "Platypus: a central bank digital currency with unlinkable transactions and privacy-preserving regulation," in *CCS*. ACM, 2022, pp. 2947–2960.

[39] X. Yi and K.-Y. Lam, "A new blind ECDSA scheme for bitcoin transaction anonymity," in *AsiaCCS*. ACM, 2019, pp. 613–620.

APPENDIX A
ARTIFACT APPENDIX

***Abstract*—Pisces is a secure design of the private and compliable cryptocurrency exchange system, where users interact with the platform in the join, deposit, exchange, and withdrawal procedures. The artifact is a prototype in Java implementing all four procedures. For each procedure, the user sub-procedure interacts with the platform sub-procedure locally via memory communication, so that we can focus on testing the computation time cost and communication size, and excluding communication time cost. The experiment is run on MacBook Air (1.6GHz Dual-Core Intel Core i5, 16GB memory). The provided instructions should work for both MacOS and Linux, but the results may vary a bit depending on the computation powder.**

A. Description & Requirements

1) How to access: The artifact source code is accessible at https://github.com/yananli117/Pisces and https://doi.org/10.5281/zenodo.8328453 with DOI: 10.5281/zenodo.8328453.

2) Hardware dependencies:

- No hardware dependency for function test.
- MacBook Air (1.6GHz Dual-Core Intel Core i5, 16GB memory) is necessary to reproduce similar time cost. Since different computation powder makes much difference in the computation time cost, It is necessary to use the same to reproduce similar results, especially for the computation cost.

3) Software dependencies:

- MacOS or Linux
- JDK 17 or later versions
- Maven 3.8.1 or later version
- mcl library and gmp library. I did all the tests on MacOS. It should also work on Linux systems. Since one dependency mcl is programmed with C, which is system-dependent, the provided compiling method works for MacOS and Linux.

4) Benchmarks: None

B. Artifact Installation & Configuration

1) Download the project. We have compiled and packaged the project into an artifact xx.jar in the provided archive.
2) Install JDK 17 or later version.
3) Install Gmp library. Gmp library is the prerequisite to using the mcl library. Please refer to the instructions gmp installment for both MacOS and Linux.
4) Compile mcl library for your system. Since the Java project uses a bilinear group which is implemented in mcl library (C/C++ implementation) for efficiency, before running the artifact to test, you should compile the underlying mcl library for your system. You can follow the instructions of Compiling mcl on Linux or macOS. To be concrete, first download the file install_fast_mcljava_linux_mac.sh. Then execute the file to compile mcl library and related headers with

```
$ ./install_fast_mcljava_linux_mac.sh $JAVA_HOME/include
  # $JAVA\_HOME is the Java installment directory
```

If it does not work, maybe you need to give permissions of the file to everyone:

```
$ chmod 777 foldername
```

Then execute the file again.
5) Finally, you are ready to test.

C. Major Claims

- (C1): The designed private and compilable cryptocurrency exchange scheme functions by supporting all four procedures: joining the platform, depositing coins to the platform, exchanging one kind of coin to another kind of coin without leaking the amount and kind of coins, and withdrawing coins to the blockchain anonymously. The implemented prototype is proven by running the experiment with four procedures, where the computation cost is reported in table I and the communication cost is reported in the paragraph *Communication cost* in section VI.
- (C2): The designed scheme is practical via comparison with the privacy-preserving incentive system (PPIS for short), another anonymous credential application. It is proven by running two schemes on the same environment and comparing the computation time cost of each related procedure, where PPIS's time cost is reported in Table II.
- (C3): The comparison between secure exchange and plain exchange and the comparison with Privacy Pass also demonstrates the practicality. Since all the needed experimental data is the same as C1, other estimated data is reported in the parts of *Comparison with plain exchange* and *Comparison with Privacy Pass and PPIS* in section VI.

D. Evaluation

1) Experiment (E1): [Pisces's computation time cost and communication size] [less than 10 human minutes + 2 computer minutes]: Pisces has four procedures: join, deposit, exchange, and withdraw. In each procedure, the user and the platform interact with each other. We test the computation time and communication size of each party in each procedure. The expected computation results are shown in table I, which is consistent with our tested results in Pisces/testlog.log. The expected communication size results are shown in *Communication cost* in Section VI of the full paper, which is also consistent with our tested results in Pisces/testlog.log and independent of the experiment environment.

[Preparation] Please follow the README in the provided repository to install the requirements. No further configurations are needed.

[Execution] Please go to the directory of Pisces (the full repository you download). and run the command:

```
$ java -jar uacs-1.0-SNAPSHOT-jar-with-dependencies.jar
```

[Results] The produced log shows the four procedures in sequence. For each procedure, the user interacts with the platform to run 100 times, the computation time is accumulated for each party over 100 iterations. The average time per iteration is shown in the terminal and log file, which should be consistent with the paper if the running environment is similar. In one iteration of each procedure, the byte length of messages each party sends is accumulated and printed out. The communication size should be independent of the running environment since the message length is inherent, not related to the environment.

2) Experiment (E2): [PPIS computation cost as a comparison] [Less than 5 human minutes + 2 computer minutes]: To show the practicality, we test another anonymous credential application PPIS in the same environment for comparison. PPIS only has three procedures: join, earn, and spend corresponding to Pisces' join, deposit, and withdraw, respectively. For each procedure, both the user and provider computation time are measured and accumulated. Then the average results among 100 iterations in milliseconds are printed.

[Preparation] No further configurations beyond E1.

[Execution] PPIS (also called UACS by the authors: updatable anonymous credential system) is implemented by the authors. This is the . We compile it package to the jar file in Pisces/uacs-1.0-SNAPSHOT-jar-with-dependencies.jar PPIS (also called UACS by the authors: updatable anonymous credential system) is implemented by the authors. This is the source code. We compile it and package it to the jar file in Pisces/uacs-1.0-SNAPSHOT-jar-with-dependencies.jar. You can directly run it with the command:

```
java -jar uacs-1.0-SNAPSHOT-jar-with-dependencies.jar
```

[Results] The produced log uacstestlog.log shows each party's running time of the three procedures in sequence in milliseconds. It also depends on the experimental environment. Even though it could be quite different for different computation powder, one sure thing is the results are comparable with Pisces' results in E1 and similar to each other as we claim in the paper.